MACIEJ BOROWIECKI
KRZYSZTOF CHECHŁACZ
KATARZYNA OLĘDZKA
AGNIESZKA SAMULSKA

RAMOWE PROGRAMY SZKOLEŃ

DLA NAUCZYCIELI SZKÓŁ PODSTAWOWYCH

Z ZAKRESU KSZTAŁCENIA MYŚLENIA ALGORYTMICZNEGO I NAUKI PROGRAMOWANIA

ORE Ośrodek Rozwoju Edukacji

MACIEJ BOROWIECKI
KRZYSZTOF CHECHŁACZ
KATARZYNA OLĘDZKA
AGNIESZKA SAMULSKA

RAMOWE PROGRAMY SZKOLEŃ

DLA NAUCZYCIELI SZKÓŁ PODSTAWOWYCH

Z ZAKRESU KSZTAŁCENIA MYŚLENIA ALGORYTMICZNEGO I NAUKI PROGRAMOWANIA

WARSZAWA 2018

Recenzja naukowa
dr Anna Beata Kwiatkowska – WMiI UMK

Redakcja naukowa
Maciej Borowiecki

Redakcja językowa i korekta
Karolina Strugińska

Opracowanie graficzne, projekt okładki
Wojciech Romerowicz
grafika: © Mimi Potter/Fotolia.com, © Arthead/Fotolia.com

Redakcja techniczna i skład
Wojciech Romerowicz

ISBN 978-83-66047-03-7

Warszawa 2018
Wydanie I

Ośrodek Rozwoju Edukacji
Aleje Ujazdowskie 28
00-478 Warszawa
www.ore.edu.pl
tel. 22 345 37 00
fax 22 345 37 70

ORE Ośrodek Rozwoju Edukacji

SPIS TREŚCI

KONCEPCJA I ZAŁOŻENIA OGÓLNE

Ramowe programy szkoleń dla nauczycieli szkół podstawowych z zakresu nauki programowania na I i II etapie edukacyjnym zostały opracowane zgodnie z nową podstawą programową edukacji informatycznej w ramach kształcenia wczesnoszkolnego oraz przedmiotu informatyka w klasach 4–8 szkoły podstawowej (Dz.U. z 2017 r., poz. 356). Należy podkreślić, że programy szkoleń dotyczą jedynie podniesienia kompetencji nauczycieli w zakresie algorytmicznego rozwiązywania problemów, myślenia komputacyjnego oraz programowania i nie wyczerpują wszystkich zapisów podstawy programowej dla edukacji informatycznej (w klasach 1–3) i przedmiotu informatyka (w klasach 4–8). Nie zastępują także studiów podyplomowych w zakresie przygotowania nauczycieli do nauczania przedmiotu informatyka, co więcej – wymaganiem wstępnym stawianym uczestnikom szkolenia (z wyjątkiem nauczycieli edukacji wczesnoszkolnej) jest posiadanie uprawnień do nauczania przedmiotu, a więc kierunkowe wykształcenie wyższe lub ukończone studia podyplomowe.

Nowa podstawa programowa w przypadku informatyki wprowadza bardzo poważne zmiany, które były postulowane przez środowiska informatyczne związane z edukacją od kilku lat. Już sama zmiana nazwy przedmiotu w szkole podstawowej z „zajęcia komputerowe” na „edukacja informatyczna” i „informatyka” świadczy o tym, że nowa podstawa kładzie nacisk na inne treści i umiejętności. Wprowadza powszechną naukę programowania na wszystkich etapach edukacyjnych. Programowanie należy tu rozumieć, jako całościowy proces rozwiązywania problemu od jego sformułowania, specyfikacji (określenia danych i wyników), poszukiwana metody rozwiązania (algorytmu/algorytmów jego rozwiązania), do opracowania i zaprogramowania rozwiązania, przetestowania i ewentualnej korekty, z wykorzystaniem odpowiednio dobranej do wieku uczniów aplikacji lub języka programowania.

Cele ogólne kształcenia informatycznego są takie same dla wszystkich etapów edukacyjnych. Opis wymagań szczegółowych ma charakter spiralny

(przyrostowy) – na każdym etapie edukacyjnym wymaga się od uczniów umiejętności zdobytych na wcześniejszych etapach edukacyjnych i rozszerza się je o umiejętności nowe. Tak też zostały przygotowane ramowe programy szkoleń dla nauczycieli.

Warto zauważyć, że umiejętności nabyte podczas zajęć informatyki, są przydatne zarówno na zajęciach innych przedmiotów, jak i w życiu codziennym. Umiejętności informatyczne są związane z rozwojem logicznego myślenia, precyzyjnego prezentowania myśli i pomysłów, zapisywania w języku formalnym tego, co rozumiemy intuicyjnie. Ponadto pomagają doskonalić umiejętność dobrej organizacji pracy oraz współpracy w grupie. Oczekuje się, że uczniowie wkraczający w zawodowe i dorosłe życie będą przygotowani do podjęcia obowiązków i wyzwań, jakie stawia przed nimi XXI wiek. Powinni zatem poznać podstawowe metody informatyki, aby w przyszłości stosować je w praktycznych sytuacjach w różnych dziedzinach.

Opracowane zostały trzy ramowe programy szkoleń (każde z nich obejmuje 40 godzin lekcyjnych zajęć stacjonarnych) oraz jeden program w wersji skróconej (10 godzin lekcyjnych zajęć stacjonarnych). Ze względu na specyfikę nauczania w klasach 1–3, w szczególności zalecenia, aby edukację informatyczną na tym poziomie prowadził nauczyciel nauczania zintegrowanego (a nie nauczyciel informatyki klas starszych) i łączył edukację informatyczną z pozostałymi edukacjami, program tego szkolenia należy traktować oddzielnie.

Treści nauczania dla klas 4–6 i 7–8 w podstawie programowej zostały sformułowane oddzielnie, w związku z tym proponowane są także dwa programy szkoleń, dla nauczających odpowiednio na poziomie klas 4–6 i 7–8. Należy je jednak traktować, jako propozycję dwuczęściowego szkolenia. Nauczyciele, którzy uczą lub planują uczyć, zarówno w klasach 4–6, jak i 7–8 powinni ukończyć obydwie części szkolenia. Dla nauczycieli, którzy już posiadają doświadczenie w realizacji treści programowych związanych z nauczaniem programowania, bądź realizowali zajęcia związane z programowaniem na zajęciach pozaszkolnych lub w ramach przedmiotu zajęcia komputerowe, przewidziana jest skrócona 10-godzinna wersja szkolenia na poziomie klas 4–6.

Każdy z programów zawiera informacje ogólne, wymagania wstępne stawiane uczestnikom szkoleń, cele szkolenia, treści nauczania, przykładowy rozkład materiału, omówienie poszczególnych tematów, przykładowe scenariusze zajęć

i zadania (w większości z rozwiązaniami). Omówiono poszczególne tematy dość obszernie, aby maksymalnie ułatwić placówkom doskonalenia nauczycieli przeprowadzenie szkolenia.

Na poziomie treści nauczania programy nie odwołują się do konkretnych narzędzi i języków programowania, mówią tylko o wykorzystaniu środowiska języka wizualnego na poziomie klas 1–3 oraz 4–6, natomiast języka tekstowego w klasach 7–8. W szkoleniu dotyczącym nauczania na poziomie klas 4–6 proponowane jest wprowadzenie podstaw języka tekstowego, aby przygotować nauczycieli i uczniów do łagodnego przejścia do programowania algorytmów w języku wysokiego poziomu w klasach najstarszych szkoły podstawowej.

W przykładowych rozkładach materiału programy odwołują się do konkretnych języków programowania. Omówienia poszczególnych tematów oraz przykładowe scenariusze zajęć i zadania zawierają często implementację rozwiązywanego problemu w konkretnym języku programowania. Jako język wizualny został wybrany najbardziej popularny obecnie Scratch, językiem programowania tekstowego jest Python (w wersji 3.*). Warto podkreślić, że translatory tych języków są powszechnie dostępne i bezpłatne. Treści nauczania można zrealizować, wybierając dowolne języki programowania wizualnego i tekstowego, należy wówczas opracować własny rozkład materiału.

Niektóre propozycje w programie dla klas 1–3 i 4–6 dotyczące wykorzystania środowiska Scratch powtarzają się, ponieważ grupy nauczycieli uczestniczących w tych szkoleniach prawdopodobnie będą rozłączne. Szkolenie na poziomie klas 1–3 jest adresowane przede wszystkim do nauczycieli edukacji wczesnoszkolnej, a pozostałe szkolenia – do nauczycieli informatyki.

AGNIESZKA SAMULSKA

RAMOWY PROGRAM SZKOLENIA DLA NAUCZYCIELI KLAS 1–3 (I ETAP EDUKACYJNY)

INFORMACJE OGÓLNE

Szkolenie jest przeznaczone dla nauczycieli edukacji wczesnoszkolnej oraz nauczycieli informatyki uczących lub planujących nauczać w klasach 1–3 szkół podstawowych. Jego głównym celem jest przygotowanie nauczycieli do realizacji nowej podstawy programowej edukacji informatycznej w ramach edukacji wczesnoszkolnej w zakresie rozumienia, analizowania i rozwiązywania problemów oraz programowania na poziomie klas 1–3. Szkolenie obejmuje 40 godzin lekcyjnych zajęć stacjonarnych.

Zajęcia mają z założenia charakter przede wszystkim warsztatowy, uczestnicy pod nadzorem prowadzącego samodzielnie rozwiązują problemy, w szczególności wcielają się w rolę ucznia. Część zajęć należy przeznaczyć na wykład i dyskusję oraz omówienie zagadnień metodycznych. Praca praktyczna pozwoli słuchaczom nabrać biegłości w posługiwaniu się narzędziami i metodami informatycznymi. Nie mniej ważna jest również refleksja pedagogiczna – w jakim celu wprowadzamy dane zagadnienia, jak zorganizować proces dydaktyczny, na co szczególnie zwrócić uwagę i jakie mogą wystąpić trudności. Podczas zajęć nie tylko prowadzący dzielą się swoją wiedzą, ale także słuchacze wymieniają się doświadczeniami.

WYMAGANIA WSTĘPNE STAWIANE UCZESTNIKOM SZKOLENIA

Uczestnik szkolenia powinien posiadać kompetencje wymienione w *Załączniku 1*, ponadto posiadać uprawnienia do nauczania edukacji wczesnoszkolnej lub informatyki w szkole podstawowej.

CELE SZKOLENIA:

- przygotowanie nauczycieli do prowadzenia zajęć edukacji informatycznej według nowej postawy programowej;
- kształcenie kompetencji nauczycieli w zakresie programowania z wykorzystaniem środowisk do programowania wizualnego;
- rozwijanie u uczniów umiejętności rozumienia, analizowania i rozwiązywania problemów – w tym myślenia algorytmicznego – poprzez dobór ciekawych zadań.

TREŚCI NAUCZANIA:

1. rozwijanie myślenia algorytmicznego dzieci młodszych poprzez gry i zabawy oraz dostosowane do ich wieku narzędzia TIK;
2. rozwiązywanie zadań, zagadek i łamigłówek prowadzących do odkrywania algorytmów;
3. rozwiązywanie problemów wymagających tworzenia sekwencji poleceń dla realizacji planu działania prowadzącego do osiągnięcia określonego celu;
4. zapisywanie efektów własnej pracy;
5. sterowanie obiektem za pomocą pojedynczych poleceń i ich sekwencji, z nawiązaniem do konkretnych form, metod i środków rozwijania u uczniów myślenia komputacyjnego;
6. wizualne programowanie prostych sytuacji lub historyjek, z uwzględnieniem pracy grupowej.

PRZYKŁADOWY ROZKŁAD MATERIAŁU:

Temat i tematy cząstkowe	Punkt podstawy programowej	Treści	Liczba godzin
1. Wprowadzenie			**2**
• Organizacja szkolenia • Dlaczego warto uczyć się programowania? Podstawa programowa edukacji informatycznej dla pierwszego etapu edukacyjnego • Jak prowadzić zajęcia z najmłodszymi	VII	1–6	0,5 0,5 1

2. Rozwijanie myślenia algorytmicznego			**9**
• Projekt „Informatyka bez komputera” • Konkurs informatyczny „Bóbr” • Gry i zabawy bez komputera • Projekt „Godzina Kodowania”	VII.1.1 VII.1.2 VII.1.3 VII.2.1	1, 2, 3, 5	1 2 3 3
3. Programowanie w ScratchJr			**3**
• Pierwsze kroki w ScratchJr • Przygotowujemy opowiadania multimedialne • Podsumowanie – giełda pomysłów	VII.1.1 VII.1.2 VII.2.1 VII.2.3	1, 3, 4, 5, 6	0,5 2 0,5
4. Programowanie w środowisku Scratch			**18**
• Pierwsze kroki w Scratchu • Opowiadania multimedialne • Gry i zabawy edukacyjne • Symulacje • Jak wykorzystać Scratcha na zajęciach • Realizujemy własne pomysły – praca zespołowa • Podsumowanie – programowanie w środowisku wizualnym	VII.1.1 VII.1.2 VII.2.1 VII.2.3	1, 3, 4, 5, 6	1 4 4 3 1 4 1
5. Praca z uczniami o różnych potrzebach edukacyjnych			**6**
• Praca z uczniem zdolnym, konkurs informatyczny „Bóbr” • Roboty i gry edukacyjne • Rozwijanie różnych zainteresowań uczniów	VII.1.1 VII.1.2 VII.1.3 VII.2.1	1, 2, 3, 5	2 3 1
6. Podsumowanie			**2**
• Scenariusze zajęć edukacyjnych uwzględniających edukację informatyczną • Najważniejsze umiejętności do zdobycia na pierwszym etapie edukacyjnym • Wsparcie dla nauczycieli – gdzie szukać inspiracji i pomocy?	VII	1–6	1 0,5 0,5

OMÓWIENIE POSZCZEGÓLNYCH TEMATÓW

1. Wprowadzenie

1.1. Organizacja szkolenia

Szkolenie rozpoczyna się od omówienia kwestii organizacyjnych oraz przedstawienia celów szkolenia i poruszanych tematów. Uczestnicy szkolenia powinni się przedstawić i krótko opowiedzieć o swojej pracy (w jakim wieku są ich uczniowie; w jakim charakterze pracują – np. nauczyciel edukacji wczesnoszkolnej, nauczyciel świetlicy, nauczyciel informatyki itd.; jakie warunki techniczne mają w placówkach do przeprowadzenia zajęć z edukacji informatycznej – np. klasa z komputerem oraz tablicą interaktywną, dostęp do pracowni komputerowej raz w tygodniu, pracownia mobilna, tablety itp.). Jest to cenna wiedza zarówno dla osoby prowadzącej szkolenie, jak i dla pozostałych uczestników szkolenia.

1.2. Dlaczego warto uczyć się programowania?

Podstawa programowa edukacji informatycznej dla pierwszego etapu edukacyjnego

Wygłoszony zostaje wykład zatytułowany: *Dlaczego warto uczyć się programowania?*, podczas którego należy m.in. przytoczyć fragmenty podstawy programowej dotyczące założeń kształcenia ogólnego w szkole podstawowej w odniesieniu do nauczania informatyki oraz omówić treści nauczania (wymagania szczegółowe) edukacji informatycznej dla pierwszego etapu edukacyjnego. Uczestnicy szkolenia powinni otrzymać wydruk stosownych fragmentów rozporządzenia (lub mieć dostęp do wersji elektronicznej fragmentów rozporządzenia).

Zasoby do wykorzystania:

- Rozmowa z prof. Mitchem Resnickiem z Massachusetts Institute of Technology (MIT) na temat potrzeby kształcenia innowacyjnych i kreatywnych osób;
- Wystąpienie prof. Mitcha Resnicka pt. *Nauczmy dzieciaki kodować*.

1.3. Jak prowadzić zajęcia z najmłodszymi

Po wykładzie powinna nastąpić dyskusja dotycząca wyzwań związanych z realizacją podstawy programowej oraz kwestii prowadzenia zajęć z dziećmi w klasach 1–3. Należy odwołać się do doświadczeń nauczycieli i ich propozycji w tej materii (prowadzący szkolenie powinien notować pomysły na tablicy).

Wnioski, które powinny pojawić się po dyskusji:

- nauka programowania powinna odbywać się interdyscyplinarnie (w połączeniu z innymi rodzajami edukacji, należy wykorzystywać nauczanie STEAM – ang. *Science*, *Technology*, *Engineering*, *Art i Mathematics* – nauki ścisłe, technologia, inżynieria, sztuka i matematyka);
- rozwiązywanie zadań, zagadek i łamigłówek prowadzących do odkrywania algorytmów powinno odbywać się także bez użycia technologii (w formie zabaw i gier bez wykorzystania komputera i innych urządzeń cyfrowych – „informatyka bez komputera”);
- zajęcia można prowadzić w różnych wariantach: w sali lekcyjnej z wykorzystaniem pracowni mobilnej, tabletów, używając własnego sprzętu (idea BYOD – ang. *Bring Your Own Device* – przynieś swe własne urządzenie), komputera i tablicy interaktywnej, laptopa oraz rzutnika w pracowni komputerowej;
- na zajęciach edukacji informatycznej można wykorzystać gry planszowe lub roboty;
- w sieci dostępnych jest wiele środowisk i aplikacji wspierających naukę programowania najmłodszych.

Zasoby do wykorzystania:

- Projekt „Zakazane piosenki” SP w Ząbkach;
- Pomysł na lekcje programowania: Lekcje programowania – część 1 i Lekcje programowania – część 2;
- Środowisko ScratchJr.

2. Rozwijanie myślenia algorytmicznego

2.1. Informatyka bez komputera

Można uczyć o informatyce – jej pojęciach i metodach – bez komputera, również poza pracownią komputerową. Ideę tę rozwinął Tim Bell z Nowej Zelandii. Takie podejście jest niezmiernie ważne podczas pracy z najmłodszymi dziećmi. Wiele zajęć

edukacji informatycznej powinno odbywać się w tej formie. Uczestnikom szkolenia należy zaprezentować przykłady filmów, w których uczniowie poznają zagadnienia informatyczne bez użycia komputera (Projekt „Computer Science Unplugged") oraz zawartość strony projektu „Informatyka dla Jasia i Joasi" propagującego to podejście. Z uczestnikami szkolenia należy zrealizować temat *Zliczanie kropek – system binarny* (http://jasijoasia.edu.pl/csu1.pdf) według podanego scenariusza.

Zasoby do wykorzystania:

- Film wprowadzający do projektu „Computer Science Unplugged";
- Film ilustrujący algorytmy sortowania;
- Film ilustrujący „grę w pomarańczę";
- Strona projektu „Informatyka dla Jasia i Joasi": http://jasijoasia.edu.pl;
- Strona Ośrodka Edukacji Informatycznej i Zastosowania Komputerów: http://programowanie.oeiizk.edu.pl/, (sekcja „Informatyka /prawie/ bez komputera").

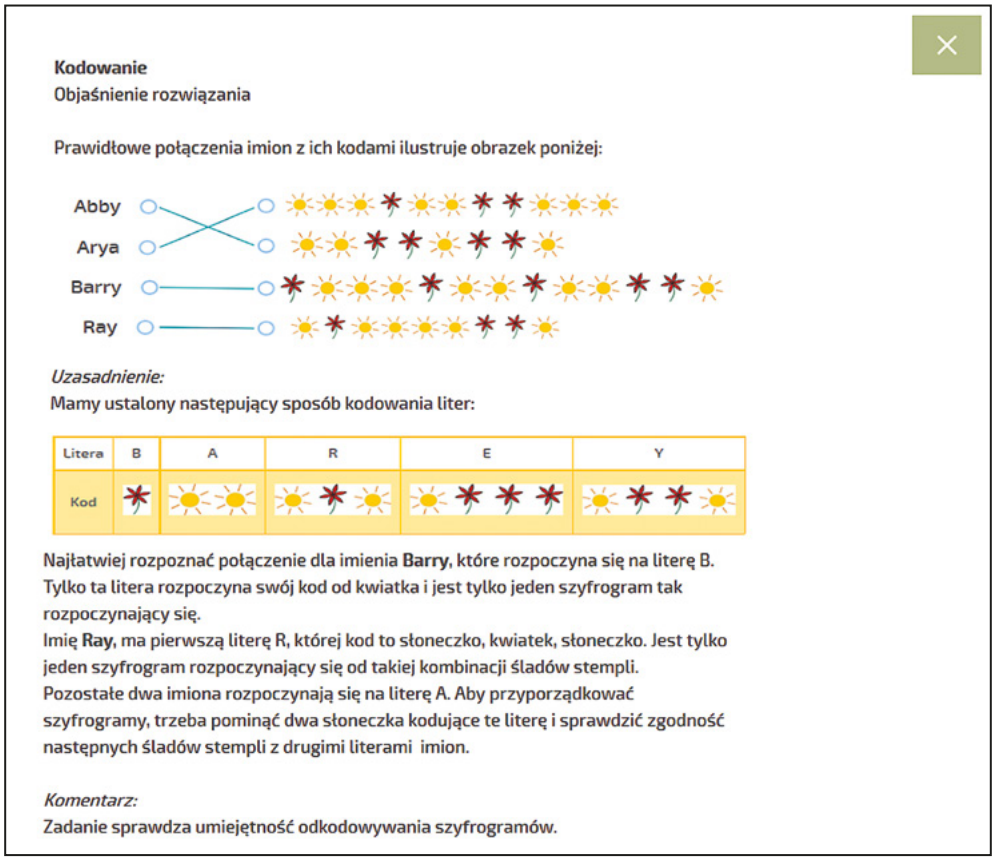
Wymienione powyżej filmy w serwisie YouTube są dostępne w angielskiej wersji językowej, więc warto włączyć polskie napisy.

2.2. Międzynarodowy Konkurs Informatyczny „Bóbr"

Głównym celem konkursu informatycznego „Bóbr" jest rozwijanie i kształtowanie myślenia algorytmicznego i komputacyjnego oraz popularyzacja posługiwania się technologią informacyjną i komunikacyjną wśród uczniów na wszystkich etapach edukacyjnych. Konkurs ma zasięg międzynarodowy i obejmuje cztery poziomy edukacyjne. Dla uczniów z klas 1–3 dedykowana jest kategoria Skrzat. Na stronie konkursu znajdują się zadania archiwalne w wersji konkursowej wraz z niezwykle cennym omówieniem (prawidłowa odpowiedź, ewentualnie uzasadnienie i komentarz), co może ułatwić pracę słuchaczy i być dla nich inspiracją. Warto wybrać kilka przykładowych zadań konkursu i omówić je z uczestnikami szkolenia.

Banner Międzynarodowego Konkursu Informatycznego „Bóbr":

Przykładowe zadanie za 4 punkty, listopad 2016 r.:



Kodowanie
Objaśnienie rozwiązania

Prawidłowe połączenia imion z ich kodami ilustruje obrazek poniżej:

Abby
Arya
Barry
Ray

Uzasadnienie:
Mamy ustalony następujący sposób kodowania liter:

Litera	B	A	R	E	Y
Kod					

Najłatwiej rozpoznać połączenie dla imienia **Barry**, które rozpoczyna się na literę B. Tylko ta litera rozpoczyna swój kod od kwiatka i jest tylko jeden szyfrogram tak rozpoczynający się.
Imię **Ray**, ma pierwszą literę R, której kod to słoneczko, kwiatek, słoneczko. Jest tylko jeden szyfrogram rozpoczynający się od takiej kombinacji śladów stempli.
Pozostałe dwa imiona rozpoczynają się na literę A. Aby przyporządkować szyfrogramy, trzeba pominąć dwa słoneczka kodujące te literę i sprawdzić zgodność następnych śladów stempli z drugimi literami imion.

Komentarz:
Zadanie sprawdza umiejętność odkodowywania szyfrogramów.

Zasoby do wykorzystania:

- Strona internetowa konkursu „Bóbr": https://www.bobr.edu.pl/.

2.3. Gry i zabawy bez komputera

Zajęcia należy podzielić na trzy części. W pierwszej z nich nauczyciele wykonują kolejne zadania algorytmiczne bez użycia komputera. Proponujemy zadania z wykorzystaniem szyfrowania (np. z wykorzystaniem dedykowanego zuchom szyfru GA-DE-RY-PO-LU-KI, szyfru opartego na tablicy kodów lub szyfru obrazkowego – przykładowe zadania 1–3).

Druga część zajęć, prowadzona w grupach 2–3 osobowych, polega na opracowaniu przez uczestników planu zajęć i kart pracy wraz z instrukcją do wykonania. Prowadzący zajęcia czynnie wspiera nauczycieli w realizacji zadania. Nauczyciele podczas pracy mogą korzystać z zasobów internetu w celu wyszukania wartościowych propozycji (np. ze strony konkursu „Bóbr") lub odwoływać się do własnych doświadczeń z tego zakresu.

Ostatnia część zajęć poświęcona jest prezentacji przygotowanych materiałów ze szczególnym uwzględnieniem kontekstu ich wykorzystania (np. odgadnięcie zaszyfrowanych wyrazów może stanowić wstęp do lekcji o darach jesieni lub zostać wykorzystane w ramach edukacji polonistycznej do odgadywania imion bohaterów baśni). W ten sposób nauczyciele stworzą bank gotowych pomysłów do realizacji z dziećmi.

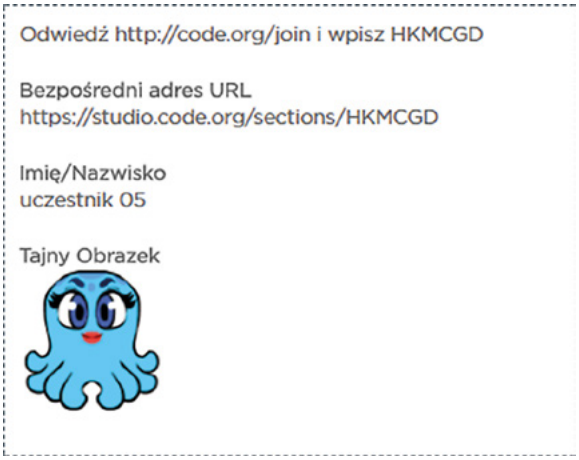
Zasoby do wykorzystania:

- Strona prezentująca szyfry harcerskie;
- Scenariusz zajęć na temat programowania bez komputera „Kodujemy kolorowo";
- Bank pomysłów: http://koduj.gov.pl/.

2.4. Projekt „Godzina Kodowania"

Przybliżamy uczestnikom szkolenia projekt „Godzina Kodowania", organizowany w ramach „Tygodnia Edukacji Informatycznej" (*Computer Science Education Week*, USA). Jest to inicjatywa adresowana do uczniów z całego świata. Można w niej uczestniczyć przez cały rok – zarówno w szkole, jak i w domu, wystarczy mieć dostęp do internetu. W ramach szkolenia należy omówić i zrealizować ze słuchaczami wybrane aktywności dedykowane dzieciom w wieku 6–8 lat – *Kurs 1*, oraz dzieciom w wieku 8–10 lat (czytającym) – *Kurs 2* (https://code.org/). Należy przed zajęciami założyć sekcje dla nauczycieli (loginy obrazkowe – *Kurs 1*, oraz loginy słowne – *Kurs 2*) i rozdać indywidualne karty z informacjami na temat zasad logowania wygenerowane przez system. Pozwoli to na pełne utożsamienie się uczestników szkolenia z rolą uczniów.

Przykładowa karta (Login obrazkowy):



Przykładowa karta (Login słowny):

Podczas indywidualnej pracy uczestników, trwającej około dwóch godzin, należy monitorować na bieżąco ich postępy i wspierać w pokonywaniu trudności. Problematyczne zadania trzeba omówić na forum (może to zrobić chętny słuchacz). Mile widziana jest pomoc koleżeńska. Po zakończeniu ćwiczeń należy pokusić się o refleksje związane z taką formą aktywności i stylem prowadzenia zajęć.

Spostrzeżenia, które powinny pojawić się w trakcie dyskusji:

- uczestnictwo w „Godzinie Kodowania" młodszych dzieci nie wymaga zakładania konta na portalu https://code.org/;
- ćwiczenia są dostosowanie do poziomu uczniów, ułożone zgodnie z ideą stopniowania trudności;
- dzieci stopniowo zaznajamiają się z kolejnymi zagadnieniami związanymi z programowaniem wizualnym i wprowadzane są w świat programowania poprzez zabawę;
- do ćwiczeń zostały opracowane filmy instruktażowe wspomagające naukę;
- ćwiczenia interaktywne są sprawdzane automatycznie – uczniowie natychmiast otrzymują informację zwrotną;
- w razie niepowodzenia można ponownie rozwiązać zadanie;
- niektóre zadania mogą być wykonane poprawnie na wiele sposobów (mniej lub bardziej optymalnie);
- zadania uczą porządkowania, rozumienia stosunków przestrzennych, układania sekwencji poleceń, testowania i poprawiania fragmentów kodu (debugowania), powtarzania poleceń, podejmowania decyzji;
- mnogość zadań pozwala na indywidualizację procesu nauczania oraz wykorzystanie materiałów z projektu „Godzina Kodowania" na różnych poziomach;
- wśród ćwiczeń występują aktywności niewymagające użycia komputera (*unplugged*).

Ostatnią część zajęć należy poświęcić na szczegółowe omówienie zawartej w projekcie procedury tworzenia i modyfikowania klas oraz sposobu śledzenia postępów uczniów. Do omówienia tego ostatniego zagadnienia można wykorzystać dane wypracowane przez uczestników szkolenia. Zaleca się, aby w praktyce przećwiczyli oni prezentowane treści, co wiąże się z koniecznością założenia kont nauczycielskich na platformie https://code.org/, utworzenia sekcji dla uczniów oraz wygenerowania kart z informacjami o logowaniu.

Zasoby do wykorzystania:

- Strona projektu „Godzina Kodowania": http://godzinakodowania.pl/;
- Strona platformy: https://code.org/;
- Strona OEIiZK – instrukcja dla nauczyciela: http://programowanie.oeiizk.edu.pl/, (sekcja „Polecamy/Godzina Kodowania").

3. Programowanie w ScratchJr

3.1. Pierwsze kroki w ScratchJr

ScratchJr to środowisko programowania wizualnego dogodne dla najmłodszych dzieci, które jeszcze nie opanowały umiejętności czytania. Środowisko działa na urządzeniach iPad oraz na tabletach z systemem Android czy Google Chrome (7-calowych lub większych). Na stronie https://www.scratchjr.org/ (dostępnej w języku angielskim) znajduje się szczegółowy opis środowiska oraz materiały instruktażowe i metodyczne dla nauczycieli. Zaleca się, żeby czynności wymagające użycia urządzeń mobilnych poprzedzać aktywnościami *unplugged*. Do tego celu warto przygotować komplet wydrukowanych bloków środowiska (materiał dostępny w postaci pliku pdf na stronie https://www.scratchjr.org/).

Uwaga: istnieje możliwość zainstalowania środowiska ScratchJr na komputerach stacjonarnych z wykorzystaniem emulatora Androida.

Zasoby do wykorzystania:

- Strona domowa Scratch: https://www.scratchjr.org/;
- Bloki ScratchJr.

3.2. Przygotowujemy opowiadania multimedialne

Uczestnicy szkolenia powinni mieć możliwość praktycznego zapoznania się ze środowiskiem (mogą pracować w parach). Należy wykonać kilka prostych ćwiczeń

wprowadzających, a następnie przejść do realizacji projektu multimedialnego (przykładowe zadanie 4). W sieci jest dostępnych wiele ciekawych scenariuszy.

Zasoby do wykorzystania:

- Karty pracy w ScratchJr;
- Scenariusze z projektu „Mistrzowie Kodowania";
- Scenariusze ze strony Super Koderzy.

3.3. Podsumowanie – giełda pomysłów

Na zakończenie należy przeprowadzić dyskusję dotyczącą tego, jakie tematy z zakresu edukacji wczesnoszkolnej (nie informatycznej) mogą być wspierane realizacją projektów w środowisku ScratchJr (np. pory roku, zwierzęta w zagrodzie, kolory jesieni itp.). Dyskusja powinna być poprzedzona krótką pracą w parach. Nauczyciele ponownie tworzą bank gotowych pomysłów do realizacji z dziećmi.

4. Programowanie w środowisku Scratch

4.1. Pierwsze kroki w Scratchu

O ile ScratchJr jest środowiskiem przeznaczonym do pracy z urządzeniami mobilnymi, to Scratch nie ma już takich ograniczeń. Zalecamy pracę w tym środowisku z dziećmi, które umieją już czytać (pod koniec pierwszej klasy lub w klasie drugiej). Wskazane jest wykonanie z dziećmi wybranych ćwiczeń dostępnych w materiałach „Godziny Kodowania" w ramach zasobu *Kurs 2*.

Zajęcia z nauczycielami należy rozpocząć od prezentacji środowiska Scratch. Wskazujemy wady i zalety pracy online i offline (m.in. ze względu na zalety pracy w chmurze). Po utworzeniu i potwierdzeniu kont należy wykonać ćwiczenia opisane w scenariuszu *Pierwsze kroki w Scratchu*. Można też na zajęciach wykorzystać karty Scratcha.

Uwaga: pełna funkcjonalność środowiska w wersji online wiąże się z koniecznością posiadania konta pocztowego. Problem stanowi zatem wiek uczniów szkoły podstawowej – dzieci te są zbyt małe, aby posiadać konto pocztowe i swobodnie się nim posługiwać. Dlatego proponowanym rozwiązaniem jest ścisła współpraca z rodzicami w celu założenia dziecku konta na portalu https://scratch.mit.edu/. Warto rodzicom przedstawić walory edukacyjne środowiska i zapewnić ich, że założenie konta nie wiąże się

z otrzymywaniem niechcianej korespondencji (tzw. spamu). Na adres podany w procesie rejestracji przyjdzie link aktywacyjny potrzebny do uzyskania pełnej funkcjonalności konta.

Zasoby do wykorzystania:

- Strona OEIiZK – opis środowiska Scratch i „Słowniczek bloczków", (sekcja „Warszawa programuje!");
- Strona OEIiZK – scenariusz ćwiczeń *Pierwsze kroki w Scratchu*, (sekcja „Mistrzowie Kodowania");
- Zabawy tematyczne Karty Scratch.

4.2. Opowiadania multimedialne

Uczestnicy szkolenia mają już wprawę w realizowaniu projektów w środowisku ScratchJr. Należy ponownie podkreślić, że dobór tematyki opowiadań multimedialnych musi mieć na celu integrację programowania z pozostałymi sferami edukacji. W proponowanym przykładzie wspieramy edukację polonistyczną. Projekt „Dialog" cechuje prostota zastosowanych skryptów oraz możliwość wizualizacji dowolnej rozmowy między duszkami. Dialog odbywa się na zasadzie znanej dzieciom z życia codziennego – w momencie, gdy jedna osoba mówi, druga cierpliwie czeka. Do przykładowej wizualizacji wybrany został fragment wiersza pt. *ZOO* Jana Brzechwy.

Uwaga: Projekty realizowane w środowisku Scratch są często rozbudowane, dlatego zaleca się ich wykonywanie krok po kroku i częste testowanie poszczególnych fragmentów kodów.

Zasoby do wykorzystania:

- Strona OEIiZK – scenariusz *1: Dialog*, (sekcja „Warszawa programuje!").

Druga część zajęć poświęcona jest multimediom. Kolejny projekt pozwala w oparciu o proste przykłady zrozumieć ideę nadawania i odbierania komunikatów. Wykorzystanie bloczków związanych z dźwiękami zwiększa jego atrakcyjność. Na scenie pojawiają się kolejno duszki: kot spacerujący w rytm znanej piosenki *Wlazł kotek na płotek*, werbel symulujący grę na różnych instrumentach, szczekający pies. Zastosowane efekty mają na celu pokazanie możliwości dźwiękowych środowiska Scratch.

Zasoby do wykorzystania:

- Strona OEIiZK – scenariusz 4: *Opowiadanie multimedialne*, (sekcja „Warszawa programuje!").

Podsumowaniem tej części zajęć powinna być kolejna giełda pomysłów na temat: realizację jakich treści z podstawy programowej innych niż z zakresu edukacji informatycznej wspiera Scratch? (np. edukację polonistyczną: czytanie, pisanie, samokształcenie).

4.3. Gry i zabawy edukacyjne

Projekty z tej grupy cechuje interakcja z użytkownikami. Stosuje się w nich sterowanie klawiaturą lub myszką. Użytkownicy przygotowanej aplikacji nie są wyłącznie biernymi obserwatorami, ale mają wpływ na to, co się dzieje na ekranie. Projektowaniu i tworzeniu gier przez uczniów towarzyszy ogromne zaangażowanie.

Projekt „Zdrowe odżywianie" stanowi przykład wykorzystania prostego sterowania. Użytkownik może kierować duszkiem za pomocą strzałek klawiatury – żuczek obraca się w lewo lub prawo o 90° i porusza się naprzód. Na scenie znajdują się także inne duszki (np. jabłko, ciastko) reprezentujące zdrowe produkty oraz żywność tzw. śmieciową. Po dotknięciu danego obiektu przez żuczka pojawia się stosowny komunikat – w zależności od kategorii duszka, który został dotknięty. „Zjedzone" przez żuczka produkty znikają ze sceny.

Należy zauważyć, że w prezentowanych scenariuszach znajdują się propozycje modyfikacji i zadań do samodzielnego wykonania. Służą one indywidualizacji procesu nauczania i mogą również zostać wykorzystane do pracy z nauczycielami. Grupa uczestników szkolenia zapewne okaże się na tyle zróżnicowana, że wiele z propozycji zostanie wykorzystanych.

Zasoby do wykorzystania:

- Strona OEIiZK – scenariusz 2: *Zdrowe odżywianie*, (sekcja „Warszawa programuje!").

„Akwarium" można zaliczyć do grupy projektów symulacyjnych, ponieważ naśladuje zachowanie ryb, ale zawiera również elementy związane z projektowaniem gier. Zakłada on interakcję z użytkownikiem oraz zliczanie punktów. Ruch ryb pływających w akwarium sprawia wrażenie chaotycznego i bezcelowego, ale gdy „pukamy w szybę" (oczywiście wirtualnie, używając

do tego celu myszki), zaciekawione zwierzęta poruszają się w stronę miejsca, w które zapukano. W akwarium znajduje się też rybka drapieżna, która może atakować inne. Powoduje to naturalną potrzebę wprowadzenia zmiennej do zliczania zjedzonych rybek.

Zasoby do wykorzystania:

- Strona OEIiZK – scenariusz 3: *Akwarium*, (sekcja „Warszawa programuje!").

Ostatnim projektem będzie stworzenie aplikacji „Porządki" (przykładowy scenariusz 1). Projekt ten stanowi kolejny sposób zintegrowania edukacji przyrodniczej i informatycznej. Uczniowie tworzą w ramach jego działania aplikację, w której uczą się sposobu segregowania odpadów, a przy okazji programują wizualnie w środowisku Scratch codzienne czynności. Wykorzystamy w niej instrukcję warunkową oraz nadawanie i odbieranie komunikatów.

4.4. Symulacje

Symulacja to odtwarzanie pewnych sytuacji za pomocą modelu. Projekty symulacyjne pozwalają na zrozumienie zjawisk występujących wokół nas. Proponujemy zaprezentować nauczycielom na zajęciach dwa tego typu projekty do wykorzystania na lekcjach edukacji wczesnoszkolnej: „Śluza wodna" i „Elektrownia wiatrowa". Nie są to projekty przeznaczone do realizacji z dziećmi. Na zajęciach z nauczycielami należy zrealizować fragment drugiego projektu – symulator wiatraka (przykładowy scenariusz 2), a na zakończenie tej części zajęć – ostatni projekt „Kot i balony" (przykładowe zadanie 5). W tym projekcie zostanie ponownie wykorzystane urządzenie zewnętrzne (mikrofon).

Zasoby do wykorzystania:

- Projekt „Śluza wodna";
- Projekt „Elektrownia wiatrowa".

4.5. Jak wykorzystać Scratcha na zajęciach

Portal Scratch zawiera miliony projektów udostępnionych przez użytkowników środowiska. Znajduje się tam wiele wartościowych pomysłów. Proponujemy przejrzenie zasobów portalu https://scratch.mit.edu/ pod kątem poszukiwania ciekawych projektów do wykorzystania na lekcjach edukacji wczesnoszkolnej. Należy zwrócić uwagę nauczycielom na fakt, że do udostępniania

projektów konieczne jest posiadanie konta potwierdzonego w procesie rejestracji – w przeciwnym wypadku użytkownik nie ma możliwości dzielenia się swoimi pracami. Wskazane jest zademonstrowanie procesu udostępniania projektu (kod embed i link do projektu). Omówienia wymaga także sposób założenia studia przydatnego m.in. do gromadzenia prac uczniowskich – tworzymy w ten sposób wirtualną ścianę z pracami dzieci. Warto też przedstawić sposób i ideę opracowywania remiksów. Na koniec każdy z nauczycieli powinien publicznie zaprezentować projekt, który znalazł, i opowiedzieć o jego walorach edukacyjnych.

4.6. Realizujemy własne pomysły – praca zespołowa

Gdy uczestnicy szkolenia mają już wprawę i pewne wyczucie związane z programowaniem w Scratchu, proponujemy samodzielne wykonanie karty okolicznościowej (przykładowe zadanie 6) oraz opracowanie i stworzenie aplikacji wspierającej realizację wybranej lekcji z zakresu edukacji wczesnoszkolnej (przykładowe zadanie 7). Podczas realizacji tego drugiego zadania zalecana jest praca w zespołach dwuosobowych oparta na praktycznym pokazie jednej z metod pracy z uczniami.

Nauczyciele muszą przejść wszystkie etapy tworzenia aplikacji:

- sprecyzowanie pomysłu,
- stworzenie planu działania (jakich elementów potrzebujemy, czy są dostępne w bibliotece środowiska, co się będzie działo w aplikacji – jaki będzie scenariusz gry),
- wykonanie aplikacji wraz z jej testowaniem krok po kroku.

W tworzonych aplikacjach nauczyciele powinni wykorzystać niektóre z wcześniej poznanych mechanizmów interakcji z użytkownikiem (sterowanie klawiaturą, przeciąganie duszków po ekranie, kliknięcie w duszka, sterowanie głosem).

4.7. Podsumowanie – programowanie w środowisku wizualnym

Czas na podsumowanie programowania wizualnego. W czasie dyskusji należy zwrócić uwagę nauczycieli na następujące kwestie:

- zanim zaczniemy programować wizualnie, programujemy w formie zabawy – bez użycia komputera;
- naukę programowania warto zacząć od „Godziny Kodowania” – *Kurs 1*;

- w pracy z uczniami najmłodszymi wykorzystujemy środowisko ScratchJr, mając na uwadze to, że edukacja informatyczna powinna wspierać inne rodzaje edukacji w ramach kształcenia zintegrowanego;
- wstępem do kolejnego etapu może być ponownie „Godzina Kodowania" – *Kurs 2*;
- programowanie wizualne w Scratchu zaczynamy od bardzo prostych projektów, powoli stopniując skalę trudności;
- źródłami materiałów i inspiracji do pracy ze ScratchJr i Scratch mogą być instrukcje pracy z programem, materiały dla nauczyciela, karty pracy, karty Scratcha (bardzo krótkie projekty), projekty „Warszawa programuje!", „Godzina Kodowania" i program edukacyjny „Mistrzowie Kodowania".

Należy także uwzględnić propozycje nauczycieli.

5. Praca z uczniami o różnych potrzebach edukacyjnych

5.1. Praca z uczniem zdolnym, Międzynarodowy Konkurs Informatyczny „Bóbr"

Uczestnikom szkolenia należy przedstawić szczegółowe informacje na temat Konkursu Informatycznego „Bóbr", w szczególności w kontekście pracy z uczniem zdolnym (strona konkursu, regulamin i zasady organizacji). W pierwszej fazie zajęć praktycznych nauczyciele rozwiązują samodzielnie zadania z roku 2016 lub 2015 (w wersji konkursowej). Druga część zajęć opiera się na wspólnym omówieniu rozwiązań wybranych zadań. Zadania konkursowe polegają na rozwiązywaniu różnorodnych problemów. Warto wybrać przynajmniej po jednym (najciekawszym) zadaniu z kategorii takich jak:

- spostrzegawczość,
- analiza informacji,
- wybór obiektów ze względu na ich cechy charakterystyczne,
- układanie zdarzeń w logicznym porządku,
- odczytywanie zakodowanej informacji; odczytywanie informacji zgodnie z przyjętym schematem,
- wykonywanie sekwencji poleceń sterowania robotem na ekranie komputera,
- powtarzanie, łączenie pojedynczych czynności w sekwencję wydarzeń, która później jest powtarzana,
- sortowanie,
- wyszukiwanie najmniejszej/największej wartości spełniającej podany warunek,

- zagadnienia grafowe,
- inne.

Zasoby do wykorzystania:

- Strona Konkursu Informatycznego „Bóbr": https://www.bobr.edu.pl/.

5.2. Roboty i gry edukacyjne

Wiele ciekawych i atrakcyjnych zajęć można przeprowadzić na poziomie klas 1–3 z wykorzystaniem robotów. Sterować robotami przeznaczonymi dla uczniów najmłodszych (np. Ozobot, WeDo, Dash&Dot, Photon, mBot) można za pomocą dedykowanych aplikacji lub programowania wizualnego. Na szkoleniu należy wspomnieć także o grach edukacyjnych, np. CodyRoby. Warto zaprezentować „na żywo" działanie tej gry (instrukcja i akcesoria do gry dostępnie na stronie http://koduj.gov.pl/). Następnie uczestnicy szkolenia, podzieleni na zespoły 2–3 osobowe, przygotowują i prezentują krótkie recenzje wybranej gry lub robota, uwzględniając dostępność oraz wartość materiałów metodycznych dla nauczyciela. Tę ostatnią aktywność zalecamy zastąpić rzeczywistym testowaniem pomocy dydaktycznych posiadanych przez placówkę organizującą szkolenie (robot, gra planszowa) i dyskusją o ich użyteczności.

Zasoby do wykorzystania:

- Strona Kodable (gra bezpłatna w wersji podstawowej);
- Gry do nauki programowania Blockly Games;
- Strona edukacyjna Trasa Świętego Mikołaja;
- Gra Run Marco;
- CodyRoby – karciana gra do nauki programowania: http://koduj.gov.pl/cody-roby-kodowanie-w-formie-gry-karcianej/;
- Program Lightbot.

Fakultatywnie na zajęciach można wykorzystać roboty i planszowe gry edukacyjne. Poniżej linki do najbardziej popularnych – znajdziemy tam także scenariusze zajęć:

- Robot Ozobot – informacje;
- Zestawy Lego WeDo;
- Bloki Scratcha w integracji z Lego WeDo;
- Strona producenta Lego WeDo;
- Scenariusze lekcji z wykorzystaniem robotów Dash i Dot;
- Robot Photon – informacje;
- Robot mBot – informacje;
- Gra Scottie Go!.

Należy zwrócić uwagę, że wymogi podstawy programowej można zrealizować bez wykorzystania robotów, które jednak wzbogacają i uatrakcyjniają zajęcia. Warto zatem wykorzystać sprzęt, którym dysponujemy, lub zakupić go, jeśli placówka ma taką możliwość.

5.3. Rozwijanie różnych zainteresowań uczniów

Ostatnią część zajęć proponujemy poświęcić na przygotowanie indywidualnej ścieżki rozwoju ucznia (np. szczególnie uzdolnionego lub zaniedbanego środowiskowo). Nauczyciele mogą pracować w parach. Opracowanie powinno zawierać krótką charakterystykę ucznia (jego deficytów lub uzdolnień) oraz propozycję zadań i aktywności stawianych przed nim. Jako alternatywne zadanie można zaproponować przygotowanie planu zajęć pozalekcyjnych (np. z algorytmiki i programowania dla najmłodszych, robotyki, gier i zabaw logicznych, projekt grupowy do wykonania itp.). Uczestnicy szkolenia powinni zaprezentować swoje propozycje.

6. Podsumowanie

6.1. Scenariusze zajęć edukacyjnych uwzględniających edukację informatyczną

Jedną z form podsumowania szkolenia może być opracowanie konspektu zajęć z zakresu edukacji wczesnoszkolnej wspieranych edukacją informatyczną. Ważne, aby zajęcia nie były dedykowane wyłącznie zagadnieniom informatycznym. Konspekt musi zawierać treści z podstawy programowej, zarysowane działania, aktywności podejmowane podczas lekcji. Praca nad konspektem powinna odbywać się w zespołach 2–3 osobowych i zakończyć się prezentacją propozycji.

Zasoby do wykorzystania:

- Scenariusz *Poznajmy język robotów* (edukacja polonistyczna, plastyczna, społeczna i matematyczna);
- Gra Code Grid Math (edukacja matematyczna);
- Strona projektu „Mistrzowie kodowania” (różne dziedziny edukacji).

6.2. Najważniejsze umiejętności do zdobycia na pierwszym etapie edukacyjnym

W formie dyskusji należy przypomnieć kompetencje konieczne do zdobycia na pierwszym etapie edukacyjnym, takie jak umiejętność:

- logicznego i algorytmicznego myślenia (rozwijanego poprzez gry i zabawy oraz dostosowane do wieku uczniów narzędzia TIK);
- rozwiązywania zadań, zagadek i łamigłówek prowadzących do odkrywania algorytmów;
- rozwiązywania problemów wymagających tworzenia sekwencji poleceń mających posłużyć realizacji planu działania prowadzącego do osiągnięcia określonego celu;
- sterowania obiektem za pomocą pojedynczych poleceń i ich sekwencji;
- wizualnego opracowywania prostych sytuacji lub historyjek z uwzględnieniem pracy grupowej;
- zapisywania efektów własnej pracy.

Uczniowie powinni umieć zastosować informatyczne podejście do rozwiązywania problemu:

- sprecyzować cel (np. stworzenie multimedialnej kartki dla mamy);
- opracować rozwiązanie (np. stworzyć plan działania: jakich elementów potrzebujemy, co się będzie działo na ekranie, jaki będzie scenariusz);
- zaprogramować rozwiązanie i je przetestować (np. wykonać aplikację w środowisku Scratch i sprawdzić jej działanie krok po kroku).

Każdy z powyższych punktów należy szczegółowo omówić, przytaczając przykłady zadań i narzędzi oraz aplikacji potrzebnych do ich zrealizowania na zajęciach edukacji wczesnoszkolnej. Wszystkie pomysły powinny zostać zanotowane na tablicy przez osobę prowadzącą szkolenie.

6.3. Wsparcie dla nauczycieli – gdzie szukać inspiracji i pomocy?

Na koniec warto przeprowadzić dyskusję dotyczącą ciekawych projektów i inicjatyw edukacyjnych skierowanych do najmłodszych, odwołując się do doświadczeń i wiedzy nauczycieli (lokalne konkursy, serwisy ze scenariuszami zajęć, projekty regionalne, ogólnopolskie i międzynarodowe itp.). Można też przypomnieć adresy wszystkich najważniejszych stron odwiedzonych podczas szkolenia:

- https://code.org/;
- http://koduj.gov.pl/;
- https://www.scratchjr.org/;
- https://scratch.mit.edu/;
- http://programowanie.oeiizk.edu.pl;
- http://wiki.mistrzowiekodowania.pl;
- https://www.bobr.edu.pl.

PRZYKŁADOWE SCENARIUSZE ZAJĘĆ

Scenariusz 1 – *Porządki*

Opis zajęć

Projekt „Porządki” adresowany do uczniów klasy 2 lub 3 to sposób na zintegrowanie edukacji przyrodniczej i informatycznej. Uczniowie tworzą aplikację, dzięki której uczą się sposobu segregowania odpadów, a przy okazji programują wizualnie codzienne czynności w środowisku Scratch. Wykorzystujemy w niej instrukcję warunkową oraz nadawanie i odbieranie komunikatów.

Rysunek 1. Projekt „Porządki”

Czas trwania

45 minut

Niezbędne zasoby

Wykorzystujemy tło minimum dwukolorowe o rozmiarze 480x360 pikseli o barwach odpowiadających kolorom pojemników do segregowania odpadów (niebieski, zielony, żółty, brązowy). W przykładowym projekcie zastosowano trzy kolory. Dodatkowo kolorowe pola podpisano nazwami segregowanych odpadów. Tło można narysować w edytorze grafiki środowiska Scratch lub przygotować w innym edytorze grafiki. Postaci duszków – pliki graficzne (svg lub png) –pobieramy z internetu (https://openclipart.org/).

Realizacja

Projekt przygotujemy etapami:

- 1. etap – przygotowanie tła i duszków z opcją przeciągania po ekranie;

- 2. etap – sprawdzanie, czy segregowane obiekty znajdują się na właściwych polach;
- 3. etap – dodanie zmiennych umożliwiających zliczanie odpadów poprawnie i błędnie posegregowanych.

Część 1 – dodajemy grafikę do projektu

Pierwszy etap projektu polega na zmianie tła oraz wczytaniu postaci duszków. Tło do projektu wczytujemy z pliku lub samodzielnie malujemy z wykorzystaniem edytora grafiki środowiska Scratch. Ta druga opcja wydłuży czas realizacji projektu. Postaci duszków wczytujemy z plików. Ze względu na to, że nazwa pliku staje się nazwą duszka, należy zadbać o jej adekwatność (np. butelka.png, słoik.png, papier.svg itp.).

Realizacja projektu polega na segregowaniu duszków symbolizujących różne rodzaje odpadów. Działanie odbywa się z użyciem techniki: przeciągnij i upuść. Standardowo w odtwarzaczu opcja przeciągania nie działa. W projekcie musimy to zmienić, zaznaczając właściwą opcję we właściwościach duszka.

Rysunek 2. Zmiana ustawień duszka – przeciąganie w odtwarzaczu

Po pierwszym etapie projekt posiada już funkcjonalność, którą można wykorzystać na zajęciach z uczniami. Mając do dyspozycji tablicę interaktywną, w ciekawy sposób możemy omówić z dziećmi zasady segregowania śmieci oraz przyporządkować kolorom pojemników składowane w nich odpady.

Część 2 – tworzymy skrypty

Drugi etap projektu umożliwia automatyczne sprawdzenie, czy odpady zostały posegregowane prawidłowo. Dzięki temu aplikacja zyskuje na atrakcyjności. W projekcie został wykorzystany duszek-przycisk (*Button2*) z biblioteki środowiska Scratch. Korzystając z edytora grafiki, na przycisku umieszczamy napis: `sprawdź`.

Sprawdzenie prawidłowości ułożenia odpadów realizowane jest poprzez nadawanie i odbieranie komunikatów. Po kliknięciu w przycisk nadany zostaje do wszystkich duszków komunikat o nazwie `sprawdzenie`. Po odebraniu komunikatu duszki-odpady sprawdzają, czy zostały umieszczone na obszarach o właściwych kolorach.

Rysunek 3. Skrypt przycisku

Rysunek 4. Skrypty duszków-odpadów

Musimy zadbać o to, aby wskazać kolor właściwy dla danego duszka w czujniku `dotyka koloru?`.

Część 3 – testujemy i udoskonalamy projekt

Przystępując do wstępnego testowania projektu, należy sprawdzić położenie duszków i pojawiające się komunikaty. W momencie gdy ta część projektu zostanie poprawnie wykonana, możemy przystąpić do następnych czynności.

Kolejno wprowadzamy następujące poprawki:

- losowanie pozycji duszków-odpadów;

  Projekt uruchamiamy skryptem zielonej flagi. Dbamy, aby duszki-odpady zmieniały swoją pozycję podczas inicjacji projektu.

Rysunek 5. Skrypty zielonej flagi duszków-odpadów

- ukrycie przycisku po jego kliknięciu;

Rysunek 6. Modyfikacja skryptu przycisku

Ukrywając duszka, należy pamiętać, aby pokazać go na początku w ramach skryptu zielonej flagi.

Rysunek 7. Skrypt zielonej flagi przycisku

Część 4 – dodajemy nową funkcjonalność

Ostatni etap projektu ma fakultatywny charakter. Wykorzystujemy dwie zmienne: do zliczania ustawień poprawnych oraz błędnych. Skrypt zerowania zmiennych przypisać można różnym obiektom, np. przyciskowi.

Rysunek 8. Modyfikacja skryptu zielonej flagi przycisku

Poprawki należy również wprowadzić w skryptach duszków-odpadów.

Rysunek 9. Modyfikacja skryptów duszków-odpadów

Podsumowanie

Podczas realizacji tego projektu omawiamy szereg zagadnień informatycznych. Mamy tu do czynienia z jednoczesnym sterowaniem wieloma obiektami na ekranie, instrukcją warunkową oraz zmiennymi. Projekt ten umożliwia wprowadzenie trudnych zagadnień w przystępny sposób. Lepiej, by nie był to pierwszy projekt,

w którym dzieci posługują się zmiennymi. Warto pojęcie zmiennej wprowadzić, realizując uprzednio prostszy projekt, w którym występuje jeden licznik.

Projekt można poprzedzić zajęciami polegającymi na wyszukaniu grafiki. Realizujemy w ten sposób kolejne treści z podstawy programowej.

Mając do dyspozycji tablicę interaktywną, można tego typu projekt tworzyć i testować zespołowo. Korzystając z przedstawionego mechanizmu, możemy porządkować różne elementy. W zależności od potrzeb związanych z realizacją kolejnych treści edukacji wczesnoszkolnej mogą to być: owoce i warzywa, zwierzęta i rośliny, figury geometryczne itd.

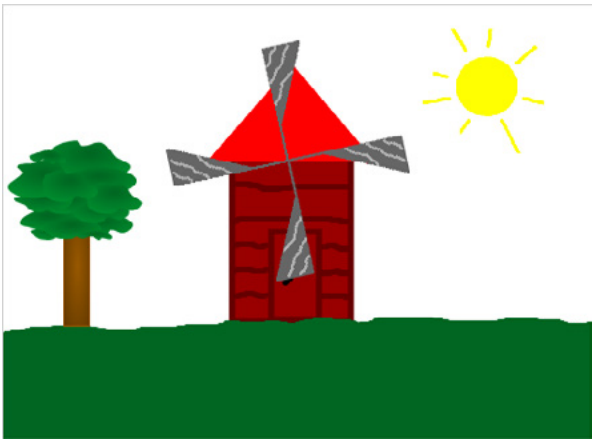
Proponowane modyfikacje i zadania do samodzielnego wykonania:

- wstawienie kolejnych duszków-odpadów;
- dodanie kolejnego koloru (brązowy – odpady biodegradowalne);
- znikanie duszków-odpadów poprawnie posegregowanych.

Scenariusz 2 – *Wiatrak*

Opis zajęć

Projekt „Wiatrak” adresowany do uczniów klasy 1 lub 2 integruje treści edukacji matematycznej, przyrodniczej, plastycznej i informatycznej. Uczniowie, tworząc symulator wiatraka, rozwijają ekspresję twórczą, rozumienie stosunków przestrzennych oraz pojęć geometrycznych, w tym zagadnień związanych z symetrią. Na podstawie obserwacji wyjaśniają istotę obserwowanego zjawiska oraz potrafią je zaprogramować wizualnie.



Rysunek 10. Projekt „Wiatrak”

Czas trwania

45 minut

Niezbędne zasoby

Jako elementy do tworzenia projektu uczniowie wykorzystują prace plastyczne wykonane w edytorze graficznym Scratcha. Ewentualnie symulacja może zostać wzbogacona o duszki pobrane z biblioteki (np. *Tree1*).

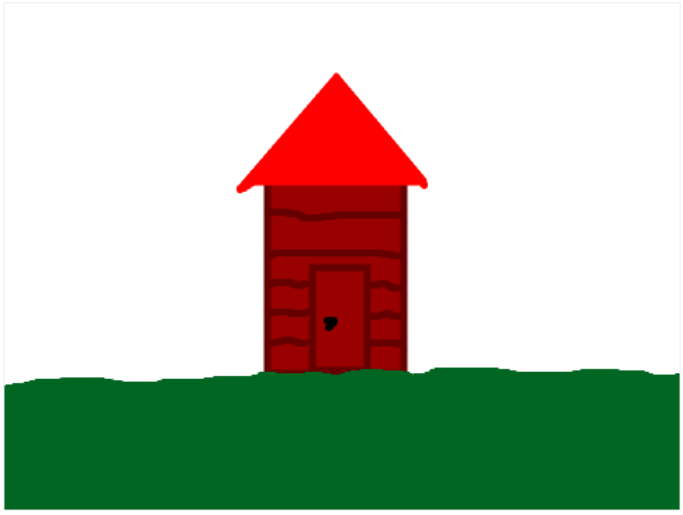
Realizacja

Projekt tworzymy etapami:

- 1. etap – przygotowanie tła i duszków w edytorze graficznym;
- 2. etap – wprawienie wiatraka w ruch;
- 3. etap – modyfikacja skryptu sterującego wiatrakiem.

Część 1 – przygotowujemy grafikę do projektu

Pierwszy etap projektu polega na przygotowaniu grafiki w edytorze Scratcha. Tło będzie stanowił rysunek budynku wiatraka oraz jego otoczenia.

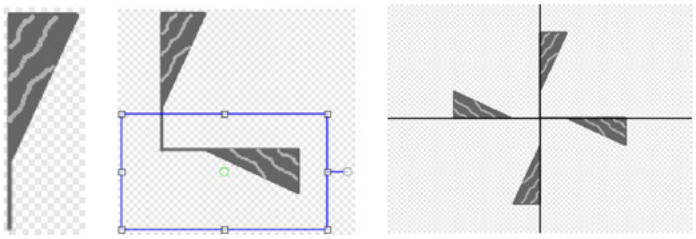


Rysunek 11. Scena projektu „Wiatrak”

Pozostałe elementy graficzne do wykorzystania w projekcie to: łopaty wiatraka, słońce, drzewo itp.

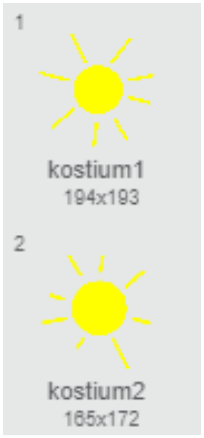
Rysunek 12. Duszki projektu „Wiatrak”

Szczególną uwagę należy poświęcić tworzeniu łopat wiatraka. Zaczynamy od narysowania pojedynczej łopaty, którą następnie powielamy i obracamy. Miejsce połączenia wszystkich czterech elementów stanowi środek tego duszka.



Rysunek 13. Etapy tworzenia łopat wiatraka

Dodatkowym elementem symulacji może być „świecące” słońce. Do uzyskania złudzenia ruchu promieni potrzebujemy dwóch kostiumów słońca nieznacznie różniących się długością promieni.



Rysunek 14. Kostiumy duszka-słońce

Część 2 – tworzymy skrypty

Drugi etap projektu związany jest z koniecznością utworzenia skryptów. Zaczynamy od zaprojektowania animacji słońca, którą zrealizujemy poprzez zmianę kostiumów.

Rysunek 15. Skrypt zielonej flagi duszka-słońce

Najważniejszy element symulacji stanowią obracające się łopaty wiatraka.

Rysunek 16. Skrypt zielonej flagi duszka-łopaty wiatraka

Warto zauważyć, że tym razem do animacji nie wykorzystujemy kolejno wyświetlających się kostiumów, a obroty zaprojektowanego duszka.

Część 3 – testujemy i udoskonalamy projekt

Gdy uruchomimy skrypt zielonej flagi, łopaty wiatraka obracają się względem środka kostiumu. Można ruch łopat spowolnić lub przyspieszyć, zmieniając wartość kąta obrotu. Ręczna modyfikacja wartości w bloczku jest pracochłonna, dlatego proponujemy wprowadzenie zmiennej, której wartość będzie wpływać na szybkość obrotu łopat.

Rysunek 17. Modyfikacja skryptu zielonej flagi duszka-łopaty wiatraka

Wyświetlając wartość zmiennej w postaci suwaka, zyskujemy interaktywność projektu.

Rysunek 18. Suwak

Podczas uważnego testowania tego rozwiązania można spostrzec, że początkowo łopaty obracają się zgodnie z założeniem: tym szybciej, im większa wartość została wybrana na suwaku. Gdy przekroczymy pewną wartość, pojawia się jednak problem: łopaty nagle zwalniają, czasem wydaje się, że obracają się w przeciwnym kierunku, a dla wartości 90 nie obracają się wcale. To ostatnie zjawisko wynika z nakładania się na siebie łopat przy obrocie o kąt 90 stopni. Aby temu zaradzić, możemy zmienić zakres suwaka (opcja: `ustaw min. i max. suwaka`). Innym rozwiązaniem jest użycie wyrażenia arytmetycznego (dzielenia).

Rysunek 19. Modyfikacja skryptu zielonej flagi duszka-łopaty wiatraka

Część 4 – dodajemy nową funkcjonalność

Ostatni etap projektu polega na wykorzystaniu czujnika „głośność", dzięki któremu symulacja staje się bardziej realistyczna.

Rysunek 20. Modyfikacja skryptu zielonej flagi duszka-łopaty wiatraka

By przetestować działanie zaprezentowanego powyżej skryptu, musimy mieć do dyspozycji mikrofon. Ponadto, podczas uruchamiania skryptu zielonej flagi, należy zezwolić w przeglądarce na użycie urządzenia zewnętrznego.

Podsumowanie

Realizując ten projekt, należy zwracać uwagę na treści z zakresu innych dziedzin edukacji. Możemy z uczniami szukać na rysunkach figur geometrycznych, analizować symetrię łopat, położenie poszczególnych elementów na scenie. Dajemy szansę ujawnienia ekspresji twórczej poprzez kreowanie symboli obiektów z ich otoczenia.

Proponowane modyfikacje i zadania do samodzielnego wykonania:

- na podstawie tego projektu można przygotować symulację jadącego samochodu, w którym obracają się koła.

Zasoby do wykorzystania:

- Projekt „Wiatrak".

PRZYKŁADOWE ZADANIA

Zadanie 1: Szyfr GA-DE-RY-PO-LU-KI (bez użycia komputera)

GA-DE-RY-PO-LU-KI – to prosty szyfr stosowany przez harcerzy. Charakteryzuje się tym, że w kluczu występuje sześć par liter. Każda z par składa się ze spółgłoski i samogłoski. Szyfrowanie polega na zamianie liter w obrębie par. Litery niewystępujące w kluczu (i spacje) nie są zamieniane. Np. dla klucza GA-DE-RY-PO-LU-KI każdą literę G zamieniamy na A i odwrotnie (A na G), D na E, E na D, itd. Napis: ALA MA KOTA po zaszyfrowaniu kluczem GA-DE-RY-PO-LU-KI ma postać: GUG MG IPTG.

Inne popularne klucze to: MA-LI-NO-WE-BU-TY, MO-TY-LE-CU-DA-KI, NO-WE-BU-TY-LI-SA, RE-GU-LA-MI-NO-WY, KO-NI-EC-MA-TU-RY.

Zaszyfruj napis: ZAPRASZAM DO WSPÓLNEJ ZABAWY W SZYFROWANIE wybranym kluczem. Odszyfruj napis: ZCUHT SYMSCJĄ YLN RMAZDJ SZTFRMWDNKD, wiedząc, że do szyfrowania użyto klucza MO-TY-LE-CU-DA-KI.

Na zajęciach z nauczycielami wybieramy dłuższe sekwencje do szyfrowania i odszyfrowania. Zabawę z dziećmi zaczynamy od zaszyfrowania krótkich wyrazów (np. imion). Warto przygotować pomoc w postaci sześciu kart dla każdego klucza, po jednej literze z pary na stronę karty. Na przykład dla klucza MA-LI-NO-WE-BU-TY będą to karty: M (rewers) i A (awers), L (rewers) i I (awers), N (rewers) i O (awers), W (rewers) i E (awers), B (rewers) i U (awers) oraz T (rewers) i Y (awers). Szyfrowanie łączymy z odszyfrowaniem. Uczniowie mogą pracować w parach według schematu: szyfrujemy wiadomość, przesyłamy adresatowi, adresat odszyfrowuje wiadomość, nadawca sprawdza, czy wiadomość została poprawnie odszyfrowana.

Propozycja innych tekstów do zaszyfrowania – dla nauczycieli i starszych uczniów: HARCERSKI SZYFR DO SUPER TAJNYCH ZADAŃ, ZUCHY I HARCERZE CHRONIĄ WIADOMOŚCI, PODMIENIAMY WYBRANE LITERY W TEKŚCIE, KLUCZE SZYFRUJĄCE ŁATWE DO ZAPAMIĘTANIA, KLUCZE SZYFRUJĄCE O DŁUGOŚCI DWANAŚCIE. Propozycja wyrazów do zaszyfrowania przez dzieci: LATO, WIOSNA, PIES, KOT, ANIA, TOMEK itp.

Zadanie 2: Szyfr z wykorzystaniem tablicy kodów (bez użycia komputera)

Tekst jawny szyfrujemy z wykorzystaniem poniższej tablicy kodów. Kod znaku to dwucyfrowa liczba – pierwsza cyfra znajduje się nad literą, druga po jej lewej stronie. Np. napis: ALA MA KOTA po zaszyfrowaniu daje ciąg liczb:
12 36 12 90 76 12 90 16 18 98 12.

	1	3	5	7	9
2	A	B	C	D	E
4	F	G	H	I	J
6	K	L	Ł	M	N
8	O	P	R	S	T
0	U	W	Y	Z	odstęp

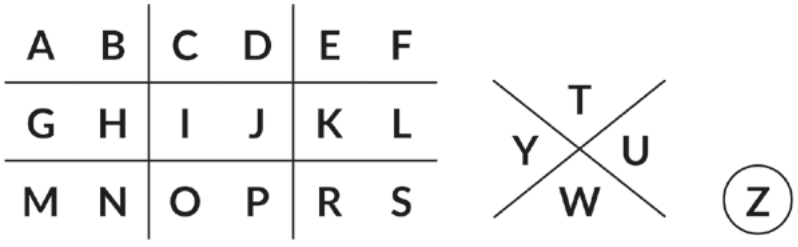
Rysunek 21. Tablica kodów

Zaszyfruj wyraz MATEMATYKA, korzystając z tablicy kodów i przekaż otrzymany ciąg liczb innej osobie do odszyfrowania.

Uwaga: Tablica kodów może mieć różną zawartość, w zależności od zestawu liter poddawanych szyfrowaniu. W powyższym przypadku występuje litera Ł, która może być zastąpiona innym znakiem.

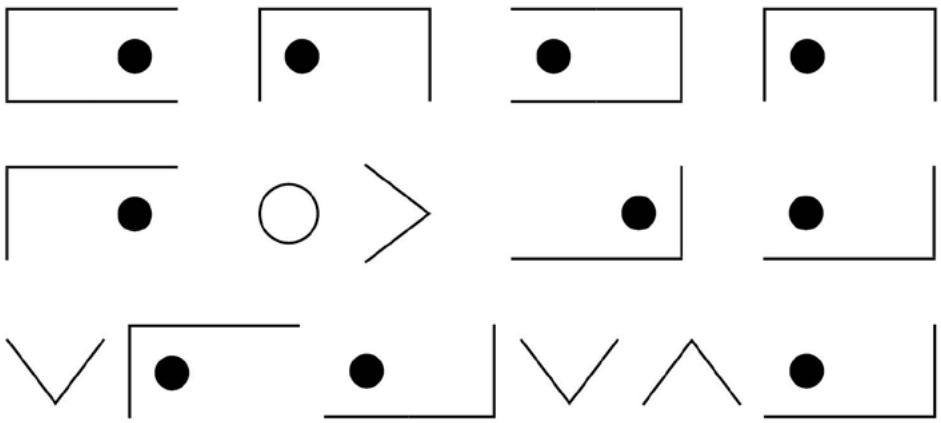
Zadanie 3: Szyfr obrazkowy (bez komputera)

Harcerski szyfr „Czekoladka" polega na rysowaniu kolejnych zaszyfrowanych liter danego wyrazu. Aby zaszyfrować literę, rysujemy fragment ramki, w którym ta litera jest zapisana, zgodnie z rysunkami poniżej. Ponieważ w jednej ramce najczęściej znajdują się dwie litery, odpowiednio usytuowana kropka pokazuje, czy mamy na myśli literę z lewej, czy z prawej strony danego fragmentu ramki. Dla liter: T, U, W, Y i Z, które występują pojedynczo, rysowanie kropki jest zbędne.



Rysunek 22. Szyfr obrazkowy

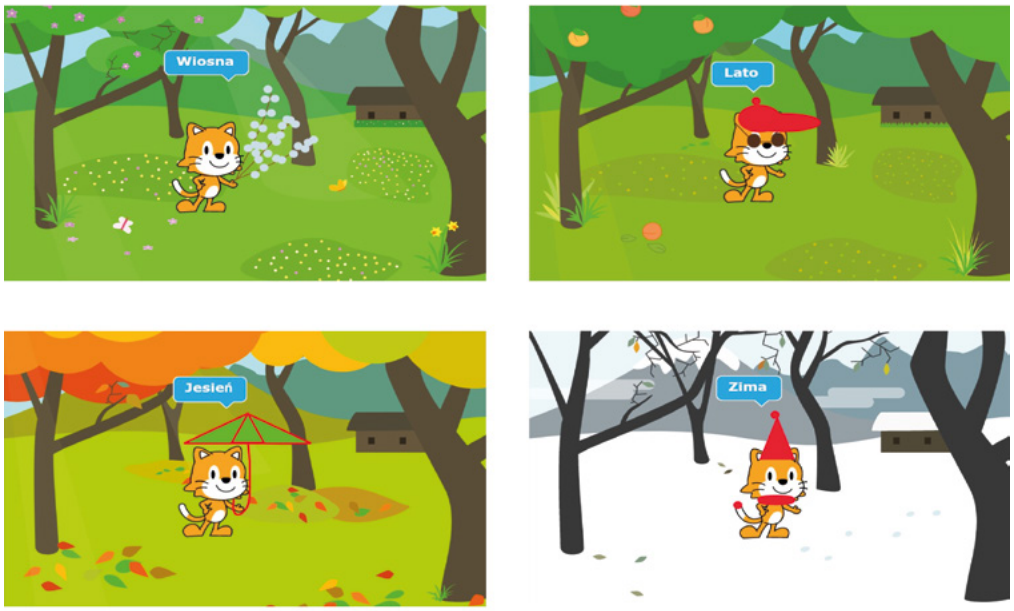
Odszyfruj zaszyfrowane wyrazy:



Odpowiedź: logo, szyba, tratwa.

Zadanie 4: Pory roku (projekt w środowisku ScratchJr)

Przygotuj projekt o nazwie „Pory roku” w środowisku ScratchJr. Animacja powinna przedstawiać kolejno pory roku i bohatera – kota z domalowanymi charakterystycznymi rekwizytami. Dla wiosny mogą to być np. bazie, dla lata – czapka z daszkiem lub okulary słoneczne, dla jesieni – parasol, a dla zimy – czapka i szalik.

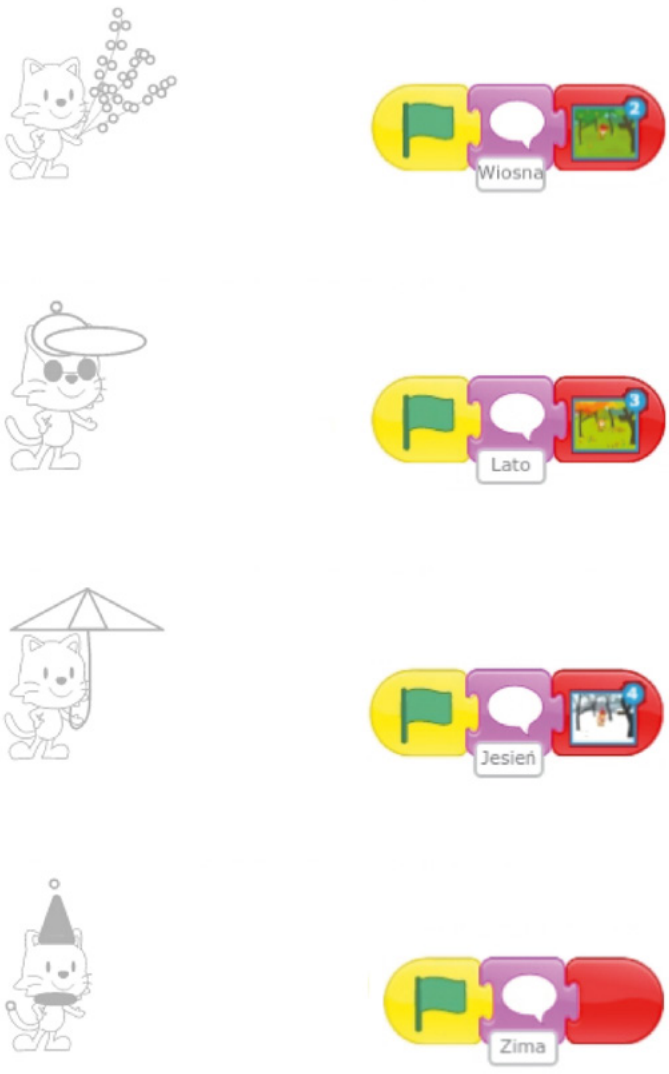


Rysunek 23. Projekt „Pory roku”

Bardzo prosty projekt, wykorzystujący sekwencję poleceń oraz dający możliwość układania obrazków w logicznym porządku, może posłużyć jako podsumowanie lekcji na temat pór roku lub opisywania pogody panującej za oknem.

W projekcie wykorzystuje się tła z biblioteki przedstawiające pory roku oraz standardowego duszka, któremu w edytorze graficznym dorysowano wybrane rekwizyty. Uwaga: w środowisku ScratchJr można wstawić maksymalnie cztery tła.

Dla poszczególnych duszków zdefiniowano następujące skrypty:



Rysunek 24. Skrypty duszków

Alternatywnie, np. realizując projekt z dziećmi nieczytającymi, bloczek z dymkiem można zastąpić bloczkami związanymi z ruchem.

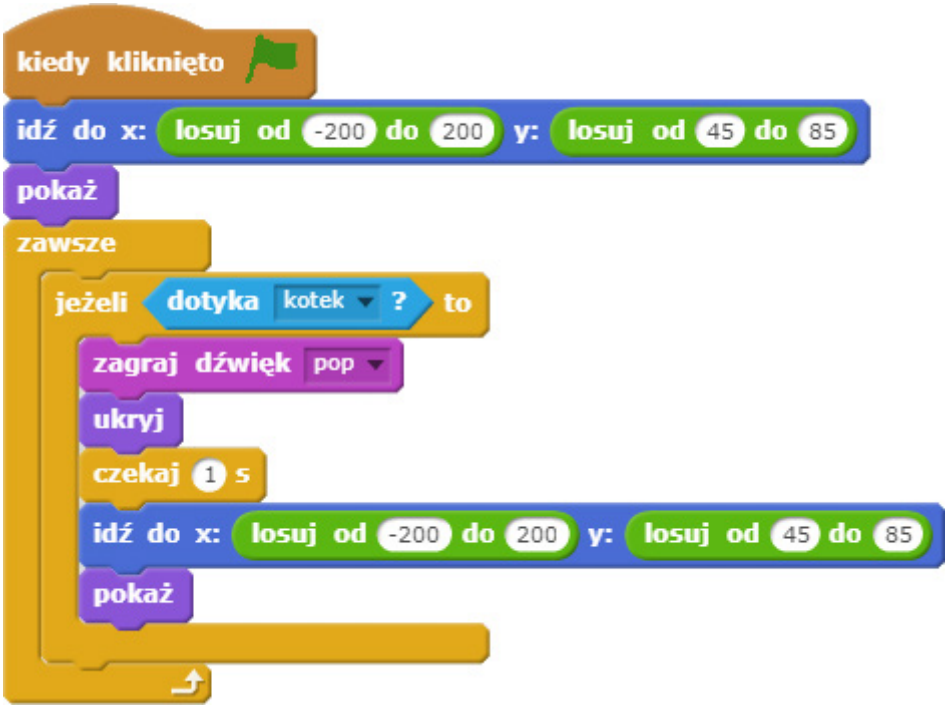
Zadanie 5: Kot i balony (projekt w środowisku Scratch)

Przygotuj projekt „Kot i balony” w środowisku Scratch. Projekt powinien opierać się na zabawie wymagającej interakcji z użytkownikiem. Podskakującym kotem, który próbuje złapać balon unoszący się w powietrzu, sterujemy za pomocą siły głosu. W zaawansowanej wersji projektu można dodać opcję zliczania punktów oraz efekty kolorystyczne.

Rysunek 25. Projekt „Kot i balony”

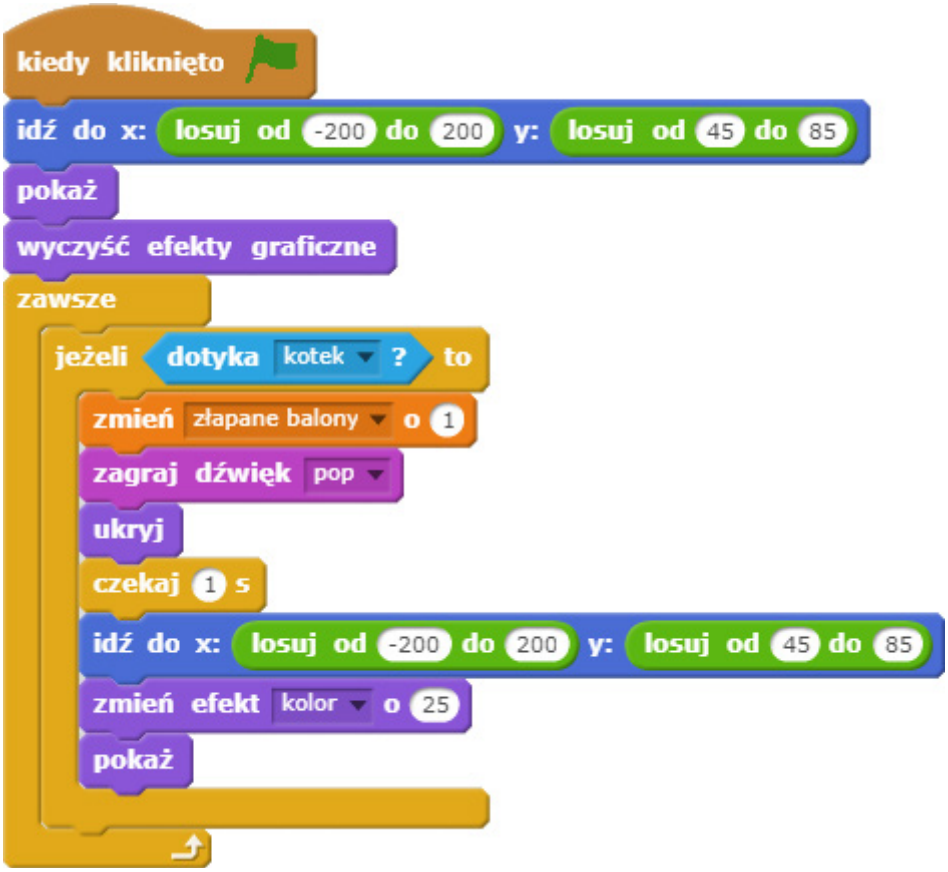
W projekcie wykorzystano tło z biblioteki (*Stage2*) oraz dwa duszki (standardowy kot oraz *Balloon1*).

Rysunek 26. Skrypty kota



Rysunek 27. Skrypt balonu

Rysunek 28. Modyfikacja skryptu kota – wersja zaawansowana



Rysunek 29. Modyfikacja skryptu balonu – wersja zaawansowana

W tym projekcie do sterowania wykorzystane zostało urządzenie zewnętrzne (mikrofon). Uruchamiając aplikację należy zezwolić na jego użycie. Problemy, zarówno wśród uczniów, jak i nauczycieli, może powodować umieszczanie we właściwych miejscach czujników: „dotyka", „głośność" oraz bloków z kategorii: „wyrażenia". Ze szczególną uwagą należy konstruować wyrażenie *-80+głośność*4*. W zaawansowanej wersji projektu do zmiany wyglądu balonów można również wykorzystać różne kostiumy duszka.

Zasoby do wykorzystania:

- Studio projektów OEIiZK: przykładowe projekty.

Zadanie 6: Kartka okolicznościowa (projekt w środowisku Scratch)

Realizacja projektu polega na zaprojektowaniu w środowisku Scratch kartki okolicznościowej – na przykład z okazji Dnia Matki. Wykonany projekt powinien mieć postać animacji, w której z rozsypanki literowej na tle serca tworzy się tytułowy napis. Efekty wizualne mogą być dowolne.

Rysunek 30. Projekt „Kartka okolicznościowa”

W projekcie wykorzystano tło z biblioteki (*Hearts1*) oraz siedem duszków (są nimi wybrane litery typu *Glow*). Standardowe litery mają kolor niebieski. Do ich pokolorowania użyto edytora graficznego Scratcha.

Rysunek 31. Przykładowy skrypt litery

Wartości w polach x i y zależą od punktu umieszczenia duszka na scenie i ustalają się automatycznie w bloczkach: „idź do” oraz „leć do” – umieszczonych w zasobniku. Po ich przeniesieniu do sekcji tworzenia skryptów wszelkie zmiany zawartości pól x i y muszą być dokonywane ręcznie, poprzez wpisanie właściwych wartości. Dlatego warto realizację projektu podzielić na dwie fazy.

W pierwszej z nich należy ustalić położenie liter na scenie (w rozsypance), a następnie dla każdej z nich bloczek: „idź do" przenieść do sekcji tworzenia skryptów. W drugiej fazie, po ustaleniu docelowego miejsca położenia liter w wyrazach, należy powtórzyć operację przeniesienia bloczków do sekcji tworzenia skryptów – ale tym razem dla bloczka: „leć do". Ponadto w projekcie dodano animację w postaci obrotu. Należy jednakże uwzględnić również inne pomysły uczestników szkolenia.

Realizując ten projekt, trzeba zwrócić uwagę na kwestie związane z umiejscowieniem duszków na scenie. Animację należy wykonywać stopniowo, krok po kroku ją testując. Proponujemy zacząć pracę z uczniami od zaprezentowania animacji jednej litery, bez wykorzystania pętli: „powtórz".

Zasoby do wykorzystania:

- Przykładowy projekt „Dla Mamy".

Zadanie 7: Realizujemy własne pomysły (projekt w środowisku Scratch)

Zadanie polega na przygotowaniu scenariusza projektu/aplikacji, wspierającego wybraną lekcję z zakresu edukacji wczesnoszkolnej, przewidzianego do realizacji w środowisku Scratch. Można wzorować się na scenariuszu z projektu „Opowiadanie multimedialne". Następnie należy stworzyć projekt/aplikację według przedstawionego pomysłu.

Podczas realizacji tego zadania zalecana jest praca w zespołach dwuosobowych, która umożliwi praktyczny pokaz jednej z metod pracy z uczniami. Nauczyciele muszą przejść wszystkie etapy tworzenia aplikacji:

- sprecyzowanie pomysłu;
- stworzenie planu działania (jakich elementów potrzebujemy, czy są one dostępne w bibliotece środowiska, co się będzie działo w aplikacji – jaki będzie scenariusz gry);
- wykonanie aplikacji oraz jej testowanie krok po kroku.

W tworzonych aplikacjach nauczyciele powinni wykorzystać niektóre z wcześniej poznanych mechanizmów interakcji z użytkownikiem (sterowanie klawiaturą, przeciąganie duszków po ekranie, kliknięcie w duszka, sterowanie głosem). Na koniec powinni zweryfikować pierwotny scenariusz i uzupełnić go o elementy, które wystąpiły w projekcie, a nie były zaplanowane.

Lekcja z uczniami będzie przebiegać według odmiennego scenariusza. Temat projektu powinien być ściśle związany z omawianymi treściami z zakresu edukacji wczesnoszkolnej. Przygotowanie scenariusza może polegać na ułożeniu w logicznym porządku zdarzeń/czynności, które są konieczne do wykonania podczas jego realizacji. Po wspólnym ustaleniu zakresu działań uczniowie w parach realizują projekt, a nauczyciel pełni rolę konsultanta i bacznego obserwatora.

Zasoby do wykorzystania:

- Studio projektów OEIiZK: przykładowe projekty.

KATARZYNA OLĘDZKA

RAMOWY PROGRAM SZKOLENIA DLA NAUCZYCIELI KLAS 4–6 (II ETAP EDUKACYJNY)

INFORMACJE OGÓLNE

Szkolenie jest przeznaczone dla nauczycieli informatyki nauczających lub planujących nauczanie w szkołach podstawowych. Jego główny cel stanowi przygotowanie nauczycieli do realizacji nowej podstawy programowej przedmiotu informatyka w szkole podstawowej w zakresie algorytmicznego rozwiązywania problemów oraz programowania na poziomie klas 4–6. Szkolenie obejmuje 40 godzin lekcyjnych zajęć stacjonarnych. Nauczyciele mogą kontynuować doskonalenie w trakcie drugiej części szkolenia przygotowującej do realizacji podstawy programowej w klasach 7 i 8.

Zajęcia powinny mieć przede wszystkim charakter warsztatowy, uczestnicy pod nadzorem prowadzącego samodzielnie rozwiązują problemy, wcielając się w rolę ucznia. Część zajęć należy przeznaczyć na wykład i dyskusję oraz omówienie zagadnień metodycznych. Praca praktyczna pozwoli słuchaczom nabrać biegłości w posługiwaniu się narzędziami i metodami informatycznymi. Niemniej ważna jest również refleksja pedagogiczna: w jakim celu wprowadzamy dane zagadnienia, jak zorganizować proces dydaktyczny, na co szczególnie zwrócić uwagę i jakie mogą wystąpić trudności. Podczas zajęć nie tylko prowadzący dzielą się swoją wiedzą, ale także słuchacze wymieniają się doświadczeniami.

WYMAGANIA WSTĘPNE STAWIANE UCZESTNIKOM SZKOLENIA

Uczestnik szkolenia powinien posiadać kompetencje wymienione w *Załączniku 1*, a ponadto mieć uprawnienia do nauczania informatyki w szkole podstawowej.

CELE SZKOLENIA:

- przygotowanie nauczycieli do prowadzenia zajęć z informatyki zgodnie z nową postawą programową w zakresie algorytmicznego rozwiązywania problemów i programowania;
- doskonalenie własne nauczycieli w zakresie algorytmiki i programowania, a także rozumienia pojęć informatycznych i metod informatyki;
- rozwijanie u uczniów umiejętności myślenia komputacyjnego i rozwiązywania problemów z życia codziennego przy pomocy narzędzi informatycznych;
- nabycie umiejętności programowania w języku wizualnym w zakresie umożliwiającym realizację podstawy programowej przedmiotu informatyka w klasach 4–6 szkoły podstawowej;
- wprowadzenie podstaw programowania w języku tekstowym.

TREŚCI NAUCZANIA:

1. Rola algorytmicznego rozwiązywania problemów, myślenia komputacyjnego i programowania w nowej podstawie programowej ze szczególnym uwzględnieniem zapisów dotyczących jej realizacji w klasach 4–6.
2. Sterowanie obiektem za pomocą sekwencji poleceń, z nawiązaniem do konkretnych form, metod i środków rozwijania u uczniów myślenia komputacyjnego.
3. Wizualne programowanie prostych sytuacji lub historyjek z wykorzystaniem poleceń sekwencyjnych, warunkowych i iteracyjnych oraz zdarzeń.
4. Porządkowanie informacji.
5. Rozwiązywanie problemów z życia codziennego oraz innych dziedzin wiedzy poprzez formułowanie i zapisywanie algorytmu.
6. Poszukiwanie w zbiorach nieuporządkowanych i uporządkowanych konkretnego elementu oraz elementów najmniejszego i największego.
7. Kształtowanie zdolności algorytmicznego rozwiązywania problemów. Wyróżnianie podstawowych kroków w procesie definiowania i algorytmicznego rozwiązywania problemu oraz ich stosowanie w praktyce.
8. Wprowadzenie do tekstowego języka programowania wysokiego poziomu, w szczególności do sterowania obiektem.
9. Wykorzystanie arkusza kalkulacyjnego do rozwiązywania zadań algorytmicznych związanych z prostymi obliczeniami.
10. Testowanie, poprawianie i prezentowanie własnych programów.

PRZYKŁADOWY ROZKŁAD MATERIAŁU:

Temat i temat cząstkowy	Punkt podstawy programowej	Treści	Liczba godzin
1. Wprowadzenie			**2**
• Organizacja szkolenia	całość	1, 2, 3	0,5
• Dlaczego warto uczyć się programowania? Podstawa programowa informatyki dla drugiego etapu edukacyjnego			0,5
• Projekt „Godzina Kodowania”			1
2. Rozwijanie myślenia algorytmicznego			**8**
• Projekt „Informatyka bez komputera”	I.1, I.2a, I.2b, I.3, II.3c	1, 3, 4, 5, 6, 7, 9	2
• Konkurs Informatyczny „Bóbr”			2
• Zadania obliczeniowe			2
• Odkrywamy algorytmy			2
3. Programowanie w środowisku Scratch			**14**
• Pierwsze kroki w Scratchu	I.2, I.3, II.1, II.2, II.4	1, 2, 3, 7, 10	1
• Przygotowujemy opowiadania multimedialne			3
• Tworzymy własne gry			3
• Opracowujemy symulacje			2
• Rysujemy			2
• Implementujemy algorytmy obliczeniowe			2
• Podsumowanie – programowanie w środowisku wizualnym			1
4. Sterowanie obiektem za pomocą sekwencji poleceń			**10**
• Grafika żółwia w Pythonie – wprowadzenie	I.2c, I.3, II.1, II.2, II.3	1, 2, 7, 8, 10	1
• Uczymy żółwia nowych słów			1
• Powtarzamy czynności			1
• Powtarzamy i podejmujemy decyzje			1
• Dzielimy problem na problemy cząstkowe			1
• Projektujemy posadzki			1
• Budujemy piramidy			1

• Rozwiązujemy zadania • Sterujemy robotem			1 2
5. Praca z uczniami o różnych potrzebach edukacyjnych			**5**
• Praca z uczniem zdolnym • Gry edukacyjne • Rozwijanie różnych zainteresowań uczniów	I.1, I.2, I.3, II	1, 2, 3, 4, 5, 6, 7	2 2 1
6. Podsumowanie			**1**
• Najważniejsze umiejętności do zdobycia na drugim etapie edukacyjnym: Co uczniowie powinni umieć przed przejściem do czwartej klasy? Czego będą się uczyć się w klasach starszych? • Zadania nauczyciela – jak radzić sobie z trudnościami i gdzie szukać pomocy?	całość	całość	0,5 0,5

OMÓWIENIE POSZCZEGÓLNYCH TEMATÓW

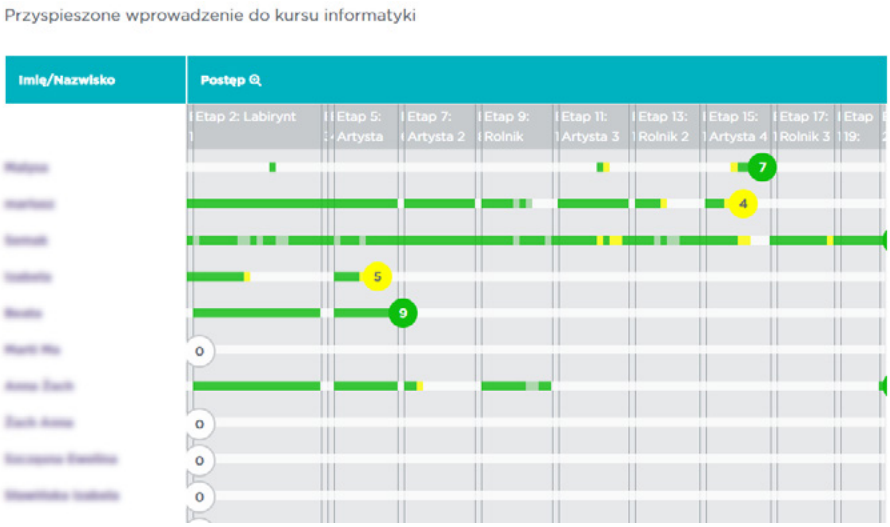
1. Wprowadzenie

Na początku szkolenia, oprócz omówienia zasad organizacji, warto przedstawić uczestnikom tematykę. W tym celu należy przeprowadzić wykład ukazujący, dlaczego warto nauczać programowania. Ponadto trzeba omówić poszczególne zapisy dotyczące informatyki zawarte w nowej podstawie programowej, zarówno w odniesieniu do założeń ogólnych programu, jak i do szczegółowych treści. Szczególną uwagę należy poświęcić wskazaniom dotyczącym rozwiązywania problemów i nauce programowania w klasach 4–6 oraz zastanowieniu się, jakie umiejętności wynoszą uczniowie z edukacji informatycznej w nauczaniu wczesnoszkolnym. Później może nastąpić dyskusja o zmianach, jakie niesie nowa podstawa programowa, oraz o tym, jak wprowadzać je w życie. Można też nawiązać do artykułów prof. Macieja M. Sysły *Wprowadzając… porządek* oraz dr Anny Beaty Kwiatkowskiej *W poszukiwaniu abstrakcyjnego modelu*.

Zasoby do wykorzystania:

- Sysło M.M., (2016), *Wprowadzając… porządek*, Toruń: Informatyka w Edukacji;
- Kwiatkowska, A.B., (2016), *W poszukiwaniu abstrakcyjnego modelu*, Toruń: Informatyka w Edukacji.

Ostatnim celem do zrealizowania podczas pierwszej części szkolenia jest zapoznanie uczestników z projektem „Godzina kodowania” (http://godzinakodowania.pl/), organizowanym rokrocznie w ramach Tygodnia Edukacji Informatycznej, stanowiącym ciekawą inicjatywę edukacyjną adresowaną do uczniów z całego świata. Projekt ten cieszy się w Polsce dużą popularnością. Można w nim uczestniczyć przez cały rok, zarówno w szkole, jak i w domu przez internet. W ramach szkolenia zaleca się rozwiązanie kilku zadań z wybranego kursu oraz przedstawienie narzędzi dostępnych dla nauczyciela (zakładanie i administrowanie kontami uczniów, monitorowanie postępów uczniów).



Rysunek 1. Widok nauczyciela – postępy uczniów

Zasoby do wykorzystania:

- Strona projektu „Godzina kodowania”: http://godzinakodowania.pl/;
- Strona platformy: https://code.org;
- Strona OEliZK – instrukcja dla nauczyciela: http://programowanie.oeiizk.edu.pl/, (sekcja „Polecamy/Godzina Kodowania”).

2. Rozwijanie myślenia algorytmicznego

2.1. Projekt „Informatyka bez komputera”

Zyskującym na popularności sposobem prezentowania zagadnień z zakresu informatyki jest prowadzenie zajęć dydaktycznych bez komputera. Jak zauważył Edsger Dijkstra: „informatyka ma tyle samo wspólnego z komputerami, co astronomia ma z teleskopami”. Można uczyć o informatyce – jej pojęciach i metodach – nie używając komputera, również poza pracownią komputerową.

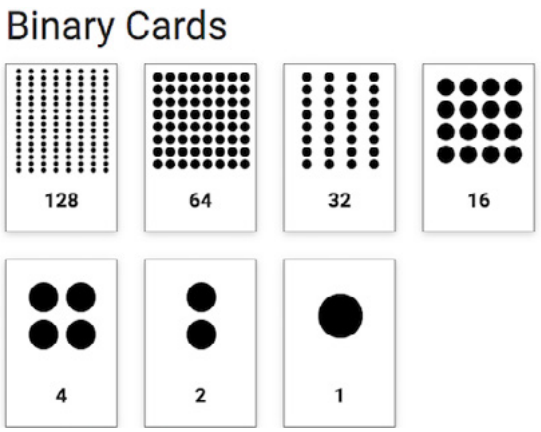
Ideę tę rozwinął Tim Bell z Nowej Zelandii. Uczniowie kształcą w ten sposób umiejętność twórczego myślenia i poznają świat komputerów poprzez pojęcia takie jak: liczby binarne, algorytm, program, sortowanie, kompresja, szyfrowanie danych. Zajęcia prowadzone w ten sposób angażują uczniów także ruchowo.

Przykładowe tematy zajęć:

✓ ***Zliczanie kropek*** **– system binarny**

Ponieważ dane w komputerach są zapisywane jako ciągi zer i jedynek, warto pokazać, w jaki sposób liczby mogą być reprezentowane przy pomocy zaledwie dwóch symboli. Przygotowujemy kartki z kropkami, a następnie zadajemy uczniom pobudzające do myślenia pytania dotyczące zapisu binarnego liczby. Poprzez odsłanianie/zasłanianie odpowiednich kart możemy zamieniać zapis dziesiętny na binarny i odwrotnie.

Wizualne reprezentacje liczb:



✓ ***Kolory jako liczby*** **– kodowanie obrazu**

Pokazujemy zapis obrazów za pomocą kodów liczbowych. Specjalnie przygotowane karty pracy – dostępne na stronie http://programowanie.oeiizk.edu.pl w zakładce „Informatyka /prawie/ bez komputera" – mogą w tym pomóc.

✓ ***Magia obracanych kart*** **– wykrywanie i korekcja błędów**

Do zajęć potrzebujemy zestawu 36 identycznych, najlepiej kwadratowych, dwukolorowych kart (z jednej strony są one czarne, z drugiej białe). Poprzez specyficzny układ kart możemy zakodować ich sumę kontrolną i w razie zmiany

jednej z nich wykryć nieprawidłowość. Ćwiczenie to wydaje się zarówno atrakcyjne, jak i pouczające.

Opracowanych scenariuszy jest więcej – są one dostępne w języku angielskim wraz z ilustrującymi je filmami (http://csunplugged.org), oraz w wersji polskiej (http://jasijoasia.edu.pl).

✓ ***Scottie Go!***

Inny pomysł na poprowadzenie zajęć informatycznych opiera się na wykorzystaniu gry „Scottie Go!". Uczniowie za pomocą kartonowych klocków układają program, który można zeskanować i uruchomić na tablecie. W zestawie przygotowano dla uczniów zadania o rosnącym poziomie trudności, a także materiały dla nauczyciela. Zajęcia ze „Scottie Go!" pozwalają doskonalić umiejętności analitycznego i logicznego myślenia, rozwijają intuicję algorytmiczną oraz uczą rozwiązywania skomplikowanych problemów i pracy w grupie. Można je wykorzystać, pracując z najmłodszymi uczniami, ale również na zajęciach z trochę starszymi. Gra stanowi ciekawe uzupełnienie typowych zajęć z użyciem komputerów.

Zasoby do wykorzystania:

- Strona internetowa Ośrodka Edukacji Informatycznej i Zastosowań Komputerów: http://programowanie.oeiizk.edu.pl/, (sekcja „Informatyka /prawie/ bez komputera");
- Strona projektu „Informatyka dla Jasia i Joasi": http://jasijoasia.edu.pl/, (zakładka „Projekt CS Unplugged", Scenariusze);
- Strona projektu „Computer Science Unplugged": https://csunplugged.org/en/;
- Strona gry „Scottie Go!": https://scottiego.com/pl/.

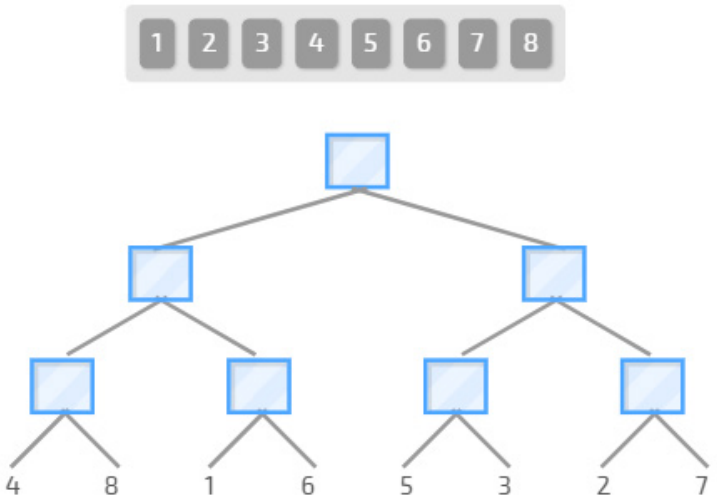
2.2. Konkurs Informatyczny „Bóbr"

Głównym celem Międzynarodowego Konkursu Informatycznego „Bóbr" jest rozwijanie oraz kształtowanie myślenia algorytmicznego i komputacyjnego, a także popularyzacja posługiwania się technologią informacyjną i komunikacyjną wśród wszystkich uczniów na każdym z etapów edukacyjnych. Konkurs ma zasięg międzynarodowy i obejmuje cztery poziomy edukacyjne. Dla uczniów klas 4–6 przeznaczona jest kategoria „Benjamin". Na stronach konkursu (https://www.bobr.edu.pl/) znajdziemy wiele inspirujących zadań rozwijających myślenie algorytmiczne. Warto, żeby uczestnicy szkolenia rozwiązali kilka zadań.

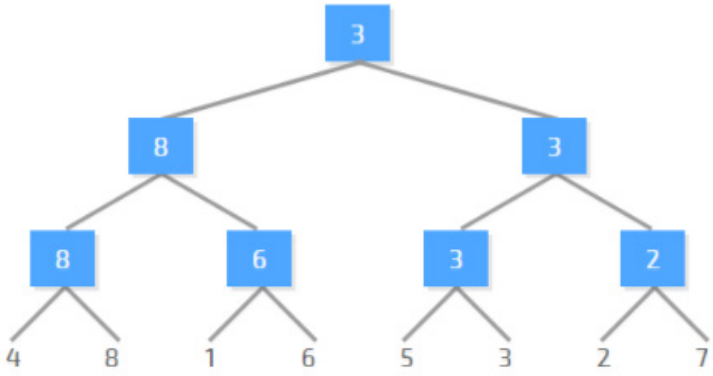
Przykład zadania:
Ośmiu członków zespołu sportowego *Beaver* potrzebuje nowego kapitana. Organizuje więc turniej, by go wybrać. Zawodnicy startują w parach. Zasady turnieju są następujące:

1. Jeśli para składa się z zawodników z parzystymi numerami, to zwycięzcą jest zawodnik z wyższym numerem.
2. Jeśli para składa się z zawodników z numerami nieparzystymi, to zwycięzcą jest zawodnik z niższym numerem.
3. Jeśli para składa się z zawodników z numerami nieparzystym i parzystym, to zwycięzcą jest zawodnik z numerem bliższym numerowi 4 (na przykład: numer 1 jest bliżej numeru 4 niż 8, ponieważ 4 – 1 = 3 i 8 – 4 = 4).

Każdy z zawodników losowo wybiera numer między 1 i 8. Również losowo zostają wybrane początkowe pary zawodników: 4 – 8; 1 – 6; 5 – 3; 2 – 7. Zwycięzcy z każdej pary kontynuują turniej. Celem uczestników szkolenia jest skonstruowanie „drzewa" turnieju i wskazanie zwycięzcy, czyli kapitana zespołu.



Rozwiązanie:



Zasoby do wykorzystania:

- Strona konkursu „Bóbr”: https://www.bobr.edu.pl/.

2.3. Zadania obliczeniowe

Do rozwiązywania zadań wymagających obliczeń można wykorzystać arkusz kalkulacyjny. Wspomagamy w ten sposób rozwój myślenia komputacyjnego. W arkuszu stosunkowo łatwo wykonuje się różne obliczenia i zapisuje algorytmy. Do zajęć z algorytmiki potrzebna jest podstawowa znajomość arkusza kalkulacyjnego.

Pierwszą grupę zadań stanowią symulacje pewnych wydarzeń, drugą – zapis algorytmów matematycznych za pomocą formuł w arkuszu kalkulacyjnym. Na zajęciach z nauczycielami warto zrealizować przynajmniej po jednym zadaniu symulacyjnym (np. przykładowy scenariusz 2) oraz zadaniu obliczeniowym.

✓ **Przykład zadania: Ryby w akwarium (przykładowy scenariusz 2)**

✓ **Przykładowe zadania: Średnia arytmetyczna**

Zadanie 1
Średnia arytmetyczna sześciu liczb: 7, x, 5, 5, 4, 6 jest równa 6. Ile jest równe x?
Odpowiedź: 9.

Zadanie 2
Uczeń otrzymał następujące oceny: 5, x, 5, 4, 4. Średnia tych ocen jest równa 4. Jaka jest najniższa, a jaka najwyższa ocena?
Odpowiedź: 2 i 5.

Zadanie 3
Zawodnicy otrzymali następujące noty:

Liczba zawodników	Nota
6	3
12	4
2	x

Średnia nota uzyskana przez zawodników wynosi 4. Ile jest równe x?
Odpowiedź: 7.

2.4. Odkrywamy algorytmy

Zgadywanie liczby, czyli „Gra w 20 pytań" to przykład algorytmicznego wyszukiwania w zbiorze uporządkowanym. Rozpoczynamy od zabawy – jedna osoba wymyśla liczbę, druga próbuje ją zgadnąć. Następnie stawiamy pytania, które powinny doprowadzić do sfomułowania algorytmu. Można też posłużyć się gotową aplikacją, która ułatwi sformułowanie algorytmu. Warto ten algorytm zapisać np. w postaci listy kroków.

Innym przykładem realizacji tego samego algorytmu jest sprawdzenie, czy uczeń o podanym nazwisku chodzi do danej klasy. Przygotowujemy kartki z nazwiskami uczniów, rozkładamy je uporządkowane alfabetycznie (analogicznie do ich zapisu w dzienniku lekcyjnym), ale odwrócone, aby nazwiska nie były widoczne. W jednym ruchu odkrywamy jedną kartkę. Zastanawiamy się, ile kartek maksymalnie musimy odkryć, aby znaleźć ucznia o danym nazwisku lub mieć pewność, że nie chodzi on do tej klasy. Następnie warto powtórzyć ćwiczenie, ale zakryte kartki rozkładamy losowo. Powtarzamy to samo pytanie. Staramy się sformułować i zapisać, np. w postaci listy kroków, algorytm odpowiadający losowemu ułożeniu kartek.

Podobne ćwiczenia można wykonać z kartami do gry. Rozszerzamy je o poszukiwanie „najmłodszej" i „najstarszej" karty. Naturalnie w ten sposób przybliżymy uczniom zagadnienia z zakresu wyszukiwania elementu w zbiorze uporządkowanym i nieuporządkowanym, znajdowania elementu minimalnego i maksymalnego.

3. Programowanie w środowisku Scratch

3.1. Pierwsze kroki w Scratchu

Rozpoczynając zajęcia, warto przedstawić założenia edukacyjne oraz siłę społeczności Scratcha. Wchodzimy na stronę środowiska: https://scratch.mit.edu/ i omawiamy ogólne założenia. Można też skorzystać z materiałów zamieszczonych na stronie: http://programowanie.oeiizk.edu.pl, zakładka „Scratch".

Następnie należy omówić, w jaki sposób rozpocząć naukę programowania w środowisku Scratch, podać informacje na temat możliwości pracy *online* i *offline* oraz procedury zakładania konta. Praktyczne ćwiczenia rozpoczynamy od krótkiego omówienia specyfiki środowiska. Następnie poruszamy duszkiem i programujemy reakcję na różne zdarzenia. Już na początku zostaje wprowadzone pojęcie algorytmu, jako przepisu działania oraz programu, czyli

zapisu algorytmu w postaci zrozumiałej dla komputera. Warto wykorzystać scenariusz zajęć *Pierwsze kroki w Scratchu*.

Zasoby do wykorzystania:

- Strona OEIiZK – opis środowiska Scratch oraz „Słowniczek bloczków", (sekcja „Warszawa programuje!");
- Strona OEIiZK – moduł I „Pierwsze kroki w Scratchu", (sekcja „Mistrzowie Kodowania").

3.2. Przygotowujemy opowiadanie multimedialne

W ramach szkolenia uczymy, jak stworzyć opowiadania multimedialne, ale także, jak je wykorzystać w procesie dydaktycznym. Uczniowie chętnie tworzą takie opowieści – z podstawowych bloczków dotyczących duszka, sceny i multimediów powstać mogą ciekawe historyjki. Są to dobre ćwiczenia zarówno dla początkujących, jak i bardziej zaawansowanych użytkowników Scratcha. Ponadto, realizując je, w sposób naturalny łączymy umiejętności programistyczne z wiedzą z zakresu innych przedmiotów.

✓ Projekt „Dialog" (zakładka „Warszawa Programuje!")

Projekt „Dialog" opiera się na wizualizacji fragmentu wiersza *ZOO* Jana Brzechwy. Dialog odbywa się między dwoma duszkami – potrzebna jest więc umiejętność synchronizacji zdarzeń. Rozpoczynamy od wybrania z biblioteki tła sceny. Następnie przygotowujemy i rozmieszczamy na scenie dwa duszki – bohaterów dialogu. Należy przydzielić im właściwe wersy wiersza i zadbać o to, aby wyświetlanie poszczególnych partii dialogu odbywało się we właściwym czasie.

✓ Multimedialna pocztówka (zakładka „Mistrzowie Kodowania")

Opisujemy, w jaki sposób można stworzyć w programie Scratch multimedialną, interaktywną pocztówkę (np. świąteczną). Opisane zostały dwa przykładowe projekty: „Św. Mikołaj wpada do komina" oraz „Zapalamy lampki na choince". Oba projekty pobudzają do kreatywności, toteż realizując je, nie trzeba ściśle trzymać się scenariusza, lecz pozwolić uczniom wdrażać ich pomysły.

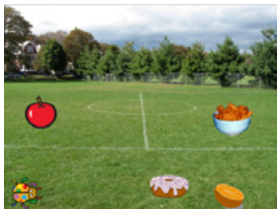
Zasoby do wykorzystania:

- Strona OEIiZK – scenariusz 1: *Dialog*, (sekcja „Warszawa programuje!”);
- Strona OEIiZK – Moduł VIII „Multimedialna pocztówka”, (sekcja „Mistrzowie Kodowania”);
- Studio z projektami „Warszawa programuje!”.

3.3. Tworzymy własne gry

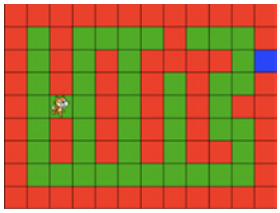
Tworzenie prostych gier to kolejna aktywność, która pochłania uczniów oraz stwarza okazję, by w sposób naturalny wprowadzić pojęcie zmiennej, np. do zliczania punktów. Przygotowując gry uczniowie, mogą zdobyć wiele umiejętności związanych z programowaniem, gdyż widzą atrakcyjny cel – stworzenie gry według własnego scenariusza. Mogą również przyswajać wiedzę z różnych dziedzin. Na przykład: wobec konieczności zdefiniowania kształtu lub opisania ruchu muszą myśleć w kategoriach matematycznych, wykorzystując algebrę i geometrię. Rozwijają umiejętności psychologiczne, socjologiczne, a także uczą się, jak się uczyć.

✓ Zdrowe odżywianie („Warszawa programuje!”)



Gra polega na sterowaniu żuczkiem za pomocą strzałek z klawiatury. Na planszy znajdują się inne obiekty reprezentujące zdrową i niezdrową żywność. Gdy żuczek „złapie” obiekt, pojawia się stosowny komunikat – w zależności od tego, czy zostało dotknięte np. jabłko, czy chipsy. Gra ma więc konkretne przesłanie edukacyjne.

✓ Kot w labiryncie („Mistrzowie Kodowania”)



Tworzymy grę, w której zadaniem gracza jest wyprowadzenie kota z labiryntu. Należy zbudować skrypty sterujące postacią za pomocą klawiatury. Można wykorzystać gotową planszę lub stworzyć własną. Dodatkowo projektujemy obsługę zdarzeń zachodzących na scenie, takich jak na przykład próba wejścia na ścianę, dotarcie do mety. Dodajemy punktację lub wzbogacamy projekt o nowe plansze – tworząc kilka poziomów gry.

✓ Gra zręcznościowa – odbijanie piłeczki („Mistrzowie Kodowania”)

Tworzymy grę, w której użytkownik steruje paletką, a jego zadaniem jest jak najdłużej odbijać piłeczkę. Może ona odbijać się od trzech ścian – na górze, z prawej i z lewej strony. Jeśli dotknie dolnej ściany – gracz przegrywa. Piłka może po pewnym czasie przyspieszać. Z zagadnień typowo programistycznych występują tu: sterowanie obiektem i zdarzenia, ale także definiowanie warunków. Pojawiają się też elementy fizyki – odbicie od ściany.

Zasoby do wykorzystania:

- Strona OEIiZK – scenariusz 2: *Zdrowe odżywianie*, (sekcja „Warszawa programuje!”);
- Strona OEIiZK – moduł III „Kot w labiryncie”, (sekcja „Mistrzowie Kodowania”);
- Strona OEIiZK – Moduł V „Odbijanie piłeczki”, (sekcja „Mistrzowie Kodowania”);
- Studio z projektami „Warszawa programuje!”.

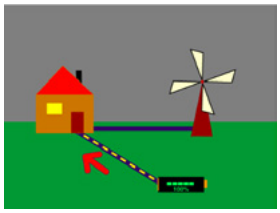
3.4. Opracowujemy symulacje

Środowisko Scratch umożliwia również przygotowanie projektów, których głównym celem jest poznawanie rzeczywistości. Budujemy specjalnie skonstruowany model jako uproszczony podzbiór rzeczywistości, korzystając z którego osoba ucząca się zdobywa – poprzez zabawę – nową wiedzę o świecie. Pobudzamy w ten sposób uczniów do formułowania pytań i poszukiwania odpowiedzi, rozwiązywania problemów teoretycznych i praktycznych. Poniżej przedstawiono dwa przykłady projektów do wykorzystania. Jeden z nich (np. „Elektrownia wiatrowa”) należy wykonać całościowo na zajęciach, natomiast drugi można przygotować w formie szablonu – w którym jedynie niektóre elementy zostaną oprogramowane – lub ewentualnie zaprezentować w wersji całkowicie opracowanej.

✓ Śluza wodna („Warszawa programuje!”)

Symulacja wyjaśnia mechanizm działania śluzy wodnej. Użytkownik obserwuje przepływanie łódki między zbiornikami o różnych poziomach wody. Opracowując projekt samodzielnie, uczniowie zapamiętują z łatwością zasady działania śluzy.

✓ **Elektrownia wiatrowa („Warszawa programuje!")**



Dzięki temu projektowi można wyjaśnić wykorzystanie siły wiatru do zasilania gospodarstwa domowego energią elektryczną. Pozwala on na symulację działania układu składającego się z elektrowni, akumulatora oraz domu zasilanego siłą wiatru bezpośrednio z wiatraka, bądź z akumulatora. W projekcie występuje rozbudowany system warunków. Programując, uczniowie poznają zależności logiczne i mają okazję zrozumieć ich zastosowanie. Dodatkowym atutem projektu jest wykorzystanie mikrofonu do symulacji działania wiatru. Im mocnej dmuchamy, tym szybciej wiatrak się kręci.

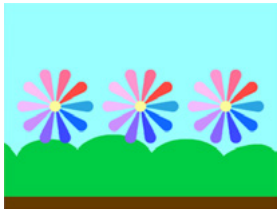
Zasoby do wykorzystania:

- Strona OEIiZK – scenariusz 5: *Śluza wodna*, (sekcja „Warszawa programuje!");
- Strona OEIiZK – scenariusz 6: *Elektrownia wiatrowa*, (sekcja „Warszawa programuje!");
- Studio z projektami „Warszawa programuje!".

3.5. Rysujemy

Po przygotowaniu kilku projektów warto przejść do zagadnień nieco trudniejszych. Z jednej strony będą to zadania oparte na wykorzystaniu grafiki żółwia w Scratchu, z drugiej – ukierunkowane na naukę algorytmiki. Implementując rysunki z wykorzystaniem pióra, warto zacząć od prostych przykładów. Jest to doskonała okazja do utrwalenia umiejętności tworzenia zapisu w języku programowania, powtarzania czynności oraz definiowania funkcji – własnych bloków. Uczniowie mogą szukać inspiracji, przeglądając projekty innych użytkowników – np. poprzez wpisanie frazy: `100% pen`.

✓ **Rabata („Warszawa programuje!")**



Użytkownik obserwuje podczas działania projektu kreślone przez duszka wielobarwne kwiatki układające się w rabatę. Zaczynamy od prostych rysunków, by potem przejść do coraz bardziej skomplikowanych. Można uczyć się metodą prób i błędów, ale warto również zachęcić do stosowania obliczeń. Jeśli chcemy uzyskać pracę efektowną pod względem estetycznym, warto różnicować grubość pisaka i kolory.

✓ Gwiazdozbiór („Warszawa programuje!”)

Po uruchomieniu gotowego projektu użytkownik obserwuje pojawiające się na niebie gwiazdki rysowane przez duszka. Przygotowując projekt, definiujemy własne bloki, by potem z gotowych poleceń, jakby „cegiełek”, budować całą prezentację. Można wprowadzić zarówno bloki bez parametru, jak i z parametrem. Opracowanie projektu stanowi doskonałe przygotowanie do dalszego kształcenia umiejętności programistycznych.

Zasoby do wykorzystania:

- Strona OEIiZK – scenariusz dodatkowy: *Rabata*, (sekcja „Warszawa programuje!”);
- Strona OEIiZK – scenariusz 9: *Gwiazdozbiór*, (sekcja „Warszawa programuje!”);
- Studio z projektami „Warszawa programuje!”.

3.6. Implementujemy algorytmy obliczeniowe

✓ Gra logiczna – zgadywanie liczby („Mistrzowie Kodowania”)

Tworzymy grę polegającą na odgadywaniu liczby wylosowanej przez komputer. Po każdej próbie gracz jest informowany, czy podana przez niego liczba jest mniejsza, większa, czy równa wartości pamiętanej przez komputer. Zabawa ma na celu odkrycie przez gracza algorytmu postępowania prowadzącego do jak najszybszego odgadnięcia wylosowanej liczby. Algorytm ten został wprowadzony na zajęciach podczas realizacji punktu 2.4. Na lekcji część z uczniów może przygotować odwrotną realizację algorytmu – gracz wymyśla liczbę, a komputer „zgaduje”.

Można także zaproponować przygotowanie skryptów do prostych obliczeń. Celem zadań może być „przeliterowanie” liczby – czyli wypisanie cyfra po cyfrze od końca (przykładowe zadanie 1) oraz znalezienie największej liczby z podanych (przykładowe zadanie 2).

Zasoby do wykorzystania:

- Strona OEIiZK – moduł VI „Zgadywanie liczby”, (sekcja „Mistrzowie Kodowania”).

3.7. Podsumowanie – programowanie w środowisku wizualnym

Powyżej zostało przedstawionych wiele zagadnień, które należy wyjaśnić podczas nauki podstaw programowania w środowisku wizualnym. Najczęściej ich omawianie stwarza okazję do pierwszego usystematyzowanego kontaktu uczniów ze światem programowania, toteż niezwykle ważne jest, by był to przekaz nie tylko czysto techniczny, ale również wzbogacony odwołaniem do głównych pojęć informatycznych. Przedstawione projekty należy traktować jako przykładowe – na zajęciach z uczestnikami szkolenia można zrealizować niektóre z nich lub dobrać inne. Należy jednak zadbać zarówno o różnorodność tematyczną, jak i o szerokie wykorzystanie konstrukcji programistycznych.

Na koniec tej części zajęć, przedstawiamy uczestnikom szkolenia informacje, gdzie można znaleźć ciekawe materiały do nauki podstaw programowania w Scratchu. Zachęcamy też do wymiany doświadczeń. Poszukując podpowiedzi, warto pamiętać o kartach Scratcha. Są one dobrze przygotowane pod względem metodycznym – przedstawiają obrazowo pojedyncze pojęcia w sposób zwięzły i zrozumiały dla ucznia (https://scratch.mit.edu/info/cards).

Część zaprezentowanych powyżej scenariuszy powstała w ramach projektu „Mistrzowie Kodowania". Polecamy nauczycielom zasoby tego projektu, które są dostępne na stronie: http://wiki.mistrzowiekodowania.pl. Można tam znaleźć kolejne projekty.

Trzecim źródłem, o którym nie można zapomnieć, jest strona główna Scratcha. Znajdują się tam przykładowe projekty – wraz z opisem, jak je wykonać – przygotowane przez twórców i współpracowników Scratcha (zakładka „Wskazówki") oraz publikowane przez użytkowników. Codziennie przybywa nowych, a na stronie głównej pojawiają się wybrane – w pewnym sensie najlepsze. Natomiast w zakładce „Przeglądaj" projekty są pogrupowane według rodzajów: animacje, sztuka, gry, muzyka, opowiadania i samouczki.

Środowisko Scratch zostało tu wybrane ze względu na jego ogromną popularność oraz bogactwo dostępnych materiałów zarówno w języku polskim, jak i angielskim. Jednak nie jest to jedyne środowisko do programowania wizualnego. Na zajęciach omawiamy, choćby skrótowo, także działanie innych środowisk – np. Kodable, Blockly. Posiadają one różne, choć zbliżone, możliwości. Na uwagę zasługuje w szczególności środowisko Blockly, które nie tylko może być adaptowane przez różne środowiska w zależności od potrzeb (np. do sterowania

robotami), ale wydaje się wartościowe samo w sobie. Ze względu na ograniczony zasób bloczków można w nim uczyć podstaw algorytmiki. Uczeń nie tyle skupia się na poznaniu środowiska, jego wielu możliwościach czy ciekawych rozwiązaniach multimedialnych, ile na oprogramowaniu algorytmów. Stąd już tylko jeden krok do programowania w języku tekstowym.

Podsumowując, w środowisku wizualnym łatwiej niż w tekstowym rozpocząć działania związane z programowaniem. Gdy przeciągamy bloczki, nie musimy troszczyć się o składnię – interfejs programu podpowiada odpowiednią formę. Trudniej też o zrobienie błędu. W miarę nabierania przez użytkowników wprawy okazuje się, że zaczynają oni postrzegać przeciąganie bloczków jako uciążliwe, gdyż są w stanie szybciej wprowadzić polecenia w trybie tekstowym. Rozbudowane skrypty stają się mało czytelne. Z czasem następuje potrzeba przejścia do środowiska tekstowego. Można zacząć np. od zadań opartych na wykorzystaniu grafiki żółwia w Pythonie.

Zasoby do wykorzystania:

- Karty Scratcha;
- Strona domowa Scratcha, „Wskazówki”: https://scratch.mit.edu/tips;
- Strona projektu „Mistrzowie Kodowania”;
- Strona Kodable (gra bezpłatna w wersji podstawowej);
- Gry do nauki programowania Blockly Games;
- Środowisko programowania Blockly.

4. Sterowanie obiektem za pomocą sekwencji poleceń

4.1. Grafika żółwia w Pythonie – wprowadzenie

Programowanie w języku tekstowym warto zacząć od wykorzystania grafiki żółwia – idei wypracowanej przez Seymoura Paperta. Stanowi ona podstawę języka Logo. Jednak motywy graficzne można również opracowywać z wykorzystaniem tego podejścia w języku Python z biblioteką Turtle. Grafika żółwia kojarzy się głównie z językiem Logo, ale warto zwrócić uwagę, że geometria żółwia była stosowana jeszcze przed powstaniem języka Logo. Występuje także w wielu narzędziach i środowiskach dedykowanych do nauki programowania dla dzieci, np. w Scratchu (projekty z wykorzystaniem pióra) czy „Godzinie kodowania” (chociaż niekoniecznie używa się w nich nazwy „grafika żółwia”) – została także wykorzystana wcześniej w tym szkoleniu.

Jakie cechy tego podejścia sprawiają, że można z jego pomocą uczyć programowania nawet małe dzieci? Można wskazać na trzy istotne aspekty:

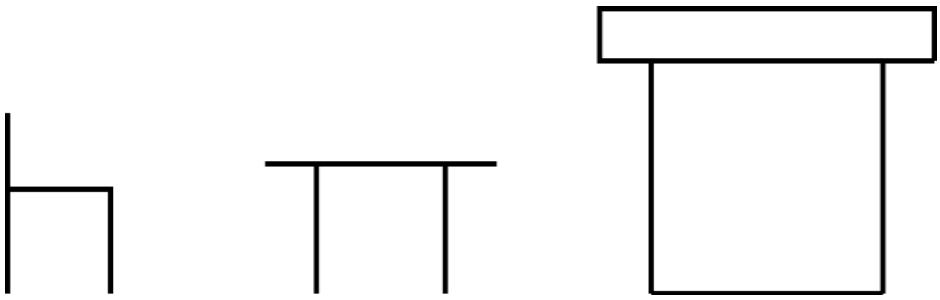
- proste i konkretne, a przez to zrozumiałe polecenia, zbliżone do języka naturalnego: „idź naprzód o... kroków", „obróć się o kąt...", „podnieś pisak" itp.;
- postać żółwia – graficzny symbol, który pozwala dziecku wyobrazić sobie wykonawcę algorytmu zapisanego w języku programowania;
- semantyka operacyjna – uczeń obserwuje działanie stworzonych przez siebie programów. Interpretacja kodu powoduje na ekranie efekt, który dość łatwo ocenić pod względem zgodności ze wzorcem. Widząc, jak żółw rysuje, uczeń sprawdza, czy wykonuje zadanie prawidłowo, może zatem znaleźć ewentualny błąd.

Podstawowe komendy:

Polecenie	Wyjaśnienie
`fd(n)`	*forward* – przesunięcie żółwia w aktualnym kierunku o n kroków
`bk(n)`	*backward* – przesunięcie żółwia przeciwnie do aktualnego kierunku o n kroków
`rt(alfa)`	*right* – obrót żółwia w prawo o kąt `alfa`
`lt(alfa)`	*left* – obrót żółwia w lewo o kąt `alfa`
`pu()`	*pen up* – żółw podnosi pisak, czyli nie rysuje podczas przemieszczania się
`pd()`	*pen down* – żółw opuszcza pisak, czyli rysuje podczas przemieszczania się

Importowanie biblioteki *Turtle: from turtle import* *

Zaczynamy pracę w języku Python od wykonywania prostych obliczeń w trybie interaktywnym, następnie wprowadzamy proste zadania graficzne – np.: rysowanie stołu, krzesła, mostu, domu, itp., tworząc ciąg poleceń w pliku.



Rysunek 2. Przykładowe rysunki

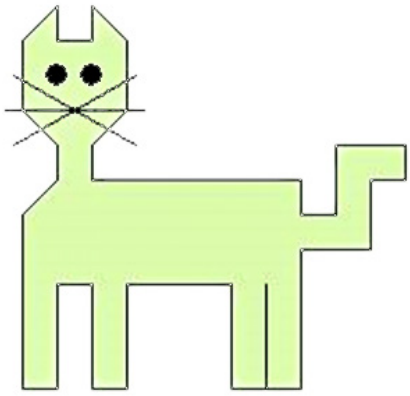
Zasoby do wykorzystania:

- Jochemczyk W., Olędzka K., (2013), *Python dla wszystkich*, Toruń: Informatyka w Edukacji, (zakładka: „O Pythonie”);
- Film *Pythonowe początki*;
- Materiały OEIiZK: http://python.oeiizk.edu.pl/, (zakładka: „Rysowanie z żółwiem”).

4.2. Uczymy żółwia nowych słów

Kod złożony z fragmentów stanowiących logiczną całość warto podzielić na elementy określające pojedyncze funkcje. Dzięki temu możemy wielokrotnie wykorzystać dany kod, a cały program staje się bardziej zwięzły i przejrzysty.

Uczestnicy szkolenia powinni spróbować narysować dobrze znane przedmioty. Najlepiej zacząć od rysunków, które zawierają tylko kąty proste, gdyż ich wykonanie jest łatwiejsze dla uczniów, a następnie przejść do wykorzystania także innych kątów. Pewną trudność mogą sprawdzać rysunki, w których występują nietypowe długości odcinków – np. przekątna kwadratu. Uczestnicy powinni wykonać również swoje własne rysunki. Na początku uczniowie chętniej rysują motywy konkretne, niż abstrakcyjne wzory. Poniżej przykładowy rysunek (uwaga: nieco pracochłonny).



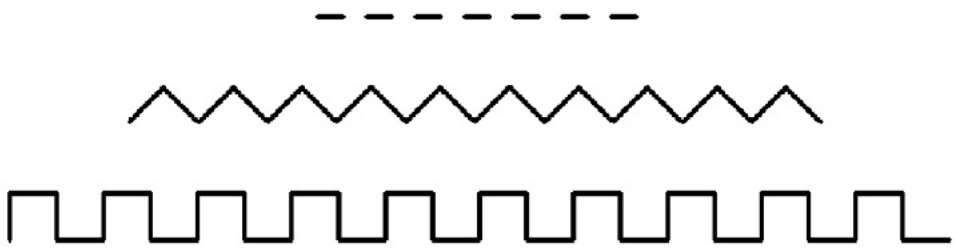
Rysunek 3. Efekt wywołania funkcji `kot()`

4.3. Powtarzamy czynności

Najlepiej zacząć od narysowania kwadratu, najpierw ręcznie – zapisując wszystkie instrukcje (i od razu je grupując), a później z wykorzystaniem instrukcji `for`. Następnie można przejść do tworzenia innych wielokątów foremnych: trójkąta, pięciokąta, sześciokąta itd. Chociaż pierwsze rysunki zapewne będą powstawać metodą prób i błędów, należy omówić, jak wyliczać kąty.

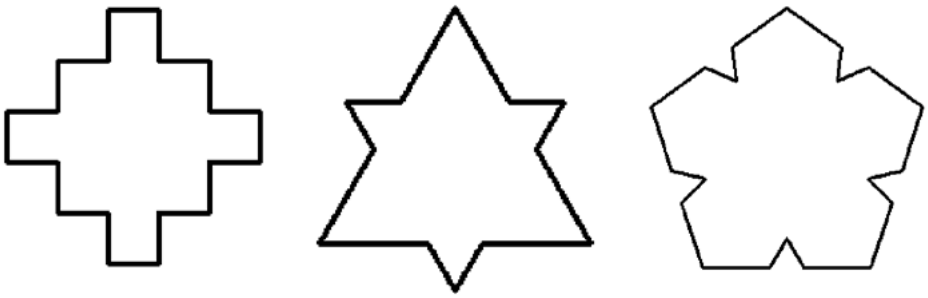
Instrukcja iteracji jest bardzo ważna w nauce programowania i nie można tego tematu zlekceważyć. Trzeba poświęcić odpowiednią ilość czasu na ćwiczenia. Zaczynamy od prostych przykładów, na których wyraźnie widać, jakie elementy się powtarzają. Następnie można przejść do trochę trudniejszych, by dobrze wyćwiczyć umiejętność stosowania instrukcji iteracji. Można też nawiązać do znanego już z programowania w Scratchu bloczka `powtórz`.

Pierwszą grupę przykładowych zadań stanowią różnego rodzaju szlaczki.



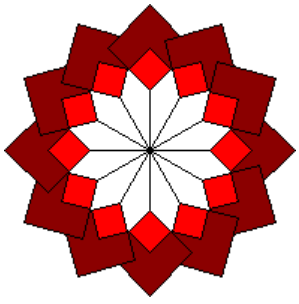
Rysunek 4. Szlaczki (przykładowe zadanie 3)

Kolejne zadania mogą opierać się na tworzeniu motywów, których rysowanie zaczyna i kończy się w tym samym miejscu.



Rysunek 5. Różne wzory (przykładowe zadanie 4)

W celu urozmaicenia motywów warto wzbogacić je o kolory.



Rysunek 6. Efekt wywołania funkcji `kwiat()` (przykładowe zadanie 5)

4.4. Powtarzamy i podejmujemy decyzje

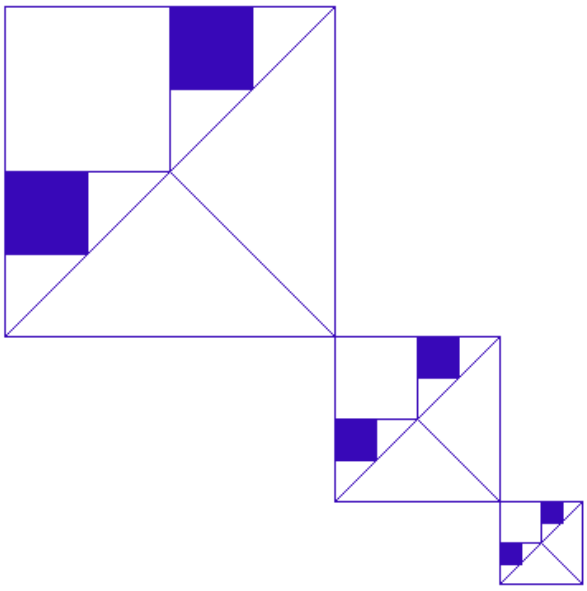
W dalszej części szkolenia można wprowadzić modyfikacje do wzorów powtarzalnych. W zależności od tego, po raz który jest wykonywana pętla, element powtarzalny może być inny lub różnić się jedynie parametrem.

✓ **Przykładowy scenariusz – *Rysowanie grzebieni* (Scenariusz 1)**

4.5. Dzielimy problem na problemy cząstkowe

Jeśli w motywie występują figury podobne do siebie – różnej wielkości, koloru lub o równej liczbie elementów powtarzających się – można napisać funkcję z parametrem. Gdy chcemy narysować mały kwadrat, średni kwadrat oraz duży kwadrat – należy zdefiniować jedną funkcję z parametrem: `kwadrat(bok)`. Definiujemy ją analogicznie jak bezparametrową. Jedyna różnica polega na wymienieniu w nagłówku wszystkich parametrów – kolejne z nich oddzielamy przecinkami.

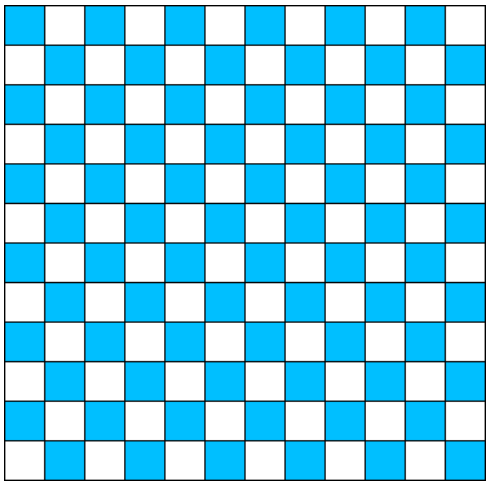
Przykład zadania:



Rysunek 7. Przykład motywu, w którym należy zdefiniować funkcję z parametrem

4.6. Projektujemy posadzki

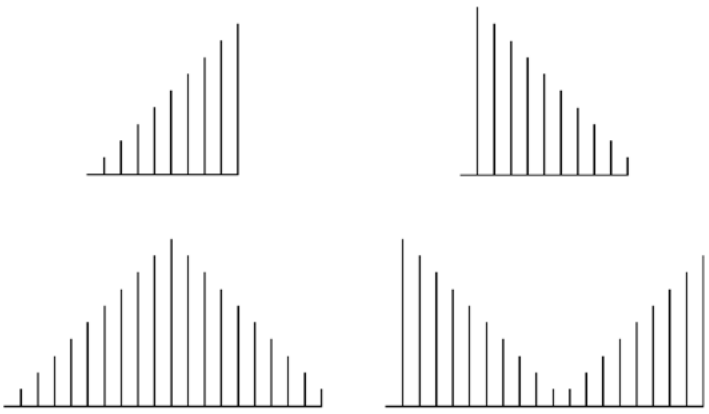
Po zrealizowaniu prostszych zadań z wykorzystaniem iteracji, należy przejść do bardziej skomplikowanych. Są to projekty typu „posadzka", w których jeden element powtarzany jest wiele razy. Wykonując je, mamy do czynienia z zagnieżdżonymi pętlami. Rozwiązanie można zapisać na wiele sposobów.



Rysunek 8. Posadzka (przykładowe zadanie 6)

4.7. Budujemy piramidy

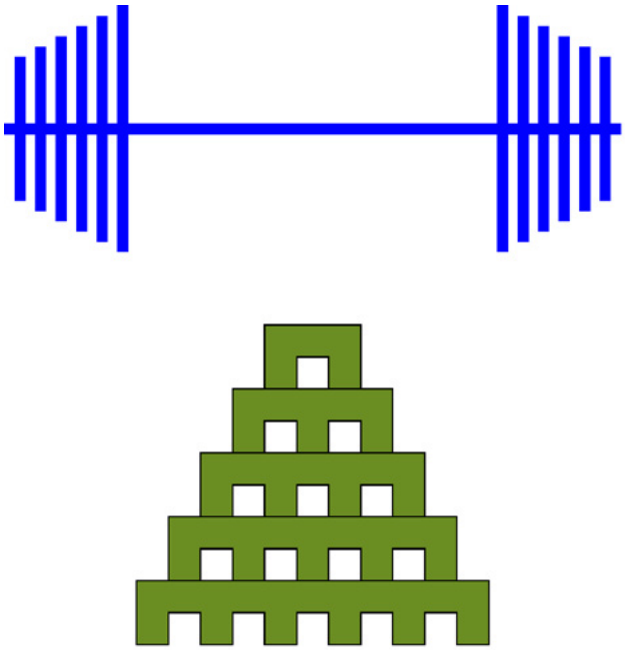
Ostatnią – najtrudniejszą – grupę zadań stanowią te oparte na wykorzystaniu wartości zmiennej sterującej pętli `for`. Należy zacząć od prostych zadań polegających na rysowaniu odcinków, prostokątów lub innych figur – od najmniejszej do największej lub na odwrót.



Rysunek 9. Różne „ząbki" (przykładowe zadanie 7)

Podobnie jak poprzednio – w celu uatrakcyjnienia rysowanych motywów warto dodać kolory.

Trudniejsze przykłady wymagają napisania kilku funkcji. Kilka takich zadań powinno się zrealizować na zajęciach z nauczycielami, by potem mogli w analogiczny sposób pracować ze zdolną młodzieżą.

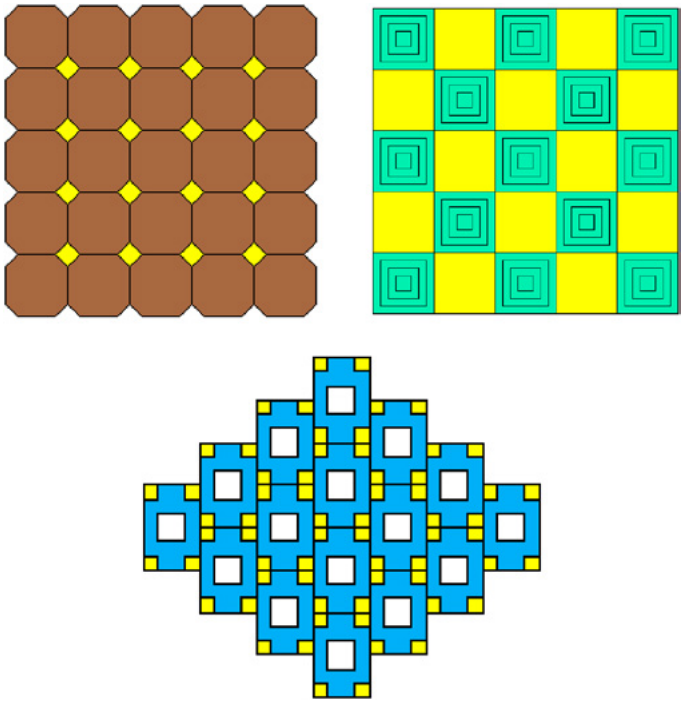


Rysunek 10. Przykłady rysunków z wykorzystaniem zmiennej sterującej pętli (przykładowe zadania 8 i 9)

4.8. Rozwiązujemy zadania

Uczniowie, rozwiązując proste problemy, poznają polecenia języka programowania i utrwalają ich znajomość. Następnie powinni przystąpić do realizowania bardziej złożonych zadań. Niekoniecznie muszą one dotyczyć trudniejszych zagadnień – na to przyjdzie czas w klasach starszych. Uczniowie dowiadują się, jak dzielić problem na problemy cząstkowe i stosować nowe umiejętności w sytuacjach typowych oraz mniej typowych. Możemy też omówić z nimi technikę projektowania algorytmów (*top-down*) i ich implementacji (*bottom-up*). Trzeba pamiętać, że są to pierwsze kroki uczniów w obszarze programowania, dlatego nie powinni czuć się przytłoczeni. Inaczej jest, gdy prowadzimy zajęcia dla nauczycieli – trzeba zadbać, by mieli szersze spojrzenie na dany aspekt.

Po wykonaniu kilku zadań omawiamy etapy rozwiązywania problemów informatycznych. Jest to dopiero początek na drodze do poznawania algorytmiki, ale warto już na tym etapie kształtować prawidłowe myślenie. Omawiamy kolejno, na czym polegają opis i analiza sytuacji problemowej, sporządzanie specyfikacji problemu, projektowanie rozwiązania, komputerowa realizacja rozwiązania oraz jego testowanie i prezentacja. Szczególnie należy podkreślić, że przy wykonywaniu konkretnego zadania najpierw trzeba przemyśleć sposób jego rozwiązania, a dopiero później zacząć pisać kod. Wiele spośród zadań można bowiem rozwiązać za pomocą różnych metod.



Rysunek 11. Przykłady trudniejszych zadań

W nauce programowania bardzo istotne jest testowanie.
Przede wszystkim staramy się na bieżąco reagować na błędy – najczęściej wychwytujemy błędy syntaktyczne. Po zakończeniu pisania kodu, powinniśmy również skrupulatnie przetestować rozwiązanie, gdyż błędy logiczne są dużo trudniejsze do wykrycia. Każde zadanie sprawdzamy, przyjmując wartości brzegowe parametrów i kilka pośrednich. Jeśli możliwych wartości parametrów jest niewiele, zaleca się przeprowadzenie testowania dla wszystkich.

Skąd powinniśmy czerpać pomysły na zadania? Przede wszystkim należy nawiązywać do zjawisk typowych dla życia codziennego. Uczniowie chętniej

zajmują się problemami, które uważają za bliskie. Można także skorzystać z różnych serwisów, są one jednak adresowane głównie do uczniów starszych. Niektóre z nich mogą być wykorzystywane w nauce na poziomie szkoły podstawowej. Na stronach Przedmiotowego Konkursu Informatycznego „miniLOGIA" (dla uczniów szkół podstawowych województwa mazowieckiego) zamieszczono wiele zadań graficznych (http://minilogia.oeiizk.waw.pl) – znajdują się wśród nich zarówno prostsze, jak i trudniejsze, co pozwala dostosować poziom trudności zadania do bieżącego stopnia zaawansowania uczniów. Zadania o wyższym stopniu trudności można też znaleźć na stronie konkursu „LOGIA" – przeznaczonego dla uczniów klas z gimnazjalnych (http://logia.oeiizk.waw.pl).

Zasoby do wykorzystania:

- Strona konkursu „miniLOGIA": http://minilogia.oeiizk.waw.pl/;
- Strona konkursu „LOGIA": http://logia.oeiizk.waw.pl;
- Strona konkursu „Bóbr": https://www.bobr.edu.pl/;
- Strona projektu „Godzina Kodowania": https://code.org/.

4.9. Sterujemy robotem

Na szkoleniu należy wspomnieć o różnych rodzajach robotów, takich jak np. Ozobot, WeDo, Dash&Dot, Photon, mBot. W zależności od posiadanych zasobów warto przeprowadzić krótkie warsztaty z wybranym oprogramowaniem i sprzętem. Ponadto uczestnicy mogą wymienić się swoimi doświadczeniami związanymi z tym, jak pozyskać wybrane pomoce dydaktyczne dla szkoły, lecz również ze sposobami wykorzystania ich w pracy z uczniami.

Zajęcia z robotyki można uzupełnić o micro:bit. Jest to mikrokomputer zaprojektowany do pracy z dziećmi, który można zaprogramować przy użyciu języka wizualnego lub tekstowego – np. Python. Warto przygotować różne projekty, np. związane z IoT (ang. *Internet of Things* – internet rzeczy).

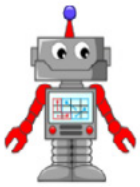
Należy zwrócić uwagę, że realizacja zapisów podstawy programowej możliwa jest bez wykorzystania robotów. Mogą one jednak wzbogacić i uatrakcyjnić zajęcia, więc warto wykorzystać sprzęt, którym już dysponujemy, lub zakupić go, jeśli placówka ma taką możliwość.

Zajęcia praktyczne związane z elementami robotyki można uzupełnić zagadnieniami o charakterze teoretycznym – np. analizując wybrane problemy z archiwów konkursu „Bóbr" (https://www.bobr.edu.pl/).

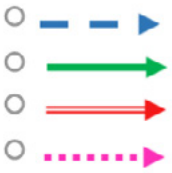
✓ Zadanie: Spacer robota

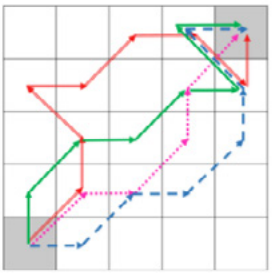
Spacer robota

Robot znajduje się w lewym dolnym rogu siatki. Chce on dojść do prawego górnego rogu. W tym celu może przechodzić z pola na sąsiednie pole i wykonywać następujące ruchy:
po skosie do góry w lewo:
po skosie do góry w prawo.
pionowo do góry:
w prawo.
Robot nie może wykonać dwóch takich samych ruchów pod rząd.



Jedna z dróg robota zaznaczonych na kolorowo nie jest zgodna z przyjętymi powyżej zasadami. Która?



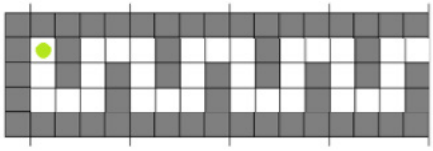


Odpowiedź: Niedozwolony jest ruch po skosie na prawo w dół – przedostatni ruch „czerwonego” robota.

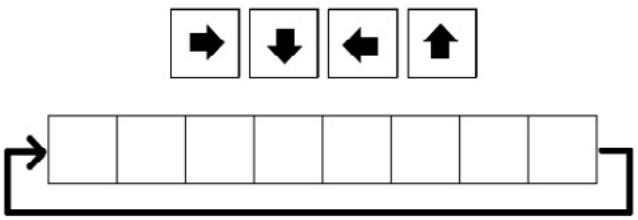
✓ Zadanie: Robot

Robot

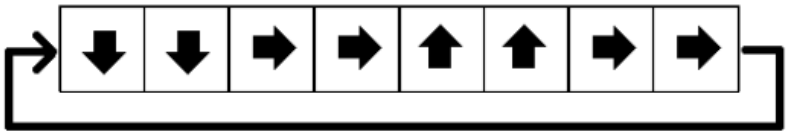
Pomóż zielonemu robotowi wyjść z labiryntu.



Przeciągnij strzałki, aby utworzyły ciąg poleceń, który robot ma wykonać 4 razy.



Odpowiedź: Ciągiem poleceń, które musi wykonać robot, jest:



Zasoby do wykorzystania:

- Robot Ozobot – informacje;
- Zestawy Lego WeDo;
- Bloki Scratcha w integracji z Lego WeDo;
- Strona producenta Lego WeDo;
- Scenariusze lekcji z wykorzystaniem robotów Dash i Dot;
- Robot Photon – informacje;
- Strona konkursu „Bóbr".

5. Praca z uczniami o różnych potrzebach edukacyjnych

Niezwykle ważnym zadaniem nauczyciela jest wyszukiwanie i rozwijanie talentów uczniów. Informatyka daje ogromne możliwości realizowania pasji uczniów, gdyż promuje nie tylko tych postrzeganych w szkole jako zdolni, ale aktywizuje słabszych. Praktycznie każdy młody człowiek może połączyć swoje zainteresowania z tą dziedziną wiedzy, by je lepiej rozwinąć. Zadaniem nauczyciela jest stymulować ucznia do nieustannego rozwoju i towarzyszyć mu w miarę możliwości na tej drodze.

W wielu szkołach prowadzone są koła zainteresowań. Na szkoleniu nauczyciele mogą wymienić się doświadczeniami na temat ich działania. Ponadto warto zachęcić uczestników do zaangażowania się w przygotowywanie uczniów do konkursów. Przykładem takiego konkursu jest wspomniany wcześniej „Bóbr" (https://www.bobr.edu.pl/). Warto, żeby nauczyciele, poznając jego ogólne założenia, rozwiązali kilka zadań z kategorii „Benjamin". Można uczestników podzielić na trzyosobowe grupy i każdej grupie przydzielić kilka zadań, a następnie wspólnie je omówić.

Na szkoleniu należy też wspomnieć, w kontekście pracy z uczniami o różnych potrzebach edukacyjnych, o możliwości wykorzystania aplikacji multimedialnych do wspomagania rozwoju uczniów ze specyficznymi trudnościami. Na przykład w Scratchu istnieje wiele opracowanych aplikacji, które wspomagają uczenie się.

Przygotować je mogą nauczyciele dla uczniów – bądź uczniowie dla siebie nawzajem. Przykłady takich projektów to „Spadające jabłuszka" i „Układanie wyrazów". W pierwszym uczniowie zbierają „dobre jabłka" – np. wyrazy poprawnie zapisane pod względem ortograficznym, w drugim z liter układają wyrazy. Gotowe projekty można modyfikować, dodając inne ćwiczenia. Gry o charakterze edukacyjnym znajdują się również na stronie http://wiki.mistrzowiekodowania.pl w punkcie: *Szkoła podstawowa (klasy 4–6)*, *zaawansowane*.

Nawet jeśli podczas szkolenia projekty te nie będą przygotowywane, warto zaprezentować je w formie pokazu. Źródłem ciekawych projektów edukacyjnych jest również strona Scratcha. Na szkoleniu można poświęcić trochę czasu na wyszukanie kilku wartościowych projektów. Uczestnicy nauczą się nie tylko, jak poszukiwać odpowiednich materiałów, ale także, jak je oceniać.

Zasoby do wykorzystania:

- Strona konkursu „Bóbr": https://www.bobr.edu.pl/;
- Studio z projektami edukacyjnymi: https://scratch.mit.edu/studios/4487109;
- Strona projektu „Mistrzowie Kodowania": http://wiki.mistrzowiekodowania.pl/;
- Strona domowa Scratch: https://scratch.mit.edu.

6. Podsumowanie

Na koniec szkolenia warto jeszcze raz wrócić do zapisów w podstawie programowej nie tylko po to, by przypomnieć treści omawiane na szkoleniu, ale aby pokazać je w szerszej perspektywie. Nauczyciel musi bowiem dobrze orientować się w materiale, którego naucza, ale także wiedzieć, co uczniowie powinni umieć, zanim przyjdą do czwartej klasy oraz czego będą się uczyć w klasach starszych. Konieczność ta wynika również ze spiralnego układu treści nauczania w podstawie programowej.

PRZYKŁADOWE SCENARIUSZE ZAJĘĆ

Scenariusz 1 – *Rysujemy grzebienie*

Opis zajęć

Głównym celem zajęć jest nauka powtarzania czynności oraz podejmowania decyzji. Uczniowie nabywają tych umiejętności i utrwalają je podczas realizacji zadania: Grzebienie, które zapisują i uruchamiają w języku Python.

Czas trwania

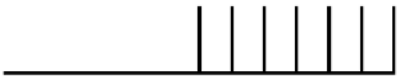
45 minut

Realizacja

Zaczynamy od narysowania grzebienia mającego ząbki jednakowej wysokości. Następnie wprowadzamy modyfikacje – co drugi lub co trzeci ząbek musi być inny. Na kolejnym etapie zajęć uczestnicy przygotowują dla siebie nawzajem grzebienie – zagadki. W ten sposób każdy ma możliwość, by zaprojektować swój własny wzór, ale także, aby opisać wzór wymyślony przez innego uczestnika.

✓ **Zadanie: Grzebień prosty**

Zdefiniuj funkcję: `grzebien(ile)`, po wywołaniu której powstanie rysunek grzebienia składającego się z trzonka oraz identycznych ząbków. Liczbę ząbków określa parametr `ile`, który może przyjmować wartości od 2 do 20. Długość trzonka wynosi 60, odległości między ząbkami 10, a wysokość ząbków 20.



`grzebien(7)`

`grzebien(10)`

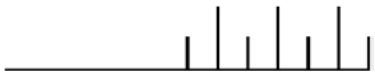
Zauważmy, że wszystkie odległości są wielokrotnościami 10, dlatego warto zdefiniować zmienną pomocniczą a, która pozwoli określić proporcje: trzonek będzie miał długość 6*a, odstęp między ząbkami a, natomiast wysokość ząbków 2*a.

Najpierw żółw przesuwa się o długość trzonka pomniejszoną o a, gdyż ten odcinek rysujemy razem z pierwszym ząbkiem. Później kolejno przesuwamy żółwia i rysujemy ząbki. Pojedynczy krok składa się z przesunięcia o a, obrotu w lewo o 90°, przesunięcia w przód o 2*a, wycofania się i obrotu w prawo o 90°.

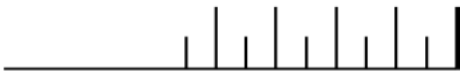
```
1.  from turtle import *
2.
3.  def grzebien(ile):
4.      a = 10
5.      fd(5*a)
6.      for i in range(ile):
7.          fd(a)
8.          lt(90)
9.          fd(2*a)
10.         bk(2*a)
11.         rt(90)
```

✓ Zadanie: Co drugi ząbek krótszy

Zdefiniuj funkcję `co_drugi(ile)`, po wywołaniu której powstanie rysunek grzebienia składającego się z trzonka oraz ząbków. Parametr `ile` określa liczbę ząbków. Ząbki mają na przemian wysokość 10 i 20. Parametr `ile` może przyjmować wartości od 2 do 20. Długość trzonka wynosi 60, a odległości między ząbkami 10.



`co_drugi(7)`



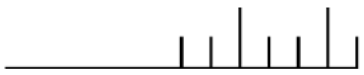
`co_drugi(10)`

Rozwiązując to zadanie, modyfikujemy funkcję napisaną poprzednio. Jedyną zmianą, jaką wprowadzamy, jest rysowanie raz wyższego, raz niższego ząbka. Uzależniamy to od wartości zmiennej sterującej pętlą: `i`. Kolejne wartości zmiennej `i` to: 0, 1, 2, 3, 4 i tak dalej, aż do: `ile-1` – wobec czego możemy rozpatrzyć resztę z dzielenia przez 2. Dla liczby parzystej otrzymamy resztę 0, a dla nieparzystej 1. Pozwoli to na zapisanie warunku, który będzie określał wysokość ząbka.

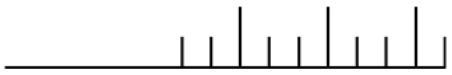
```
1.  def co_drugi(ile):
2.      a = 10
3.      fd(5*a)
4.      for i in range(ile):
5.          fd(a)
6.          lt(90)
7.          if i % 2 == 1:
8.              fd(2*a)
9.              bk(2*a)
10.         else:
11.             fd(a)
12.             bk(a)
13.         rt(90)
```

✓ Zadanie: Co trzeci ząbek wyższy

W kolejnym kroku komplikujemy zadanie – teraz co trzeci ząbek musi być wyższy.



co_trzeci(7)



co_trzeci(10)

```
1.  def co_trzeci(ile):
2.      a = 10
3.      fd(5*a)
4.      for i in range(ile):
5.          fd(a)
6.          lt(90)
7.          if i % 3 == 2:
8.              fd(2*a)
9.              bk(2*a)
10.         else:
11.             fd(a)
12.             bk(a)
13.         rt(90)
```

✓ **Zadanie: Projektujemy własne wzory**

Kolejnym zadaniem, jakie stawiamy przed uczestnikami, jest zaprojektowanie własnego wzoru. Korzystając z wcześniej napisanych programów i modyfikując je, projektujemy inne wzory. Następnie uczestnicy wymieniają się pomysłami i próbują wzajemnie odtworzyć zaprojektowane przez siebie wzory. Pozwala to z jednej strony na kreatywność, a z drugiej na rozwijanie różnych umiejętności – analizy rysunku, modyfikowania kodu programu, testowania.

Podsumowanie

W czasie tych zajęć koncentrujemy się na opisaniu w języku formalnym (programowania) zależności, które można dostrzec na rysunku. Analizujemy motyw, by sformułować zależność, a następnie zapisać ją w języku Python. Ćwiczymy rozwiązywanie problemu z zastosowaniem instrukcji iteracji i instrukcji warunkowej. Uczestnicy poznali już wcześniej te konstrukcje, wykonując zadania wymagające użycia Scratcha i arkusza kalkulacyjnego.

Podczas programowania warto zwrócić uwagę na błędy popełniane przy pisaniu programów. Trzeba nie tylko pomóc je znaleźć i poprawić, ale także pokazać, jak je wyszukiwać. Często początkujący programiści mają z tym problem.

Naturalne rozszerzenie projektu opiera się na dodawaniu wzorów kolorowych. Można zatem zaproponować inny regularnie powtarzający się motyw i rozszerzyć go.

Scenariusz 2 – *Ryby w akwarium*

Opis zajęć

Głównym celem zajęć jest przygotowanie symulacji w arkuszu kalkulacyjnym. Pokazujemy, jak zapisać algorytm rozwiązujący problem za pomocą arkusza kalkulacyjnego. Podczas zajęć zapisujemy algorytm za pomocą formuł.

Czas trwania

45 minut

Realizacja

Sformułowanie problemu

Stefan hoduje ryby w dużym akwarium. Na początku ma 10 ryb. Każdego dnia liczba ryb podwaja się, jeżeli jednak przekroczy 500 – ginie 9/10 z nich. Ponieważ liczba ryb musi być wyrażona liczbą całkowitą, po każdym dniu liczbę ryb, które zostają, należy zaokrąglić. Oblicz, ile będzie ryb po 100 dniach. Wskaż największą i najmniejszą liczbę ryb w rozważanym okresie.

Zajęcia rozpoczynamy od przedstawienia uczestnikom problemu do rozwiązania. Prezentujemy główną ideę oraz dokładnie omawiamy każdy warunek. Wskazane jest, aby rozpocząć symulację, licząc ręcznie lub z wykorzystaniem kalkulatora. Jeśli mamy do dyspozycji tablicę multimedialną, warto ją w tej części zajęć wykorzystać. Realizujemy najpierw właściwą symulację, dopiero później znajdujemy maksimum i minimum. Na koniec podsumowujemy całe zajęcia.

Symulacja krok po kroku

Na początku mamy 10 ryb. Drugiego dnia – i każdego następnego dnia – liczba ryb się podwaja. Zapisujemy kolejne wartości na tablicy. Dopiero w momencie, kiedy liczba ryb przekroczy próg 500, część ryb ginie. Po szóstym dniu jest 320 ryb, wobec tego następnego dnia jest ich 640. Ósmego dnia liczba ryb wynosi 64, gdyż 9/10 ginie.

Lista kroków

W momencie, gdy wszyscy uczestnicy rozumieją, jak powinien działać algorytm, warto zapisać go bardziej formalnym językiem.

1. Przypisz w zmiennej `liczba_ryb` wartość 10.
2. Powtórz 100 razy:
    - Jeśli liczba `liczba_ryb`>500, to zmiennej `liczba_ryb` przypisz wartość `0,1*liczba_ryb`; wartość zaokrąglij do części całkowitej. W przeciwnym przypadku zmiennej `liczba_ryb` przypisz wartość `2*liczba_ryb`.
3. Wypisz wartość zmiennej `liczba_ryb`.

Rozwiązanie problemu w arkuszu kalkulacyjnym

Przygotowujemy arkusz, w którym będziemy przeprowadzać symulację.

	A	B	C	D	E	F	G
1	**Zadanie – ryby w akwarium**						
2					po 100 dniach	150	
3	1	10			min	10	
4	2	20			max	944	
5	3	40					
6	4	80					
7	5	160					
8	6	320					
9	7	640					
10	8	64					
11	9	128					
12	10	256					
13	11	512					
14	12	51					

Wpisujemy wartość początkową, kolejne dni, a następnie odpowiednią formułę.

	A	B
2		
3	1	10
4	2	=JEŻELI(B3>500;ZAOKR(0,1*B3;0);2*B3)
5	3	=JEŻELI(B4>500;ZAOKR(0,1*B4;0);2*B4)
6	4	=JEŻELI(B5>500;ZAOKR(0,1*B5;0);2*B5)
7	5	=JEŻELI(B6>500;ZAOKR(0,1*B6;0);2*B6)

wartość początkowa

wpisana formuła

skopiowane formuły

Następnie formułę kopiujemy, przeciągając ją na niższe komórki. Na koniec warto zadbać o odpowiednie sformatowanie. Ważnym elementem pracy jest porównanie wyników otrzymanych wcześniej z tymi, które mamy w arkuszu. Pozwala nam to ocenić poprawność rozwiązania.

Obliczenie maksimum i minimum

Podobnie jak poprzednio, najpierw wykonujemy symulację „na sucho", prezentując, jak działa algorytm znajdowania największego i najmniejszego elementu. Kolejne kroki zapisujemy na tablicy, a następnie implementujemy znajdowanie maksimum w arkuszu kalkulacyjnym. Implementacja jednak jest tutaj dużo prostsza, gdyż korzystamy z wbudowanych funkcji `min` i `max`.

Podsumowanie

Podczas zajęć przeprowadziliśmy symulację obliczeń dotyczących liczby ryb w akwarium w określonym czasie. Zaczęliśmy od budowania modelu, potem przeszliśmy do analizy algorytmu i próby jego zapisu w postaci listy kroków, by dojść do obliczeń w arkuszu kalkulacyjnym. Końcowy etap pracy to testowanie. Pokazujemy w ten sposób kolejne kroki prowadzące do rozwiązywania problemów. Liczy się przede wszystkim pomysł rozwiązania, a narzędzia informatyczne wspomagają jedynie ten proces.

W ramach rozszerzenia można zaproponować wykonanie symulacji z wykorzystaniem środowiska Scratch, co stanowi ciekawe doświadczenie dla uczestników szkolenia.

PRZYKŁADOWE ZADANIA

Zadanie 1: „Przeliteruj” liczbę

Zadanie polega na wypisaniu cyfr liczby, poczynając od najmniej znaczącej, czyli od końca. Np. dla liczby 2018 należy kolejno wypisać: 8, 1, 0 i 2.

Rozwiązując zadanie, wykorzystujemy podstawowe operacje na liczbach całkowitych:
2018/10 = 201 reszty 8
201/10 = 20 reszty 1
20/10 = 2 reszty 0
2/10 = 0 reszty 2.

Zapisujemy algorytm w postaci listy kroków:

1. Wczytaj liczbę.
2. Dopóki liczba > 0, wykonuj:
    - Oblicz resztę z dzielenia liczby przez 10.
    - Wypisz resztę.
    - Odejmij od liczby resztę z dzielenia.
    - Podziel liczbę przez 10.

Na podstawie powyższego algorytmu możemy ułożyć skrypt w Scratchu. Należy zwrócić uwagę, że w języku Scratch nie występuje operacja dzielenia całkowitego, dlatego od liczby odejmujemy resztę z dzielenia.

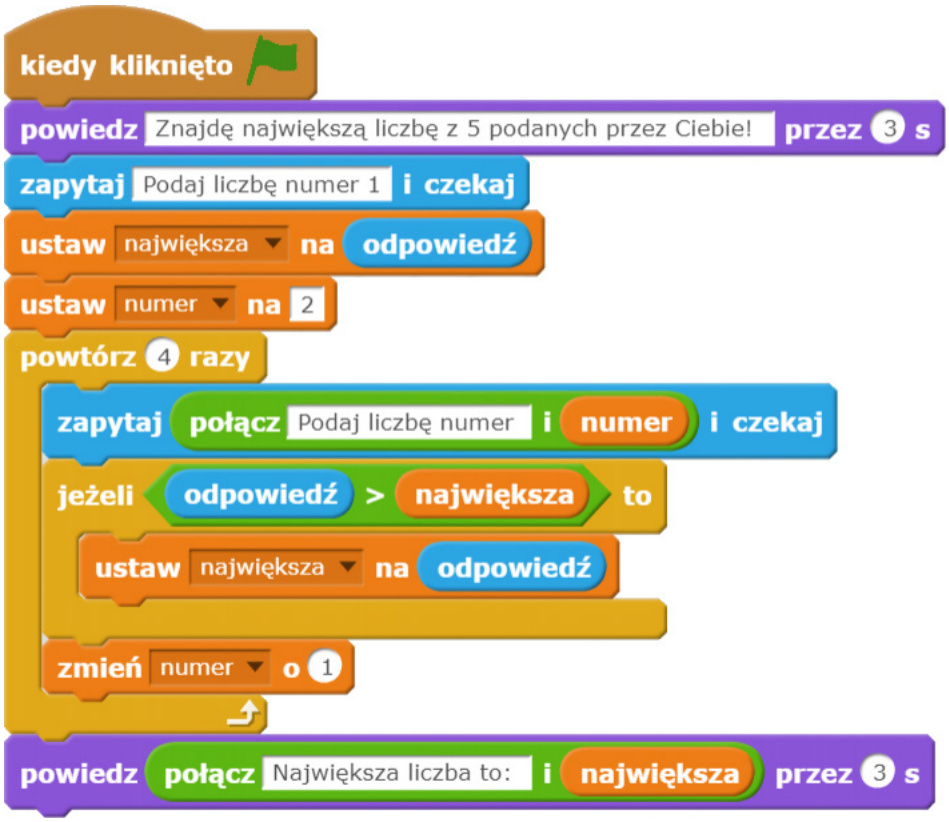
Rysunek 12. Rozwiązanie zadania: „Przeliteruj” liczbę

Zadanie 2: Największa liczba

Zadanie polega na znalezieniu największej liczby spośród kilku podanych. Podobnie, jak w poprzednim zadaniu, najpierw zapisujemy algorytm w postaci listy kroków. Przyjmując, że zostanie podanych pięć liczb, tworzymy następujący algorytm:

1. Wczytaj liczbę.
2. Zapamiętaj w zmiennej `największa` wczytaną liczbę.
3. Powtórz 4 razy:
    - Wczytaj liczbę.
    - Jeśli jest ona większa od `największa`, to:
        - Zapamiętaj w zmiennej `największa` wczytaną liczbę.
4. Wypisz wartość zmiennej `największa`.

Na podstawie powyższego algorytmu możemy ułożyć skrypt w Scratchu.

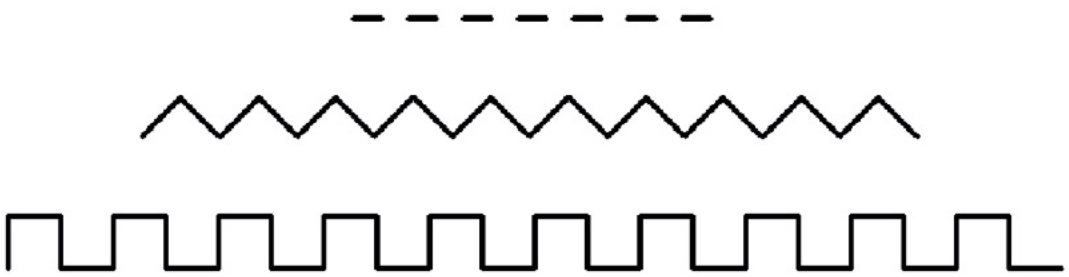


Rysunek 13. Rozwiązanie zadania: Największa liczba

Zadanie daje wiele możliwości modyfikacji, np. wczytania, z ilu liczb poszukujemy największej, lub poszukiwania liczby najmniejszej.

Zadanie 3: Szlaczki

Napisz funkcje: `linia_przerywana()`, `gory()`, `mur()`, po wywołaniu których powstaną rysunki takie jak poniżej. Długości odcinków wynoszą 30.



Rysunek 14. Szlaczki

Analizując powyższe rysunki, należy odpowiedzieć na pytanie: jaki element się powtarza i ile razy. W pierwszym przykładzie są to odcinek i przerwa, a układ powtarza się 7 razy. W drugim przykładzie jest to jedna góra, która powtarza się 10 razy – żółw stoi odchylony od poziomu pod kątem 45° w górę i powtarza 10 razy rysowanie odcinka do góry, obrót o 90° w prawo, rysowanie odcinka do dołu oraz obrót o 90° w lewo. W trzecim przykładzie mamy do narysowania 10 ząbków.

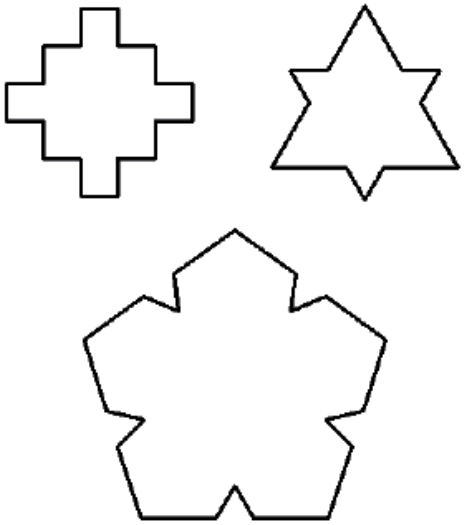
```
1.  from turtle import *
2.
3.  pensize(3)
4.
5.  def linia_przerywana():
6.      a = 30
7.      for i in range(7):
8.          fd(a/2); pu(); fd(a/2); pd()
9.
10.
11. def gory():
12.     a = 30
13.     lt(45)
14.     for i in range(10):
15.         fd(a); rt(90); fd(a); lt(90)
```

```
16.
17.
18.  def mur():
19.      a = 30
20.      lt(90)
21.      for i in range(10):
22.          fd(a); rt(90); fd(a); rt(90)
23.          fd(a); lt(90); fd(a); lt(90)
```

Żeby narysować powyższe rysunki, należy wywołać zdefiniowane funkcje: `linia_przerywana()`, `gory()`, `mur()`.

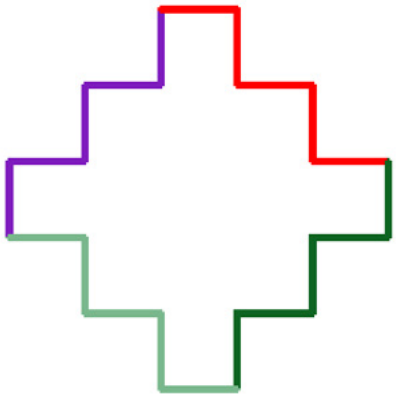
Zadanie 4: Różne wzory

Napisz funkcje: `krzyzyk()`, `gwiazdka()`, `kwiatek()`, po wywołaniu których powstaną rysunki takie jak poniżej. Długości krótszych odcinków wynoszą 30, a dłuższych 60.



Rysunek 15. Różne wzory

Rozwiązywanie zadania zaczynamy od odpowiedzi na pytanie: jaki element się powtarza. W pierwszym przykładzie jest to fragment złożony z trzech niepełnych ząbków. Ten układ powtarza się 4 razy.



W drugim przykładzie powtarza się bok trójkąta z wypustką. Zauważmy, że mamy odcinki długości 30 i 60, czyli drugi z nich jest dwa razy dłuższy niż pierwszy. Układ powtarza się 3 razy. Kąty, o jakie żółw się obraca, to 60° i 120°.

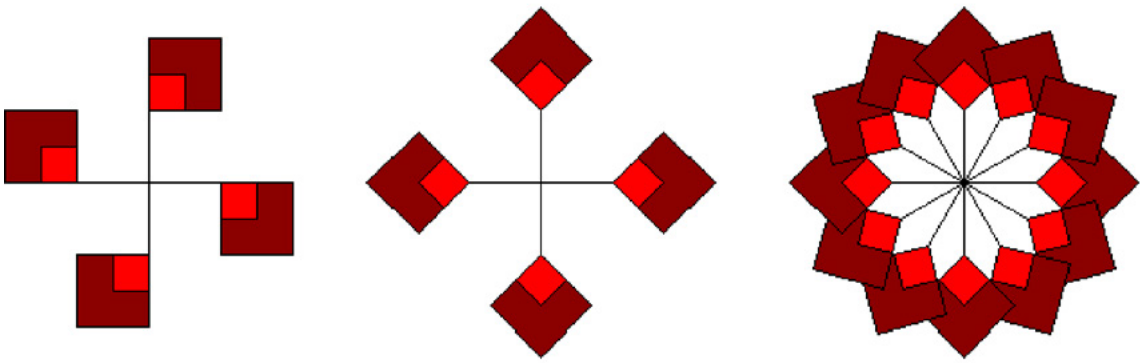
Trzeci przykład o tyle przypomina drugi, że występuje tu bok wielokąta z wypustką. Natomiast wypustka jest skierowana do wewnątrz, a powstały kształt przypomina pięciokąt.

```
1.  from turtle import *
2.
3.  pensize(3)
4.
5.  def krzyzyk():
6.      a = 30
7.      lt(90)
8.      for i in range(4):
9.          fd(a); rt(90)
10.         fd(a); lt(90)
11.         fd(a); rt(90)
12.         fd(a); lt(90)
13.         fd(a); rt(90)
14.
15.
```

```
16.  def gwiazdka():
17.      a = 30
18.      lt(120)
19.      for i in range(3):
20.          fd(2*a); rt(60)
21.          fd(a); lt(120)
22.          fd(a); rt(60)
23.          fd(2*a); lt(120)
24.
25.
26.  def kwiatek():
27.      a = 30
28.      for i in range(5):
29.          fd(2*a); lt(60)
30.          fd(a); rt(120)
31.          fd(a); lt(60)
32.          fd(2*a); lt(72)
```

Zadanie 5: Kwiat

Napisz funkcje: `kwiat1()`, `kwiat2()`, `kwiat3()`, po wywołaniu których powstaną rysunki takie jak poniżej. Długość pałąka wynosi 40, a boki kwadratów 40 i 20.



Rysunek 16. Różne kwiaty

W przykładzie pierwszym rysunek składa się z pałąka i klocka zbudowanego z dwóch kwadratów: większego – ciemnoczerwonego i mniejszego – czerwonego. Wygodnie będzie więc zdefiniować funkcję pomocniczą: `klocek()`. Dla każdego

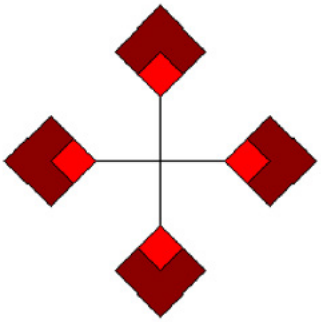
kwadratu określamy kolor wypełnienia, rozpoczynamy zamalowywanie, rysujemy kwadrat i kończymy zamalowanie. Należy zwrócić uwagę, że najpierw rysujemy duży kwadrat, a w następnej kolejności mały.

```
def klocek():
    fillcolor("darkred")
    begin_fill()
    for i in range(4):
        fd(40)
        rt(90)
    end_fill()

    fillcolor("red")
    begin_fill()
    for i in range(4):
        fd(20)
        rt(90)
    end_fill()
```

Rysując wzór, korzystamy ze zdefiniowanej wcześniej funkcji: `klocek()`.

```
def kwiat1():
    for i in range(4):
        fd(40)
        klocek()
        bk(40)
        rt(90)
```

W kolejnym kroku rysujemy drugi motyw.



Od poprzedniego wzoru różni się on położeniem klocka względem pałąka – przed narysowaniem klocka i po narysowaniu dodajemy obrót.

```
def kwiat2():
    for i in range(4):
        fd(40)
        lt(45)
        klocek()
        rt(45)
        bk(40)
        rt(90)
```

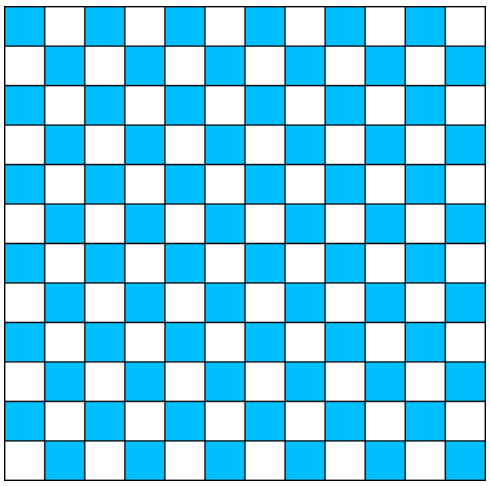
Teraz wystarczy narysować 12 elementów zamiast 4, aby otrzymać gotowe rozwiązanie. Kąt między kolejnymi elementami wynosi 360/12. Rozwiązanie zadania przedstawia się następująco:

```
from turtle import *

def klocek():
    fillcolor("darkred")
    begin_fill()
    for i in range(4):
        fd(40)
        rt(90)
    end_fill()

    fillcolor("red")
    begin_fill()
    for i in range(4):
        fd(20)
        rt(90)
    end_fill()

```

```
19.  def kwiat3():
20.      for i in range(12):
21.          fd(40)
22.          lt(45)
23.          klocek()
24.          rt(45)
25.          bk(40)
26.          rt(360/12)
```

Zadanie 6: Szachownica

Szachownica jest kwadratem, który składa się z parzystej liczby małych kwadratów. Napisz funkcję: `szachownica(n)`, po wywołaniu której zostanie narysowany motyw taki, jak poniżej. Parametr n jest liczbą naturalną parzystą i może przyjmować wartości od 2 do 20. Bok małego kwadratu wynosi 30.



Rysunek 17. Szachownica

Analizę zadania rozpoczynamy od wyodrębnienia elementu powtarzającego się. Jest nim kwadrat, wypełniony raz niebieskim, raz białym kolorem.

Funkcja `kwad()` – rysująca kwadrat – ma dwa parametry: długość boku kwadratu i kolor jego zamalowania. Ponadto zdefiniujemy funkcję pomocniczą: `skok(a, b)`, po wywołaniu której żółw przesunie się bez rysowania o a do przodu i b w prawo. Ułatwi to znacznie przemieszczanie żółwia i skróci zapis.

Rozpoczynamy od przemieszczenia żółwia w lewy dolny róg rysunku – tak, aby wzór powstał dokładnie na środku. Główna część rozwiązania opiera się na działaniu zagnieżdżonej pętli – n razy powtarzamy rysowanie rzędu. Kolory kwadratu układają się naprzemiennie: raz niebieski, raz biały – w zależności od parzystości sumy numerów wiersza i kolumny.

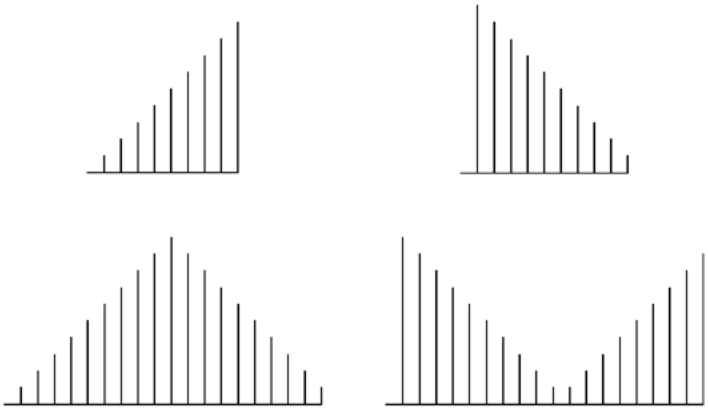
```
1.  from turtle import *
2.
3.
4.  def kwad(bok, kolor):
5.      fillcolor(kolor)
6.      begin_fill()
7.      for i in range(4):
8.          fd(bok)
9.          rt(90)
10.     end_fill()
11.
12.
13. def skok(a, b):
14.     pu()
15.     fd(a)
16.     rt(90)
17.     fd(b)
18.     lt(90)
19.     pd()
20.
21.
```

```
22.  def szachownica(n):
23.
24.      bok = 30
25.      skok(-n/2*bok, -n/2*bok)
26.
27.      for i in range(n):
28.          for j in range(n):
29.              if (i + j)%2 == 0:
30.                  kwad(bok, "DeepSkyBlue")
31.              else:
32.                  kwad(bok, "White")
33.              skok(bok, 0);
34.          skok(-bok*n, bok)
```

Warto zwrócić uwagę, że przedstawione powyżej rozwiązanie jest jednym z wielu możliwych. Można bowiem rysować np. tylko niebieskie kwadraty oraz zewnętrzną obwódkę.

Zadanie 7: Różne ząbki

Napisz funkcje: `zabki1()`, `zabki2()`, `zabki3()`, `zabki4()`, po wywołaniu których powstaną rysunki takie jak poniżej. Długości sąsiednich odcinków różnią się o 10.



Rysunek 18. Różne ząbki

Pierwszy rysunek składa się z odcinków coraz to dłuższych. Pierwszy pionowy odcinek od lewej ma długość 10, drugi 20, trzeci 30 itd. Ogólnie można to wyrazić wzorem: `10*i`, gdzie `i` przybiera wartości od 1 do 9.

Drugi rysunek ma pionowe odcinki coraz to krótsze, czyli najpierw 100, później 90, 80 itd. aż do 10. Podobnie jak poprzednio można to wyrazić wzorem: `10*i`, przy czym `i` maleje – przybiera wartości od 10 do 1. Alternatywnym rozwiązaniem jest rysowanie od prawej do lewej, które jednak utrudniłoby wykonywanie kolejnych zadań.

Przykład trzeci to złożenie rysunku pierwszego i drugiego, a czwarty – drugiego i pierwszego.

```
1.  from turtle import *
2.
3.
4.  def zabki1():
5.      a = 10
6.      for i in range(1, 10):
7.          fd(a)
8.          lt(90)
9.          fd(a*i)
10.         bk(a*i)
11.         rt(90)
12.
13.
14. def zabki2():
15.     a = 10
16.     for i in range(10, 0, -1):
17.         fd(a)
18.         lt(90)
19.         fd(a*i)
20.         bk(a*i)
21.         rt(90)
22.
23.
```

```
24.  def zabki3():
25.      zabki1()
26.      zabki2()
27.
28.
29.  def zabki4():
30.      zabki2()
31.      zabki1()
```

Zadanie 8: Sztanga

Napisz funkcję o nazwie: `sztanga(n)`, która będzie rysowała taką sztangę, jak na rysunku poniżej. Parametr n określa liczbę ciężarków po jednej stronie. Może on przyjmować wartości z zakresu od 1 do 12. Grubość ciężarków wynosi 10, odstępy między ciężarkami również wynoszą 10. Wysokość pierwszego – największego ciężarka zawsze wynosi 250, a każdy następny jest mniejszy o 20. Szerokość sztangi jest stała i wynosi 600.

Rysunek 19. Sztanga

Sztanga jest symetryczna. Jeśli narysujemy odważniki po jednej stronie, możemy skorzystać z rozwiązania i dorysować drugą stronę. W związku z tym definiujemy pomocniczą funkcję: `odwazniki(n)` rysującą odważniki po jednej stronie sztangi. Warto także zdefiniować drugą pomocniczą funkcję, która rysuje prostokąt. Zarówno poprzeczka, jak i odważniki są prostokątami. Rysowanie prostokąta można zaczynać od środka jednego z boków. Ułatwi to rysowanie odważników – między jednym a drugim odważnikiem przesuwamy się o 20.

W funkcji głównej zaczynamy rysowanie od poprzeczki. Później przesuwamy żółwia do lewej części rysunku i rysujemy odważniki. Następnie przemieszczamy

żółwia do prawej części, obracamy o 180° i ponownie wywołujemy funkcję rysującą odważniki.

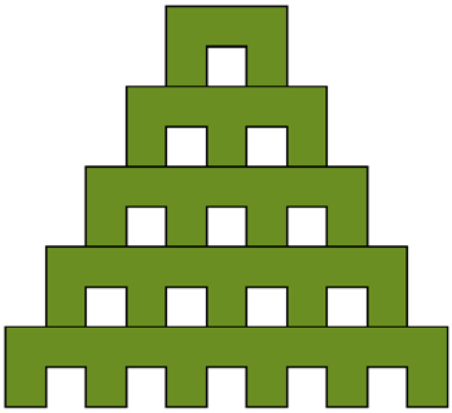
```
from turtle import *

def prost(a, b):
    # prostokąt od środka jednego boku
    lt(90)
    begin_fill()
    for i in range(2):
        fd(a/2);rt(90)
        fd(b);rt(90)
        fd(a/2)
    end_fill()
    rt(90)

def odwazniki(n):
    for j in range(13-n, 13):
        fd(10)
        prost(20*j, 10)
        fd(10)
    #powrot
    fd(300-20*n)

def sztanga(n):
    color("blue")

    #poprzeczka
    lt(90);bk(5)
    prost(600, 10)
    lt(90)


```

```
    #odważniki po lewej stronie
    fd(300);rt(180)
    lt(90);fd(5);rt(90)
    odwazniki(n)


    #odważniki po prawej stronie
    fd(300);rt(180)
    odwazniki(n)
```

Jeśli rysowanie trwa długo, można je przyspieszyć, wywołując: `tracer(0)`; `sztanga(6)`; `update()`.

Zadanie 9: Pałac

Napisz funkcję: `palac(liczba_pieter)`, która tworzy rysunek pałacu o danej liczbie pięter z zakresu od 1 do 10. Długości krótkich odcinków wynoszą 20.



Rysunek 20. Pałac

Pałac składa się z pięter. Tworzymy go od góry do dołu, rysując pojedynczy poziom i przemieszczając żółwia do miejsca, w którym powinien zaczynać się kolejny. Poziomy oznaczamy kolejno cyframi od 1 do liczby pięter.

Zauważmy, że numer poziomu oznacza jednocześnie liczbę białych elementów występujących w jego obrębie. Do rysowania poziomu będziemy potrzebować funkcji pomocniczej: `poziom(n, bok)`, gdzie n – to numer poziomu, a bok – długość krótkiego odcinka. Każdy poziom składa się z n ząbków i poprzeczek

pionowych oraz długiej poprzeczki poziomej. Długości pionowych poprzeczek wynoszą 2*bok, a poziomej 2*n*bok+bok.

```
from turtle import *

def poziom(n, bok):
    begin_fill()
    for i in range(n):
        fd(bok); lt(90)
        fd(bok); rt(90)
        fd(bok); rt(90)
        fd(bok); lt(90)
    fd(bok); lt(90)
    fd(2*bok); lt(90)
    fd(2*n*bok+bok); lt(90)
    fd(2*bok); lt(90)
    end_fill()


def palac(n):
    bok = 20
    fillcolor("olivedrab")
    for i in range(1, n+1):
        poziom(i, bok)
        pu()
        lt(180); fd(bok)
        lt(90); fd(2*bok)
        lt(90)
        pd()
```

Przykładowe wywołanie: `tracer(0); palac(5); update()`.

KATARZYNA OLĘDZKA

RAMOWY PROGRAM SZKOLENIA DLA NAUCZYCIELI KLAS 4–6 (II ETAP EDUKACYJNY) – WERSJA SKRÓCONA

INFORMACJE OGÓLNE

Szkolenie jest przeznaczone dla uczestników szkolenia dotyczącego nauczania w klasach 7–8, którzy nie ukończyli szkolenia odnoszącego się do nauczania w klasach 4–6, ale deklarują znajomość zagadnień zawartych w jego programie. Obejmuje ono 10 godzin lekcyjnych zajęć stacjonarnych. Nauczyciele kontynuują doskonalenie podczas szkolenia przygotowującego do realizacji podstawy programowej w klasach 7–8.

Szkolenie prezentuje wybrane zagadnienia z programu szkolenia dla nauczycieli klas 4–6, wprowadzające w tematykę poruszaną na szkoleniu dotyczącym najstarszych klas szkoły podstawowej. Obejmuje ono przede wszystkim wprowadzenie do programowania w języku tekstowym oraz powtórzenie i utrwalenie umiejętności budowania w języku wizualnym prostych projektów dotyczących przetwarzania liczb naturalnych.

WYMAGANIA WSTĘPNE STAWIANE UCZESTNIKOM SZKOLENIA

Uczestnik szkolenia powinien posiadać:

- kompetencje wymienione w *Załączniku 1*;
- uprawnienia do nauczania przedmiotu informatyka w szkole podstawowej;
- umiejętności z zakresu rozwiązywania problemów algorytmicznych z wykorzystaniem komputera, w szczególności programowania w języku wizualnym z wykorzystaniem zdarzeń, instrukcji warunkowych i iteracyjnych oraz korzystania ze zmiennych;

- doświadczenie w prowadzeniu zajęć z dziećmi na poziomie klas 4–6 w zakresie rozwiązywania problemów z wykorzystaniem wizualnego środowiska programowania.

CELE SZKOLENIA:

- przygotowanie nauczycieli do prowadzenia zajęć z informatyki z zakresu algorytmicznego rozwiązywania problemów i programowania zgodnie z nową postawą programową;
- doskonalenie własne nauczycieli w zakresie stosowania algorytmiki i programowania, a także rozumienia pojęć informatycznych i metod informatyki;
- utrwalenie umiejętności programowania w języku wizualnym;
- wprowadzenie podstaw programowania w języku tekstowym.

TREŚCI NAUCZANIA:

1. rola algorytmicznego rozwiązywania problemów, myślenia komputacyjnego i programowania w nowej podstawie programowej ze szczególnym uwzględnieniem zapisów dla klas 4–6;
2. sterowanie obiektem za pomocą sekwencji poleceń;
3. poszukiwanie – w zbiorach nieuporządkowanych i uporządkowanych – konkretnego elementu oraz elementów najmniejszego i największego;
4. kształtowanie zdolności algorytmicznego rozwiązywania problemów; wyróżnianie podstawowych kroków w formułowaniu i algorytmicznym rozwiązywaniu problemu oraz ich stosowanie w praktyce;
5. wprowadzenie do tekstowego języka programowania wysokiego poziomu;
6. testowanie, poprawianie i prezentowanie własnych programów.

PRZYKŁADOWY ROZKŁAD MATERIAŁU

Temat i tematy cząstkowe	Punkt podstawy programowej	Treści	Liczba godzin
1. Wprowadzenie			1
• Organizacja szkolenia • Podstawa programowa informatyki dla drugiego etapu edukacyjnego	całość	1	0,5 0,5

2. Programowanie w środowisku Scratch			**2**
• Implementujemy algorytmy obliczeniowe	I.2, I.3, II.1, II.2	1, 2, 3, 4, 6	2
3. Sterowanie obiektem za pomocą sekwencji poleceń			**7**
• Grafika żółwia w Pythonie – wprowadzenie • Uczymy żółwia nowych słów • Powtarzamy czynności • Powtarzamy i podejmujemy decyzje • Dzielimy problem na problemy cząstkowe • Projektujemy posadzki • Budujemy piramidy • Rozwiązujemy zadania	I.2c, I.3, II.1, II.2	1, 2, 3, 5, 6	0,5 0,5 1 1 1 1 1 1

OMÓWIENIE POSZCZEGÓLNYCH TEMATÓW

1. Wprowadzenie

Rozpoczynając szkolenie, należy omówić zasady jego organizacji oraz specyfikę mającą na celu przygotowanie do właściwego szkolenia. Można przeprowadzić krótką dyskusję, dlaczego warto nauczać programowania. Ponadto trzeba omówić poszczególne zapisy dotyczące informatyki zawarte w nowej podstawie programowej, zarówno w odniesieniu do założeń ogólnych programu, jak i szczegółowych treści. Szczególną uwagę należy poświęcić zapisom dotyczącym rozwiązywania problemów i nauce programowania w klasach 4–6 oraz refleksji dotyczącej tego, jakie umiejętności wynoszą uczniowie z edukacji informatycznej w nauczaniu wczesnoszkolnym. Trzeba także przedstawić pełny program szkolenia dla klas 4–6 oraz treści, które będą omawiane na tym szkoleniu.

2. Programowanie w środowisku Scratch

Zakładamy, że uczestnicy szkolenia posiadają już umiejętności i doświadczenie z zakresu programowania wizualnego. Prawdopodobnie programowali w środowisku Scratch. Jeśli korzystali z innego środowiska programowania wizualnego, należy zwięźle omówić możliwości Scratcha.

2.1. Implementujemy algorytmy obliczeniowe

✓ **Gra logiczna – zgadywanie liczby („Mistrzowie Kodowania”)**

Tworzymy grę polegającą na odgadywaniu liczby wylosowanej przez komputer. Po każdej próbie gracz jest informowany, czy podana przez niego liczba jest mniejsza, większa, czy równa wartości pamiętanej przez komputer. Zabawa ma na celu odkrycie przez gracza algorytmu postępowania prowadzącego do jak najszybszego odgadnięcia wylosowanej liczby. Warto także przygotować odwrotną realizację algorytmu – gracz wymyśla liczbę, a komputer zgaduje.

Następnie należy rozwiązać kilka prostych problemów obliczeniowych i zaimplementować je w Scratchu. Celem zadań może być „przeliterowanie” liczby, czyli wypisanie cyfra po cyfrze od końca (przykładowe zadanie 1) oraz znalezienie największej liczby spośród podanych (przykładowe zadanie 2).

Zasoby do wykorzystania:

- Strona OEIiZK – Moduł VI „Zgadywanie liczby”, (sekcja „Mistrzowie Kodowania”).

3. Sterowanie obiektem za pomocą sekwencji poleceń

Na tym etapie szkolenia należy zrealizować prawie cały punkt 4 programu pełnego szkolenia (z wyjątkiem ostatnich aktywności: „Sterujemy robotem”). Podstawowe zadania prawdopodobnie zajmą mniej czasu, dzięki nabytym już wcześniej doświadczeniem słuchaczy.

3.1. Grafika żółwia w Pythonie – wprowadzenie

Programowanie w języku tekstowym warto zacząć od wykorzystania grafiki żółwia – idei wypracowanej przez Seymoura Paperta, stanowiącej podstawę języka Logo. Motywy graficzne można również opracowywać z wykorzystaniem tego podejścia w języku Python z biblioteką Turtle. Grafika żółwia kojarzy się głównie z językiem Logo, ale warto zwrócić uwagę, że geometria żółwia była stosowana, przed powstaniem języka Logo. Występuje w wielu narzędziach i środowiskach przeznaczonych do nauki programowania dla dzieci, np. w Scratchu (projekty z wykorzystaniem pióra) czy „Godzinie Kodowania” (chociaż niekoniecznie używa się w nich nazwy „grafika żółwia”). Została także wykorzystana na tym szkoleniu.

Jakie cechy tego podejścia sprawiają, że można z jego pomocą uczyć programowania nawet małe dzieci? Należy wskazać na trzy istotne aspekty:

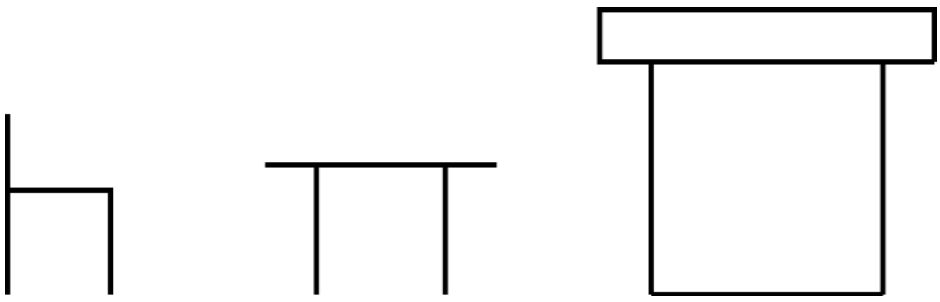
- proste i konkretne, a przez to zrozumiałe polecenia, zbliżone do języka naturalnego – „idź naprzód o… kroków", „obróć się o kąt…", „podnieś pisak" itp.;
- postać żółwia – graficzny symbol, który pozwala dziecku wyobrazić sobie wykonawcę algorytmu zapisanego w języku programowania;
- semantyka operacyjna – uczeń obserwuje działanie stworzonych przez siebie programów. Interpretacja kodu powoduje na ekranie efekt, który dość łatwo ocenić pod względem zgodności ze wzorcem. Widząc, jak żółw rysuje, uczeń sprawdza, czy wykonuje zadanie prawidłowo, może zatem znaleźć ewentualny błąd.

Podstawowe komendy:

Polecenie	Wyjaśnienie
`fd(n)`	*forward* – przesunięcie żółwia w aktualnym kierunku o n kroków
`bk(n)`	*backward* – przesunięcie żółwia przeciwnie do aktualnego kierunku o n kroków
`rt(alfa)`	*right* – obrót żółwia w prawo o kąt *alfa*
`lt(alfa)`	*left* – obrót żółwia w lewo o kąt *alfa*
`pu()`	*pen up* – żółw podnosi pisak, czyli nie rysuje podczas przemieszczania się
`pd()`	*pen down* – żółw opuszcza pisak, czyli rysuje podczas przemieszczania się

Importowanie biblioteki Turtle: from turtle import *

Zaczynamy pracę w języku Python od prostych obliczeń w trybie interaktywnym, następnie wprowadzamy proste zadania graficzne – np.: rysowanie stołu, krzesła, mostu, domu, itp., tworząc ciąg poleceń w pliku.



Rysunek 1. Przykładowe rysunki

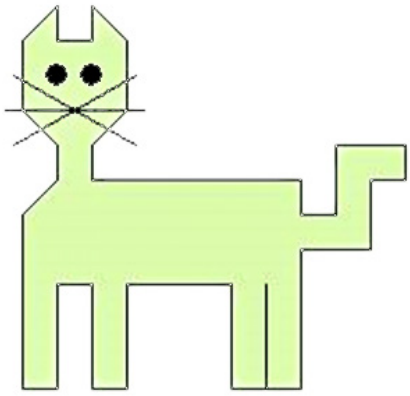
Zasoby do wykorzystania:

- Jochemczyk W., Olędzka K., (2013), *Python dla wszystkich*, Toruń: Informatyka w Edukacji, (zakładka „O Pythonie”);
- Film *Pythonowe początki*;
- Materiały na stronie: http://python.oeiizk.edu.pl, (zakładka: „Rysowanie z żółwiem”).

3.2. Uczymy żółwia nowych słów

Kod złożony z fragmentów stanowiących logiczną całość warto podzielić na pojedyncze funkcje. Dzięki temu możemy wielokrotnie wykorzystać dany kod, a cały program staje się bardziej zwięzły i przejrzysty.

Uczestnicy szkolenia powinni spróbować narysować dobrze znane przedmioty. Najlepiej zacząć od rysunków, które zawierają tylko kąty proste, gdyż ich wykonanie jest łatwiejsze dla uczniów, a następnie przejść do wykorzystania także innych kątów. Pewną trudność mogą też stanowić rysunki, w których występują nietypowe długości odcinków – np. przekątna kwadratu. Uczestnicy powinni wykonać również swoje własne rysunki. Na początkowym etapie uczniowie chętniej rysują motywy konkretne niż abstrakcyjne wzory. Poniżej przykładowy rysunek (uwaga: nieco pracochłonny).



Rysunek 2. Efekt wywołania funkcji `kot()`

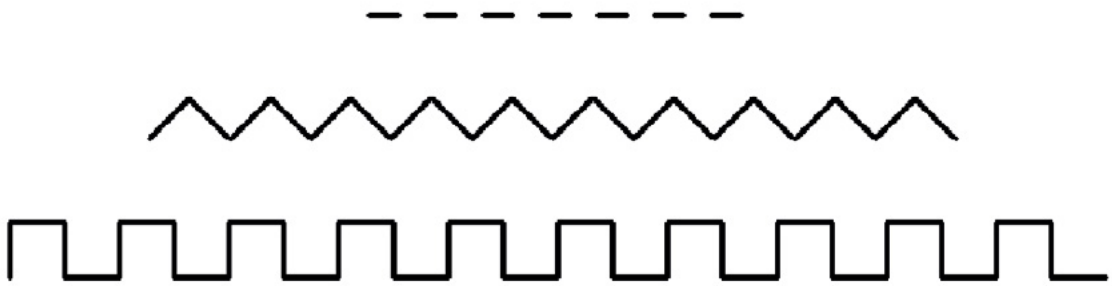
3.3. Powtarzamy czynności

Najlepiej zacząć od narysowania kwadratu, najpierw ręcznie – zapisując wszystkie instrukcje (i od razu je grupując), a później z wykorzystaniem instrukcji `for`. Następnie można przejść do tworzenia innych wielokątów foremnych: trójkąta,

pięciokąta, sześciokąta itd. Chociaż pierwsze rysunki na pewno będą powstawać metodą prób i błędów, należy omówić, jak wyliczyć kąty.

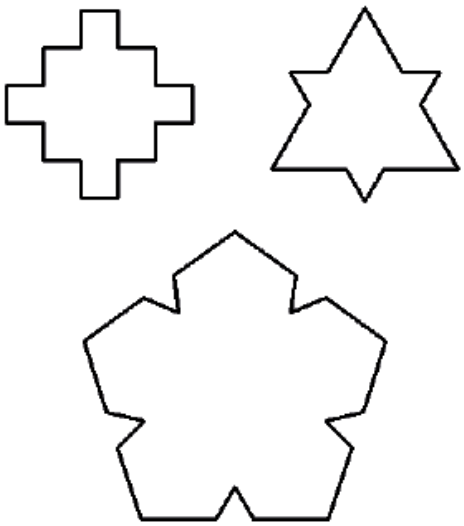
Instrukcja iteracji jest bardzo ważna w nauce programowania i nie można tego tematu zlekceważyć. Trzeba poświęcić odpowiednią ilość czasu na ćwiczenia. Zaczynamy od prostych przykładów, na których wyraźnie widać, jakie elementy się powtarzają. Następnie można przejść do trochę trudniejszych, by dobrze wyćwiczyć umiejętność stosowania instrukcji iteracji. Można też nawiązać do programowania w Scratchu i bloczka „powtórz".

Pierwszą grupę przykładowych zadań stanowią różnego rodzaju szlaczki.



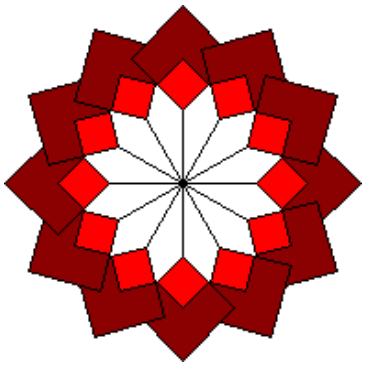
Rysunek 3. Szlaczki (przykładowe zadanie 3)

Drugą grupę czynności tworzą różnego rodzaju motywy, w których rysowanie zaczyna i kończy się w tym samym miejscu.



Rysunek 4. Różne wzory (przykładowe zadanie 4)

Dla urozmaicenia warto stworzone motywy wzbogacić o kolory.



Rysunek 5. Efekt wywołania funkcji `kwiat()` (przykładowe zadanie 5)

3.4. Powtarzamy i podejmujemy decyzje

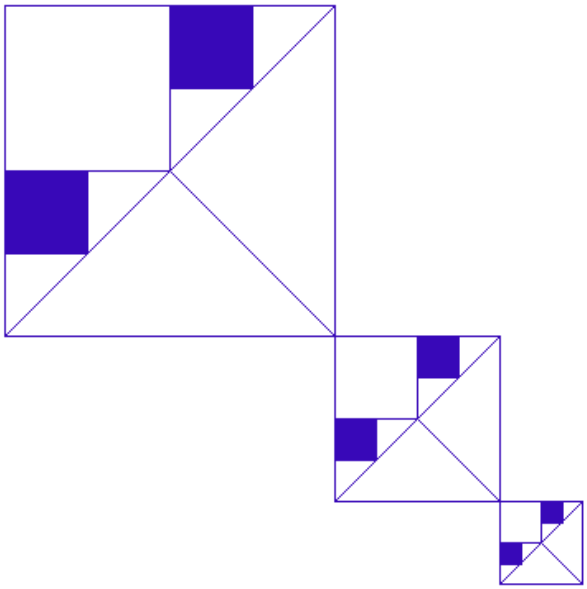
W dalszej części szkolenia można wprowadzić modyfikacje do wzorów powtarzalnych. W zależności od tego, po raz który jest wykonywana pętla, element powtarzalny może być zupełnie inny lub różnić się jedynie parametrem.

✓ **Przykładowy scenariusz – *Rysowanie grzebieni***

3.5. Dzielimy problem na problemy cząstkowe

Jeśli w danym motywie występują figury podobne do siebie – różnej wielkości, koloru lub o równej liczbie elementów powtarzających się – można napisać funkcję z parametrem. Przypuśćmy, że chcemy narysować mały kwadrat, średni kwadrat oraz duży kwadrat, należy wówczas zdefiniować jedną funkcję z parametrem: `kwadrat(bok)`. Funkcję z parametrami definiuje się analogicznie jak bezparametrową. Jedyna różnica polega na wymienieniu w nagłówku funkcji wszystkich jej parametrów. Parametry oddzielamy przecinkami.

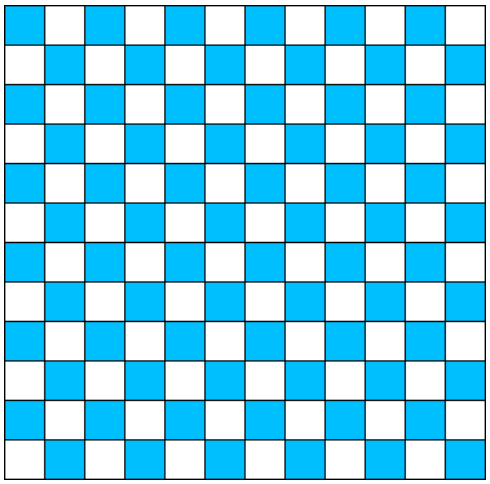
Przykład zadania:



Rysunek 6. Przykład motywu, w którym należy zdefiniować funkcję z parametrem

3.6. Projektujemy posadzki

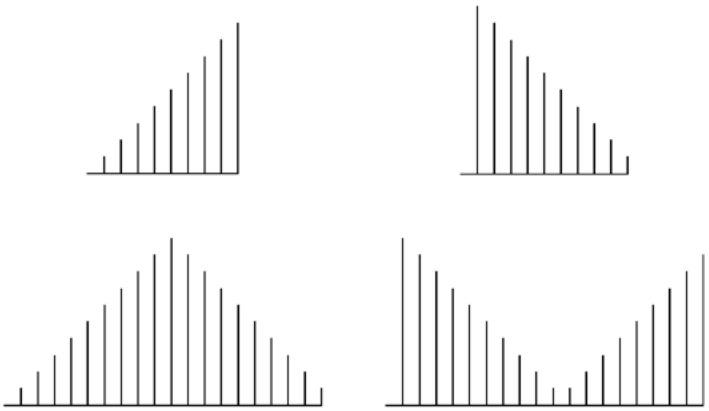
Po rozwiązaniu prostszych zadań z wykorzystaniem iteracji, należy przejść do bardziej skomplikowanych. Są to zadania typu „posadzka", w których jeden element powtarzany jest wiele razy. Wykonując je, mamy do czynienia z zagnieżdżonymi pętlami. Rozwiązanie można zapisać na wiele sposobów.



Rysunek 7. Posadzka (przykładowe zadanie 6)

3.7. Budujemy piramidy

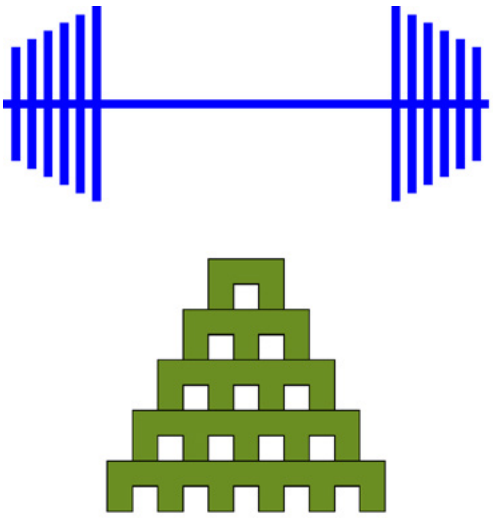
Ostatnią – najtrudniejszą – grupę zadań stanowią te oparte na wykorzystaniu wartości zmiennej sterującej pętli `for`. Należy zacząć od prostych realizacji polegających na rysowaniu odcinków, prostokątów lub innych figur – od najmniejszej do największej lub na odwrót.



Rysunek 8. Różne „ząbki” (przykładowe zadanie 7)

Podobnie jak poprzednio – dla uatrakcyjnienia rysowanych motywów warto dodać kolory.

Trudniejsze przykłady wymagają napisania kilku funkcji. Kilka takich zadań powinno się zrealizować na zajęciach z nauczycielami, by potem mogli w analogiczny sposób pracować ze zdolną młodzieżą.

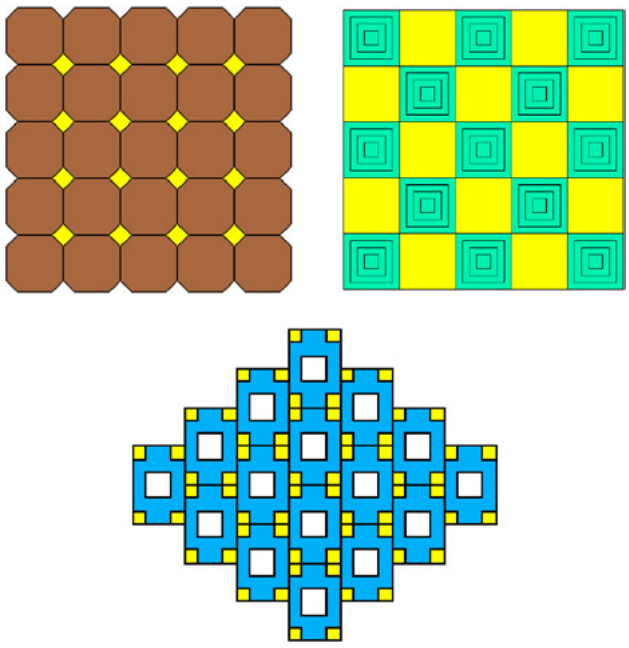


Rysunek 9. Przykłady rysunków z wykorzystaniem zmiennej sterującej pętli (przykładowe zadania 8 i 9)

3.8. Rozwiązujemy zadania

Uczniowie, rozwiązując proste problemy, poznają polecenia języka programowania i utrwalają ich znajomość. Następnie powinni przystąpić do rozwiązywania bardziej złożonych zadań. Niekoniecznie muszą one dotyczyć trudniejszych zagadnień – na to przyjdzie czas w klasach starszych. Uczniowie dowiadują się, jak dzielić problem na problemy cząstkowe i stosować nowe umiejętności w sytuacjach typowych oraz mniej typowych. Możemy też omówić technikę projektowania algorytmów (*top-down*) i ich implementacji (*bottom-up*). Trzeba pamiętać, że są to pierwsze kroki uczniów w obszarze programowania, dlatego nie powinni czuć się przytłoczeni. Inaczej jest, gdy prowadzimy zajęcia dla nauczycieli – trzeba zadbać, by mieli szersze spojrzenie.

Po wykonaniu kilku zadań omawiamy etapy rozwiązywania problemów informatycznych. Jest to dopiero początek na drodze do poznawania algorytmiki, ale warto już na tym etapie kształtować prawidłowe myślenie. Omawiamy kolejno, na czym polegają opis i analiza sytuacji problemowej, sporządzanie specyfikacji problemu, projektowanie rozwiązania, komputerowa realizacja rozwiązania oraz jego testowanie i prezentacja. Szczególnie należy podkreślić, że przy wykonywaniu konkretnego zadania najpierw trzeba przemyśleć sposób jego rozwiązania, a dopiero później zacząć pisać kod. Wiele spośród zadań można bowiem rozwiązać za pomocą różnych metod.



Rysunek 10. Przykłady trudniejszych zadań

W nauce programowania bardzo istotne jest testowanie.
Przede wszystkim staramy się na bieżąco reagować na błędy – najczęściej wychwytujemy błędy syntaktyczne. Po zakończeniu pisania kodu, powinniśmy również skrupulatnie przetestować nasze rozwiązanie, gdyż błędy logiczne są dużo trudniejsze do wykrycia. Każde zadanie sprawdzamy, przyjmując wartości brzegowe parametrów i kilku pośrednich. Jeśli możliwych wartości parametrów jest niewiele, zaleca się przeprowadzenie testowania dla wszystkich.

Skąd czerpać pomysły na zadania? Po pierwsze z życia codziennego. Uczniowie chętniej zajmują się problemami, które uważają za bliskie. Można też skorzystać z różnych serwisów, są one jednak adresowane głównie do uczniów starszych. Niektóre z nich mogą być wykorzystywane w nauce na poziomie szkoły podstawowej. Na stronach Przedmiotowego Konkursu Informatycznego „miniLOGIA" dla uczniów szkół podstawowych województwa mazowieckiego zamieszczono wiele zadań graficznych (http://minilogia.oeiizk.waw.pl) – znajdują się wśród nich zarówno prostsze, jak i trudniejsze, można więc dobrać zadania do poziomu uczniów. Zadania o wyższym stopniu trudności można też znaleźć na stronie konkursu „LOGIA" dla uczniów klas gimnazjalnych (http://logia.oeiizk.waw.pl).

Zasoby do wykorzystania:

- Strona konkursu „miniLOGIA": http://minilogia.oeiizk.waw.pl;
- Strona konkursu „LOGIA": http://logia.oeiizk.waw.pl;
- Strona konkursu „Bóbr": http://www.bobr.edu.pl;
- Strona projektu „Godzina Kodowania": https://code.org.

PRZYKŁADOWY SCENARIUSZ ZAJĘĆ

Scenariusz 1 – *Rysujemy grzebienie*

Opis zajęć

Głównym celem zajęć jest nauka powtarzania czynności oraz podejmowania decyzji. Uczniowie nabywają tych umiejętności i utrwalają je podczas realizacji zadania: Grzebienie – które zapisują i uruchamiają w języku Python.

Czas trwania

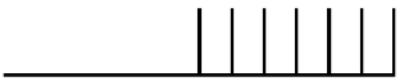
45 minut

Realizacja

Zaczynamy od narysowania grzebienia mającego ząbki jednakowej wysokości. Następnie wprowadzamy modyfikacje – co drugi lub co trzeci ząbek musi być inny. Na kolejnym etapie zajęć uczestnicy przygotowują dla siebie nawzajem grzebienie – zagadki. W ten sposób każdy ma możliwość, by zaprojektować swój własny wzór, ale także aby opisać wzór wymyślony przez innego uczestnika.

✓ Zadanie: Grzebień prosty

Zdefiniuj funkcję: `grzebien(ile)`, po wywołaniu której powstanie rysunek grzebienia składającego się z trzonka oraz identycznych ząbków. Liczbę ząbków określa parametr `ile`, który może przyjmować wartości od 2 do 20. Długość trzonka wynosi 60, odległości między ząbkami 10, a wysokość ząbków 20.



`grzebien(7)`

`grzebien(10)`

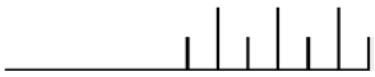
Zauważmy, że wszystkie odległości są wielokrotnościami 10, dlatego warto zdefiniować zmienną pomocniczą a, która pozwoli określić proporcje: trzonek będzie miał długość 6*a, odstęp między ząbkami a, natomiast wysokość ząbków 2*a.

Najpierw żółw przesuwa się o długość trzonka pomniejszoną o a, gdyż ten odcinek rysujemy razem z pierwszym ząbkiem. Później kolejno przesuwamy żółwia i rysujemy ząbki. Pojedynczy krok składa się z przesunięcia o a, obrotu w lewo o 90°, przesunięcia wprzód o 2*a, wycofania się i obrotu w prawo o 90°.

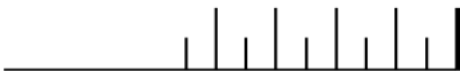
```
1.  from turtle import *
2.
3.  def grzebien(ile):
4.      a = 10
5.      fd(5*a)
6.      for i in range(ile):
7.          fd(a)
8.          lt(90)
9.          fd(2*a)
10.         bk(2*a)
11.         rt(90)
```

✓ Zadanie: Co drugi ząbek krótszy

Zdefiniuj funkcję: `co_drugi(ile)`, po wywołaniu której powstanie rysunek grzebienia składającego się z trzonka oraz ząbków. Parametr `ile` określa liczbę ząbków. Ząbki mają na przemian wysokość 10 i 20. Parametr `ile` może przyjmować wartości od 2 do 20. Długość trzonka wynosi 60, a odległości między ząbkami 10.



`co_drugi(7)`



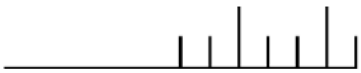
`co_drugi(10)`

Rozwiązując to zadanie, modyfikujemy funkcję napisaną poprzednio. Jedyną zmianą, jaką wprowadzamy, jest rysowanie raz wyższego, raz niższego ząbka. Uzależniamy to od wartości zmiennej sterującej pętlą `i`. Kolejne wartości zmiennej `i` to: 0, 1, 2, 3, 4 i tak dalej, aż do `ile-1` – wobec czego możemy rozpatrzyć resztę z dzielenia przez 2. Dla liczby parzystej otrzymamy resztę 0, a dla nieparzystej 1. Pozwoli to na zapisanie warunku, który będzie określał wysokość ząbka.

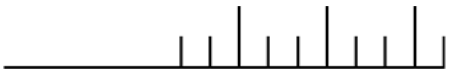
```
def co_drugi(ile):
    a = 10
    fd(5*a)
    for i in range(ile):
        fd(a)
        lt(90)
        if i % 2 == 1:
            fd(2*a)
            bk(2*a)
        else:
            fd(a)
            bk(a)
        rt(90)
```

✓ Zadanie: Co trzeci ząbek wyższy

W kolejnym kroku utrudniamy zadanie – teraz co trzeci ząbek musi być wyższy.



co_trzeci(7)



co_trzeci(10)

```
def co_trzeci(ile):
    a = 10
    fd(5*a)
    for i in range(ile):
        fd(a)
        lt(90)
        if i % 3 == 2:
            fd(2*a)
            bk(2*a)
        else:
            fd(a)
            bk(a)
        rt(90)
```

✓ Zadanie: Projektujemy własne wzory

Kolejnym zadaniem, jakie stawiamy przed uczestnikami, jest zaprojektowanie własnego wzoru. Korzystając z wcześniej napisanych programów i modyfikując je, projektujemy inne wzory. Następnie uczestnicy wymieniają się pomysłami i próbują wzajemnie odtworzyć zaprojektowane przez siebie wzory. Pozwala to z jednej strony na kreatywność, a z drugiej na rozwijanie różnych umiejętności – analizy rysunku, modyfikowania kodu programu, testowania.

Podsumowanie

W czasie zajęć koncentrujemy się na opisaniu w języku programowania zależności, uwidocznionych na rysunku. Analizujemy motyw, by sformułować zależność, a następnie zapisać ją w języku Python. Ćwiczymy rozwiązanie problemu z zastosowaniem instrukcji iteracji i instrukcji warunkowej. Uczestnicy poznali te konstrukcje, wykonując zadania wymagające użycia Scratcha i arkusza kalkulacyjnego.

Podczas programowania warto zwrócić uwagę na błędy popełniane przy pisaniu programów. Trzeba nie tylko pomóc je odszukać i poprawić, ale także pokazać, jak je wyszukiwać. Często początkujący programiści mają z tym problem.

Naturalne rozszerzanie projektu opiera się na dodawaniu wzorów kolorowych. Można zatem zaproponować inny regularnie powtarzający się motyw i rozszerzyć go.

PRZYKŁADOWE ZADANIA

Zadanie 1: „Przeliteruj” liczbę

Zadanie polega na wypisaniu cyfr liczby, poczynając od najmniej znaczącej, czyli od końca. Np. dla liczby 2018 należy kolejno wypisać: 8, 1, 0 i 2.

Rozwiązując zadanie, wykorzystujemy podstawowe operacje na liczbach całkowitych:
2018/10 = 201 reszty 8
201/10 = 20 reszty 1
20/10 = 2 reszty 0
2/10 = 0 reszty 2

Zapisujemy algorytm w postaci listy kroków:

1. Wczytaj liczbę.
2. Dopóki `liczba` > 0, wykonuj:
    - Oblicz resztę z dzielenia liczby przez 10.
    - Wypisz resztę.
    - Odejmij od liczby resztę z dzielenia.
    - Podziel liczbę przez 10.

Na podstawie powyższego algorytmu możemy ułożyć skrypt w Scratchu. Należy zwrócić uwagę, że w języku Scratch nie występuje operacja dzielenia całkowitego, dlatego od liczby odejmujemy resztę z dzielenia.

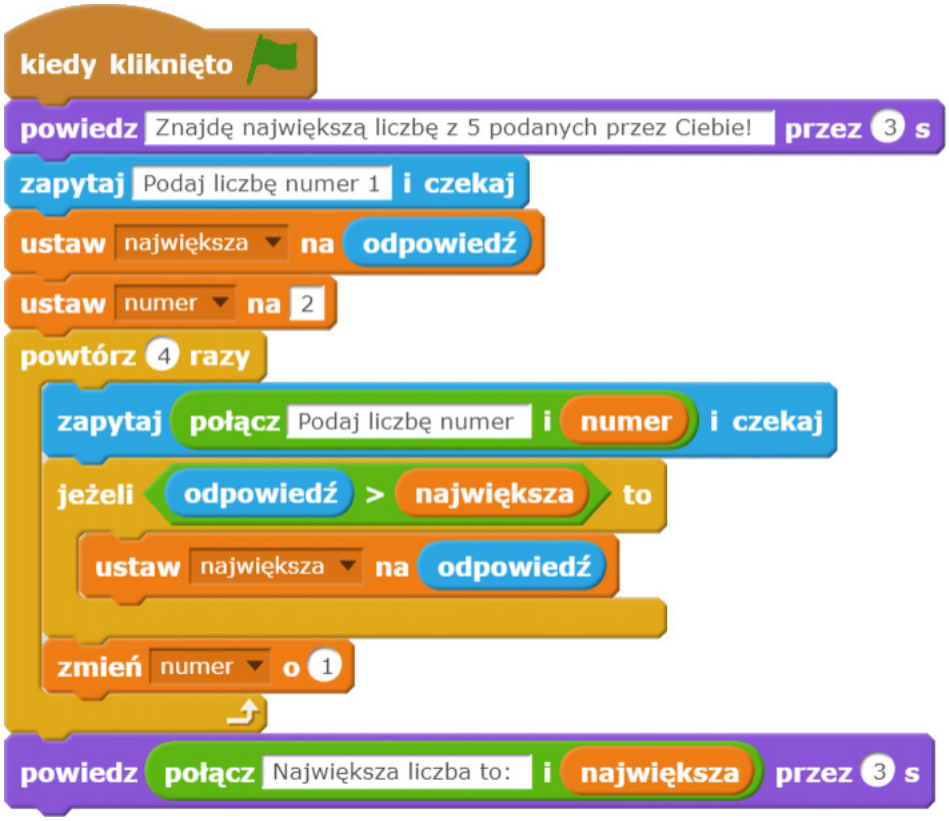
Rysunek 11. Rozwiązanie zadania: „Przeliteruj” liczbę

Zadanie 2: Największa liczba

Zadanie polega na znalezieniu największej liczby spośród kilku podanych. Podobnie jak w poprzednim zadaniu, najpierw zapisujemy algorytm w postaci listy kroków. Przyjmując, że zostanie podanych pięć liczb, tworzymy następujący algorytm:

1. Wczytaj liczbę.
2. Zapamiętaj w zmiennej `największa` wczytaną liczbę.
3. Powtórz 4 razy:
    - Wczytaj liczbę.
    - Jeśli jest ona większa od `największa`, to:
        - Zapamiętaj w zmiennej `największa` wczytaną liczbę.
4. Wypisz wartość zmiennej `największa`.

Na podstawie powyższego algorytmu możemy ułożyć skrypt w Scratchu.

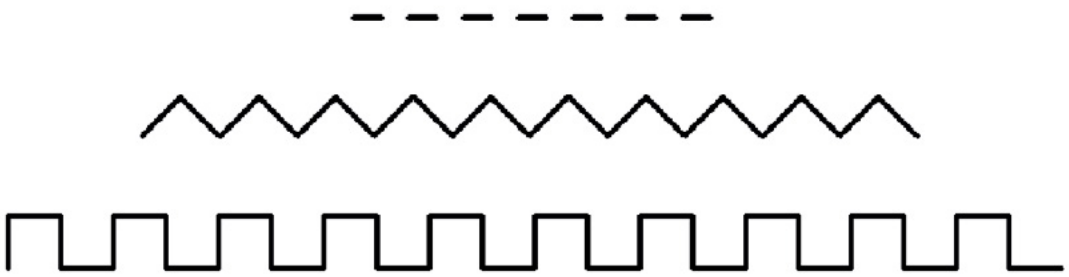


Rysunek 12. Rozwiązanie zadania: Największa liczba

Zadanie daje wiele możliwości wprowadzenia modyfikacji, np. wczytania, z ilu liczb poszukujemy największej, czy poszukiwania liczby najmniejszej.

Zadanie 3: Szlaczki

Napisz funkcje: `linia_przerywana()`, `gory()`, `mur()`, po wywołaniu których powstaną rysunki takie jak poniżej. Długości odcinków wynoszą 30.



Rysunek 13. Szlaczki

Analizując powyższe rysunki, trzeba odpowiedzieć na pytanie: jakie elementy się powtarzają i ile razy. W pierwszym przykładzie są to odcinek i przerwa. Ten układ powtarza się 7 razy. W drugim przykładzie jest to jedna góra, która powtarza się 10 razy – żółw stoi odchylony od poziomu pod kątem 45° w górę i powtarza 10 razy rysowanie odcinka do góry, obrót o 90° w prawo, rysowanie odcinka do dołu oraz obrót o 90° w lewo. W trzecim przykładzie mamy do narysowania 10 ząbków.

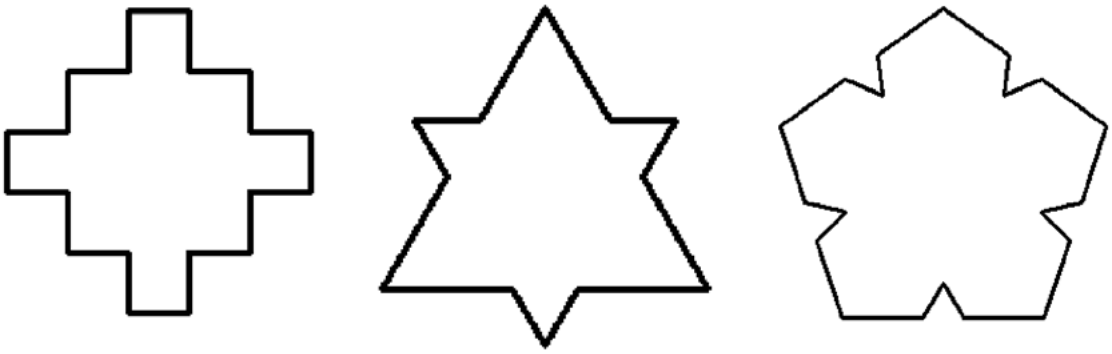
```
1.  from turtle import *
2.
3.  pensize(3)
4.
5.  def linia_przerywana():
6.      a = 30
7.      for i in range(7):
8.          fd(a/2); pu(); fd(a/2); pd()
9.
10.
11. def gory():
12.     a = 30
13.     lt(45)
14.     for i in range(10):
15.         fd(a); rt(90); fd(a); lt(90)
```

```
16.
17.
18.  def mur():
19.      a = 30
20.      lt(90)
21.      for i in range(10):
22.          fd(a); rt(90); fd(a); rt(90)
23.          fd(a); lt(90); fd(a); lt(90)
```

Żeby narysować powyższe rysunki, należy wywołać zdefiniowane funkcje: `linia_przerywana()`, `gory()`, `mur()`.

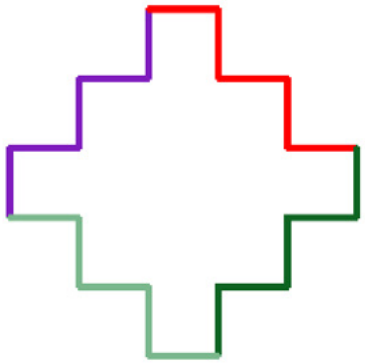
Zadanie 4: Różne wzory

Napisz funkcje: `krzyzyk()`, `gwiazdka()`, `kwiatek()`, po wywołaniu których powstaną rysunki takie jak poniżej. Długość krótszych odcinków wynosi 30, a dłuższych 60.



Rysunek 14. Różne wzory

Rozwiązywanie zadania zaczynamy od odpowiedzi na pytanie: jaki element się powtarza. W pierwszym przykładzie jest to fragment złożony z trzech niepełnych ząbków. Ten układ powtarza się 4 razy.



W drugim przykładzie powtarza się bok trójkąta z wypustką. Zauważmy, że mamy odcinki o długości 30 i 60, czyli drugi z nich jest dwa razy dłuższy niż pierwszy. Układ powtarza się 3 razy. Kąty, o jakie żółw się obraca, to 60° i 120°.

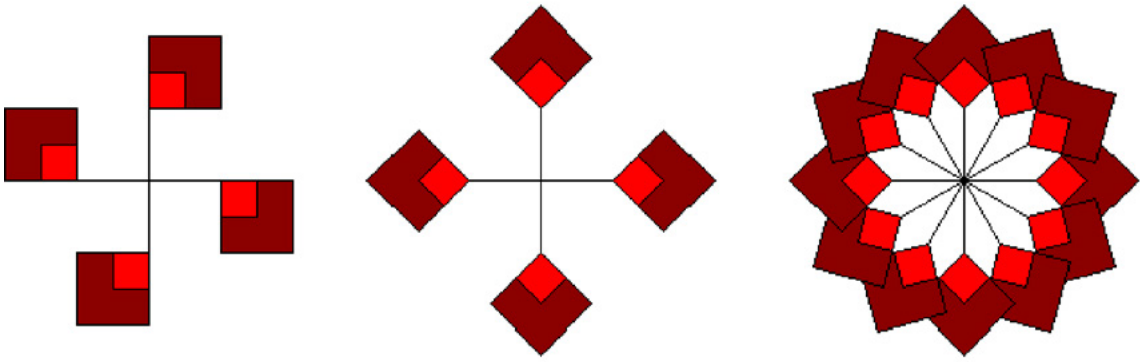
Trzeci przykład o tyle przypomina drugi, że występuje tu bok wielokąta z wypustką. Natomiast wypustka jest skierowana do wewnątrz, a powstały kształt przypomina pięciokąt.

```
1.  from turtle import *
2.
3.  pensize(3)
4.
5.  def krzyzyk():
6.      a = 30
7.      lt(90)
8.      for i in range(4):
9.          fd(a); rt(90)
10.         fd(a); lt(90)
11.         fd(a); rt(90)
12.         fd(a); lt(90)
13.         fd(a); rt(90)
14.
15.
```

```
16.  def gwiazdka():
17.      a = 30
18.      lt(120)
19.      for i in range(3):
20.          fd(2*a); rt(60)
21.          fd(a); lt(120)
22.          fd(a); rt(60)
23.          fd(2*a); lt(120)
24.
25.
26.  def kwiatek():
27.      a = 30
28.      for i in range(5):
29.          fd(2*a); lt(60)
30.          fd(a); rt(120)
31.          fd(a); lt(60)
32.          fd(2*a); lt(72)
```

Zadanie 5: Kwiat

Napisz funkcje: `kwiat1()`, `kwiat2()`, `kwiat3()`, po wywołaniu których powstają rysunki takie jak poniżej. Długość pałąka wynosi 40, a boki kwadratów 40 i 20.



Rysunek 15. Różne kwiaty

W przykładzie pierwszym rysunek składa się z pałąka i klocka zbudowanego z dwóch kwadratów: większego (ciemnoczerwonego) i mniejszego (czerwonego). Wygodnie będzie więc zdefiniować funkcję pomocniczą: `klocek()`. Dla każdego

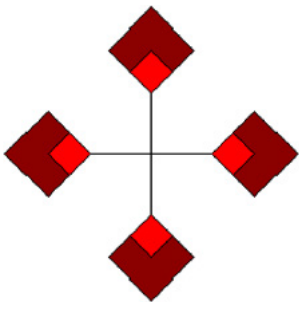
kwadratu ustawiamy kolor wypełnienia, rozpoczynamy zamalowywanie, rysujemy kwadrat i kończymy zamalowanie. Należy zwrócić uwagę, że najpierw rysujemy duży kwadrat, a w następnej kolejności mały.

```
def klocek():
    fillcolor("darkred")
    begin_fill()
    for i in range(4):
        fd(40)
        rt(90)
    end_fill()

    fillcolor("red")
    begin_fill()
    for i in range(4):
        fd(20)
        rt(90)
    end_fill()
```

Rysując wzór, korzystamy ze zdefiniowanej wcześniej funkcji: `klocek()`.

```
def kwiat1():
    for i in range(4):
        fd(40)
        klocek()
        bk(40)
        rt(90)
```

W kolejnym kroku rysujemy drugi motyw.



Od poprzedniego wzoru różni się on położeniem klocka względem pałąka – przed narysowaniem klocka i po narysowaniu dodajemy obrót.

```
def kwiat2():
    for i in range(4):
        fd(40)
        lt(45)
        klocek()
        rt(45)
        bk(40)
        rt(90)
```

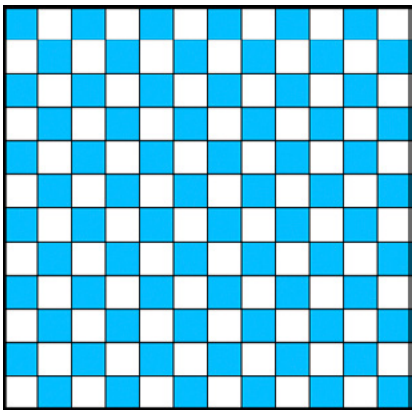
Teraz wystarczy narysować 12 elementów zamiast 4, aby otrzymać gotowe rozwiązanie. Kąt między elementami wynosi 360/12. Rozwiązanie zadania przedstawia się następująco:

```
from turtle import *

def klocek():
    fillcolor("darkred")
    begin_fill()
    for i in range(4):
        fd(40)
        rt(90)
     end_fill()

    fillcolor("red")
    begin_fill()
    for i in range(4):
        fd(20)
        rt(90)
    end_fill()

```

```
19.  def kwiat3():
20.      for i in range(12):
21.          fd(40)
22.          lt(45)
23.          klocek()
24.          rt(45)
25.          bk(40)
26.          rt(360/12)
```

Zadanie 6: Szachownica

Szachownica jest kwadratem, który składa się z parzystej liczby małych kwadratów. Napisz funkcję: `szachownica(n)`, po wywołaniu której zostanie narysowany motyw taki, jak poniżej. Parametr n jest liczbą naturalną parzystą i może przyjmować wartości od 2 do 20. Bok małego kwadratu wynosi 30.



Rysunek 16. Szachownica

Analizę zadania rozpoczynamy od wyodrębnienia elementu powtarzającego się. Jest nim kwadrat, wypełniony raz niebieskim, raz białym kolorem. Funkcja: `kwad()` – rysująca kwadrat – ma dwa parametry: długość boku kwadratu i kolor jego zamalowania. Ponadto zdefiniujemy funkcję pomocniczą: `skok(a, b)`, po wywołaniu której żółw przesunie się bez rysowania o a do przodu i b w prawo. Ułatwi to znacznie przemieszczanie żółwia i skróci zapis.

Rozpoczynamy od przemieszczenia żółwia w lewy dolny róg rysunku – tak, aby wzór powstał dokładnie na środku. Główna część rozwiązania zadania opiera się na działaniu zagnieżdżonej pętli – n razy powtarzamy rysowanie rzędu.

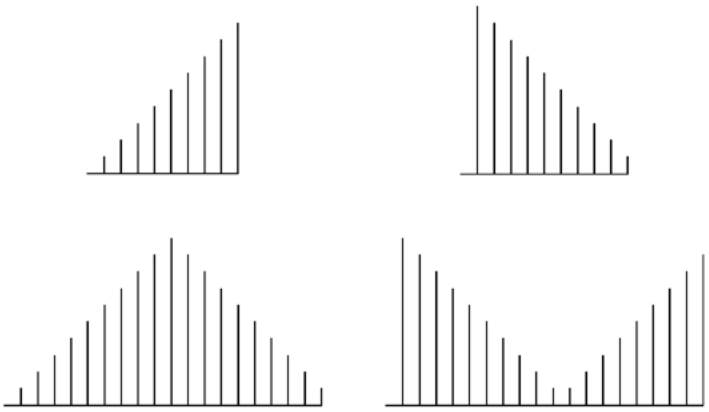
Kolory kwadratu układają się naprzemiennie: raz niebieski, raz biały – w zależności od parzystości sumy numerów wiersza i kolumny.

```
from turtle import *


def kwad(bok, kolor):
    fillcolor(kolor)
    begin_fill()
    for i in range(4):
        fd(bok)
        rt(90)
    end_fill()


def skok(a, b):
    pu()
    fd(a)
    rt(90)
    fd(b)
    lt(90)
    pd()


def szachownica(n):

    bok = 30
    skok(-n/2*bok, -n/2*bok)

    for i in range(n):
        for j in range(n):
            if (i + j)%2 == 0:
                kwad(bok, "DeepSkyBlue")
            else:
                kwad(bok, "White")
            skok(bok, 0);
        skok(-bok*n, bok)
```

Warto zwrócić uwagę, że przedstawione powyżej rozwiązanie jest jednym z wielu możliwych. Można bowiem rysować np. tylko niebieskie kwadraty oraz zewnętrzną obwódkę.

Zadanie 7: Różne ząbki

Napisz funkcje: `zabki1()`, `zabki2()`, `zabki3()`, `zabki4()`, po wywołaniu których powstaną rysunki takie jak poniżej. Długości sąsiednich odcinków różnią się o 10.



Rysunek 17. Różne ząbki

Pierwszy rysunek składa się z odcinków coraz to dłuższych. Pierwszy pionowy odcinek od lewej ma długość 10, drugi 20, trzeci 30, itd. Ogólnie można to wyrazić wzorem: `10*i`, gdzie `i` przybiera wartości od 1 do 9.

Drugi rysunek ma pionowe odcinki coraz to krótsze, czyli najpierw 100, następnie 90, 80 itd. aż do 10. Podobnie jak poprzednio można to wyrazić wzorem: `10*i`, przy czym `i` teraz maleje – przybiera wartości od 10 do 1. Alternatywnym rozwiązaniem jest rysowanie od prawej do lewej, które jednak utrudniłoby wykonywanie kolejnych zadań. Przykład trzeci opiera się na złożeniu rysunku pierwszego i drugiego, a czwarty – drugiego i pierwszego.

```
from turtle import *


def zabki1():
    a = 10
    for i in range(1, 10):
        fd(a)
        lt(90)
        fd(a*i)
        bk(a*i)
        rt(90)


def zabki2():
    a = 10
    for i in range(10, 0, -1):
        fd(a)
        lt(90)
        fd(a*i)
        bk(a*i)
        rt(90)


def zabki3():
    zabki1()
    zabki2()


def zabki4():
    zabki2()
    zabki1()
```

Zadanie 8: Sztanga

Napisz funkcję o nazwie: `sztanga(n)`, która będzie rysowała taką sztangę, jak na rysunku poniżej. Parametr n określa liczbę ciężarków po jednej stronie. Może on przyjmować wartości z zakresu od 1 do 12. Grubość ciężarków wynosi 10, odstępy między ciężarkami również wynoszą 10. Wysokość pierwszego – największego ciężarka zawsze wynosi 250, a każdy następny jest mniejszy o 20. Sztanga ma stałą szerokość wynoszącą 600.

Rysunek 18. Sztanga

Sztanga jest symetryczna. Jeśli narysujemy odważniki po jednej stronie, możemy skorzystać z rozwiązania i dorysować drugą stronę. W związku z tym definiujemy pomocniczą funkcję: `odwazniki(n)` rysującą odważniki po jednej stronie sztangi. Warto także zdefiniować drugą pomocniczą funkcję, która rysuje prostokąt. Zarówno poprzeczka, jak i odważniki są prostokątami. Rysowanie prostokąta można zaczynać od środka jednego z boków. Ułatwi to rysowanie odważników – między jednym, a drugim odważnikiem będziemy się przesuwać o 20.

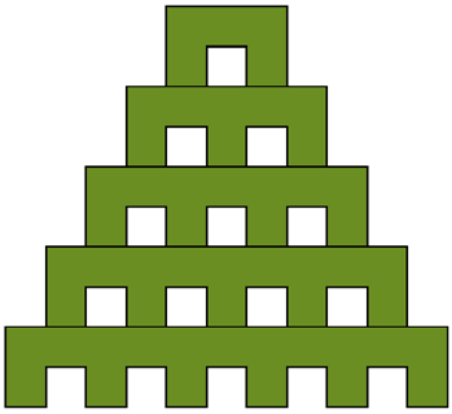
W funkcji głównej zaczynamy rysowanie od poprzeczki. Później przesuwamy żółwia do lewej części rysunku i rysujemy odważniki. Następnie przemieszczamy żółwia do prawej części rysunku, obracamy o 180° i ponownie wywołujemy funkcję rysującą odważniki.

```
from turtle import *

def prost(a, b):
    # prostokąt od środka jednego boku
    lt(90)
    begin_fill()
    for i in range(2):
        fd(a/2);rt(90)
        fd(b);rt(90)
        fd(a/2)
    end_fill()
    rt(90)

def odwazniki(n):
    for j in range(13-n, 13):
        fd(10)
        prost(20*j, 10)
        fd(10)
    #powrot
    fd(300-20*n)

def sztanga(n):
    color("blue")

    #poprzeczka
    lt(90);bk(5)
    prost(600, 10)
    lt(90)


    #odważniki po lewej stronie
    fd(300);rt(180)
    lt(90);fd(5);rt(90)
    odwazniki(n)


    #odważniki po prawej stronie
    fd(300);rt(180)
    odwazniki(n)
```

Jeśli rysowanie trwa długo, można je przyspieszyć, wywołując: `tracer(0); sztanga(6); update()`.

Zadanie 9: Pałac

Napisz funkcję: `palac(liczba_pieter)`, która tworzy rysunek pałacu o danej liczbie pięter z zakresu od 1 do 10. Długości krótkich odcinków wynoszą 20.



Rysunek 19. Pałac

Pałac składa się z pięter. Tworzymy go od góry do dołu, rysując pojedynczy poziom i przemieszczając żółwia do miejsca, gdzie powinien zaczynać się kolejny. Poziomy oznaczamy kolejno cyframi od 1 do liczby pięter.

Zauważmy, że numer poziomu oznacza jednocześnie liczbę białych elementów występujących w jego obrębie. Do rysowania poziomu będziemy potrzebować funkcji pomocniczej `poziom(n, bok)`, gdzie `n` – to numer poziomu, a `bok` – długość krótkiego odcinka. Każdy poziom składa się z `n` ząbków i poprzeczek pionowych oraz długiej poprzeczki poziomej. Długości pionowych poprzeczek wynoszą `2*bok`, a poziomej `2*n*bok+bok`.

```
from turtle import *

def poziom(n, bok):
    begin_fill()
    for i in range(n):
        fd(bok); lt(90)
        fd(bok); rt(90)
        fd(bok); rt(90)
        fd(bok); lt(90)
    fd(bok); lt(90)
    fd(2*bok); lt(90)
    fd(2*n*bok+bok); lt(90)
    fd(2*bok); lt(90)
    end_fill()


def palac(n):
    bok = 20
    fillcolor("olivedrab")
    for i in range(1, n+1):
        poziom(i, bok)
        pu()
        lt(180); fd(bok)
        lt(90); fd(2*bok)
        lt(90)
        pd()
```

Przykładowe wywołanie: `tracer(0); palac(5); update()`.

KRZYSZTOF CHECHŁACZ

RAMOWY PROGRAM SZKOLENIA DLA NAUCZYCIELI KLAS 7–8 (II ETAP EDUKACYJNY)

INFORMACJE OGÓLNE

Szkolenie jest przeznaczone dla nauczycieli informatyki uczących lub planujących nauczać w szkołach podstawowych. Jego główny cel stanowi przygotowanie nauczycieli do realizacji nowej podstawy programowej przedmiotu informatyka w szkole podstawowej w zakresie algorytmicznego rozwiązywania problemów oraz programowania na poziomie klas 7 i 8. Szkolenie obejmuje 40 godzin lekcyjnych zajęć stacjonarnych.

Zajęcia powinny mieć charakter przede wszystkim warsztatowy – uczestnicy pod nadzorem prowadzącego samodzielnie rozwiązują problemy, w szczególności wcielają się w rolę ucznia. Część zajęć należy przeznaczyć na wykład i dyskusję oraz omówienie zagadnień metodycznych. Praca praktyczna pozwoli słuchaczom nabrać biegłości w posługiwaniu się narzędziami i metodami informatycznymi. Nie mniej ważna jest również refleksja pedagogiczna – w jakim celu wprowadzamy dane zagadnienia, jak zorganizować proces dydaktyczny, na co szczególnie zwrócić uwagę i jakie mogą wystąpić trudności. Podczas zajęć nie tylko prowadzący dzielą się swoją wiedzą, ale także słuchacze wymieniają się doświadczeniami.

WYMAGANIA WSTĘPNE STAWIANE UCZESTNIKOM SZKOLENIA

Uczestnik szkolenia powinien posiadać kompetencje wymienione w *Załączniku 1*, a ponadto:

- posiadać uprawnienia do nauczania przedmiotu informatyka w szkole podstawowej;
- posiadać wiedzę i umiejętności realizowane na szkoleniu dla nauczycieli informatyki szkół podstawowych w zakresie nauki programowania

w klasach 4–6 szkoły podstawowej – ukończone 40-godzinne szkolenie lub ukończone 10-godzinne szkolenie wstępne (jeśli uczestnik deklaruje znajomość zagadnień z zakresu programu szkolenia dla klas 4–6).

CELE SZKOLENIA:

- przygotowanie nauczycieli do prowadzenia zajęć z informatyki zgodnie z nową postawą programową w zakresie algorytmicznego rozwiązywania problemów i programowania;
- doskonalenie własne nauczycieli w zakresie algorytmiki i programowania, a także rozumienia pojęć informatycznych i metod informatyki;
- rozwijanie u uczniów umiejętności myślenia komputacyjnego i rozwiązywania problemów z życia codziennego przy pomocy narzędzi informatycznych;
- nabycie umiejętności programowania w języku tekstowym wysokiego poziomu w zakresie umożliwiającym realizację podstawy programowej przedmiotu informatyka w klasach 7–8 szkoły podstawowej.

TREŚCI NAUCZANIA:

1. Rola algorytmicznego rozwiązywania problemów, myślenia komputacyjnego i programowania w nowej podstawie programowej ze szczególnym uwzględnieniem zapisów dotyczących klas 7–8. Wprowadzenie do myślenia abstrakcyjnego.
2. Kształtowanie umiejętności algorytmicznego rozwiązywania problemów. Wyróżnianie podstawowych kroków w formułowaniu i algorytmicznym rozwiązywaniu problemu oraz ich stosowanie w praktyce (od sformułowania i specyfikacji problemu po zaprogramowanie i testowanie rozwiązania).
3. Różne sposoby przedstawiania algorytmów: język naturalny, schemat blokowy, lista kroków, pseudokod, język programowania.
4. Rozwiązywanie problemów z wykorzystaniem algorytmów dotyczących liczb naturalnych, wyszukiwania elementu w zbiorze nieuporządkowanym i uporządkowanym, porządkowanie elementów.
5. Omówienie sposobów komputerowej reprezentacji wartości logicznych, liczb naturalnych, znaków i tekstów.
6. Wykorzystanie wizualnych i tekstowych języków programowania w edukacji informatycznej. Przejście od języka wizualnego do tekstowego.
7. Programowanie wybranych algorytmów w języku tekstowym z wykorzystaniem instrukcji wejścia/wyjścia, zmiennych prostych

i tablic, operacji arytmetycznych i logicznych, instrukcji warunkowych i iteracyjnych, funkcji bezparametrowych i z parametrami.

8. Projektowanie, tworzenie, testowanie i poprawianie własnych programów, w szczególności służących do sterowania obiektem.
9. Wykorzystanie dostępnych pomocy dydaktycznych i oprogramowania (np. arkusza kalkulacyjnego) do demonstracji i modyfikowania działania algorytmów.
10. Stosowanie metod wywodzących się z informatyki do rozwiązywania problemów z innych dziedzin.

PRZYKŁADOWY ROZKŁAD MATERIAŁU:

Temat i tematy cząstkowe	Punkt podstawy programowej	Treści	Liczba godzin
1. Wprowadzenie			**1**
• Organizacja szkolenia • Podstawa programowa informatyki dla drugiego etapu edukacyjnego: klasy 7–8	całość	1	0,5 0,5
2. Od problemu do algorytmu			**4**
• Etapy rozwiązywania problemu • Przykłady problemów algorytmicznych z życia codziennego • Algorytmy dotyczące liczb naturalnych • Różne sposoby zapisu algorytmów	I.1, I.2a, I.5, II.4	1, 2, 3, 8	1 1 1 1
3. Od programowania wizualnego do tekstowego			**4**
• Przykład implementacji algorytmu dotyczącego liczb naturalnych w języku wizualnym • Przykład algorytmu zapisanego w języku tekstowym wysokiego poziomu (np. przy pomocy automatycznej translacji z języka wizualnego) • Dyskusja na temat możliwości języków wizualnych • Wprowadzenie do abstrakcji, parametry, pojęcie funkcji	I.1, I.2a, I.4, II.1, II.4	1, 2, 6, 7, 8, 9	1 2 0,5 0,5

4. Programowanie w Pythonie			**9**
• Instrukcje wejścia/wyjścia • Tworzenie własnych funkcji obliczeniowych z parametrami • Wykorzystanie instrukcji warunkowych i iteracyjnych w rozwiązywanych zadaniach • Rozwiązywanie zadań dotyczących operacji na liczbach naturalnych z wykorzystaniem algorytmów z badaniem podzielności, m.in. wyodrębnianie cyfr liczby i algorytm Euklidesa • Przykłady różnych algorytmów rozwiązania tego samego problemu	I.1, I.2a, II.1, II.4	3, 4, 6, 7, 8	1 1 2 3 2
5. Wyszukiwanie i porządkowanie informacji, przetwarzanie danych			**14**
• Informatyka bez komputera • Reprezentacja liczb, znaków i napisów, tablice (listy) • Rozwiązywanie zadań związanych z wyszukiwaniem elementu w zbiorach nieuporządkowanych i uporządkowanych • Rozwiązywanie zadań związanych z porządkowaniem zbiorów • Przykłady zadań związanych z przetwarzaniem danych, które nie są liczbami	I.1, I.2b, I.3, I.4, II.1, II.4	1, 4, 5, 6, 7, 8, 9	2 2 3 4 3
6. Rozwiązywanie problemów metodami wywodzącymi się z informatyki			**5**
• Realizacja algorytmów w arkuszu kalkulacyjnym • Przykłady problemów z innych dziedzin • Wykorzystanie języków programowania do sterowania (np. robotem lub obiektem na ekranie)	I.1, I.2b, I.5, II.1, II.2, II.3c, II.4	1, 2, 3, 6, 8, 9, 10	2 1 2
7. Podsumowanie			**3**
• Zadania nauczyciela: gdzie szukać pomocy i inspiracji; czego uczniowie będą uczyć się w szkole ponadpodstawowej? • Rozwijanie zainteresowań, praca z uczniem zdolnym, konkursy informatyczne	całość	1–10	1 2

OMÓWIENIE POSZCZEGÓLNYCH TEMATÓW

1. Wprowadzenie

Wstępna część szkolenia służy omówieniu zasad jego organizacji.
Należy przedstawić cele szkolenia, nawiązać do szkolenia dotyczącego zakresu klas 4–6 w wersji 40-godzinnej lub 10-godzinnej (uczestnicy szkolenia powinni wcześniej ukończyć jedno z tych szkoleń). Jeśli uczestnicy nie kontynuują szkolenia, to powinni opowiedzieć o sobie i swoich doświadczeniach (charakter pracy, wiek uczniów, warunki techniczne w ich placówce do prowadzenia zajęć informatycznych). Pozwoli to osobie prowadzącej szkolenie lepiej dostosować się do potrzeb grupy, natomiast uczestnikom wzajemnie się poznać.

Należy omówić treści nauczania z podstawy programowej przedmiotu informatyka sformułowane dla całego II etapu edukacyjnego, zwracając uwagę szczególnie na te elementy (cele, treści), których realizacja rozpoczyna się w klasach 4–6 i jest kontynuowana w klasach 7–8, zgodnie ze spiralnym (przyrostowym) modelem nauczania. Zaakcentować należy, że w klasach 7–8 istnieje konieczność zapisywania określonych algorytmów w języku programowania. Uczestnicy szkolenia powinni mieć dostęp do treści załączników do *Rozporządzenia Ministra Edukacji Narodowej z dnia 14 lutego 2017 r. w sprawie podstawy programowej* (Dz.U. z 2017 r., poz. 356), które będą omawiane.

2. Od problemu do algorytmu

2.1. Etapy rozwiązywania problemu

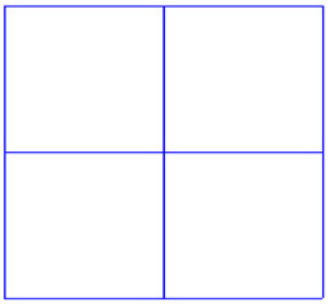
W klasach 7–8 uczniowie powinni zostać wdrożeni do rozwiązywania problemów z wykorzystaniem komputera, a podczas rozwiązywania problemu świadomie wyróżniać poszczególne etapy:

a) określenie zadania – problemu,
b) określenie danych do zadania,
c) określenie spodziewanego celu, tj. wyników rozwiązania zadania,
d) znalezienie i sformułowanie rozwiązania – algorytmu,
e) zaprogramowanie rozwiązania,
f) przetestowanie i ewentualna korekta rozwiązania.

Należy omówić te kroki na przykładzie problemu, który jest prosty i znany uczniom. W tej części szkolenia warto odwołać się do treści realizowanych

podczas szkolenia dotyczącego nauki programowania w klasach 4–6, np. do przykładów sterowania obiektem na ekranie.

Przykład:
Zadanie polega na narysowaniu poniższej figury.

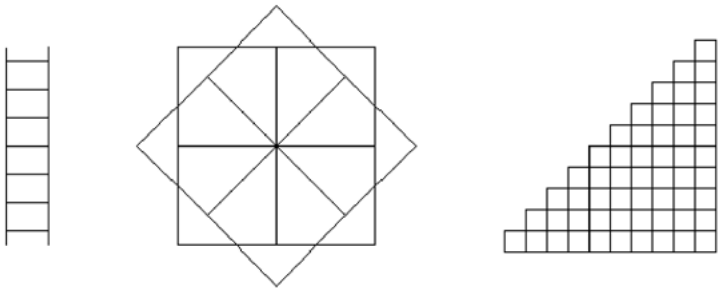


Rysunek 1. Przykładowy problem – figura do narysowania

Należy określić dane do zadania. W tym przypadku jest to długość boku kwadratu – ale którego? Mamy do wyboru długość boku dużego (zewnętrznego) lub małego kwadratu. Określamy jako daną np. długość boku zewnętrznego kwadratu. Wynikiem zadania jest utworzony rysunek.

Poszukujemy metody rozwiązania. Istnieją różne algorytmy rozwiązania (np. rysowanie czterech kwadratów lub tylko jednego – zewnętrznego – i krzyża w środku). Po wybraniu algorytmu, przystępujemy do zaprogramowania rozwiązania. Słuchacze mogą wybrać, czy implementujemy rozwiązanie zadania w środowisku Scratch, czy w języku Python (wiedza wyniesiona ze szkolenia w zakresie nauczania klas 4–6 umożliwia realizację w obu językach). Następnie testujemy napisany program i usuwamy ewentualne błędy.

Przykłady innych rysunków, dla których można podać różne algorytmy rysowania:



Rysunek 2. Przykłady rysunków

2.2. Przykłady problemów algorytmicznych z życia codziennego

Rozwiązywane problemy powinny odnosić się do różnych dziedzin życia, nie tylko do informatyki. Wskazane jest, by ukazywały zastosowanie programowania w tych obszarach. Na przykład: można poddać analizie problem zwykłego wchodzenia po schodach. Określamy operacje elementarne (podnieś nogę, postaw na kolejnym stopniu itp.), wskazujemy na konieczność sprawdzania, w jakiej sytuacji jesteśmy oraz podejmowania decyzji i powtarzania (czy jest kolejny stopień, kiedy kończymy, kiedy powtarzamy czynności wcześniej wykonane itp.).

Warto także odwołać się do problemu z życia codziennego, który wymaga wykorzystania umiejętności matematycznych. Na przykład: możemy zapytać o to, na ile sposobów można dokonać podziału całej klasy na równoliczne zespoły. Faktycznie zadanie sprowadza się do tego, aby policzyć, ile dzielników ma liczba określająca liczność klasy (liczbę uczniów). Możemy przy tym przeprowadzić dyskusję prowadzącą do określenia, jaki podział jest dopuszczalny – np. czy zezwalamy na tworzenie zespołów 1-osobowych, bądź pozostawienie jednego zespołu obejmującego wszystkich uczniów klasy. W ten sposób uczymy precyzyjnego określenia danych wejściowych i spodziewanych wyników.

Na stronie internetowej Konkursu Informatycznego „Bóbr” (http://bobr.edu.pl) znajdziemy też wiele ciekawych zadań dotyczących problemów algorytmicznych z życia codziennego. Zadaniem uczestników szkolenia może być sformułowanie problemu z życia codziennego lub innych dziedzin nauki – na poziomie uczniów końcowych klas szkoły podstawowej. Ćwiczenie można wykonać np. w zespołach dwuosobowych. Następnie uczestnicy szkolenia powinni przedstawić swoje propozycje.

2.3. Algorytmy dotyczące liczb naturalnych

Jeden z problemów został podany powyżej – podział klasy na równoliczne zespoły, czyli znalezienie dzielników liczby uczniów (w tym przypadku określenie, ile jest takich dzielników). Słuchacze powinni sprecyzować problem, następnie określić dane i spodziewany wynik. Następnie spróbować znaleźć rozwiązanie, czyli podać algorytm i zapisać go np. w języku naturalnym.

Problem: Na ile sposobów można podzielić klasę na równoliczne, co najmniej dwuosobowe zespoły, aby można było rozegrać zawody pomiędzy zespołami.

Dane: Liczba całkowita dodatnia – liczebność klasy.
Wynik: Liczba naturalna – ile jest możliwych podziałów.
Algorytm: Zespoły muszą być co najmniej dwuosobowe, muszą być też co najmniej dwa zespoły (jeśli mamy rozegrać zawody w obrębie klasy, to cała klasa nie może być zespołem). W związku z tym należy przeglądać wszystkie liczby od 2 do połowy liczebności klasy oraz sprawdzać, czy kolejna liczba jest dzielnikiem danej wejściowej (liczebności klasy). Jeśli tak, to powiększamy wynik o jeden (na początku przyjmujemy, że wynik jest zerem).

Warto przedyskutować z uczestnikami szkolenia, czy to jedyny możliwy sposób rozwiązania tego problemu, czy nie można tego zrobić sprawniej? Warto też zapytać, w jakiej sytuacji uzyskamy wynik równy zeru, oznaczający brak możliwości podziału klasy na zespoły.

Można także w tym punkcie szkolenia nawiązać do algorytmu Euklidesa znajdowania największego wspólnego dzielnika. Możemy sformułować problem następująco: mamy dwie liczby – długości boków prostokąta; chcemy znaleźć największy kwadrat, którym (powielając go) możemy wypełnić prostokąt. Można skorzystać z wizualizacji algorytmu dostępnych na stronie (http://programowanie.oeiizk.edu.pl).

Zasoby do wykorzystania:

- Strona OEIiZK: Algorytm Euklidesa – pokaz, (zakładka „Processing”);
- Strona OEIiZK: Algorytm Euklidesa – aplikacja interaktywna, (zakładka „Processing”).

Kolejny problem, do którego można nawiązać, został poruszony w przykładowym zadaniu z poprzedniej części szkolenia – „Przeliteruj liczbę”.

Uczestnicy szkolenia (pracując w zespołach) mogą spróbować sformułować jakiś problem dotyczący liczb naturalnych, do rozwiązania którego, oprócz sekwencji poleceń, potrzebne są takie elementy jak: powtarzanie i podejmowanie decyzji oraz operacje arytmetyczne na liczbach naturalnych. Następnie zespoły powinny wymienić się problemami i spróbować znaleźć oraz zapisać algorytm rozwiązania zadania. Ćwiczymy w ten sposób, oprócz rozwiązywania „klasycznych” problemów algorytmicznych, także kreatywność nauczycieli w formułowaniu zadań dla uczniów.

2.4. Różne sposoby zapisu algorytmów

Algorytmy możemy zapisywać w różnej formie – poprzednio zapisywaliśmy je w języku naturalnym oraz w języku programowania (np. jeśli uruchamialiśmy rysowanie figury w Scratchu lub Pythonie). W części szkolenia dotyczącej poziomu klas 4–6 zapisywaliśmy algorytmy także w postaci listy kroków. Inne popularne formy zapisu to pseudokod oraz schemat blokowy. Warto w tym miejscu podkreślić, że zapis w języku programowania powinien być poprzedzony użyciem innej formy. Warto zwrócić szczególną uwagę na zapis w pseudokodzie, gdyż najłatwiej na tej podstawie zaimplementować algorytm w języku programowania. W tym miejscu należy też zwrócić uwagę słuchaczom, że nie każda poruszana kwestia wymaga programowania. Dla niektórych problemów będziemy tylko poszukiwać rozwiązania (algorytmu) i zapisywać je w jednej z dostępnych form.

Uczestnicy szkolenia powinni spróbować zapisać algorytmy dotyczące liczb naturalnych omawiane w poprzednim punkcie w postaci listy kroków, schematu blokowego lub pseudokodu.

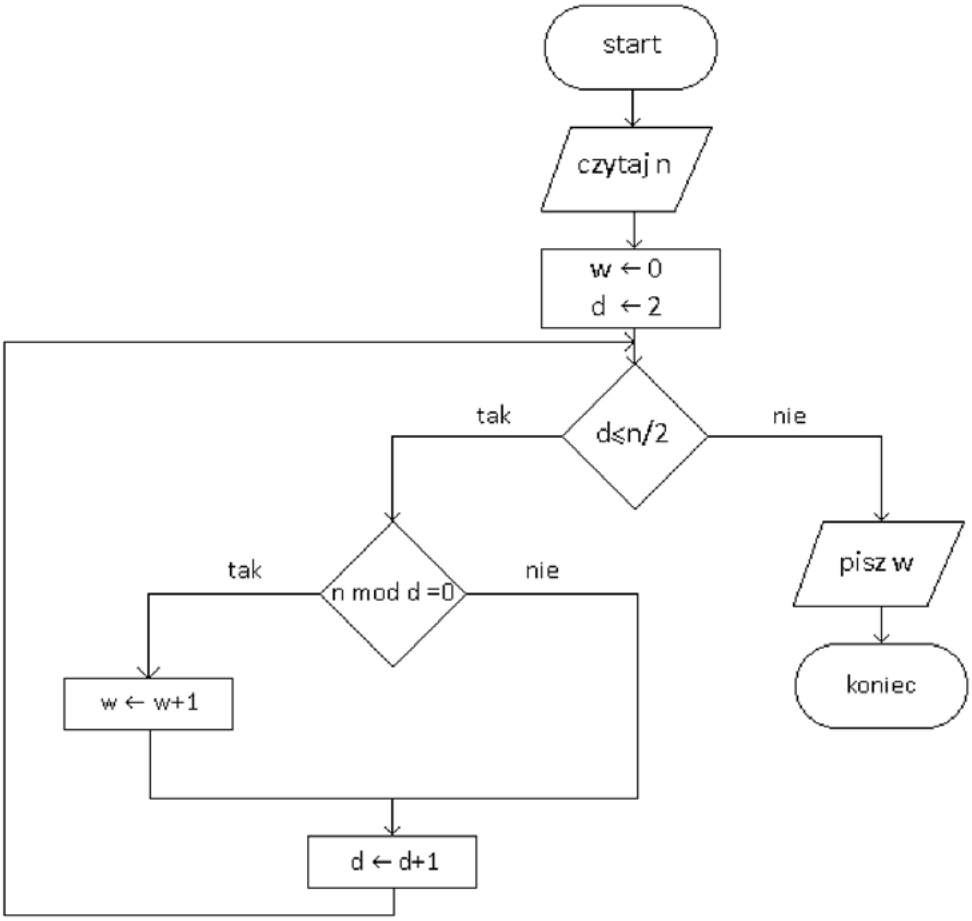
Przykład: Zapis algorytmu dotyczącego problemu liczby podziałów klasy na równoliczne zespoły.
Dane: n – liczba uczniów klasy;
Wynik: w – liczba możliwych podziałów na równoliczne zespoły.

Lista kroków:

1. Wczytaj n.
2. Przypisz zmiennej w wartość 0.
3. Dla zmiennej d przyjmującej wartości od 2 do $n/2$ wykonuj:
    - Jeżeli reszta z dzielenia n przez d jest równa 0, to:
        - przypisz zmiennej w wartość w+1;
    - Przypisz zmiennej d wartość d+1.
4. Wypisz w.

Pseudokod:

wczytaj n
$w \leftarrow 0$
dla $d = 2...n/2$ **wykonuj**
    jeżeli n **mod** $d = 0$ **to** $w \leftarrow w + 1$
    $d \leftarrow d + 1$
wypisz w



Rysunek 3. Schemat blokowy

3. Od programowania wizualnego do tekstowego

3.1. Przykład implementacji algorytmu dotyczącego liczb naturalnych w języku wizualnym

W tej części szkolenia dochodzimy do implementacji algorytmu z użyciem komputera i konkretnego narzędzia programistycznego – w tym przypadku języka wizualnego. Należy jednak pamiętać i akcentować, że implementowanie algorytmu stanowi tylko jeden z wielu kroków na drodze do rozwiązania zadania.

Uczestnicy szkolenia umieją już tworzyć projekty w środowisku Scratch, więc nie powinno stanowić problemu zaimplementowanie wybranych algorytmów. Można najpierw powtórzyć implementację zadania: „Przeliteruj liczbę" – ze szkolenia dotyczącego nauczania klas 4–6. Następnie warto napisać skrypty realizujące wybrany algorytm spośród wcześniej omówionych.

3.2. Przykład algorytmu zapisanego w języku tekstowym wysokiego poziomu (np. przy pomocy automatycznej translacji z języka wizualnego)

W tym punkcie szkolenia należy zrealizować przykładowy scenariusz 1. Dotyczy on problemu znajdowania największego wspólnego dzielnika. Algorytm jest implementowany najpierw w Scratchu, a następnie w środowisku Google Blockly (korzysta ono z bloków bardzo podobnych do używanych w Scratchu). Środowisko to umożliwia automatyczną translację na szereg języków tekstowych, w tym na język Python.

Uczestnicy szkolenia znają już podstawy programowania w języku Python (kurs dotyczący klas 4–6 lub uzupełniający) w zakresie rozwiązywania problemów z wykorzystaniem grafiki żółwia. Tym razem implementują pierwszy problem obliczeniowy, a także definiują funkcję obliczającą wartość. Poruszana jest również kwestia różnych algorytmów rozwiązania tego samego problemu, modyfikujemy algorytm Euklidesa (wersja z resztą z dzielenia). Przy okazji dyskutujemy o liczbie kroków koniecznych do wykonania, by wyliczyć wynik – rozważania te stanowią wstęp do analizy złożoności czasowej algorytmów.

3.3. Dyskusja na temat możliwości języków wizualnych

Porównujemy języki/środowiska programowania wizualnego i tekstowego – zarówno ich możliwości, jak i wygodę użytkowania oraz szybkość tworzenia kodu. Należy wskazać ograniczenia środowiska Scratch – brak możliwości definiowania funkcji określających wynik oraz brak operacji dzielenia całkowitego. Namawiamy do programowania tekstowego na zajęciach z uczniami. Możliwość łagodnego przejścia od programowania wizualnego do tekstowego zapewnia zastosowanie narzędzia do automatycznej translacji kodu. Przy prostych zadaniach dobre rezultaty daje użycie środowiska Google Blockly, w którym możemy utworzyć kod, zestawiając (wizualnie) klocki-bloczki i zobaczyć, jak wygląda rozwiązanie m.in. w języku Python (scenariusz zajęć realizowany w poprzednim punkcie).

3.4. Wprowadzenie do abstrakcji, parametry, pojęcie funkcji

Rozważamy różne podejścia do wprowadzania danych i prezentacji wyników. Docelowym rozwiązaniem jest takie, w którym wyraźnie zostaną rozróżnione: określenie danych, właściwa realizacja algorytmu, wypisanie wyniku. Do tego

celu niezbędne jest wprowadzenie mechanizmu funkcji z parametrami, mającymi możliwość określenia wyniku. W ten sposób jesteśmy w stanie zaimplementować algorytm tak, by szczegóły jego realizacji podczas wywołania pozostawały niejako ukryte dla osoby korzystającej z utworzonej funkcji. Ponadto w takiej sytuacji zmiana danych nie powoduje konieczności ingerowania w kod programu. Zauważmy, że już tak postępowaliśmy, realizując przykładowy scenariusz 1. Warto zwrócić uwagę, że dla uczniów często zagadnienia te okazują się trudne.

4. Programowanie w Pythonie

W tej części szkolenia zarówno wprowadzamy elementy języka programowania Python, jak i utrwalamy umiejętność ich wykorzystywania. Należy jednak pamiętać, że najlepsze efekty uzyskamy, gdy wprowadzanie nowych konstrukcji potraktujemy jako odpowiedź na konieczność zwiększania repertuaru środków używanych do zapisu kolejnych algorytmów stanowiących rozwiązanie określonego problemu. W trakcie szkolenia kładziemy zatem nacisk na umiejętność rozwiązywania problemów: rozpoczynamy od ich sformułowania i przechodzimy przez kolejne etapy rozwiązania.

Uczestnicy już na poprzednim szkoleniu w zakresie sterowania obiektem na ekranie zostali wprowadzeni do programowania tekstowego (z wykorzystaniem języka Python). Poznali podstawowe konstrukcje języka, korzystali ze zmiennych, definiowali funkcje, używali funkcji z parametrami.

4.1. Instrukcje wejścia/wyjścia

W języku Python można pracować w trybie bezpośrednim (interaktywnym, w którym polecenie jest wykonywane od razu po jego wpisaniu i zatwierdzeniu) lub z użyciem funkcji (skryptowym). Tryb bezpośredni okazuje się szczególnie przydatny, gdy chcemy szybko sprawdzić działanie np. rzadko używanej funkcji. W trybie bezpośrednim automatycznie wypisywana jest wartość ostatniego wyliczonego wyrażenia, nie ma potrzeby używania wbudowanego polecenia: `print()`. Zwykle jednak będziemy tworzyć funkcje.

Przykład zadania:
Rozważmy problem zamiany wartości zmiennych. Na przykład: jeśli zmiennej x przypisana została wartość 3, a zmiennej y wartość 5, chcemy, aby po wykonaniu programu było odwrotnie, czyli aby wartość zmiennej y wynosiła 3. W jakiej kolejności należy ustawić poniższe instrukcje?

```
x = y
y = pom
pom = x
```

Warto pokazać zastosowanie kilku najczęściej używanych funkcji z biblioteki `math` i przetestować ich działanie w trybie bezpośrednim.

```
from math import *

print(sqrt(2)) # pierwiastek kwadratowy
print(round(2.3)) # zaokrąglenie
print(floor(2.7)) # podłoga
print(ceil(2.3)) # sufit
print(pi) #liczba pi
kto = input('Jak masz na imię? ') #wczytanie danych, napis
ile = int(input('Ile masz lat? ')) #wczytanie danych, konwersja na liczbę całkowitą
print(ord('a')) # kod ASCII znaku
print(chr(97)) # znak o podanym kodzie ASCII
```

4.2. Tworzenie własnych funkcji obliczeniowych z parametrami

Prostym zadaniem może być obliczenie średniej arytmetycznej dwóch liczb.

```
def srednia(a, b):
    return (a + b) / 2
```

W ten sposób przypominamy pojęcie funkcji z parametrami. Na tym etapie nie akcentujemy tego, że wynik nie jest typu całkowitoliczbowego.

Przykłady zadań:
Korzystając z poniższego wzoru, napisz funkcję: `CnaF(c)`, której wynikiem dla podanej temperatury w stopniach Celsjusza jest temperatura w stopniach Farenheita.

$$F = 32 + \frac{9}{5} \times C$$

Wynikiem `CnaF(0)` jest 32, wynikiem `CnaF(100)` jest 212.

Napisz funkcję odwrotną: `FnaC(f)`, która zamienia stopnie Fahrenheita na Celsjusza.

$C = \frac{5}{9} \times (F - 32)$

Wynikiem `FnaC(32)` jest 0, wynikiem `FnaC(212)` jest 100.

Uczestnicy szkolenia, pracując parami, powinni wymyślić kilka prostych problemów obliczeniowych, następnie zapisać odpowiednie funkcje (jedna osoba proponuje problem do rozwiązania dla drugiej).

4.3. Wykorzystanie instrukcji warunkowych i iteracyjnych w rozwiązywanych zadaniach

Uczestnicy szkolenia korzystali już wcześniej z instrukcji warunkowych i iteracyjnych. Teraz powinni wykonać kilka prostych zadań obliczeniowych z wykorzystaniem takich instrukcji.

Przykłady zadań:

✓ Zadanie: Maksymalny element

Rozważamy problem znajdowania elementu maksymalnego. Zadanie polega na napisaniu funkcji: `maksymalny(x, y)`, której wynikiem jest większa z dwóch liczb lub, gdy są one równe – dowolna.

Wynikiem funkcji: `maksymalny(1, 3)` jest 3, wynikiem `maksymalny(4, 2)` jest 4, wynikiem `maksymalny(2, 2)` jest 2.

W poniższym kodzie jest błąd. Proszę go znaleźć.

```
1.  def maksymalny(x, y):
2.    if x < y:
3.      return x
4.    else:
5.      return y
```

✓ Zadanie: Zegar

Na zegarze elektronicznym wyświetlają się godziny, minuty i sekundy. Zdefiniuj funkcję: `ile(g, m, s)`, która wyznaczy, ile sekund upłynęło od północy do czasu wskazywanego na zegarze. Parametrami tej funkcji są trzy liczby określające odpowiednio: `g` – godziny, `m` – minuty i `s` – sekundy wskazywane przez zegar.

Np. wynikiem `ile(7, 3, 2)` jest 25382.

✓ Zadanie: Potęgowanie

Ania uczy się potęgować. Zaczęła od liczby 2. Wie, że:
$2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, ...
Zauważyła, że ostatnia cyfra wyniku powtarza się cyklicznie: 2, 4, 8, 6. Jeśli wykładnik potęgi to liczba: 1, 5, 9, 13, 17 itd., to ostatnią cyfrą jest 2. Jeśli wykładnik potęgi to liczba: 2, 6, 10, 14, 18 itd., to ostatnią cyfrą jest 4 itd. Tak wciągnęła ją zabawa, że postanowiła sprawdzić, jakie wyniki uzyska dla liczby 3.

Zdefiniuj funkcję o nazwie: `spr(n)`, która zweryfikuje obliczenia Ani dla tej liczby. Parametrem funkcji jest wykładnik potęgi (liczba z zakresu od 1 do 1000000). Wynikiem funkcji jest ostatnia cyfra rezultatu potęgowania 3^n.

Np. wynikiem `spr(1)` jest 3.

✓ Zadanie: Numery mieszkań

Numery mieszkań w pewnym bloku są trzycyfrowe i składają się z numeru klatki (1 lub 2), piętra (od 0 do 3) i numeru kolejnego mieszkania na piętrze (od 1 do 3). W każdej klatce i na każdym piętrze jest tyle samo mieszkań. Wypisz kolejne numery wszystkich mieszkań. Funkcję napisz tak, by łatwo można było zmienić liczbę klatek, pięter i mieszkań na piętrze.

```
1.  def mieszkania(maxk, maxp, maxm):
2.      for k in range(1, maxk+1):
3.          for p in range(maxp+1):
4.              for m in range(1,maxm+1):
5.                  print(100*k+10*p+m)
```

4.4. Rozwiązywanie zadań związanych z operacjami na liczbach naturalnych z wykorzystaniem algorytmów z badaniem podzielności, m.in. wyodrębnianie cyfr liczby i algorytm Euklidesa

Na początku tego etapu szkolenia należy rozwiązać przykładowe zadanie 1. Polega ono na wypisaniu cyfr liczby, nawiązuje do zadania „Przeliteruj liczbę" wykorzystywanego na poprzednim szkoleniu.

Kolejne zadanie może polegać na sumowaniu cyfr liczby.

```
1.  def suma_cyfr(liczba):
2.      suma = 0
3.      while liczba > 0:
4.          suma = suma + liczba % 10
5.          liczba = liczba // 10
6.      return suma
```

Podobnie możemy szukać największej cyfry w liczbie.

```
1.  def najwieksza_cyfra(liczba):
2.      najwieksza = 0
3.      while liczba > 0:
4.          if liczba % 10 > najwieksza:
5.              najwieksza = liczba % 10
6.          liczba = liczba // 10
7.      return najwieksza
```

W ten sposób przygotowujemy uczestników do rozwiązywania zadań związanych z porządkowaniem danych. Podobnych problemów można sformułować o wiele więcej – choćby: „Ile jest siódemek w danej liczbie?" lub: „Czy w danej liczbie jest więcej cyfr parzystych, czy nieparzystych, a może po równo?" (istnieją trzy możliwości, więc jest to dobre zadanie na wykorzystanie zagnieżdżonej instrukcji warunkowej).

Na tym etapie szkolenia warto zaprogramować rozwiązanie omawianego wcześniej zadania, dotyczącego podziału klasy na równoliczne grupy. Warto też podać przykład zadania, w którym należy wykorzystać algorytm Euklidesa.

✓ Zadanie: Drużyna

W klubie trenuje n zawodników grających na pozycji obrońcy i `m` zawodników grających w ataku. Na treningu, wykorzystując wszystkich zawodników, trener musi zestawić jak największą liczbę drużyn, w których będzie po tyle samo obrońców i atakujących. Napisz funkcję: `ile(n, m)`, której wynikiem będzie liczba obrońców występujących w każdej z drużyn.

Np. wynikiem `ile(80, 32)` jest 5.

Nie należy stronić od rozwiązywania nieco ambitniejszych zadań. Analizując problemy matematyczne, możemy np. rozwiązać zadanie: „Znajdź n-tą liczbę doskonałą, gdzie n to dana liczba naturalna". Jest to dobre zadanie do zastosowania konstrukcji, w której jedna z samodzielnie zbudowanych funkcji wywołuje inną funkcję (ćwiczymy podział problemu na podproblemy), wynik nie jest oczywisty i złożoność czasowa przyjętego algorytmu ma praktyczne znaczenie (przykładowe zadanie 3).

Podczas szkolenia należy również zaprezentować choć jeden przykład zadania, w którym poprawnie sformułowany algorytm rozwiązania eliminuje konieczność wykonywania wielu żmudnych czynności. Przykład taki może stanowić zadanie: „Czy iloczyn cyfr danej liczby naturalnej jest liczbą pierwszą?".

Pierwszym pomysłem, jaki się narzuca, jest policzenie iloczynu cyfr i sprawdzenie, czy jest on liczbą pierwszą. Tymczasem można postąpić inaczej – wystarczy zauważyć, że warunkiem koniecznym i wystarczającym do tego, by iloczyn cyfr był liczbą pierwszą, jest to, by w liczbie występowały tylko jedynki (może też nie być ani jednej) i dokładnie jedna cyfra spośród 2, 3, 5, 7.

Jeszcze kilka przykładów zadań:

✓ Zadanie: Ile mamy dni?

Czy zastanawiałaś/eś się kiedyś, ile dni dotąd przeżyłeś/aś? Zadanie jest na pozór proste. Trzeba jednak pamiętać o uwzględnieniu lat przestępnych. Rok przestępny to taki rok, który jest podzielny przez 4, ale nie jest podzielny przez 100 lub jest podzielny przez 400. Napisz funkcję: `imd(rok, miesiąc, dzień)`. Wartością funkcji jest liczba dni, która upłynęła od danej daty do dzisiaj.

✓ Zadanie: Ślimak na słupie

Zdefiniuj funkcję: `kiedy(x, y, z)`. Wynikiem funkcji jest liczba określająca, którego dnia ślimak znajdzie się na szczycie dziesięciometrowego słupa. Ślimak pierwszego dnia startuje u podstawy słupa. Każdego dnia wspina się o x centymetrów. Każdej nocy osuwa się o y centymetrów, chyba że natrafi na półkę – wtedy zatrzymuje się na niej. Półki znajdują się co z centymetrów, licząc od podstawy słupa. Zakładamy, że x jest większe od y.

Wynikiem funkcji `kiedy(300, 100, 100)` jest 4, wynikiem `kiedy(5, 3, 2)` jest 250.

✓ Zadanie: Suma cyfr

Zdefiniuj funkcję `suma_jed(liczba)`, której wartością jest liczba jednocyfrowa. Funkcja powinna liczyć sumę cyfr parametru: `liczba`, następnie sumę cyfr policzonej sumy, itd. – aż suma będzie jednocyfrowa.

Wynikiem funkcji `suma_jed(12356)` jest 8, wynikiem `suma_jed(123)` jest 6.

4.5. Przykłady różnych algorytmów rozwiązania tego samego problemu

Realizując przykładowy scenariusz 1, korzystaliśmy z algorytmu Euklidesa znajdowania największego wspólnego dzielnika zarówno z odejmowaniem, jak i z resztą z dzielenia. Porównaliśmy te algorytmy pod kątem liczby wykonywanych operacji. W przykładowym zadaniu 3 (realizowanym ewentualnie w poprzednim punkcie), także występują dwa algorytmy obliczania sumy dzielników liczby o różnej złożoności (liczbie wykonywanych operacji). Warto w tym miejscu po raz kolejny powrócić do problemu podziału klasy na równoliczne grupy i znaleźć algorytm wykonujący mniejszą liczbę operacji.

```
wczytaj n
w ← 0
d ← 2
dopóki d*d<n wykonuj
        jeżeli n mod d = 0 to w ← w + 2
        d ← d + 1
jeżeli d*d = n to w ← w + 1
wypisz w
```

Przykład zadania:
Napisz funkcję: `piramida(n, bok)`, której wynikiem będzie łączna długość odcinków tworzących piramidę. Pierwszy parametr określa liczbę poziomów piramidy, a drugi – długość boku każdego z kwadratów.

Rysunek 4. Piramidy o 4 i 5 poziomach

Można zaproponować przeanalizowanie poniższych rozwiązań. Po pierwsze należy zweryfikować, czy wszystkie są prawidłowe – jak to sprawdzić?
Czy na podstawie rozwiązania można odtworzyć pomysł na jego zaprojektowanie?

```
1.  def piramida1(n, bok):
2.      wy = 0
3.      for i in range(n):
4.          wy = wy + 2 *(n - i) + 1
5.      return (wy + n) * bok
6.
7.  def piramida2(n, bok):
8.      x = 0
9.      for i in range(n):
10.         x = x + (n - i) * bok * 3 - (n - i - 1) * bok
11.     return x + n * bok
12.
13. def piramida3(n, bok):
14.     l = 0
15.     b = n
16.     while b > 0:
17.         l = b * bok + l
18.         b = b - 1
19.     l = l + 3 * n * bok
20.     c = n - 1
21.     h = 0
22.     while c > 0:
23.         h = c * bok + h
24.         c = c - 1
25.     l = l + h
26.     return l
```

```
27.
28.  def piramida4 (n, bok):
29.     return (n + 3) * n * bok
```

Podsumowanie

Podstawowy zestaw instrukcji języka Python, który powinni opanować uczestnicy szkolenia, obejmuje instrukcję przypisania wartości – podstawienia (=), wprowadzanie wartości przez użytkownika (`input`), wypisywanie wyniku (`print`), iteracje (`while, for`), instrukcję warunkową (`if`) oraz informacje o tworzeniu wyrażeń arytmetycznych (operatory: `+, -, *, **, /, //, %`; nawiasy) i logicznych (porównania, spójniki logiczne, nawiasy). Używamy funkcji i korzystamy z wbudowanych funkcji pierwotnych. Z metodycznego punktu widzenia ważne jest, aby poszczególne elementy wprowadzać wówczas, gdy stają się potrzebne do zaimplementowania konkretnego algorytmu. Nie należy wprowadzać wszystkich jednocześnie.

Dobieramy zadania o coraz wyższym stopniu trudności. Powinny one wymagać zaprojektowania algorytmu rozwiązania, w części zadań możemy zasugerować algorytm.

Zasoby do wykorzystania:

- Podręcznik Wikibooks: Zanurkuj w Pythonie;
- Materiały „Koduj z Klasą" dotyczące Pythona;
- Samouczek Pythona: http://www.learnpython.org/pl/;
- Python w szkole – materiały OEIiZK;
- Code Academy (ang.): https://www.codecademy.com/learn/learn-python;
- Interaktywny samouczek Pythona: https://snakify.org/.

5. Wyszukiwanie i porządkowanie informacji, przetwarzanie danych

5.1. Informatyka bez komputera

W tej części szkolenia warto odwołać się do materiałów z serwisu http://jasijoasia.edu.pl, scenariuszy projektu „CS Unplugged".
Omawianie zagadnień dotyczących przetwarzania danych można rozpocząć bez używania komputera. W poniższej tabelce znajdują się propozycje scenariuszy do ewentualnego wykorzystania. Niektóre z nich mogły już zostać zrealizowane podczas poprzedniej części szkolenia dla klas 4–6. W zależności od poziomu grupy

wybieramy gotowe scenariusze lub proponujemy własne. Wskazane jest podzielić uczestników na grupy i każdej grupie przydzielić do opracowania po jednym temacie. Później powinna nastąpić krótka prezentacja efektów ich pracy.

Na początku są tylko dane *O reprezentacji informacji*	*Zanim wprowadzimy komputer* *O algorytmach*
• *Zliczanie kropek* – system binarny • *Kolory jako liczby* – kodowanie obrazu • *To można odtworzyć!* – kompresja tekstu • *Magia obracanych kart* – wykrywanie i korekcja błędów • *20 pytań* – teoria informacji	• *Wojna morska* – algorytmy przeszukiwania • *Najlżejsze i najcięższe* – algorytmy sortowania • *Zrobić to szybciej* – sieci sortujące

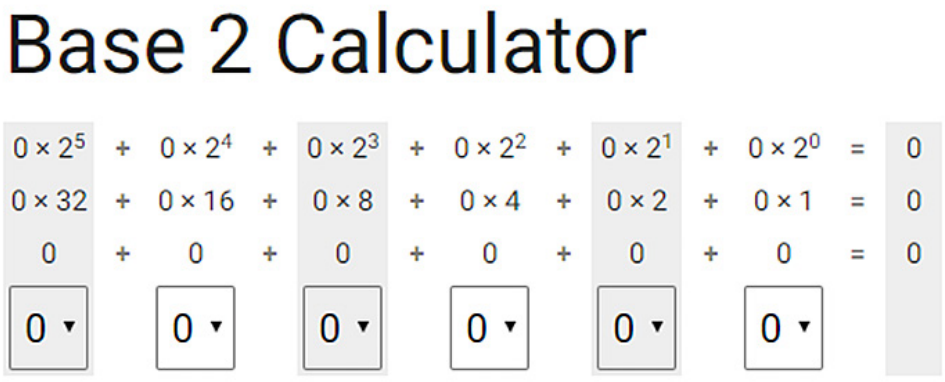
W tej części szkolenia można także wykorzystać zadania konkursu „Koala". Jest on organizowany w dwóch kategoriach – jedna dotyczy uczniów klas 7–8. Rywalizacja polega na rozwiązywaniu zadań matematyczno-informatycznych bez korzystania z komputera.

Zasoby do wykorzystania:

- Strona: http://jasijoasia.edu.pl/ (projekt „CS Unplugged", Scenariusze);
- Strona konkursu „Koala": http://jasijoasia.edu.pl/koala/index.htm.

5.2. Reprezentacja liczb, znaków i napisów, tablice (listy)

Podstawą reprezentacji informacji w komputerze jest system dwójkowy. Oprócz przeprowadzenia wspominanych na szkoleniu dla klas 4–6 ćwiczeń z wykorzystaniem kart binarnych, można rozważyć na bardziej abstrakcyjnym poziomie problem przeliczania z systemu na system.



Rysunek 5. Kalkulator binarny

Uczestnicy otrzymują polecenie zapisania konkretnej liczby w systemie dwójkowym, a następnie jej przeliczenia oraz odwrotnie. Można też postawić dodatkowe pytania: Ile liczb możemy zapisać za pomocą dwóch cyfr, trzech cyfr, *n* cyfr? Jaką najmniejszą i jaką największą liczbę można zapisać? Należy krótko omówić reprezentację znaków w komputerze – przekazać informację o kodach ASCII (ang. *American Standard Code for Information Interchange*) oraz wartościach logicznych. Czasami w zadaniach związanych z przetwarzaniem znaków/napisów łatwiej jest operować na kodach-liczbach, niż bezpośrednio na znakach. Przykładem mogą być zadania dotyczące szyfrowania napisu, np. z wykorzystaniem „szyfru Cezara". Od wykorzystania napisów jest już bardzo blisko do zastosowania tablic jako zbiorów elementów tego samego typu. W języku Python tablice implementowane są za pomocą list.

Podsumowanie tej części może polegać na znalezieniu algorytmu rozwiązania następującego zadania (przykładowe zadanie 2):

Znajdź najmniejszą liczbę naturalną, która nie jest sumą liczb znajdujących się w podanym uporządkowanym ciągu liczb naturalnych.

Na przykład dla ciągu [1, 2, 3, 4, 100] wynikiem jest 11, a dla [1, 1, 2, 3, 5, 18, 21] wynikiem jest 13.

Zasoby do wykorzystania:

- Strona OEIiZK: http://programowanie.oeiizk.edu.pl, (sekcja: „Informatyka /prawie/ bez komputera");
- Jochemczyk W., Olędzka K., (2016), *Ważenie a system binarny*, Toruń: Informatyka w Edukacji.

5.3. Rozwiązywanie zadań związanych z wyszukiwaniem elementu w zbiorach nieuporządkowanych i uporządkowanych

Ważne, aby w tej części szkolenia ukazać, jak powszechne jest wyszukiwanie i porządkowanie informacji – jak wiele problemów sprowadza się do wykonywania tych operacji. Rozwiązujemy odpowiednio dobrane zadania.

Omówienie zagadnień związanych z wyszukiwaniem elementu w tablicy rozpoczynamy od wyszukiwania w tablicy nieuporządkowanej. Na przykład: Wpisz swoje imię, a komputer odpowie, czy jesteś chłopcem, czy dziewczynką. Piszemy funkcję, której parametrem jest imię dziecka, a wynikiem napis: „chłopiec" lub „dziewczynka". W najprostszej wersji wystarcza analiza ostatniego znaku imienia:

```
1.  def dziecko(imie):
2.     if imie[len(imie) - 1] == 'a':
3.        return 'dziewczynka'
4.     else:
5.        return 'chłopiec'
```

Powyższa wersja funkcji wydaje się jednak niedoskonała – nie uwzględnia, że istnieją imiona żeńskie, które nie kończą się literą a, ani że zdarzają się imiona męskie, których ostatnią literą jest a. Tworzymy listy wyjątków (zadbajmy, by znalazły się w niej odpowiednie imiona osób z grupy, z którą prowadzimy zajęcia, o ile są reprezentowane):

```
1.  meskie_w = ['Kuba', 'Barnaba', 'Dyzma', 'Kosma', 'Bonawentura', 'Jarema']
2.  zenskie_w = ['Inez', 'Dolores', 'Beatrycze']
```

Rozbudowana wersja funkcji będzie wyglądała następująco:

```
1.  def dziecko(imie):
2.     if imie[len(imie) - 1] == 'a':
3.        if jest_elementem(imie, meskie_w):
4.           return 'chłopiec'
5.        else:
6.           return 'dziewczynka'
7.     else:
8.        if jest_elementem(imie, zenskie_w):
9.           return 'dziewczynka'
10.       else:
11.          return 'chłopiec'
```

Brakuje już tylko funkcji: `jest_elementem`. Najprościej można zapisać ją, korzystając z operacji: `in` (należy do zbioru).

```
1.  def jest_elementem(imie, imiona):
2.     return imie in imiona
```

Warto rozwiązać także zadanie, w którym będziemy musieli samodzielnie przejrzeć zbiór nieuporządkowany – na przykład polegające na wypisaniu tych liczb, które w bezpośrednim sąsiedztwie mają liczby mniejsze od siebie.

```
1.  def szukaj(liczby):
2.      for i in range(1,len(liczby)-1):
3.          if liczby[i]>liczby[i-1] and liczby[i]>liczby[i+1]:
4.              print(liczby[i])
```

Kolejnym zagadnieniem jest wyszukiwanie w tablicy uporządkowanej, dla ustalenia uwagi: niemalejąco. Możemy w tym przypadku postąpić sprytniej – nie musimy przeglądać po kolei wszystkich elementów. Warto w tym miejscu nawiązać do realizowanej podczas poprzedniego szkolenia gry w zgadywanie liczby. Rozpoczynamy od określenia zakresu poszukiwań – jest nim cała tablica, od początkowego do ostatniego elementu. Następnie zawężamy ten zakres w następujący sposób: sprawdzamy element znajdujący się pośrodku zakresu: jeśli jest on mniejszy od poszukiwanego – w poszukiwaniach będziemy brać pod uwagę tylko elementy stojące na dalszych pozycjach niż środkowy, w przeciwnym przypadku – te dalsze elementy odrzucamy. Czynności powtarzamy. Kończymy, gdy pozostanie jeden element.

```
1.  def jest_elementem_uporz(poszukiwany, a):
2.      imin = 0
3.      imax = len(a) - 1
4.      while imin < imax:
5.          s = (imin + imax) // 2
6.          if a[s] < poszukiwany:
7.              imin = s + 1
8.          else:
9.              imax = s
10.     return a[imin] == poszukiwany
```

Rozważmy następujące zadanie: Dany jest co najmniej trzyelementowy zbiór liczb określających długości odcinków. Czy z każdego zestawu trzech odcinków o długościach wziętych z tego zbioru można zbudować trójkąt? Rozwiązanie można sprowadzić do znalezienia dwóch najmniejszych elementów oraz elementu największego (przykładowe zadanie 4).

Znalezienie najmniejszego (podobnie jak największego) elementu w zbiorze nie jest zadaniem trudnym. Należy jednak uświadomić sobie, w jakim celu to robimy. Możliwe są dwa podejścia, których realizacja nieco się różni. W pierwszym z nich interesuje nas jedynie wartość najmniejszego elementu, a w drugim – umiejscowienie najmniejszego elementu (w tablicy-liście reprezentującej zbiór w komputerze). Odpowiednie fragmenty programów mogą wyglądać następujco (a – oznacza tablicę, w której szukamy najmniejszego elementu; zakładamy, że najmniejszy jest element początkowy, następnie przeglądamy kolejne elementy i gdy znajdziemy wśród nich mniejszy, zmieniamy wartość zmiennej):

```
1.  def mini(a):
2.      minimum = a[0]
3.      for i in range(1, len(a)):
4.          if a[i] < minimum:
5.              minimum = a[i]
6.      return minimum
```

oraz

```
1.  def imini(a):
2.      iminimum = 0
3.      for i in range(1, len(a)):
4.          if a[i] < a[iminimum]:
5.              iminimum = i
6.      return iminimum
```

Dla a = [22, 44, 66, 88, 11, 33, 55, 77, 99] wynikiem `mini(a)` jest 11 (tj. wartość najmniejszego elementu), a wynikiem `imini(a)` jest 4 (tj. wartość indeksu dla elementu najmniejszego – bo a[4]=11 – przypominamy, że elementy numerujemy od zera).

Warto z uczestnikami szkolenia zastanowić się, jak rozwiązać zadanie dotyczące trójkątów (znalezienie dwóch najmniejszych liczb i największej).

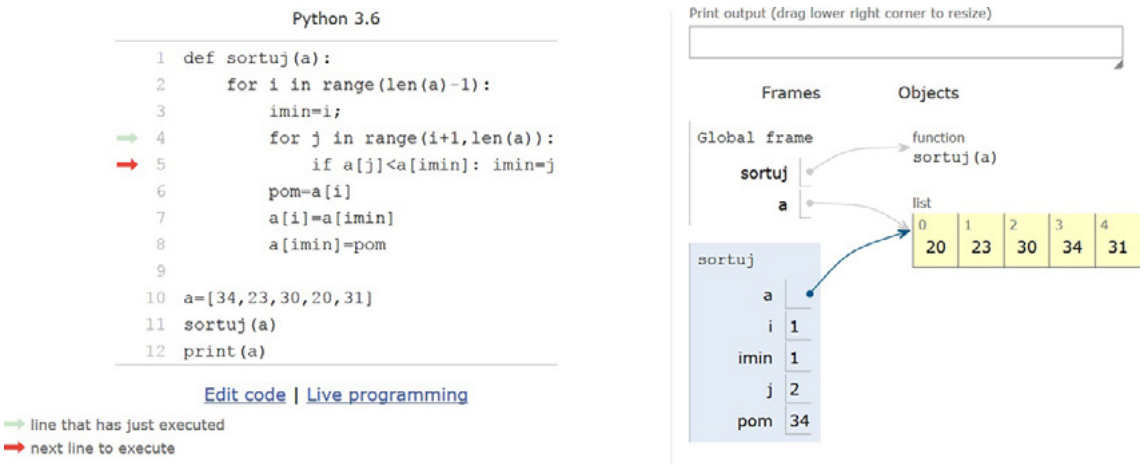
5.4. Rozwiązywanie zadań związanych z porządkowaniem zbiorów

Porządkowanie elementów zbioru jest jednym z zadań najczęściej wykonywanych przez komputery. Warto porządkować zbiory – w zbiorze uporządkowanym łatwiej wyszukujemy elementy, bezproblemowo uzyskujemy dostęp do elementu

najmniejszego i największego itd. Nawiązujemy do praktycznych sytuacji, takich jak np. układanie we właściwej kolejności kartek, które nam się rozsypały (ale na szczęście były ponumerowane). Podczas szkolenia prosimy uczestników, by zaproponowali algorytmy porządkowania. Okaże się, że propozycji jest wiele – istnieją różne algorytmy rozwiązania tego zadania. Prawie wszystkie opierają się na wykorzystaniu jako podstawowej operacji porównania dwóch elementów sortowanego zbioru (tablicy). Poszczególne elementy przestawiane są w taki sposób, by w końcu otrzymać dane uporządkowane.

Jednym z popularnych algorytmów porządkowania jest sortowanie przez proste wybieranie. Załóżmy, że chcemy tablicę posortować niemalejąco. Zatem na początkowej pozycji powinien znaleźć się element najmniejszy, na kolejnej – najmniejszy z pozostałych itd. Algorytm sortowania przez proste wybieranie polega na wyszukaniu elementu mającego się znaleźć na żądanej pozycji i zamianie miejscami z tym, który był tam dotychczas. Najpierw znajdujemy element najmniejszy w całej tablicy i zamieniamy go z początkowym. W kolejnym kroku znajdujemy element najmniejszy w części tablicy bez elementu początkowego i zamieniamy go z drugim. Podobnie w kolejnych krokach – znajdujemy najmniejszy z elementów nieposortowanej jeszcze części tablicy i zamieniamy go z pierwszym w tej części. Wielokrotnie powtarzamy tę akcję dla nieuporządkowanej części danych, aż cała tablica będzie posortowana. Zwróćmy uwagę, że wykorzystujemy algorytm poszukiwania miejsca minimum omówiony w poprzednim punkcie.

Na zajęciach warto zapisać ten algorytm w pseudokodzie. Można też zaprezentować wizualizację algorytmu, korzystając z serwisu http://pythontutor.com/.



Rysunek 6. Wizualizacja algorytmu sortowania przez wybieranie

Pewne algorytmy porządkowania są przydatne jedynie dla określonych specyficznych danych. Jednocześnie niecelowe jest używanie ich w innych sytuacjach. Jeśli na przykład w tablicy mamy zapisaną sekwencję wyników *n* rzutów sześcienną kostką do gry, chcąc ją uporządkować, wykorzystamy algorytm sortowania przez zliczanie. Ma on zastosowanie w sytuacji porządkowania zbiorów całkowitoliczbowych, a do takich należy zbiór wyników rzutów kostką. Sortowanie przez zliczanie wymaga wykorzystania dodatkowej struktury danych, w której na początku wyliczamy i zapamiętujemy liczby wystąpień poszczególnych wartości z sortowanej tablicy. Kiedy już wyliczymy te liczby, wypełniamy tablicę niejako od nowa, odpowiednimi danymi.

```
1.  def sortuj(a):
2.      b=[0 for i in range(6)]
3.      for i in range(len(a)):
4.          b[a[i]-1]=b[a[i]-1]+1
5.      k=0
6.      for i in range(6):
7.          for j in range(b[i]):
8.              a[k]=i+1
9.              k=k+1
```

Tablice – listy są zawsze indeksowane od zera. Możliwe wyniki rzutów kostką znajdują się w zakresie od 1 do 6, więc przy zliczaniu od wyniku odejmujemy 1, a wypełniając tablicę – dodajemy 1. Warto w tym miejscu zastanowić się, jak postępować, gdy nie znamy zakresu danych.

Omawiając algorytmy porządkowania danych, przeprowadzamy dyskusję nt. złożoności czasowej i pamięciowej algorytmów. W przypadku algorytmu sortowania przez zliczanie zastanawiamy się, kiedy warto go stosować.

Zasoby do wykorzystania:

- Strona wizualizacji kodów Pythona: http://pythontutor.com/.

5.5. Przykłady zadań związanych z przetwarzaniem danych, które nie są liczbami

W tym punkcie można zrealizować przykładowy scenariusz 2. Omawiany w nim problem polega na sprawdzeniu, czy dwa słowa są swoimi anagramami.

Przykład zadania:
Napisz funkcję: `ile(napis)`, której wynikiem będzie liczba unikatowych znaków występujących w napisie.
Np. wynikiem `ile('abrakadabra hokus pokus')` jest 11.

6. Rozwiązywanie problemów metodami wywodzącymi się z informatyki

6.1. Realizacja algorytmów w arkuszu kalkulacyjnym

Rozwiązywanie problemów algorytmicznych nie zawsze wymaga użycia języka programowania wysokiego poziomu. Często wystarczy użyć innego narzędzia wyspecjalizowanego w przetwarzaniu danych. Jako przykład posłuży tu arkusz kalkulacyjny. Możemy w nim wykonywać obliczenia i zapisywać algorytmy. Nie trzeba znać działania wszystkich funkcji arkusza – do rozwiązywania problemów algorytmicznych zwykle wystarczą podstawowe operacje: działania arytmetyczne, funkcja warunkowa, sumowanie. Arkusz kalkulacyjny wydaje się szczególnie godny polecenia do realizowania algorytmów związanych z obliczeniami. Po wpisaniu ogólnego wzoru ciągu, można wygodnie śledzić, jak zmieniają się kolejne wyrazy zadanego ciągu, a także próbować wykrywać zależności, stawiać i weryfikować hipotezy. Wizualizacja, np. w postaci wykresu, także może pomóc w tych czynnościach. Po wpisaniu formuł łatwo przeprowadzać symulacje dla różnych danych.

Jako praktyczne ćwiczenie warto rozwiązać z uczestnikami szkolenia przykładowe zadanie 5.

6.2. Przykłady problemów z innych dziedzin

Podczas szkolenia należy nawiązywać do zadań z różnych dziedzin. W naturalny sposób narzucają się naszej uwadze problemy matematyczne. Warto jednak wskazywać do rozwiązania problemy dotyczące również pozostałych przedmiotów nauczanych w szkole i innych sfer działalności człowieka.

Z dziedziny fizyki można rozważyć np. zagadnienie spadku swobodnego ciał, z zakresu geografii – znajdowanie wśród danego zbioru punktów (współrzędnych geograficznych) tych wysuniętych najbardziej na południe, a z obszaru języków (zarówno polskiego jak i obcych) – tworzenie losowych zdań, w których poszczególne elementy (podmiot, orzeczenie i dopełnienie) pochodzą z określonych zbiorów słów i muszą zostać użyte w odpowiedniej formie.

W ten sposób pokazujemy, że informatyka integruje się z niemal wszystkimi innymi dziedzinami i staje się ich nieodłącznym elementem. W uczniach, którzy widzą jej konkretne zastosowania, wzbudzamy zainteresowanie tą dziedziną.

Większość nauczycieli informatyki posiada przeważnie inne kierunkowe wykształcenie, a przygotowanie do nauczania informatyki uzyskali na studiach podyplomowych. Niech zatem każdy ze słuchaczy zaproponuje problem do rozwiązania z wykorzystaniem metod algorytmicznych z zakresu dziedziny, w której zdobył wykształcenie.

6.3. Wykorzystanie języków programowania do sterowania (np. robotem lub obiektem na ekranie)

Ta część szkolenia jest zależna od możliwości placówki organizującej szkolenie – od tego, czy i jakim sprzętem dysponuje. Szkolenie nauczycieli powinno zawierać elementy oparte na wykorzystaniu języka programowania do sterowania. To, czym będziemy sterować (np. robotem), czy będziemy programować (np. układ *Arduino*), zależy głównie od możliwości placówki. Na rynku jest dostępnych wiele różnych robotów z możliwością ich programowania w języku wizualnym lub tekstowym.

Sterowanie nie musi ograniczać się tylko do zastosowania robotów. Micro:bit to niepozorna płytka urządzenie, która mieści się w kieszeni, a jest mikrokomputerem zaprojektowanym do pracy z dziećmi i młodzieżą. Można zaprogramować go w języku wizualnym lub tekstowym. Jednym z możliwych języków jest Python, co stanowi dużą zaletę. Programujemy, używając komputera, a następnie przesyłamy gotowy kod do urządzenia. Podobnie działają: BeCreo i Arduino – modułowe zestawy do nauki programowania z elementami robotyki.

Jeśli placówka nie dysponuje żadnym sprzętem, to nie należy rezygnować z tematyki sterowania obiektem. W takiej sytuacji sterujemy obiektem na ekranie. Obiekt poruszający się po ekranie może rysować lub wchodzić w interakcje z innymi obiektami – możliwości jest bardzo wiele. Na stronie konkursu

http://logia.oeiizk.waw.pl/ znajdziemy wiele trudniejszych zadań związanych ze sterowaniem obiektem na ekranie i przetwarzaniem struktur danych.

Zasoby do wykorzystania:

- Strona konkursu „Logia": http://logia.oeiizk.waw.pl/;
- Strona zestawu BeCreo: http://becreo.eu/index.php/pl/.

7. Podsumowanie

Szkolenie koncentruje się na umiejętnościach, jakie uczeń zdobywa na II etapie edukacyjnym. Uczestnicy szkolenia powinni także otrzymać informacje o tym, czego uczniowie będą się uczyć na kolejnym etapie edukacyjnym w szkole ponadpodstawowej i jakie zmiany niesie w tym obszarze reforma oświaty. Warto także przedstawić zasady organizacji egzaminu maturalnego z informatyki.

Podczas szkolenia wykorzystuje się język Python. Należy przedstawić uczestnikom także inne języki tekstowe, które mogą być alternatywnie wykorzystywane, np. Java, JavaScript, Processing, C++. Warto przeprowadzić krótką dyskusję na temat funkcjonalności tych języków oraz wyboru języka właściwego do realizacji podstawy programowej w zależności od poziomu i możliwości uczniów, a także umiejętności zdobytych przez nich w niższych klasach.

Podstawa programowa zakłada stopniowe przechodzenie na poziomie klasy 7 na język tekstowy. Można także realizować podstawę, używając języka wizualnego – zwłaszcza gdy wcześniej uczniowie mieli mało zajęć dotyczących algorytmiki i programowania. Należy wówczas wykorzystać środowisko, które umożliwia definiowanie funkcji zwracających wynik, np. Google Blockly lub Snap!

Może okazać się, że napotykamy grupę uczniów wyjątkowo zdolnych i/lub wyjątkowo zainteresowanych omawianymi tu treściami. Warto zainspirować ich do dalszej pracy. Pomocne będą tu strony konkursów i olimpiad informatycznych oraz strony powiązane z nimi. Stanowią one obszerną bazę zadań, często posiadają też w swych zasobach kursy związane z algorytmiką czy omówienia zadań konkursowych. Niektóre portale umożliwiają wysyłanie rozwiązanych zadań i automatyczne ich sprawdzanie. Linki do poszczególnych portali zostały podane poniżej. Warto wskazać je uczestnikom szkolenia i krótko omówić. Istotne, by nauczyciele zachęcali swoich uczniów do udziału w konkursach informatycznych. Dają one uczniom możliwości rozwoju w dziedzinie informatyki także poza regularnymi zajęciami w szkole.

Zasoby do wykorzystania:

- Strona konkursu „LOGIA”: http://logia.oeiizk.waw.pl/;
- Strona konkursu „Bóbr”: https://www.bobr.edu.pl/
  (kategoria „Junior” dla uczniów klas 7–8);
- Strona konkursu „Koala”: http://jasijoasia.edu.pl/koala/index.htm;
- Strona Olimpiady Informatycznej Gimnazjalistów: http://oig.edu.pl/
  (omówienia zadań);
- Strona Olimpiady Informatycznej: https://www.oi.edu.pl/
  (zakładki: książeczki, linki);
- Strona Młodzieżowej Akademii Informatycznej: https://main2.edu.pl/news/
  (kursy, zadania, możliwość wysyłania i sprawdzania zadań);
- Portal treningowy Olimpiady Informatycznej: https://szkopul.edu.pl/
  (kursy, zadania, możliwość wysyłania i sprawdzania zadań);
- Portal sphere online judge http://pl.spoj.com/
  (zadania, konkursy, możliwość wysyłania i sprawdzania zadań);
- Akademia Khana: https://pl.khanacademy.org/
  (informatyka, algorytmy).

PRZYKŁADOWE SCENARIUSZE ZAJĘĆ

Scenariusz 1: *Od bloków do programowania tekstowego – na przykładzie algorytmu Euklidesa*

Opis zajęć

Głównym celem zajęć jest przedstawienie, jak można, pracując z uczniami, płynnie przejść od programowania wizualnego do tekstowego. Rozważamy problem znalezienia największego wspólnego dzielnika dwóch liczb naturalnych. Na początku zajęć formułujemy problem, określamy dane i oczekiwany wynik, zaproponowane rozwiązanie zapisujemy w postaci algorytmu w pseudokodzie. Algorytm zostanie zaimplementowany najpierw w Scratchu, a następnie w środowisku Google Blockly, które umożliwia automatyczną zamianę m. in. na język Python. Kolejnym krokiem będzie wprowadzenie funkcji oraz modyfikacja algorytmu.

Czas trwania

90 minut

Realizacja

Zaczynamy od sformułowania problemu. Dane są dwie liczby naturalne większe od zera, chcemy znaleźć ich największy wspólny dzielnik. W szkole pojęcie największego wspólnego dzielnika jest wprowadzane wcześniej na matematyce. Warto przedyskutować, jak uczniowie znajdują tradycyjnie największy wspólny dzielnik.

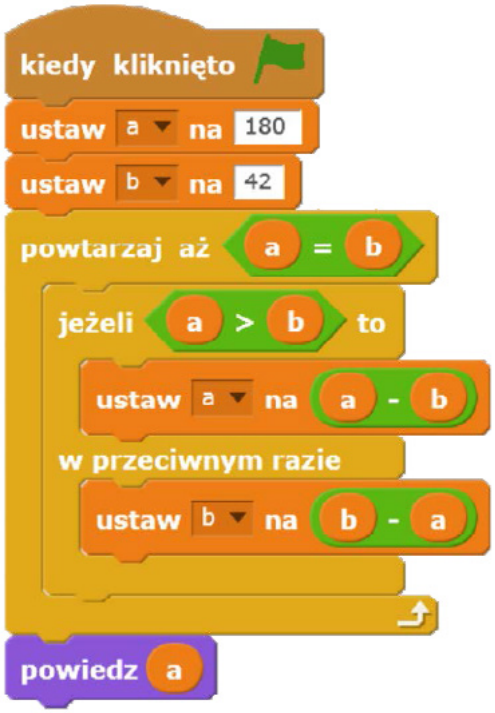
Na przykład dzielniki liczby 180 to: 1, 2, 3, 4, 5, 6, 9, 10, 12, 15, 18, 20, 30, 36, 45, 60, 90 oraz 180, a dzielniki liczby 42 to: 1, 2, 3, 6, 7, 14, 21 oraz 42. Czyli wyznaczamy dzielniki obu liczb i poszukujemy największej liczby występującej wśród dzielników obu liczb. W powyższym przykładzie widzimy, że taką liczbą jest 6.

Wypisywanie wszystkich dzielników każdej z dwóch danych liczb jest czynnością żmudną i nie wydaje się konieczne – np. widać, że szukany największy wspólny dzielnik nie może być większy od mniejszej z liczb (więc w powyższym przykładzie nie ma sensu branie pod uwagę liczb: 45, 60, 90 oraz 180, będących dzielnikami 180). Algorytm związany z wypisywaniem (pamiętaniem) wszystkich dzielników byłby niewygodny do zapisania, ze względu na konieczność użycia dodatkowych struktur danych. Można myśleć

o algorytmie, w którym analizujemy – w kolejności malejącej – kolejne liczby od mniejszej z dwóch danych liczb ku jedynce i szukamy pierwszej takiej, która jest dzielnikiem obu. Jednak to także wydaje się być postępowaniem nieefektywnym. Po przedyskutowaniu różnych możliwych rozwiązań, koncentrujemy się na algorytmie Euklidesa. Możemy skorzystać z wizualizacji algorytmu dostępnej na stronie http://programowanie.oeiizk.edu.pl/ (zakładka „Processing").

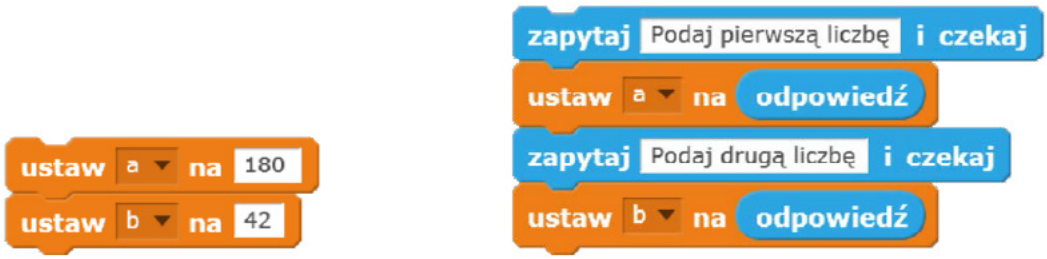
Uwaga: Powyższą część zajęć można pominąć, jeśli uczestnikom szkolenia (uczniom) jest już znany algorytm Euklidesa na odejmowanie (zagadnienia te były omawiane na wcześniejszych zajęciach). Wówczas przechodzimy od razu do zapisu algorytmu w pseudokodzie i implementacji w Scratchu.

Dane: a, b — liczby całkowite dodatnie.
Wynik: największy wspólny dzielnik liczb a i b.
dopóki a ≠ b **wykonuj**
    jeżeli a > b
        a ← a − b
    w przeciwnym przypadku
        b ← b − a
wypisz a



Rysunek 7. Algorytm Euklidesa na odejmowanie (Scratch)

Dane wprowadzone do programu w powyższym przykładzie są stałymi liczbami, których zmiana wymaga modyfikacji kodu. Możemy zastąpić przypisanie stałych wartości ich podaniem przez użytkownika.



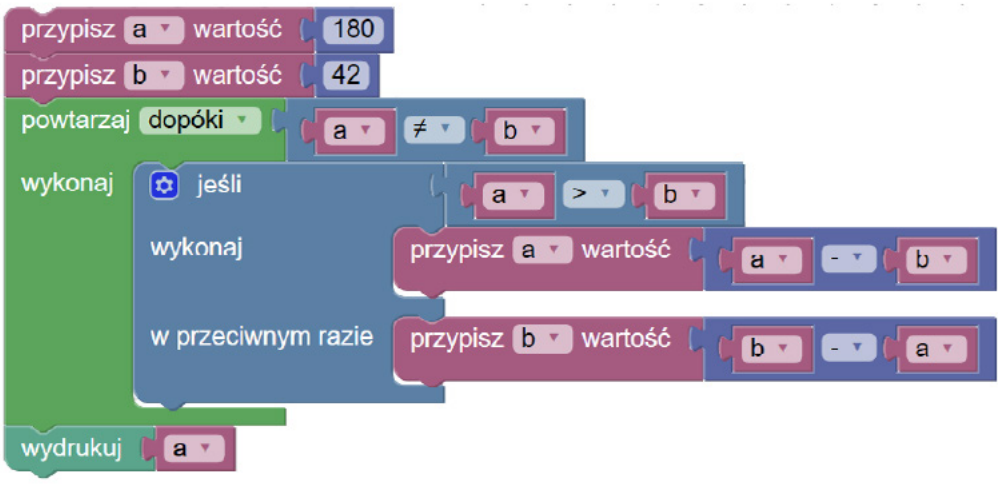
Rysunek 8. Określenie danych

Podobnie proces wpisywania wyniku może zostać rozbudowany. Warto wyróżnić trzy etapy:

- określenie danych,
- właściwą realizację algorytmu,
- wypisanie wyniku.

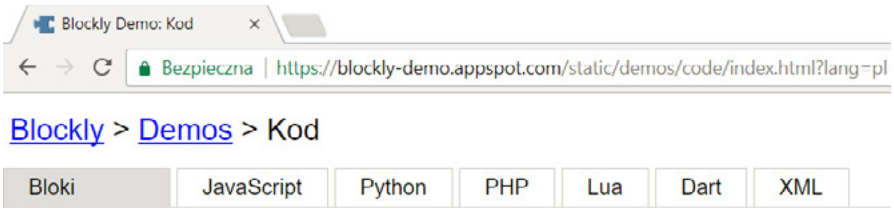
Koncentrujemy się w tym scenariuszu na algorytmie, więc rozbudowywanie obsługi wejścia/wyjścia nie jest aż tak istotne. Pozostaniemy przy dwóch stałych liczbach.

Zapisujemy jeszcze raz ten sam algorytm, korzystając z innego narzędzia do programowania wizualnego – Google Blockly (https://blockly-demo.appspot.com/static/demos/code/index.html?lang=pl, łatwiej wpisać w wyszukiwarkę Google Blockly Demo Code i otworzyć z poziomu wyszukiwarki).



Rysunek 9. Algorytm Euklidesa na odejmowanie (Google Blockly)

W Google Blockly możemy automatycznie przetłumaczyć kod na różne języki, w tym na język Python. Wystarczy kliknąć w wybraną zakładkę.



Rysunek 10. Zakładki Google Blockly

Poniżej znajduje się algorytm zapisany w Pythonie wygenerowany automatycznie (z pominięciem dwóch pierwszych wierszy).

```
a = 180
b = 42
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
print(a)
```

Rysunek 11. Algorytm Euklidesa na odejmowanie (Python – kod generowany automatycznie)

Powyższy kod można przekopiować do pliku i uruchomić w środowisku Python. Większość języków programowania ma możliwość tworzenia funkcji pozwalających na zapamiętanie ciągu poleceń pod pojedynczą nazwą, z możliwością podania parametrów – danych. Szczególnie godne polecenia są języki, w których istnieje możliwość tworzenia funkcji, które oprócz danych wejściowych mają możliwość udostępnienia wyniku modułowi wywołującemu.

W języku Scratch możemy tworzyć nowe bloki, a nawet przekazywać do nich parametry. Niestety nie ma on możliwości tworzenia funkcji, czyli przekazywania wyników działania do bloku, który ją wywołuje. Komunikacja jest możliwa jedynie za pomocą zmiennych globalnych. Praktyka ta generuje złe intuicje, gdyż zaburza rozumienie przepływu danych w pisanym programie.

Zastosujmy mechanizm funkcji do zapisu algorytmu Euklidesa w języku Python. Na początku zapisu pojawia się nowy wiersz stanowiący nagłówek funkcji – znajduje się w nim informacja o nazwie funkcji i o jej dwóch

parametrach – danych. Po obliczeniu wyniku, nie jest on wypisywany, ale udostępniany modułowi wywołującemu jako wynik funkcji.

```
1.  def nwd(a, b):
2.     while a != b:
3.        if a > b:
4.           a = a - b
5.        else:
6.           b = b - a
7.     return a
```

Powyższy kod może zostać także automatycznie wygenerowany z poziomu Google Blockly – w tym środowisku można definiować funkcje zwracające wynik. W zależności od poziomu wiedzy słuchaczy (uczniów) można skorzystać z tej możliwości. Dalsze przykłady będą zapisywane jedynie w języku Python.

Zastanawiamy się nad sprawnością działania algorytmu: czy można go poprawić, aby wykonywał mniej operacji? Warto na tablicy ręcznie (lub w arkuszu kalkulacyjnym) wykonać algorytm na odejmowanie, np. dla liczb: 1024 i 48. Jedna liczba jest znacznie większa od drugiej i wielokrotnie zmniejszamy pierwszą liczbę (otrzymując po kolei 976, 928, ..., 16). Szybko zauważamy, że możemy wielokrotne odejmowanie zastąpić resztą z dzielenia. Wówczas wynik otrzymamy dużo sprawniej. Należy zwrócić uwagę, kiedy zmodyfikowany algorytm się kończy. Poprzednio obie liczby były równe, teraz jedna z nich przyjmuje wartość zero. Wynik będzie dostępny w zmiennej, która nie uległa wyzerowaniu.

dopóki a ≠ 0 i b ≠ 0 **wykonuj**
        jeżeli a > b
                a ← a **mod** b
        w przeciwnym przypadku
                b ← b **mod** a
jeżeli a > b
        zwróć a
w przeciwnym przypadku
        zwróć b

mod (*modulo*) to popularne oznaczenie operacji arytmetycznej: reszta z dzielenia. Zauważmy, że warunek pętli możemy zapisać prościej. Jeśli obydwie liczby mają

być różne od zera, to ich iloczyn też. Analogicznie z wynikiem – jeśli mamy zwrócić niezerową wartość, to wynikiem może być suma liczb, z których jedna jest zerem.

dopóki a * b ≠ 0 **wykonuj**
    jeżeli a > b
        a ← a **mod** b
    w przeciwnym przypadku
        b ← b **mod** a
zwróć a + b

```
1.  def nwd(a, b):
2.      while a * b != 0:
3.          if a > b:
4.              a = a % b
5.          else:
6.              b = b % a
7.      return a + b
```

Sam zapis można jeszcze usprawnić, formułując algorytm bez użycia instrukcji warunkowej. Zakładamy, że pierwsza liczba jest większa. Jeśli nie, to pierwszy obrót pętli dokonuje tylko zamiany wartości zmiennych.

```
1.  def nwd(a, b):
2.      while b != 0:
3.          pom = b
4.          b = a % b
5.          a = pom
6.      return a
```

Pamiętajmy, żeby każdą wersję programu testować, czyli wywoływać funkcję dla różnych danych. Pozwala to poprawić ewentualne błędy.

Podsumowanie

Algorytm Euklidesa został wybrany w tym scenariuszu ze względu na jego znaczenie dla informatyki oraz edukacji, a także z powodu prostoty jego zapisu

z wykorzystaniem pętli warunkowej i instrukcji warunkowej. Może on też stać się punktem wyjścia do dyskusji na temat kosztu czasowego (złożoności obliczeniowej) zmieniającego się w zależności od zastosowanej metody. Podobny scenariusz zajęć, wprowadzający płynne przejście od języka wizualnego do tekstowego, można oczywiście zrealizować na przykładzie innego algorytmu dotyczącego obliczeń na liczbach całkowitych. Jako rozszerzenie możemy zaproponować rozwiązanie jakiegoś zadania z wykorzystaniem algorytmu znajdowania największego wspólnego dzielnika.

Przykład zadania:
Kupujemy spodki i filiżanki. Spodki są pakowane po 8 sztuk, a filiżanki po 6 sztuk. Jaką najmniejszą liczbę opakowań spodków, a jaką filiżanek należy kupić, by po postawieniu wszystkich filiżanek na spodkach nie pozostał ani jeden spodek? W tym zadaniu należy wyznaczyć najmniejszą wspólną wielokrotność dwóch danych liczb – a do tego przyda się umiejętność wyznaczenia największego wspólnego dzielnika.

Scenariusz 2 – *Anagramy*

Opis zajęć

Głównym celem zajęć jest ukazanie zastosowania poznanych wcześniej metod, związanych z przetwarzaniem i porządkowaniem danych. Rozważany problem polega na sprawdzeniu, czy dwa słowa są swoimi anagramami.

Czas trwania

45 minut

Realizacja

Zaczynamy od sformułowania problemu oraz określenia danych i spodziewanego wyniku. Dane są dwa słowa (napisy bez spacji) złożone z małych liter alfabetu łacińskiego. Chcemy sprawdzić, czy są swoimi anagramami. Dane słowo jest anagramem drugiego, jeśli składa się z tych samych liter ułożonych w innej kolejności (jest permutacją jego liter). Na przykład anagramami są słowa: *barok*, *kobra* i *robak*, a także słowa: *markotny* i *romantyk*. Wynikiem jest wartość logiczna *True* (prawda) kiedy dane słowa są anagramami lub wartość *False* (fałsz) – w przeciwnym przypadku. Zastanawiamy się wspólnie ze słuchaczami

(uczniami) nad algorytmem (algorytmami) rozwiązania problemu. Mogą paść różne propozycje. Analizujemy je.

Ponieważ jedno słowo miało powstać z drugiego przez zamianę kolejności liter, możemy próbować przestawiać litery jednego słowa, aby otrzymać drugie. Problem sprowadza się do tworzenia kolejnych słów poprzez przestawianie liter (czyli generowanie kolejnych permutacji) i porównywania z drugim słowem. Takie rozwiązanie wydaje się zawiłe i trudne do realizacji. Może więc można poprzestawiać litery jednego i drugiego słowa, tak aby otrzymać dwa identyczne? A jeśli nie są identyczne, żeby można było stwierdzić, że dane dwa słowa nie są swoimi anagramami. Zauważamy, że możemy litery obydwu wyrazów ustawić alfabetycznie (czyli posortować) i porównać otrzymane słowa.

Poznaliśmy już metodę sortowania przez wybór. Można ją tu zastosować do uporządkowania liter. Wiele języków oferuje gotowe polecenia do porządkowania elementów, także Python. W przedstawionym poniżej rozwiązaniu wykorzystamy tę metodę.

Uwaga: Język Python traktuje napisy (*stringi*) inaczej niż inne języki, w których napis jest traktowany jak tablica znaków. Nie można w prosty sposób zamieniać (przestawiać znaków napisu). Należy zatem zamienić napis na listę znaków.

```
1.  def anagramy(a, b):
2.      a=list(a); a.sort()
3.      b=list(b); b.sort()
4.      return a == b
```

Inny pomysł, jaki może się pojawić w dyskusji, to zliczenie częstotliwości wystąpień poszczególnych liter. Nie musimy później tworzyć posortowanych napisów (czyli sortować przez zliczanie), wystarczy porównać, czy częstości są takie same. Do zliczania wykorzystujemy pomocnicze tablice (listy). Małych liter alfabetu łacińskiego jest 26, więc tablica będzie indeksowana od 0 do 25. Musimy przyporządkować każdej literze liczbę z tego zakresu. Używamy kodów ASCII (ang. *American Standard Code for Information Interchange*) –jeśli na zajęciach nie była wcześniej omówiona reprezentacja znaków, należy to w tym miejscu zrobić. Małe litery alfabetu łacińskiego mają kody zaczynające się od 97, więc wystarczy od kodu litery odjąć 97. Na początku należy utworzyć tablicę

26 liczników i ją wyzerować. Następnie przeglądamy słowo litera po literze i powiększamy właściwy licznik o 1.

Zliczanie częstotliwości liter stanowi osobny problem, ponadto musimy zliczyć częstotliwości dla dwóch słów. Dlatego definiujemy pomocniczą funkcję.

```
1.  def zlicz(a):
2.      licz = [0 for i in range(26)]
3.      for litera in a:
4.          licz[ord(litera) - 97] += 1 #zwiększenie wartości zmiennej o 1
5.      return licz
6.
7.  def anagramy(a, b):
8.      zlicz_a = zlicz(a)
9.      zlicz_b = zlicz(b)
10.     return zlicz_a == zlicz_b
```

Podsumowanie

Na tych zajęciach przetwarzaliśmy napisy, zastosowaliśmy zliczanie i sortowanie do rozwiązania problemu. Operowaliśmy także kodami znaków ASCII. Zastosowaliśmy dwa różne algorytmy rozwiązywania problemu. Mimo że zarówno sam problem, jak i algorytmy jego rozwiązania mogą się wydawać skomplikowane, ich implementacja w języku programowania Python jest niezwykle prosta. Proponowane rozszerzenie tej problematyki dla uczniów zdolnych stanowić może propozycja przygotowania bazy – listy słów, a następnie wyszukania dla danego słowa wszystkich jego anagramów z listy. Bardziej złożonym zadaniem może być wyszukanie wszystkich par słów – anagramów znajdujących się na liście.

PRZYKŁADOWE ZADANIA

Zadanie 1: „Przeliteruj" liczbę

Zadanie polega na wypisaniu cyfr liczby w kolejności od ostatniej (najmniej znaczącej) do pierwszej (najbardziej znaczącej). Np. dla liczby 2018 należy kolejno wypisać: 8, 1, 0 i 2. Zadanie to było już rozwiązywane podczas pierwszej części szkolenia dla klas 4–6 i zapisane w języku Scratch. Warto jednak ponownie je rozwiązać, programując tym razem rozwiązanie w języku Python. Analogicznie jak poprzednio, wykorzystujemy podstawowe działania arytmetyczne, najpierw zapisując algorytm w pseudokodzie.

Dane: a — liczba całkowita dodatnia.
Wynik: cyfry liczby a w kolejności od najmniej znaczącej.

dopóki a > 0 **wykonuj**
    wypisz a **mod** 10
    a = a **div** 10

mod oznacza resztę z dzielenia, a **div** dzielenie całkowite.

```
1.  def wyp_cyf(a):
2.      while a > 0:
3.          print(a % 10)
4.          a = a // 10
```

Porównując rozwiązania w Scratchu i Pythonie, zwracamy uwagę, że w Pythonie mamy do dyspozycji operację dzielenia całkowitego (//).

Zadanie 2: Najmniejsza suma

Wyszukujemy najmniejszą liczbę całkowitą dodatnią, która nie jest sumą liczb znajdujących się w podanym uporządkowanym niemalejąco ciągu liczb naturalnych. Na przykład dla ciągu: [1, 2, 3, 4, 100] wynikiem jest 11, a dla [1, 1, 2, 3, 5, 18, 21] wynikiem jest 13. Przeglądamy kolejne elementy ciągu i rozważamy, które liczby można otrzymać w wyniku ich sumowania. Zauważamy, że jeśli w danej liście początkowym elementem nie jest jedynka, to poszukiwaną liczbą jest 1 (bo na pewno nie da się uzyskać jedynki w wyniku sumowania kolejnych liczb, gdyż są większe niż 1, ze względu na uporządkowanie ciągu).

Natomiast jeżeli jedynka jest początkowym elementem ciągu, to analizujemy dalej. Trzeba zauważyć, że aby w wyniku sumowania udało się uzyskać liczbę 2, drugim elementem ciągu musi być 1 lub 2. W pierwszym z tych przypadków liczbę 2 uzyskujemy, sumując dwie jedynki, a w drugim – po prostu wskazujemy dwójkę.

Po chwili zastanowienia dochodzimy do wniosku, że w wyniku sumowania uda się uzyskać każdą z liczb od 1 do sumy wszystkich przejrzanych wcześniej wyrazów ciągu. Po przejrzeniu pewnej liczby wyrazów mamy zatem dwie możliwości:

1) kolejnym wyrazem ciągu jest liczba równa co najwyżej sumie wcześniejszych wyrazów ciągu, powiększonej o 1 – wówczas kontynuujemy przeglądanie;
2) kolejny wyraz ciągu jest większy o więcej niż 1 od sumy poprzednich (przejrzanych) wyrazów – w tej sytuacji, skoro nie da się uzyskać w wyniku sumowania liczby większej o 1 niż dotychczasowa suma, to poszukiwaną liczbą jest ta suma powiększona o 1.

Zapis w pseudokodzie może być następujący:
Dane: n — liczba całkowita dodatnia,
a — uporządkowany niemalejąco ciąg liczb naturalnych o długości n
Wynik: najmniejsza liczba całkowita dodatnia niebędąca sumą liczb z ciągu a

```
suma = 0
i  = 0
dopóki a[i] <= suma + 1 wykonuj
        suma = suma + a[i]
        i = i + 1
zwróć suma + 1
```

Zastanówmy się, czy powyższy algorytm jest prawidłowy. Czy można podać dane, dla których działa błędnie? Jeśli uczestnicy nie potrafią wskazać błędu, warto zapisać powyższy algorytm w Pythonie i uruchomić – np. dla danych: [1, 2, 3, 4].

Zapomnieliśmy o tym, że ciąg — lista stanowiąca daną – jest skończony.
Jeśli w wyniku analizy kolejnych elementów listy, akceptując je, dojdziemy do jej końca, to podczas wykonywania funkcji pojawi się błąd – próba odwołania do nieistniejącego elementu listy. Musimy poprawić ten błąd. Modyfikujemy warunek kontynuowania pętli tak, by zatrzymała się najpóźniej wówczas, gdy dojdziemy do końca listy.

```
suma = 0
i = 0
dopóki i < n oraz a[i] <=  suma + 1 wykonuj
        suma = suma + a[i]
        i = i + 1
zwróć suma + 1
```

```
1.  def nsc(a):
2.      suma = 0
3.      i = 0
4.      while i < len(a) and a[i] <= suma + 1:
5.          suma = suma + a[i]
6.          i = i + 1
7.      return suma + 1
```

Nowy warunek pętli jest dość skomplikowany. Należy zwrócić uwagę na kolejność podwarunków. Zmiana kolejności spowodowałaby błąd wykonania – nadal próbowalibyśmy odwołać się do elementu tablicy położonego za ostatnim, czyli nieistniejącego. W wielu językach programowania, w tym w Pythonie, warunki logiczne, o ile nie zachodzi taka konieczność, nie są wyliczane „do końca". W tym przypadku – jeśli pierwszy człon koniunkcji jest fałszywy (tj. zanalizowaliśmy już wszystkie elementy listy), wartość drugiego członu nie jest wyliczana, bo i tak całość ma wartość: fałsz.

Możemy uniknąć tego problemu, wykonując pętlę *n* razy i kończąc obliczenia we wnętrzu pętli w momencie, kiedy znamy już szukaną sumę.

```
suma = 0
dla i = 0, ... , n-1 wykonuj
        jeżeli a[i] <= suma + 1
                suma = suma + a[i]
        w przeciwnym przypadku
                zwróć suma + 1
zwróć suma + 1
```

```
1.  def nsc(a):
2.      suma = 0
3.      for i in range(len(a)):
4.          if a[i] <= suma + 1:
5.              suma = suma + a[i]
6.          else:
7.              return suma + 1
8.      return suma + 1
```

Powyższe zadanie stanowi przykład problemu, w którego rozwiązywaniu najistotniejszy okazuje się dobór algorytmu. Nie trzeba badać wszystkich możliwych do uzyskania sum – zresztą byłoby to bardzo czasochłonne. Sam zapis algorytmu jest bardzo krótki i zwarty.

Zadanie 3: Liczba doskonała

Zadanie polega na znalezieniu n–tej liczby doskonałej, dla danego całkowitego n > 0. Liczba doskonała to taka liczba naturalna, której suma dzielników właściwych (tj. mniejszych od niej) jest równa tej liczbie. Pierwszą (najmniejszą) taką liczbą jest 6 (6=1+2+3), drugą 28 (28=1+2+4+7+14). Określenie następnych to zadanie dla komputera, trzeba bowiem wykonać wiele działań.

Algorytm opiera się na liczeniu sumy dzielników właściwych kolejnych liczb („kandydatów" na liczby doskonałe) i zliczaniu znalezionych liczb. Gdy osiągniemy wartość n, wynikiem jest ostatnia analizowana liczba. Zapis algorytmu w pseudokodzie wygląda następująco:

Dane: n — liczba całkowita dodatnia.
Wynik: liczba będąca n-tą liczbą doskonałą.

```
kandydat = 1
licznik  = 0
dopóki licznik < n wykonuj
        kandydat = kandydat + 1
        jeżeli suma_dw(kandydat) = kandydat
              licznik = licznik + 1
zwróć kandydat
```

Projektując algorytm, nie zajmowaliśmy się szczegółami – jak policzyć sumę dzielników właściwych. Jako pomocnicze ćwiczenie warto wykonać zadanie polegające na wypisaniu wszystkich dzielników właściwych. Można zauważyć, że potencjalne dzielniki pochodzą z zakresu od 1 do połowy liczby.

dla d = 1, ... , `liczba` **div** 2 **wykonuj**
    jeżeli `liczba` % d = 0
        wypisz d

```
1.  def wypisz_dw(liczba):
2.      for d in range(1, liczba//2 + 1):
3.          if liczba % d == 0:
4.              print(d)
```

Zamiast wypisywać będziemy sumować dzielniki. Wartością początkową sumy może być wartość 1, a pierwszym potencjalnym dzielnikiem liczba 2.

`suma` = 1
dla d = 2, ... , `liczba` **div** 2 **wykonuj**
    jeżeli `liczba` % d = 0
        `suma` = `suma` + d
zwróć `suma`

```
1.  def suma_dw(liczba):
2.      suma = 1
3.      for d in range(2, liczba//2 + 1):
4.          if liczba % d == 0:
5.              suma = suma + d
6.      return suma
```

Uruchamiamy funkcję. Dla parametrów: 1, 2 i 3 wynik pojawia się szybko, dla 4 – musimy trochę poczekać, a dla 5 – wyliczenia trwają nieznośnie długo. Powinniśmy zatem poprawić efektywność algorytmu. Udoskonalamy działanie funkcji liczącej sumę dzielników właściwych. Zauważamy, że jeśli pewna liczba d jest dzielnikiem n, to jest nim również n/d. Od razu do wyliczanej sumy dzielników dodajemy oba znalezione dzielniki. Widzimy, że nie trzeba sprawdzać wszystkich liczb od 1 do n/2, wystarczy zatrzymać się przy wartości pierwiastka

kwadratowego z n. Dbamy o to, aby pierwiastek z danej liczby, o ile jest całkowity, został uwzględniony w sumie dzielników, ale tylko jednokrotnie.

```
suma = 1
d  = 2
dopóki d * d < liczba wykonuj
        jeżeli liczba % d = 0
                suma = suma + d + liczba div d
        d = d + 1
jeżeli d * d == liczba
        suma = suma + d
zwróć suma
```

```
1.  def suma_dw(liczba):
2.      suma = 1
3.      d = 2
4.      while d * d < liczba:
5.          if liczba % d == 0:
6.              suma = suma + d + liczba // d
7.          d = d + 1
8.      if d * d == liczba:
9.          suma = suma + d
10.     return suma
```

Po dokonaniu poprawek wyliczenia trwają istotnie krócej. Zadanie pozwala doświadczyć, że efektywność przyjętego algorytmu ma praktyczne znaczenie. Wdrażamy uczestników szkolenia do pisania własnych funkcji i podziału rozwiązywanego problemu na podproblemy.

Zadanie 4: Trójkąty

Dany jest co najmniej trzyelementowy zbiór liczb określających długości odcinków. Czy z każdego zestawu trzech odcinków o długościach wziętych z tego zbioru można zbudować trójkąt? Trójkąt da się zbudować, gdy suma dugości dwóch dowolnych odcinków będzie liczbą większą od długości trzeciego.

Naszej uwadze narzuca się algorytm, w którym analizujemy wszystkie możliwe trójki liczb wziętych ze zbioru. Implementacja tego algorytmu wymaga trzech zagnieżdżonych pętli, a przy dużym rozmiarze danych wykonanie programu trwałoby bardzo długo (warto przy okazji policzyć, ile jest możliwych trójek i jak szybko rośnie ich liczność przy zwiększaniu liczby elementów zbioru). Istnieją jednak bardziej efektywne algorytmy. Na przykład: jeśli uporządkujemy elementy w zbiorze od najmniejszego do największego, wystarczy sprawdzić tylko jedną trójkę, składającą się z dwóch pierwszych elementów i elementu ostatniego.

Można także zauważyć, że porządkowanie całego zbioru nie jest niezbędne – wystarczy znaleźć w zbiorze dwa najmniejsze elementy i element największy. Jeśli suma dwóch najmniejszych okaże się większa od największego, to z każdych trzech odcinków o długościach wziętych ze zbioru można zbudować trójkąt.

Poszukiwanie dwóch najmniejszych elementów i elementu największego można przeprowadzić podczas jednokrotnego przejrzenia danej listy. Przyglądamy się dwóm pierwszym elementom listy – mniejszy z nich jest początkowym „kandydatem” na najmniejszą wartość z całej listy, a większy jest początkowym „kandydatem” na drugą najmniejszą wartość z całej listy (tj. najmniejszą wśród pozostałych) oraz na największą wartość z całej listy.

Następnie przeglądamy pozostałe elementy listy i odpowiednio modyfikujemy wartości zmiennych pomocniczych. Po przejrzeniu całej listy zmienne: `min1` i `min2` zawierają wartości dwóch najmniejszych elementów listy, a zmienna `maks` – wartość największego elementu listy. Wystarczy teraz sprawdzić warunek trójkąta dla tej trójki liczb i określić wynik.

```
1.  def trojkat(a):
2.      if a[0] < a[1]:
3.          min1 = a[0]
4.          min2 = a[1]
5.          maks = a[1]
6.      else:
7.          min1 = a[1]
8.          min2 = a[0]
9.          maks = a[0]
10.     for i in range(2, len(a)):
11.         if a[i] < min2:
12.             if a[i] > min1:
13.                 min2 = a[i]
14.             else:
15.                 min2 = min1
16.                 min1 = a[i]
17.         else:
18.             if a[i] > maks:
19.                 maks = a[i]
20.     if min1 + min2 > maks:
21.         return 'tak'
22.     else:
23.         return 'nie'
```

Zadanie na początku wydawało się trudne. Spodziewaliśmy się, że trzeba będzie wykonać wiele obliczeń, aby uzyskać wynik. Okazało się jednak, że wystarczyło przejrzeć jednokrotnie daną listę. Warto zatem poświęcić trochę czasu, aby utworzyć i zapisać optymalny algorytm.

Zdefiniowana funkcja daje wynik będący słowem: „tak” lub „nie”. Jeśli w danej grupie szkoleniowej poziom wiedzy uczestników jest wysoki, wykonując to zadanie, warto wprowadzić pojęcie funkcji o wartości logicznej – wówczas w programie zamiast ostatnich czterech wierszy wystarczy napisać:

```
20.     return min1 + min2 > maks
```

Zadanie 5: Pierwiastek kwadratowy

Dla danej liczby większej od 1 wyznacz jej pierwiastek kwadratowy z dokładnością do 0,01. Do rozwiązania użyj arkusza kalkulacyjnego.

Arkusz dysponuje funkcją, która wylicza pierwiastek kwadratowy zadanej liczby. W rozwiązywanym zadaniu nie chodzi jednak o wykorzystanie tej funkcji. Rozwiązujący zadanie może skorzystać tylko z czterech podstawowych działań arytmetycznych i liczenia wartości bezwzględnej.

Zauważamy, że pierwiastek jest większy od 1, a mniejszy od danej liczby. Staramy się zawęzić zakres naszych poszukiwań (zwiększając lewy kraniec przedziału lub zmniejszając prawy), aż do momentu, w którym długość przedziału stanie się mniejsza niż 0,01 – wówczas na pewno dokładność wyliczenia będzie równa 0,01 (lub wyższa).

W każdym kolejnym kroku wyliczamy środek przedziału (średnią arytmetyczną lewego i prawego krańca) i sprawdzamy, czy po pomnożeniu wartości przez siebie otrzymujemy daną liczbę – a może mniej lub więcej. Na tej podstawie podejmujemy decyzję: jeśli otrzymamy mniej niż wynosi wartość danej liczby – „przesuwamy" lewy kraniec przedziału na dotychczasowy środek, a w przeciwnym przypadku – „przesuwamy" prawy kraniec przedziału na dotychczasowy środek. Wynikiem jest dowolna z liczb z przedziału, dla którego zakończyliśmy obliczenia. Możemy zapisać algorytm w pseudokodzie.

Dane: a — liczba większa od 1.
Wynik: pierwiastek kwadratowy z liczby a, wyznaczony z dokładnością do 0,01.

```
lewy ← 1
prawy ← a
dopóki prawy – lewy > 0,01 wykonuj
        środek ← ( lewy + prawy ) / 2
        jeżeli środek * środek < a
                lewy ← środek
        w przeciwnym przypadku
                prawy ← środek
wypisz lewy
```

Tworzymy arkusz w taki sposób, by pierwszy wiersz zawierał nagłówki (środek przedziału, lewy kraniec, prawy kraniec, długość przedziału). W wierszu nr 2 wpisujemy jedynkę (komórka `B2`) i liczbę, której pierwiastek obliczamy (komórka `C2` zaznaczona na żółto, by użytkownik lepiej widział, gdzie ma wpisać daną). W komórce `D2` wpisujemy formułę `=C2-B2`. Komórce `C2` przypisujemy nazwę: *dana* (jeśli tego nie zrobimy, to dalej dla tej komórki trzeba będzie jawnie używać adresowania bezwzględnego – `C$2`).

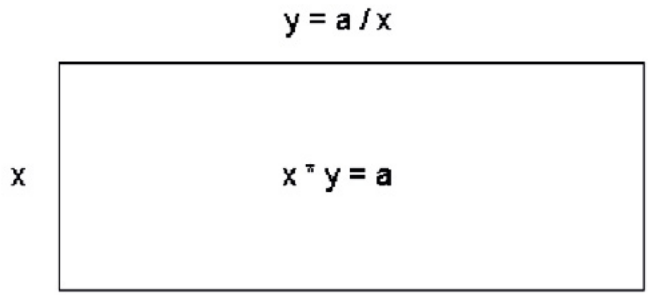
W wierszu nr 3 w pierwszej kolumnie obliczamy środek przedziału na podstawie danych z poprzedniego wiersza (komórka `A3` zawiera `=(B2+C2)/2`), zaś w komórkach `B3` i `C3` obliczamy nowe krańce przedziału, w którym szukamy pierwiastka (formuły odpowiednio `=JEŻELI(A3*A3<dana;A3;B2)` i `=JEŻELI(A3*A3<dana;C2;A3)`). Formuły z komórek `A3`, `B3`, `C3`, `D2` kopiujemy (przeciągamy) w dół o kilkanaście wierszy.

	A	B	C	D
1	środek	lewy	prawy	dł.przedziału
2		1	12	11
3	6,5	1	6,5	5,5
4	3,75	1	3,75	2,75
5	2,375	2,375	3,75	1,375
6	3,0625	3,0625	3,75	0,6875
7	3,40625	3,40625	3,75	0,34375
8	3,578125	3,40625	3,578125	0,171875
9	3,492188	3,40625	3,4921875	0,0859375
10	3,449219	3,4492188	3,4921875	0,0429688
11	3,470703	3,4492188	3,4707031	0,0214844
12	3,459961	3,4599609	3,4707031	0,0107422
13	3,465332	3,4599609	3,465332	0,0053711
14	3,462646	3,4626465	3,465332	0,0026855
15	3,463989	3,4639893	3,465332	0,0013428
16	3,464661	3,4639893	3,4646606	0,0006714
17	3,464325	3,4639893	3,464325	0,0003357

Rysunek 12. Obliczanie wartości pierwiastka kwadratowego metodą bisekcji

Poszukując wyniku, znajdujemy pierwszy wiersz, w którym w kolumnie D znajduje się wartość mniejsza niż 0,01. Dla danej 12 jest to wiersz nr 13. Z tego wiersza odczytujemy zawartość komórki w kolumnie B – to jest wynik. W przykładzie powyżej to: 3,4599609. Równie dobrze wynikiem mogłaby być dowolna liczba z przedziału [3,4599609, 3,465332].

Poniżej rozwiązujemy to samo zadanie innym sposobem. W tej metodzie również zawężamy przedział, w którym znajduje się pierwiastek danej liczby. W kolejnym kroku jako nowe krańce przedziału przyjmiemy środek i iloraz dzielenia danej liczby przez środek. Możemy zinterpretować to graficznie: na początku mamy prostokąt o długościach boków x=1 i y=a, chcemy tak zmieniać długości boków prostokąta, aby otrzymać w przybliżeniu kwadrat o polu a.



Rysunek 13. Obliczanie wartości pierwiastka kwadratowego metodą Newtona-Raphsona

Dane: a — liczba większa od 1
Wynik: pierwiastek kwadratowy z liczby a, wyznaczony z dokładnością do 0,01.

x ← 1
y ← a
dopóki | y − x | > 0,01 **wykonuj**
    x ← (x + y) / 2
    y ← a / x
wypisz x
Realizacja algorytmu w arkuszu może wyglądać następująco:

	A	B	C
1	x	y	dł.przedziału
2	1	12	11
3	6,5	1,8461538	4,6538462
4	4,1730769	2,875576	1,2975009
5	3,5243265	3,4049059	0,1194206
6	3,4646162	3,4635871	0,0010291

Rysunek 14. Obliczanie wartości pierwiastka kwadratowego metodą Newtona-Raphsona

W wierszu nr 2 wpisujemy 1 (komórka A2) i liczbę, której pierwiastek obliczamy (komórka B2 zaznaczona na żółto, by użytkownik lepiej widział, gdzie ma wpisać daną). W komórce C2 wpisujemy formułę =MODUŁ.LICZBY(B2-A2).

Komórce B2 przypisujemy nazwę: *dana* (jeśli tego nie zrobimy, to dalej dla tej komórki trzeba będzie jawnie używać adresowania bezwzględnego – B$2).

W wierszu nr 3, w pierwszej kolumnie obliczamy środek przedziału na podstawie danych z poprzedniego wiersza (komórka A3 zawiera =(A2+B2)/2), zaś w komórce B3 iloraz, o którym mowa wyżej (formuła =dana/A3). Formuły z komórek A3, B3, C2 kopiujemy (przeciągamy) w dół o kilkanaście wierszy. Wyniku szukamy podobnie jak poprzednio. Dla danej 12 jest to na przykład 3,4646162. Widzimy, że druga metoda wymaga mniej powtórzeń, pozwala szybciej uzyskać wynik. Warto przedyskutować te spostrzeżenia z uczestnikami szkolenia.

Udało się nam pokazać praktyczne zastosowanie arkusza kalkulacyjnego do rozwiązania problemu algorytmicznego. Warto także podkreślić, że w pierwszym algorytmie stosowaliśmy metodę algorytmiczną poznaną przy wyszukiwaniu elementu w zbiorze uporządkowanym.

ZAŁĄCZNIK 1
WYMAGANIA WSTĘPNE – KOMPETENCJE TIK

Zaprezentowane poniżej wymagania należy traktować jako wstępne. Oznacza to, że odpowiednie kompetencje TIK powinien posiadać każdy nauczyciel, który chce uczestniczyć w szkoleniu, zarówno dotyczącym poziomu klas 1–3, jak i klas 4–6 oraz 7–8. Nie są to jednak wszystkie wymagania. Pozostałe z nich opisane zostały w każdym z programów szkolenia w punkcie: *Wymagania wstępne stawiane uczestnikom szkolenia*.

Uczestnik szkolenia:

1. Posługuje się sprawnie komputerem w zakresie zarządzania folderami i plikami oraz uruchamiania programów.
2. Umie przeglądać informacje w internecie i wyszukiwać informacje online, sprawnie posługuje się przeglądarką internetową.
3. Umie wyrazić swoje potrzeby informacyjne; umie selekcjonować właściwe informacje pośród wyników wyszukiwania.
4. Umie porównać różne źródła informacji.
5. Wie, jak zapisać pliki i treści (na przykład teksty, zdjęcia, muzykę, pliki wideo i strony internetowe). Wie, jak powrócić do zapisanych plików i treści.
6. Umie korzystać z kilku narzędzi komunikacji elektronicznej, aby kontaktować się z innymi osobami, stosując zaawansowane funkcje tych narzędzi.
7. Umie dzielić się plikami i treściami z innymi osobami za pośrednictwem różnych narzędzi (poczta elektroniczna, przesyłanie załączników; chmura, dyski wirtualne).
8. Umie współpracować z innymi osobami, korzystając z możliwości TIK.
9. Zna zasady netykiety i umie je zastosować we własnych zachowaniach.
10. Umie kształtować własną tożsamość wirtualną i kontrolować swoje ślady w sieci.

11. Umie tworzyć treści cyfrowe w różnych formatach, na różnych platformach i w różnych środowiskach, umie wykorzystać różnorodne narzędzia cyfrowe, aby tworzyć oryginalne treści cyfrowe.
12. Umie edytować, przetwarzać i modyfikować treści stworzone przez siebie lub przez innych.
13. Wie, w jaki sposób różne rodzaje licencji wpływają na informacje i zasoby, których używa i które tworzy.
14. Wie, jak chronić swoje urządzenia cyfrowe (na przykład poprzez instalowanie oprogramowania antywirusowego, stosowanie haseł), rozwija znane sposoby dbania o bezpieczeństwo.
15. Umie zadbać o ochronę swoją i innych osób, rozumie ogólne zasady ochrony danych osobowych, a także jest świadomy, w jaki sposób dane są zbierane i wykorzystywane.
16. Wie, jak chronić siebie i innych przed cyberprzemocą, rozumie ryzyko dla zdrowia wynikające z korzystania z TIK (od ergonomii po uzależnienie od technologii).
17. Umie poprosić o wsparcie techniczne, kiedy narzędzia TIK nie działają zgodnie z oczekiwaniami albo kiedy korzysta z nowych programów, urządzeń lub aplikacji.
18. Umie korzystać w ograniczonym zakresie z TIK przy rozwiązywaniu problemów, umie wybrać narzędzia cyfrowe do wykonywania rutynowych zadań.
19. Wie, że narzędzia TIK mogą być twórczo używane, i potrafi je w ten sposób wykorzystywać w określonym zakresie.
20. Posługuje się w ograniczonym zakresie arkuszem kalkulacyjnym (budowanie formuł, obliczenia, tworzenie wykresów).

ZAŁĄCZNIK 2
WYKAZ ADRESÓW INTERNETOWYCH

Akademia Khana:
https://pl.khanacademy.org/ (informatyka, algorytmy)

Algorytm Euklidesa – aplikacja interaktywna,
http://programowanie.oeiizk.edu.pl (zakładka „Processing")

Algorytm Euklidesa – pokaz:
http://programowanie.oeiizk.edu.pl (zakładka „Processing")

Bank pomysłów koduj.gov:
http://koduj.gov.pl/

Blockly:
https://blockly-demo.appspot.com/static/demos/code/index.html?lang=pl

Blockly Games:
https://blockly-games.appspot.com/?lang=pl

Bloki ScratchJr:
https://www.scratchjr.org/pdfs/blocks.pdf

Code Academy (ang.):
https://www.codecademy.com/learn/learn-python

Dash i Dot:
http://nauczyciele.makewonder.pl/scenariusze-lekcji.html

Edukacja matematyczna:
https://www.digipuzzle.net/minigames/codegrid/codegrid_math_till_ten.htm?language=english&linkback=../../education

Edukacja polonistyczna, plastyczna, społeczna, matematyczna:
http://superkoderzy.pl/scenariusze-lekcji/najmlodsi-programuja (lekcja 1)

Film *Pythonowe początki*:
https://www.youtube.com/watch?v=n-mFQ2JqO8o

„Godzina Kodowania" – instrukcja dla nauczyciela:
http://programowanie.oeiizk.edu.pl/ (zakładka „Godzina kodowania")

Gra CodyRoby:
http://koduj.gov.pl/cody-roby-kodowanie-w-formie-gry-karcianej/

Grafika żółwia:
http://python.oeiizk.edu.pl (zakładka: „Rysowanie z żółwiem")

Gra „Scottie Go!":
https://scottiego.pl/

Gra „Run Marco":
https://www.allcancode.com/runmarco

„Gra w pomarańczę":
https://www.youtube.com/watch?v=WforXEBMm5k

Interactive Python 3 tutorial (ang.):
https://snakify.org/

Jochemczyk W., Olędzka K., (2013), *Python dla wszystkich*, Toruń: Informatyka w Edukacji:
http://python.oeiizk.edu.pl/ (zakładka: „O Pythonie")

Jochemczyk W., Olędzka K., (2016), *Ważenie a system binarny*, Toruń: Informatyka w Edukacji:
http://iwe.mat.umk.pl/archiwum/iwe2016//materials/II_Drukarnia_IwE/08.pdf

Karty pracy Scratcha:
https://scratch.mit.edu/info/cards

Karty pracy w ScratchJr (ang.):
https://www.scratchjr.org/teach/activities

Kodable – gra bezpłatna w wersji podstawowej:
https://game.kodable.com/

„Koduj z Klasą" mate*riały dotyczące Pyth*ona:
http://python101.readthedocs.io/pl/latest/

„Kodujemy kolorowo" – kodowanie graficzne:
http://www.oswajamyprogramowanie.edu.pl/2017/10/kodujemy-kolorowo-offlineowo.html

Konkurs Informatyczny „Bóbr":
https://www.bobr.edu.pl/

Konkurs „Koala":
http://jasijoasia.edu.pl/koala/index.htm

Konkurs „LOGIA":
http://logia.oeiizk.waw.pl

Konkurs „miniLOGIA":
http://minilogia.oeiizk.waw.pl

Kwiatkowska, A.B., (2016), *W poszukiwaniu abstrakcyjnego modelu*, Toruń: Informatyka w Edukacji:
http://iwe.mat.umk.pl/archiwum/iwe2016//materials/II_Drukarnia_IwE/03.pdf

Lekcje programowania – instrukcje WSiP:
https://www.wsip.pl/e-spotkania/lekcje-programowania
https://www.wsip.pl/e-spotkania/lekcje-programowania-czesc-2

Lightbot:
http://lightbot.com/flash.html

mBot:
https://trobot.pl/sklep/makeblock-roboty-edukacyjne-dla-dzieci/makeblock-mbot-v1-1-blue-bluetooth/?gclid=EAIaIQobChMIi6T2k_vw3AIVTM-yCh3Fhge5EAAYASAAEgJmPPD_BwE

Młodzieżowa Akademia Informatyczna:
https://main2.edu.pl/news/

Olimpiada Informatyczna:
https://www.oi.edu.pl/ (zakładki: książeczki, linki)

Olimpiada Informatyczna Gimnazjalistów:
http://oig.edu.pl/ (omówienia zadań)

Ozobot:
https://www.edu-sense.com/pl/

Photon:
https://photonrobot.com/pl/

Portal sphere online judge:
http://pl.spoj.com/ (zadania, konkursy, możliwość wysyłania i sprawdzania zadań)

Portal treningowy Olimpiady Informatycznej:
https://szkopul.edu.pl/ (konkursy, zadania, możliwość wysyłania i sprawdzania zadań)

Projekt SP w Ząbkach:
http://modelnowoczesnejszkoly2017.sp3zabki.pl/ramy-projektu-interdyscyplinarnego-p-t-zakazane-piosenki/

Projekty edukacyjne środowiska Scratch dla klas 1–3:
https://scratch.mit.edu/studios/4487110/

Projekty edukacyjne środowiska Scratch dla klas 4–6:
https://scratch.mit.edu/studios/4487107

Projekt „Computer Science Unplugged” film:
https://www.youtube.com/watch?v=voqghyZbZxo

Projekt „Dla Mamy”:
https://scratch.mit.edu/projects/188249692/

Projekt „Elektrownia wiatrowa”:
https://scratch.mit.edu/projects/85857008

Projekt „Mistrzowie Kodowania” – moduły:
http://programowanie.oeiizk.edu.pl/#!/mistrzowie_kodowania

Projekt „Wiatrak”:
https://scratch.mit.edu/projects/188318193/

Projekt „Śluza wodna”:
https://scratch.mit.edu/projects/85845238/

Środowisko wizualizacji programów w Pythonie:
http://pythontutor.com/

Materiały OEIiZK – „Python w szkole”:
http://python.oeiizk.edu.pl/

Rozmowa z prof. Mitchem Resnickiem z Massachusetts Institute of Technology (MIT) na temat potrzeby kształcenia innowacyjnych i kreatywnych osób:
https://www.youtube.com/embed/sTXa7QUYxaI?rel=0

Samouczek Pythona:
http://www.learnpython.org/pl/

Scenariusze lekcji Super Koderzy:
http://superkoderzy.pl/scenariusze-lekcji/najmlodsi-programuja/

Scenariusze OEIiZK – zakładka „Warszawa programuje”:
http://programowanie.oeiizk.edu.pl/#!/wp_scratch

Scenariusze z projektu „Mistrzowie Kodowania”:
http://wiki.mistrzowiekodowania.pl/index.php?title=Strona_g%C5%82%C3%B3wna

Scenariusze „Informatyka dla Jasia i Joasi”
http://jasijoasia.edu.pl/

Scenariusz ćwiczeń „Pierwsze kroki w Scratchu”:
http://programowanie.oeiizk.edu.pl (zakładka „Mistrzowie Kodowania”)

„Słowniczek bloczków” i opis środowiska Scratch:
http://programowanie.oeiizk.edu.pl/ (zakładka „Warszawa programuje!”)

Sortowanie – film:
https://www.youtube.com/watch?v=cVMKXKoGu_Y

Strona domowa Scratch Junior:
https://www.scratchjr.org/

Strona domowa środowiska Scratch:
https://scratch.mit.edu

Strona internetowa Ośrodka Edukacji Informatycznej i Zastosowań Komputerów:
http://programowanie.oeiizk.edu.pl/, (zakładka „Informatyka /prawie/ bez komputera”)

Strona platformy:
https://code.org/

Strona projektu „Computer Science Unplugged”:
https://csunplugged.org/en/

Strona projektu „Informatyka dla Jasia i Joasi”:
http://jasijoasia.edu.pl

Strona zestawu BeCreo:
http://becreo.eu/index.php/pl/

Sysło M.M., (2016), Wprowadzając... porządek, Toruń: Informatyka w Edukacji:
http://iwe.mat.umk.pl/archiwum/iwe2016//materials/II_Drukarnia_IwE/04.pdf

Szyfry harcerskie:
http://www.szyfry.matw.pl/

Trasa Świętego Mikołaja:
https://santatracker.google.com/village.html

WeDo – oprogramowanie do pobrania:
https://education.lego.com/en-us/downloads/wedo-2/software

WeDo – produkty Lego:
https://www.akcesedukacja.pl/lego-wedo-2-0/

WeDo – Scratch:
https://scratch.mit.edu/wedo

Wystąpienie prof. Mitcha Resnicka pt. *Nauczmy dzieciaki kodować*:
https://www.ted.com/talks/mitch_resnick_let_s_teach_kids_to_code?language=pl#t-535610

Zanurkuj w Pythonie – podręcznik Wikibooks:
https://pl.wikibooks.org/wiki/Zanurkuj_w_Pythonie

Zasoby „Mistrzowie Kodowania":
http://wiki.mistrzowiekodowania.pl