Projekt współfinansowany ze środków
PFRON będących w dyspozycji
Samorządu Województwa Wielkopolskiego

Podręcznik nauczyciela

Algorytmika i programowanie w Pythonie z wykorzystaniem robotów SPIKE Prime

Jakub Piasecki

„Rozwój kompetencji społecznych i cyfrowych dla niewidomych i słabowidzących”
Projekt współfinansowany ze środków PFRON będących w dyspozycji Samorządu Województwa Wielkopolskiego
Stowarzyszenie "Pomóż dziecku niewidomemu" z siedzibą w Owińskach, Plac Przemysława 9

Spis treści

Rozdział 1. Wstęp

1.1. Cel podręcznika

Celem podręcznika jest wsparcie nauczyciela w prowadzeniu zajęć z algorytmiki i programowania w sposób **dostępny, spójny i metodycznie uporządkowany**. Materiał nie ma charakteru instrukcji krok po kroku, lecz stanowi zestaw ram dydaktycznych, scenariuszy oraz wskazówek, które można adaptować do realnych warunków pracy.

Podręcznik zakłada, że nauczyciel ma podstawowe kompetencje informatyczne oraz doświadczenie dydaktyczne. Jego rolą jest porządkowanie procesu uczenia się, a nie zastępowanie autonomii nauczyciela.

1.2. Charakterystyka grupy uczniów

Zajęcia są przeznaczone dla uczniów niewidomych oraz niedowidzących, pracujących w grupach mieszanych. Zakładamy, że uczniowie korzystają z narzędzi wspomagających, takich jak czytniki ekranu (np. NVDA) oraz funkcje powiększania interfejsu.

Konsekwencją tego założenia jest: - ograniczenie komunikacji wizualnej, - nacisk na opis słowny algorytmów i działania programów, - wykorzystywanie terminala jako głównego kanału informacji zwrotnej, - praca w parach sprzyjająca wymianie strategii i wsparciu.

1.3. Założenia organizacyjne zajęć

Zajęcia realizowane są w małych grupach (do 6 uczniów), głównie w parach. Preferujemy dobór par mieszanych (uczeń niewidomy + niedowidzący), co sprzyja współpracy i komunikacji.

Struktura kursu obejmuje lekcje pojedyncze (45 minut) oraz podwójne (2 × 45 minut), szczególnie w blokach projektowych. Zajęcia prowadzone są przy komputerach, z regularnym wykorzystaniem robota jako wykonawcy algorytmu.

1.4. Rola robota w nauczaniu algorytmiki i programowania

Robot SPIKE Prime pełni w kursie rolę **kontekstu motywującego i wykonawcy instrukcji**, a nie celu samego w sobie. Nie skupiamy się na konstrukcji mechanicznej ani komunikacji wizualnej robota.

Robot służy do:

- wykonywania algorytmów zapisanych w Pythonie,
- generowania informacji zwrotnej poprzez dźwięk i ruch,
- wzmacniania rozumienia zależności między kodem a działaniem programu.

1.5. Zgodność z podstawą programową i programem ZPE

Kurs jest zgodny z podstawą programową przedmiotu Informatyka dla liceum ogólnokształcącego (zakres podstawowy) oraz z założeniami programu ZPE. Realizuje cele związane z: projektowaniem algorytmów, programowaniem w języku wysokiego poziomu, analizą i testowaniem programów, współpracą i komunikacją w pracy zespołowej.

Rozdział 2. Standard kodu Python dla uczniów niewidomych

2.1. Dlaczego standard kodu jest kluczowy

Dla uczniów niewidomych i niedowidzących czytelność kodu ma znaczenie fundamentalne. Spójny standard kodu ułatwia: pracę z czytnikiem ekranu, orientację w strukturze programu, analizę błędów oraz współpracę w parach.

Standard kodu wprowadzamy od pierwszych lekcji i konsekwentnie go stosujemy przez cały kurs.

2.2. Struktura pliku programu

Każdy program posiada stałą, rozpoznawalną strukturę:

- importy na początku pliku,
- definicję funkcji main(),
- wywołanie programu w bloku if __name__ == "__main__":.

Taka struktura ułatwia uczniom orientację w kodzie i sprzyja pracy z czytnikiem ekranu.

2.3. Nazewnictwo i czytelność kodu

Stosujemy: proste, jednoznaczne nazwy zmiennych; nazwy w języku angielskim, ale zrozumiałe znaczeniowo; konsekwentne wcięcia i odstępy.

Unikamy skrótów i nazw jednego znaku, o ile nie są one wcześniej ustalone i zrozumiałe.

2.4. Komunikacja z użytkownikiem przez terminal

Terminal jest podstawowym kanałem komunikacji programu z użytkownikiem. Programy:

- informują, co aktualnie się dzieje,
- opisują podejmowane decyzje,
- sygnalizują zakończenie działania.

Komunikaty tekstowe są formułowane jasno i konsekwentnie.

2.5. Praca w parach i zasady współtworzenia kodu

Praca w parach zakłada wspólną odpowiedzialność za kod. Współtworzenie programu obejmuje: - wspólne planowanie algorytmu, - omawianie zmian przed ich wprowadzeniem, - czytanie i komentowanie kodu partnera.

Standard kodu pełni funkcję wspólnego języka, który ułatwia komunikację i współpracę.

Rozdział 3. Lekcja 1 – Algorytm jako opis działania

3.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca na forum klasy oraz praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka + opcjonalnie jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

3.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie algorytm jako uporządkowany opis czynności prowadzących do celu,
- uczeń potrafi zapisać prosty algorytm w postaci sekwencji kroków,
- uczeń dostrzega znaczenie jednoznaczności instrukcji,
- uczeń zapisuje algorytm w języku Python w postaci prostego programu.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.1 – rozumienie problemu i opisywanie sposobu jego rozwiązania,
- II.1 – tworzenie prostych programów jako realizacji algorytmu,
- IV – współpraca w parach i komunikacja w trakcie rozwiązywania problemu.

3.3. Przebieg lekcji

Etap 1. Wprowadzenie – algorytm jako zadanie techniczne

Lekcję rozpoczynamy od przedstawienia prostego zadania technicznego związanego z robotem. Na tym etapie nie korzystamy jeszcze ze sprzętu ani z komputera. Skupiamy się na samym problemie: w jaki sposób przekazać robotowi polecenia, aby wykonał określone zadanie.

Rozmowa prowadzi do wniosku, że aby robot mógł działać poprawnie, potrzebuje dokładnego opisu czynności – krok po kroku. Ten opis nazywamy algorytmem.

Etap 2. Wspólne tworzenie algorytmu w formie tekstowej

Następnie wspólnie z uczniami tworzymy algorytm w postaci listy ponumerowanych kroków. Algorytm zapisywany jest w języku naturalnym i dotyczy wcześniej omówionego zadania technicznego.

Na tym etapie celowo wprowadzamy ograniczenia:

- algorytm składa się wyłącznie z sekwencji kroków,
- nie stosujemy warunków ani powtórzeń,
- każdy krok powinien być możliwie jednoznaczny.

Wspólne formułowanie algorytmu pozwala wychwycić nieprecyzyjne sformułowania i uświadamia uczniom, że nawet drobne niejasności mogą prowadzić do błędów w wykonaniu.

Etap 3. Uczeń w roli robota – wykonawca instrukcji

W kolejnym etapie jeden z uczniów wciela się w rolę robota, czyli wykonawcy zapisanych instrukcji. Zadaniem „robota” jest realizowanie kolejnych kroków algorytmu dokładnie tak, jak zostały zapisane.

Uczeń pełniący tę rolę:

- nie interpretuje poleceń,
- nie poprawia algorytmu,
- zatrzymuje się, jeśli któryś krok jest niejednoznaczny.

Pozostali uczniowie obserwują wykonanie algorytmu i wspólnie analizują, które elementy opisu wymagają doprecyzowania. Ćwiczenie to bardzo wyraźnie pokazuje, że algorytm działa tylko w takim zakresie, w jakim jest jasno i jednoznacznie sformułowany.

Etap 4. Zapis algorytmu w Pythonie i uruchomienie na robocie

Po przetestowaniu algorytmu przechodzimy do jego zapisu w języku Python. Pracując w parach, uczniowie tworzą prosty program uruchamiany bezpośrednio na robocie.

Na tym etapie Python pełni rolę formalnego zapisu algorytmu. Każdy krok algorytmu jest reprezentowany przez osobną instrukcję print(), a program zawiera prostą strukturę zgodną z przyjętym standardem kodu.

Przykładowa forma programu:

```
"""
Lekcja 1 – Algorytm jako opis działania
"""

from pybricks.hubs import PrimeHub
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    print("START: program uruchomiony")
    print("Krok 1: ...")
    print("Krok 2: ...")
    print("Krok 3: ...")
    print("END: program zakonczony")


if __name__ == "__main__":
    main()
```

Program jest uruchamiany na robocie, a komunikaty w terminalu pozwalają uczniom śledzić kolejne kroki algorytmu.

Etap 5. Podsumowanie i refleksja

Lekcję kończymy krótkim podsumowaniem. Wspólnie z uczniami omawiamy, czym był algorytm, dlaczego jednoznaczność opisu okazała się kluczowa oraz jakie elementy programowania pojawią się na kolejnych zajęciach.

Zwracamy uwagę, że w tej lekcji algorytm był wyłącznie sekwencją kroków, a w kolejnych zajęciach zostanie on rozszerzony o nowe konstrukcje.

3.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i opiera się na:

- obserwacji udziału uczniów w tworzeniu algorytmu,
- analizie poprawności i kompletności zapisu algorytmu,
- umiejętności opisania działania programu własnymi słowami.

3.5. Wskazówki metodyczne

W tej lekcji szczególnie istotne jest tempo pracy oraz jasne oddzielenie pojęcia algorytmu od kodu programu. Warto poświęcić wystarczająco dużo czasu na etap wspólnego tworzenia algorytmu i jego „wykonania" przez ucznia w roli robota.

Jeśli grupa pracuje szybciej, można zaproponować drobne modyfikacje algorytmu i sprawdzić ich wpływ na wykonanie. w przypadku trudności dobrze sprawdza się powrót do opisu kroków w języku naturalnym.

Rozdział 4. Lekcja 2 – Program i środowisko pracy

4.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut
Forma pracy: samodzielna praca uczniów oraz praca w parach
Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (wyłącznie kostka inteligentna),
- kabel USB / połączenie do uruchamiania programu na robocie.

4.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie różnicę między algorytmem a programem komputerowym,

- uczeń zna podstawowe elementy środowiska pracy (plik programu, uruchomienie, terminal),
- uczeń potrafi uruchomić prosty program w języku Python na robocie,
- uczeń odczytuje komunikaty w terminalu i interpretuje je jako informację o działaniu programu.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.1 – analizowanie problemu i sposobu jego rozwiązania,
- II.1 – tworzenie i uruchamianie prostych programów,
- III – posługiwanie się urządzeniami cyfrowymi i środowiskiem programistycznym.

4.3. Przebieg lekcji

Etap 1. Wprowadzenie – od algorytmu do programu

Lekcję rozpoczynamy od krótkiego nawiązania do poprzednich zajęć. Przypominamy, że algorytm był uporządkowanym opisem czynności, który robot potrafił wykonać tylko wtedy, gdy był jednoznaczny.

Następnie stawiamy pytanie, co musi się wydarzyć, aby taki opis mógł zostać wykonany samodzielnie przez robota. w ten sposób wprowadzamy pojęcie programu oraz środowiska, w którym program jest tworzony i uruchamiany.

Etap 2. Środowisko pracy – samodzielna eksploracja

Uczniowie rozpoczynają samodzielną pracę na swoich stanowiskach. Zamiast pokazu frontalnego zachęcamy ich do spokojnej eksploracji środowiska pracy: edytora kodu, pliku programu oraz terminala.

Na tym etapie skupiamy się na zrozumieniu ról poszczególnych elementów:

- plik programu jako miejsce zapisu poleceń,
- uruchomienie programu jako moment rozpoczęcia działania,
- terminal jako podstawowe źródło informacji zwrotnej.

Nauczyciel wspiera uczniów komentarzem i pomocą indywidualną, bez szczegółowego omawiania składni języka.

Etap 3. Minimalna struktura programu

W kolejnym kroku uczniowie tworzą bardzo prosty program w języku Python. Jest to wersja celowo uproszczona, która pozwala skupić się na idei programu, a nie na pełnym standardzie kodu.

Program zawiera:

- krótki komentarz opisujący jego cel,
- kilka instrukcji print(),
- wyraźny początek i koniec działania.

Przykładowa forma programu:

```
"""
Lekcja 2 – Program i środowisko pracy
"""

print("START: program uruchomiony")
print("Program dziala")
print("END: program zakonczony")
```

Na tym etapie nie wprowadzamy jeszcze funkcji main() ani rozbudowanej struktury programu. Do tych elementów wrócimy w kolejnych lekcjach.

Etap 4. Robot jako wykonawca – sygnał dźwiękowy

Po zapisaniu programu uczniowie uruchamiają go bezpośrednio na kostce SPIKE Prime. Aby uzyskać jednoznaczną informację zwrotną, zamiast ruchu robota wykorzystujemy dźwięk.

Program zostaje rozszerzony o prostą instrukcję generującą sygnał dźwiękowy, co pozwala uczniom powiązać uruchomienie programu z fizyczną reakcją robota.

Przykładowa modyfikacja programu:

```
from pybricks.hubs import PrimeHub

hub = PrimeHub()

print("START: program uruchomiony")
hub.speaker.beep()
print("DZWIEK: sygnal dzwiekowy")
print("END: program zakonczony")
```

Etap 5. Zaplanowany błąd

W dalszej części lekcji celowo wprowadzamy do programu drobny błąd, na przykład literówkę w nazwie metody lub pominięcie nawiasów. Uczniowie ponownie uruchamiają program i obserwują komunikat błędu w terminalu.

Celem tego etapu nie jest szczegółowe debugowanie, lecz pokazanie, że komunikat o błędzie jest informacją, która pomaga zrozumieć, co w programie nie działa zgodnie z oczekiwaniami.

Etap 6. Podsumowanie i refleksja

Lekcję kończymy krótkim podsumowaniem. Wspólnie z uczniami porządkujemy pojęcia algorytmu, programu i środowiska pracy oraz zwracamy uwagę na rolę terminala jako podstawowego narzędzia komunikacji z programem.

Zapowiadamy, że w kolejnych zajęciach uczniowie będą pracować nad kolejnością wykonywania instrukcji i jej wpływem na działanie programu.

4.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację samodzielności uczniów w pracy ze środowiskiem,
- poprawność uruchomienia programu na robocie,
- umiejętność interpretacji komunikatów wyświetlanych w terminalu.

4.5. Wskazówki metodyczne

W tej lekcji kluczowe znaczenie ma spokojne tempo pracy oraz umożliwienie uczniom samodzielnego oswojenia się ze środowiskiem programistycznym. Warto podkreślać, że nawet bardzo prosty program jest pełnoprawnym programem.

Zaplanowany błąd powinien być drobny i łatwy do zlokalizowania. Jego celem jest obniżenie napięcia związanego z popełnianiem błędów oraz pokazanie, że komunikaty w terminalu są naturalnym elementem pracy programisty.

Rozdział 5. Lekcja 3 – Instrukcje sekwencyjne

5.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: samodzielna praca uczniów

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

5.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie pojęcie sekwencji jako kolejności wykonywania instrukcji,
- uczeń obserwuje wpływ zmiany kolejności poleceń na działanie programu,
- uczeń stosuje instrukcję `wait()` do sterowania czasem działania programu,
- uczeń poznaje podstawową strukturę programu z funkcją `main()`.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.1 – rozumienie problemu i opisywanie sposobu jego rozwiązania,
- II.1 – tworzenie i modyfikowanie prostych programów komputerowych,

- III – posługiwanie się środowiskiem programistycznym i urządzeniami cyfrowymi.

5.3. Przebieg lekcji

Etap 1. Wprowadzenie – kolejność ma znaczenie

Lekcję rozpoczynamy od nawiązania do poprzednich zajęć, podczas których uczniowie uruchamiali proste programy i obserwowali ich działanie. Zwracamy uwagę, że program wykonywał instrukcje dokładnie w takiej kolejności, w jakiej zostały zapisane.

Stawiamy pytanie, co stanie się, jeśli zmienimy kolejność instrukcji w programie, i zapowiadamy, że odpowiedź na to pytanie uczniowie sprawdzą samodzielnie.

Etap 2. Eksperyment z kolejnością instrukcji

Uczniowie pracują samodzielnie, modyfikując prosty program zawierający kilka instrukcji `print()`. Ich zadaniem jest zmiana kolejności linii kodu i ponowne uruchomienie programu.

Celem ćwiczenia jest obserwacja, że komputer nie interpretuje znaczenia poleceń, lecz wykonuje je dokładnie w kolejności zapisu. Kolejność instrukcji bezpośrednio wpływa na kolejność komunikatów pojawiających się w terminalu.

Etap 3. Sekwencja w czasie – instrukcja `wait()`

W kolejnym kroku wprowadzamy instrukcję `wait()`, która pozwala wstrzymać działanie programu na określony czas. Wyjaśniamy, że parametr podawany w tej instrukcji oznacza czas w milisekundach, czyli tysięcznych częściach sekundy.

Uczniowie rozszerzają program, wstawiając krótkie przerwy pomiędzy kolejnymi instrukcjami. Dzięki temu mogą zaobserwować, że program działa krok po kroku nie tylko w przestrzeni, ale również w czasie.

Etap 4. Sekwencjaa robot – dźwięk i ruch

Program zostaje rozszerzony o działania robota. Najpierw wykorzystujemy dźwięk jako prostą informację zwrotną, a następnie dodajemy bardzo krótki ruch jednego silnika.

Działania robota są ułożone w sekwencji, na przykład: dźwięk – przerwa – ruch – przerwa – dźwięk. Uczniowie mogą sprawdzić, jak zmiana kolejności instrukcji wpływa na zachowanie robota.

Przykładowa struktura programu:

```
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    motor = Motor(Port.A)

    print("START")
    hub.speaker.beep()
    wait(500)
    motor.run_time(200, 500)
    wait(500)
    hub.speaker.beep()
    print("END")


if __name__ == "__main__":
    main()
```

Etap 5. Podsumowanie – sekwencja jako fundament programu

Lekcję kończymy krótkim podsumowaniem. Zwracamy uwagę, że każda bardziej złożona konstrukcja programistyczna opiera się na sekwencji instrukcji oraz że zmiana kolejności poleceń prowadzi do zmiany działania programu.

Zapowiadamy, że w kolejnych zajęciach pojawi się pytanie, czy wszystkie instrukcje muszą zawsze zostać wykonane.

5.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację samodzielnej pracy uczniów,
- poprawność modyfikacji kolejności instrukcji,
- umiejętność wyjaśnienia wpływu sekwencji na działanie programu.

5.5. Wskazówki metodyczne

Wprowadzając instrukcję `wait()`, warto poświęcić chwilę na omówienie jednostek czasu i ich znaczenia w praktyce. Dla części uczniów pomocne może być porównanie milisekund do czasu trwania dźwięku lub ruchu silnika.

Funkcja `main()` pojawia się w tej lekcji jako element porządkujący kod. Nie jest konieczne szczegółowe omawianie jej roli technicznej – wystarczy podkreślić, że pomaga ona wyraźnie oddzielić przygotowanie programu od jego uruchomienia.

Rozdział 6. Lekcja 4 – Zmienne jako pamięć programu

6.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca samodzielna z możliwością krótkiej pracy w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

6.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie zmienną jako element programu przechowujący wartość,
- uczeń stosuje zmienne do sterowania czasem i prędkością działania robota,
- uczeń potrafi zmieniać zachowanie programu poprzez modyfikację wartości zmiennych,
- uczeń interpretuje komunikat o użyciu niezdefiniowanej zmiennej jako informację o błędzie.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.1 – opisywanie sposobu rozwiązania problemu i analiza algorytmu,
- II.1 – tworzenie i modyfikowanie prostych programów komputerowych,
- III – posługiwanie się środowiskiem programistycznym i urządzeniami cyfrowymi.

6.3. Przebieg lekcji

Etap 1. Wprowadzenie – co w programie może się zmieniać

Lekcję rozpoczynamy od krótkiego nawiązania do poprzednich zajęć. Zwracamy uwagę, że w programach wykorzystywaliśmy już konkretne liczby, na przykład w instrukcji `wait()` lub przy sterowaniu silnikiem.

Stawiamy pytanie, co można zmienić w programie, aby jego działanie było inne, bez przepisywania całego kodu. w ten sposób wprowadzamy potrzebę stosowania zmiennych.

Etap 2. Pierwsza zmienna – nazwa zamiast liczby

Uczniowie pracują z fragmentem znanego programu, w którym pojawia się instrukcja wait(500). Zamiast bezpośredniego użycia liczby wprowadzamy zmienną przechowującą czas oczekiwania.

Przykładowa modyfikacja kodu:

```
from pybricks.tools import wait

wait_time_ms = 500

wait(wait_time_ms)
```

Zwracamy uwagę, że zmienna przechowuje wartość i że jej nazwa informuje, co ta wartość oznacza. Nie wprowadzamy jeszcze pojęć typów danych ani stałych.

Etap 3. Zmienna wpływa na działanie robota

W kolejnym kroku uczniowie rozszerzają program o sterowanie robotem. Oprócz zmiennej określającej czas wprowadzamy zmienną przechowującą prędkość silnika.

Uczniowie samodzielnie modyfikują wartości zmiennych i obserwują, jak wpływa to na zachowanie robota.

Przykładowa struktura programu:

```
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    motor = Motor(Port.A)

    speed = 200
    wait_time_ms = 500
```

```
    print("START")
    hub.speaker.beep()
    wait(wait_time_ms)
    motor.run_time(speed, wait_time_ms)
    hub.speaker.beep()
    print("END")


if __name__ == "__main__":
    main()
```

Etap 4. Porządkowanie programu za pomocą zmiennych

W dalszej części lekcji uczniowie porządkują program, umieszczając wszystkie zmienne w jednym miejscu na początku funkcji `main()`. Zwracamy uwagę, że taka organizacja ułatwia wprowadzanie zmian i analizę programu.

Na tym etapie dopuszczamy krótką pracę w parach, szczególnie jeśli uczniowie potrzebują wsparcia w zrozumieniu roli poszczególnych zmiennych.

Etap 5. Zaplanowany błąd – niezdefiniowana zmienna

Celowo wprowadzamy do programu błąd polegający na użyciu zmiennej, która nie została wcześniej zdefiniowana. Uczniowie uruchamiają program i obserwują komunikat błędu w terminalu.

Celem tego etapu jest pokazanie, że program nie może korzystać z wartości, których nie zna, oraz że komunikat o błędzie jednoznacznie wskazuje miejsce problemu.

Etap 6. Podsumowanie – zmienna jako pamięć programu

Lekcję kończymy podsumowaniem, w którym porządkujemy pojęcie zmiennej jako elementu programu przechowującego wartość. Zwracamy uwagę, że zmienne pozwalają sterować zachowaniem programu i przygotowują grunt pod podejmowanie decyzji w kolejnych lekcjach.

6.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację samodzielnej pracy uczniów,
- poprawność użycia zmiennych w programie,
- umiejętność wyjaśnienia wpływu zmiennych na działanie robota,
- interpretację komunikatu o niezdefiniowanej zmiennej.

6.5. Wskazówki metodyczne

Wprowadzając zmienne, warto konsekwentnie używać nazw odzwierciedlających znaczenie przechowywanych wartości. Pomaga to uczniom w rozumieniu programu i jego dalszym rozwijaniu.

Zaplanowany błąd z użyciem niezdefiniowanej zmiennej powinien być prosty i jednoznaczny. Jego celem jest oswojenie uczniów z komunikatami o błędach oraz wzmocnienie przekonania, że błędy są naturalnym elementem pracy z programem.

Rozdział 7. Lekcja 5 – Instrukcja warunkowa `if`

7.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

7.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie ideę decyzji w algorytmie,
- uczeń stosuje instrukcję warunkową if i else,
- uczeń potrafi zapisać prosty warunek oparty na zmiennej,
- uczeń obserwuje różne zachowania robota w zależności od spełnienia warunku,
- uczeń rozróżnia błąd składniowy od błędu logicznego.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów zawierających decyzje,
- II.1 – tworzenie programów z wykorzystaniem instrukcji sterujących,
- IV – współpraca w parach przy rozwiązywaniu problemów.

7.3. Przebieg lekcji

Etap 1. Wprowadzenie – czy program zawsze musi robić to samo?

Lekcję rozpoczynamy od nawiązania do pracy ze zmiennymi. Zwracamy uwagę, że choć zmienne pozwalały zmieniać zachowanie programu, to nadal wszystkie instrukcje były wykonywane po kolei.

Stawiamy pytanie, czy program może wykonać tylko część instrukcji – w zależności od konkretnej sytuacji. w ten sposób wprowadzamy potrzebę podejmowania decyzji przez program.

Etap 2. Warunek w języku naturalnym

Zanim przejdziemy do kodu, formułujemy wspólnie proste warunki w języku naturalnym, na przykład: „jeśli czas oczekiwania jest długi, robot wyda dźwięk,
w przeciwnym razie wykona ruch”.

Ograniczamy się do jednego warunku i dwóch możliwych działań. Celem tego etapu jest zrozumienie logiki decyzji bez skupiania się na składni języka Python.

Etap 3. Instrukcja if i else w Pythonie

Następnie zapisujemy warunek w języku Python, wykorzystując znane uczniom zmienne. Wyjaśniamy, że warunek jest pytaniem, na które program odpowiada „tak" albo „nie".

Przykładowa struktura kodu:

```
if wait_time_ms > 500:
    hub.speaker.beep()
else:
    motor.run_time(speed, wait_time_ms)
```

Podkreślamy znaczenie wcięć oraz fakt, że tylko jeden z bloków (if albo else) zostanie wykonany.

Etap 4. Warunek a zachowanie robota

Uczniowie pracując w parach modyfikują wartości zmiennych i obserwują, jak zmienia się zachowanie robota. w zależności od spełnienia warunku robot wydaje dźwięk lub wykonuje krótki ruch.

Dla zachowania dostępności każdą decyzję programu uzupełniamy komunikatem tekstowym w terminalu, informującym, która gałąź instrukcji warunkowej została wykonana.

Etap 5. Zaplanowany błąd – warunek logiczny

W dalszej części lekcji proponujemy warunek, który w praktyce nigdy nie zostanie spełniony, na przykład przez ustawienie progu znacznie wyższego niż używana wartość zmiennej.

Uczniowie obserwują, że program działa poprawnie, ale zawsze wykonuje tę samą gałąź instrukcji warunkowej. Analizujemy wspólnie, dlaczego tak się dzieje i co należałoby zmienić.

Etap 6. Podsumowanie – decyzja w algorytmie

Lekcję kończymy podsumowaniem, w którym porządkujemy pojęcie instrukcji warunkowej. Zwracamy uwagę, że decyzje są kluczowym elementem algorytmów i pozwalają programom reagować na różne sytuacje.

Zapowiadamy, że w kolejnych zajęciach pojawią się bardziej złożone warunki oraz łączenie kilku decyzji.

7.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację współpracy uczniów w parach,
- poprawność zapisu instrukcji if i else,
- umiejętność wyjaśnienia, dlaczego dana gałąź warunku została wykonana,
- rozpoznanie błędu logicznego w warunku.

7.5. Wskazówki metodyczne

Na tym etapie warto zwracać szczególną uwagę na wcięcia w kodzie, ponieważ mają one kluczowe znaczenie dla poprawnego działania instrukcji warunkowej. Nie jest konieczne formalne omawianie zasad składni – wystarczy konsekwentne wskazywanie poprawnych przykładów.

Dobrze sprawdza się porównywanie decyzji programu do prostych sytuacji codziennych, pod warunkiem zachowania jasnego i technicznego języka opisu.

Rozdział 8. Lekcja 6 – Warunki złożone (and, or, not)

8.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

8.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie potrzebę stosowania więcej niż jednego warunku w decyzji algorytmu,
- uczeń stosuje operatory logiczne `and` i `or` w instrukcji warunkowej,
- uczeń rozumie sens operatora `not` i potrafi go rozpoznać w kodzie,
- uczeń potrafi zapisać decyzję w postaci `if` / `elif` / `else`,
- uczeń analizuje zachowanie robota w zależności od spełnienia złożonych warunków.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów zawierających złożone decyzje,
- II.1 – tworzenie programów z wykorzystaniem instrukcji sterujących,
- IV – współpraca w parach i wspólna analiza rozwiązań.

8.3. Przebieg lekcji

Etap 1. Wprowadzenie – kiedy jedna decyzja nie wystarcza

Lekcję rozpoczynamy od nawiązania do poprzednich zajęć z instrukcją `if`. Zwracamy uwagę, że dotychczas decyzja programu była podejmowana na podstawie jednego warunku.

Stawiamy pytanie, co zrobić w sytuacji, gdy decyzja zależy od więcej niż jednej wartości, na przykład czasu i prędkości. w ten sposób wprowadzamy potrzebę łączenia warunków.

Etap 2. Warunki złożone w języku naturalnym

Przed przejściem do kodu uczniowie w parach formułują warunki złożone w języku naturalnym. Pracujemy na znanych już zmiennych, takich jak `wait_time_ms` i `speed`.

Przykłady:

- „robot wyda dźwięk, jeśli czas jest długi **i** prędkość jest duża”,
- „robot poruszy się, jeśli czas jest krótki **lub** prędkość jest mała”,
- „robot nie wykona ruchu, jeśli warunek nie jest spełniony”.

Na tym etapie wprowadzamy pojęcia „i”, „lub” oraz wspominamy o znaczeniu zaprzeczenia („nie”).

Etap 3. Operatory `and`, `or` i `not` w Pythonie

Następnie przechodzimy do zapisu warunków w języku Python. Pokazujemy, w jaki sposób słowa „i” oraz „lub” są zapisywane jako operatory `and` i `or`.

Wspominamy również o operatorze `not`, podkreślając, że służy on do odwracania warunku, jednak na tym etapie nie wymaga samodzielnego stosowania przez uczniów.

Etap 4. Decyzja wielowariantowa – if / elif / else

Program zostaje rozszerzony o więcej niż dwie możliwe reakcje robota. Wprowadzamy konstrukcję if / elif / else, która pozwala sprawdzić kolejne warunki w ustalonej kolejności.

Przykładowa struktura programu:

```
if wait_time_ms > 700 and speed > 300:
    print("DECYZJA: warunek 1")
    hub.speaker.beep()
elif wait_time_ms > 500 or speed > 200:
    print("DECYZJA: warunek 2")
    motor.run_time(speed, wait_time_ms)
else:
    print("DECYZJA: warunek 3")
    hub.speaker.beep()
    hub.speaker.beep()
```

Zwracamy uwagę, że tylko jeden z bloków zostanie wykonany, nawet jeśli kilka warunków byłoby logicznie spełnionych.

Etap 5. Eksperymentowanie i analiza warunków

Uczniowie w parach modyfikują wartości zmiennych i przed uruchomieniem programu przewidują, która gałąź instrukcji warunkowej zostanie wykonana.

Po uruchomieniu programu porównują przewidywania z rzeczywistym zachowaniem robota, analizując, dlaczego dany warunek został spełniony lub pominięty.

Etap 6. Zaplanowany błąd – mylący warunek logiczny

Proponujemy warunek logicznie poprawny, ale niezgodny z intuicją uczniów, na przykład użycie operatora or zamiast and. Uczniowie obserwują, że program działa inaczej, niż się spodziewali.

Analizujemy wspólnie, jak drobna zmiana w warunku logicznym może znacząco wpłynąć na działanie algorytmu.

Etap 7. Podsumowanie – decyzje oparte na wielu warunkach

Lekcję kończymy podsumowaniem, w którym porządkujemy różnice między operatorami `and`, `or` i `not` oraz zwracamy uwagę na rolę kolejności warunków w konstrukcji `if` / `elif` / `else`.

Zapowiadamy, że kolejnym krokiem będzie wielokrotne podejmowanie decyzji przez program, czyli pętle.

8.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację współpracy uczniów w parach,
- poprawność zapisu warunków złożonych,
- umiejętność przewidywania zachowania programu,
- analizę błędów logicznych w warunkach.

8.5. Wskazówki metodyczne

Wprowadzając warunki złożone, warto konsekwentnie odnosić się do języka naturalnego i stopniowo przechodzić do zapisu formalnego. Pomaga to uczniom budować poprawne modele mentalne działania algorytmu.

Operator `not` powinien być traktowany jako element uzupełniający – istotne jest, aby uczniowie rozumieli jego sens, nawet jeśli nie będą go jeszcze samodzielnie stosować w kodzie.

Rozdział 9. Lekcja 7 – Testowanie i analiza działania algorytmu

9.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

9.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie, że program należy sprawdzić przed jego uruchomieniem,
- uczeń potrafi przygotować prosty przypadek testowy,
- uczeń przewiduje działanie programu na podstawie kodu,
- uczeń rozróżnia błąd składniowy od błędu logicznego,
- uczeń analizuje zachowanie robota w odniesieniu do algorytmu.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – analiza algorytmu i przewidywanie jego działania,
- II.1 – testowanie i modyfikowanie programów,
- III – świadome korzystanie z informacji zwrotnej (terminal, robot),
- IV – współpraca w parach przy rozwiązywaniu problemów.

9.3. Przebieg lekcji

Etap 1. Wprowadzenie – czy program zawsze działa poprawnie?

Lekcję rozpoczynamy od krótkiej rozmowy na temat dotychczasowej pracy z programami. Zwracamy uwagę, że program, który się uruchamia, nie zawsze musi działać zgodnie z naszym zamysłem.

Stawiamy pytanie, skąd możemy wiedzieć, co program zrobi **zanim** go uruchomimy. w ten sposób wprowadzamy pojęcie testowania.

Etap 2. Przypadek testowy w języku naturalnym

W parach uczniowie analizują krótki fragment znanego programu (z warunkiem lub pętlą). Zanim uruchomią kod, opisują słownie:

- jakie wartości mają zmienne na początku,
- ile razy wykona się pętla lub która gałąź warunku zostanie wybrana,
- jakie działania wykona robot (dźwięk, ruch).

Ten opis traktujemy jako **przypadek testowy** – przewidywanie działania algorytmu.

Etap 3. Uruchomienie programu i porównanie z przewidywaniem

Uczniowie uruchamiają program i porównują rzeczywiste zachowanie robota z wcześniej przygotowanym opisem.

Każdy etap działania programu jest dodatkowo sygnalizowany komunikatem tekstowym w terminalu, co ułatwia analizę bez konieczności obserwacji wizualnej robota.

Etap 4. Analiza rozbieżności – gdzie jest błąd?

Jeśli zachowanie programu różni się od przewidywanego, uczniowie wspólnie analizują możliwe przyczyny:

- błąd w rozumowaniu (błędny przypadek testowy),
- błąd logiczny w warunku lub pętli,
- błąd składniowy zauważony przez interpreter Pythona.

Nauczyciel na bieżąco moderuje dyskusję, pomagając oddzielić błąd w algorytmie od błędu w zapisie kodu.

Etap 5. Modyfikacja programu i ponowny test

Uczniowie wprowadzają niewielką zmianę w kodzie (np. inną wartość zmiennej lub zmieniony warunek) i ponownie przygotowują krótki przypadek testowy.

Dopiero po przewidzeniu działania programu uruchamiają go ponownie.

Etap 6. Podsumowanie – testowanie jako część algorytmu

Lekcję kończymy podsumowaniem, w którym podkreślamy, że testowanie nie jest dodatkiem do programowania, lecz jego integralną częścią.

Zwracamy uwagę, że świadome przewidywanie działania programu zmniejsza liczbę przypadkowych błędów i ułatwia dalszą pracę z kodem.

9.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację współpracy uczniów w parach,
- poprawność formułowania przypadków testowych,
- umiejętność porównania przewidywanego i rzeczywistego działania programu,
- świadome rozróżnianie błędów logicznych i składniowych.

9.5. Wskazówki metodyczne

W tej lekcji szczególnie ważne jest spowolnienie tempa pracy i konsekwentne wymaganie **przewidywania działania programu przed jego uruchomieniem**.

Nie należy traktować testowania jako osobnego, technicznego etapu. Warto podkreślać, że jest to naturalna forma myślenia algorytmicznego.

Na co uważać poznawczo:

- uczniowie mogą chcieć uruchamiać program bez analizy – warto to świadomie hamować,
- jednoczesna analiza kodu i zachowania robota może być obciążająca,
- w razie trudności należy wrócić do bardzo prostych przypadków testowych (jedna zmienna, jedna decyzja).

Rozdział 10. Lekcja 8 – Pętla `while`

10.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

10.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie potrzebę wielokrotnego wykonywania instrukcji,
- uczeń stosuje pętlę `while` do powtarzania fragmentu programu,
- uczeń kontroluje liczbę powtórzeń za pomocą zmiennej–licznika,

- uczeń rozpoznaje sytuację prowadzącą do pętli nieskończonej,
- uczeń obserwuje powtarzalne działanie robota jako efekt pętli.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów zawierających iterację,
- II.1 – tworzenie programów z wykorzystaniem pętli,
- IV – współpraca w parach przy rozwiązywaniu problemów.

10.3. Przebieg lekcji

Etap 1. Wprowadzenie – kiedy jeden test to za mało

Lekcję rozpoczynamy od nawiązania do testowania programów w poprzednich zajęciach. Zwracamy uwagę, że aby sprawdzić różne przypadki, wielokrotnie uruchamialiśmy program po wprowadzeniu zmian.

Stawiamy pytanie, co zrobić, aby program sam wykonywał te same instrukcje wiele razy, bez konieczności ręcznego powtarzania uruchomienia.

Etap 2. Powtarzanie w języku naturalnym

Zanim przejdziemy do zapisu w Pythonie, opisujemy ideę pętli w języku naturalnym. Formułujemy wspólnie schemat: „dopóki warunek jest spełniony, wykonuj te instrukcje; gdy warunek przestanie być spełniony, zakończ działanie”.

Podkreślamy, że pętla zawsze opiera się na warunku oraz na elemencie, który ten warunek zmienia.

Etap 3. Pierwsza pętla `while` – licznik

Wprowadzamy prostą pętlę `while`, w której liczba powtórzeń kontrolowana jest przez zmienną–licznik. Wyjaśniamy, że warunek pętli jest sprawdzany przed każdym kolejnym wykonaniem jej wnętrza.

Przykładowa struktura kodu:

```
counter = 0

while counter < 3:
    print("PETLA: krok", counter)
    counter = counter + 1
```

Zwracamy uwagę, że jeśli warunek jest fałszywy już na początku, pętla nie wykona się ani razu.

Etap 4. Pętlaa robot – powtarzalne działanie

Uczniowie rozszerzają pętlę o działania robota. w każdej iteracji pętli robot wydaje dźwięk, wykonuje krótki ruch silnika i czeka określony czas.

Dzięki temu uczniowie mogą jednoznacznie powiązać liczbę obiegów pętli z liczbą wykonanych akcji robota.

Przykładowa struktura programu:

```
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    motor = Motor(Port.A)

    counter = 0

    while counter < 3:
        print("PETLA: krok", counter)
        hub.speaker.beep()
        motor.run_time(200, 300)
        wait(300)
        counter = counter + 1
```

```
    print("KONIEC PETLI")


if __name__ == "__main__":
    main()
```

Etap 5. Zaplanowany błąd – pętla nieskończona

Celowo usuwamy lub komentujemy instrukcję zwiększającą wartość licznika. Uczniowie obserwują, że program nie kończy działania, ponieważ warunek pętli nigdy nie przestaje być spełniony.

Zwracamy uwagę na konieczność istnienia mechanizmu zakończenia pętli oraz pokazujemy, jak bezpiecznie przerwać działanie programu.

Etap 6. Podsumowanie – pętla jako narzędzie automatyzacji

Lekcję kończymy podsumowaniem, w którym porządkujemy pojęcie pętli while. Podkreślamy, że pętla łączy w sobie warunek, sekwencję instrukcji oraz zmienną sterującą.

Zapowiadamy, że w kolejnych zajęciach pojawi się wygodniejszy sposób sterowania liczbą powtórzeń.

10.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację współpracy uczniów w parach,
- poprawność zastosowania pętli while,
- umiejętność wyjaśnienia, dlaczego pętla się kończy lub nie kończy,
- świadome reagowanie na sytuację pętli nieskończonej.

10.5. Wskazówki metodyczne

Wprowadzając pętlę while, warto konsekwentnie wracać do języka naturalnego i schematu „dopóki… wykonuj…”. Pomaga to uczniom utrzymać poprawny model mentalny działania pętli.

Szczególną uwagę warto zwrócić na zaplanowany błąd z pętlą nieskończoną. Jest to moment kluczowy dla zrozumienia roli zmiennej sterującej i warunku zakończenia pętli.

Rozdział 11. Lekcja 9 – Pętla for

11.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

11.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie różnicę między pętlą while a pętlą for,
- uczeń stosuje pętlę for do wykonywania instrukcji określoną liczbę razy,
- uczeń korzysta z funkcji range() w prostych przypadkach,
- uczeń obserwuje powtarzalne działanie robota sterowane pętlą for.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów zawierających iterację,

- II.1 – tworzenie programów z wykorzystaniem pętli,
- IV – współpraca w parach przy rozwiązywaniu problemów.

11.3. Przebieg lekcji

Etap 1. Wprowadzenie – czy zawsze potrzebny jest licznik?

Lekcję rozpoczynamy od nawiązania do poprzednich zajęć z pętlą while. Przypominamy, że liczba powtórzeń była kontrolowana ręcznie przez zmienną–licznik.

Stawiamy pytanie, czy w sytuacji, gdy z góry wiemy, ile razy dana czynność ma się wykonać, musimy każdorazowo pilnować licznika. w ten sposób wprowadzamy ideę pętli for.

Etap 2. Od „powtórz N razy" do pętli for

Uczniowie w parach opisują słownie, co należy zrobić, aby wykonać daną czynność, na przykład cztery razy. Wspólnie dochodzimy do wniosku, że w takim przypadku liczba powtórzeń jest znana z góry.

Na tej podstawie wprowadzamy pętlę for jako konstrukcję przeznaczoną do wykonywania instrukcji określoną liczbę razy.

Etap 3. Pierwsza pętla for i funkcja range()

Przechodzimy do zapisu pętli for w języku Python. Zaczynamy od prostego przykładu, wykorzystując funkcję range() w postaci z dwoma parametrami.

Przykładowa struktura kodu:

```
for i in range(1, 4):
    print("PETLA FOR: krok", i)
```

Wyjaśniamy, że zapis range(1, 4) oznacza kolejne wartości od 1 do 3 oraz że liczba 4 nie jest włączona do zakresu.

Etap 4. Pętla fora robot – powtarzalne działanie

Uczniowie zastępują znaną im pętlę while pętlą for i sterują za jej pomocą działaniem robota. w każdej iteracji robot wydaje dźwięk, wykonuje krótki ruch silnika i informuje o numerze kroku w terminalu.

Przykładowa struktura programu:

```
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    motor = Motor(Port.A)

    for i in range(1, 4):
        print("PETLA FOR: krok", i)
        hub.speaker.beep()
        motor.run_time(200, 300)
        wait(300)

    print("KONIEC PETLI FOR")


if __name__ == "__main__":
    main()
```

Etap 5. Zaplanowany problem – zakres range()

W dalszej części lekcji celowo zmieniamy zakres pętli, na przykład ustawiając range(1, 3), i analizujemy, ile razy program się wykona. Uczniowie obserwują, że liczba iteracji jest mniejsza, niż mogli się spodziewać.

Na tej podstawie utrwalamy zasadę, że górna granica zakresu w funkcji range() nie jest włączona do iteracji.

Etap 6. Podsumowanie – kiedy `for`, a kiedy `while`

Lekcję kończymy podsumowaniem, w którym porównujemy pętle `for` i `while`. Zwracamy uwagę, że pętla `for` jest szczególnie wygodna wtedy, gdy liczba powtórzeń jest znana z góry.

Zapowiadamy, że w kolejnych zajęciach pojawi się praca na zbiorach danych oraz bardziej złożone algorytmy.

11.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację współpracy uczniów w parach,
- poprawność zastosowania pętli `for`,
- umiejętność wyjaśnienia różnicy między `for` i `while`,
- świadome korzystanie z funkcji `range()`.

11.5. Wskazówki metodyczne

W tej lekcji warto konsekwentnie podkreślać różnicę w zastosowaniu pętli `for` i `while`, bez wartościowania jednej z nich jako „lepszej”. Kluczowe jest pokazanie, że są to narzędzia do różnych problemów.

Zagadnienie zakresu funkcji `range()` dobrze jest omawiać na przykładach i poprzez eksperymentowanie, zamiast wprowadzania formalnych definicji.

Rozdział 12. Lekcja 10 – Zagnieżdżanie konstrukcji (warunek w pętli)

12.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: analiza indywidualna kodu oraz praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

12.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie, że instrukcje warunkowe mogą być wykonywane wielokrotnie wewnątrz pętli,
- uczeń poprawnie łączy pętlę `for` z instrukcją `if` / `elif` / `else`,
- uczeń czyta i interpretuje strukturę programu na podstawie wcięć,
- uczeń obserwuje różne zachowania robota w kolejnych iteracjach pętli.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów zawierających iterację i decyzje,
- II.1 – tworzenie programów z wykorzystaniem zagnieżdżonych konstrukcji sterujących,
- III – analiza i modyfikacja działania programu.

12.3. Przebieg lekcji

Etap 1. Wprowadzenie – decyzja podejmowana wielokrotnie

Lekcję rozpoczynamy od nawiązania do pracy z pętlą for oraz instrukcją warunkową if. Przypominamy, że pętla pozwala powtarzać instrukcje, a warunek umożliwia podjęcie decyzji.

Stawiamy pytanie, czy decyzja może być podejmowana za każdym razem, gdy pętla wykonuje kolejny krok. w ten sposób wprowadzamy ideę zagnieżdżania konstrukcji.

Etap 2. Zagnieżdżanie w języku naturalnym

Zanim przejdziemy do kodu, opisujemy algorytm w języku naturalnym. Przykładowy schemat brzmi: „powtórz działanie kilka razy; jeśli to pierwszy krok – wykonaj jedno działanie, jeśli ostatni – inne, w pozostałych przypadkach – jeszcze inne".

Uczniowie indywidualnie analizują opis i wskazują, które elementy są powtarzane, a które sprawdzane przy każdym powtórzeniu. Następnie w parach omawiają swoje wnioski.

Etap 3. Instrukcja if wewnątrz pętli for

Przechodzimy do zapisu algorytmu w języku Python. Pokazujemy pętlę for, w której wewnątrz znajduje się instrukcja if / elif / else zależna od wartości zmiennej pętli.

Przykładowa struktura kodu:

```
for i in range(1, 6):
    if i == 1:
        print("PIERWSZY KROK")
    elif i == 5:
        print("OSTATNI KROK")
    else:
        print("SRODEK PETLI")
```

Zwracamy uwagę, że instrukcja warunkowa jest sprawdzana w każdym obiegu pętli oraz że wcięcia jednoznacznie pokazują zależności między instrukcjami.

Etap 4. Zagnieżdżaniea robot – trzy warianty reakcji

Uczniowie rozszerzają program o działania robota. w zależności od wyniku instrukcji warunkowej robot:

- w pierwszym kroku wydaje pojedynczy dźwięk,
- w krokach środkowych wykonuje krótki ruch silnika,
- w ostatnim kroku wydaje dwa dźwięki.

Każda decyzja jest sygnalizowana komunikatem tekstowym w terminalu, co pozwala śledzić działanie algorytmu bez potrzeby obserwacji wizualnej.

Etap 5. Zaplanowany błąd – znaczenie wcięć

W dalszej części lekcji celowo zmieniamy wcięcie instrukcji warunkowej lub jej fragmentu. Uczniowie obserwują, że program działa inaczej, mimo że same instrukcje nie zostały zmienione.

Analizujemy wspólnie, że w języku Python wcięcia są elementem składni i bezpośrednio wpływają na strukturę algorytmu.

Etap 6. Podsumowanie – algorytm warstwowy

Lekcję kończymy podsumowaniem, w którym porządkujemy pojęcie zagnieżdżania konstrukcji. Zwracamy uwagę, że algorytm może mieć strukturę warstwową: pętla steruje powtarzaniem, a warunek – decyzją podejmowaną w każdym kroku.

Zapowiadamy, że kolejnym etapem pracy będzie operowanie na większych zbiorach danych.

12.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację pracy uczniów w parach,
- poprawność zagnieżdżenia instrukcji warunkowej w pętli,
- umiejętność interpretacji struktury programu na podstawie wcięć,
- świadome rozpoznawanie błędów wynikających z niewłaściwego wcięcia.

12.5. Wskazówki metodyczne

Podczas tej lekcji szczególnie istotne jest konsekwentne zwracanie uwagi na strukturę kodu i wcięcia. Dla uczniów korzystających z czytników ekranu warto werbalizować poziomy zagnieżdżenia oraz zachęcać do czytania kodu linia po linii.

Zagnieżdżanie konstrukcji nie powinno być przedstawiane jako nowa, trudna koncepcja, lecz jako naturalne łączenie znanych elementów algorytmu.

Rozdział 13. Lekcja 11 – Listy jako sekwencje danych

13.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: analiza indywidualna kodu oraz praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

13.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie potrzebę przechowywania wielu wartości w jednej strukturze,
- uczeń tworzy prostą listę liczb w języku Python,
- uczeń wykorzystuje pętlę `for` do przetwarzania elementów listy,
- uczeń interpretuje działanie programu opartego na danych.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów operujących na zbiorach danych,
- II.1 – tworzenie programów przetwarzających dane,
- III – analiza działania programu i modyfikacja danych.

13.3. Przebieg lekcji

Etap 1. Wprowadzenie – wiele wartości zamiast jednej

Lekcję rozpoczynamy od przypomnienia, że dotychczas zmienna przechowywała zawsze jedną wartość, a pętla pozwalała wykonywać instrukcje wiele razy. Zwracamy uwagę, że w praktyce często chcemy pracować na całym zestawie danych, a nie na pojedynczej liczbie.

Stawiamy pytanie, jak można w programie zapisać kilka powiązanych ze sobą wartości w jednym miejscu.

Etap 2. Lista w języku naturalnym

Zanim przejdziemy do zapisu w Pythonie, opisujemy pojęcie listy w języku naturalnym. Posługujemy się przykładami zbiorów jednego typu, takich jak zestaw czasów oczekiwania czy kolejnych prędkości ruchu robota.

Uczniowie indywidualnie analizują przykłady, a następnie w parach omawiają różnice między pojedynczą zmienną a sekwencją wartości.

Etap 3. Pierwsza lista w Pythonie

Wprowadzamy składnię listy w najprostszej postaci, bez dodatkowych operacji na danych.

Przykładowy zapis:

times = [300, 500, 700]

Wyjaśniamy, że lista:

- przechowuje wiele wartości,
- zachowuje kolejność elementów,
- w tym etapie zawiera tylko liczby.

Etap 4. Lista i pętla for – przetwarzanie danych

Łączymy listę z pętlą for, wykorzystując znany już mechanizm iteracji. Program przechodzi kolejno przez wszystkie elementy listy i wykonuje dla nich te same instrukcje.

Przykładowa struktura programu:

```
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    motor = Motor(Port.A)

    times = [300, 500, 700]

    for t in times:
        print("Czas:", t)
        hub.speaker.beep()
        motor.run_time(200, t)
        wait(t)
```

```
    print("KONIEC PROGRAMU")


if __name__ == "__main__":
    main()
```

Każdy obieg pętli jest sygnalizowany komunikatem tekstowym w terminalu, co pozwala śledzić działanie programu bez obserwacji wizualnej robota.

Etap 5. Zaplanowany przypadek – pusta lista

W dalszej części lekcji tworzymy pustą listę i uruchamiamy program ponownie. Uczniowie obserwują, że pętla `for` nie wykonuje żadnej iteracji, ale program kończy się poprawnie.

Analizujemy, że brak elementów w liście nie jest błędem, lecz informacją, którą algorytm musi uwzględnić.

Etap 6. Podsumowanie – lista jako dane dla algorytmu

Lekcję kończymy podsumowaniem, w którym porządkujemy pojęcie listy jako sekwencji danych. Zwracamy uwagę, że algorytm może być opisany nie tylko przez instrukcje sterujące, ale również przez dane, na których operuje.

Zapowiadamy, że w kolejnych zajęciach pojawi się przetwarzanie danych z wykorzystaniem warunków.

13.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację pracy uczniów w parach,
- poprawność tworzenia listy i iteracji po jej elementach,
- umiejętność wyjaśnienia, co dzieje się przy pustej liście,

- świadome korzystanie z komunikatów w terminalu.

13.5. Wskazówki metodyczne

Podczas wprowadzania list warto konsekwentnie unikać pojęć indeksu i modyfikowania struktury danych. Skupienie się na liście jako źródle danych pozwala utrzymać niski poziom obciążenia poznawczego.

Zalecane jest częste łączenie listy z pętlą `for`, aby utrwalić naturalny sposób przetwarzania sekwencji wartości.

Na co uważać poznawczo

- W tej lekcji pojawia się nowy byt pojęciowy (lista), dlatego warto ograniczyć liczbę jednoczesnych nowości.
- Uczniowie powinni najpierw zrozumieć sens listy jako całości, zanim zaczniemy mówić o elementach pojedynczych.
- Dla uczniów korzystających z czytnika ekranu istotne jest głośne werbalizowanie kolejnych iteracji pętli.
- Jeśli pojawiają się trudności, można wrócić do bardzo krótkiej listy (np. dwóch elementów).

Rozdział 14. Lekcja 12 – Przetwarzanie danych: lista + warunek

14.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut
Forma pracy: praca w parach (z dopuszczeniem krótkiej analizy indywidualnej kodu)
Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

14.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń przetwarza elementy listy w pętli `for`,
- uczeń stosuje instrukcję warunkową do podejmowania decyzji dla każdego elementu listy,
- uczeń rozumie, że dane sterują zachowaniem algorytmu,
- uczeń analizuje działanie programu na podstawie komunikatów w terminalu.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów operujących na danych i zawierających decyzje,
- II.1 – tworzenie programów przetwarzających dane,
- III – analiza i modyfikacja działania programu.

14.3. Przebieg lekcji

Etap 1. Wprowadzenie – dane jako źródło decyzji

Lekcję rozpoczynamy od nawiązania do poprzednich zajęć z listami. Przypominamy, że lista pozwala przechowywać wiele wartości, a pętla `for` umożliwia ich kolejne przetwarzanie.

Stawiamy pytanie, czy program może **podejmować decyzję osobno dla każdej wartości** znajdującej się w liście. w ten sposób wprowadzamy ideę łączenia danych z instrukcją warunkową.

Etap 2. Algorytm w języku naturalnym

W parach opisujemy algorytm słownie:

- przejdź po wszystkich elementach listy,

- dla każdego elementu sprawdź warunek,
- w zależności od wyniku wykonaj jedno z działań.

Podkreślamy, że struktura algorytmu jest stała, a zmieniają się jedynie dane.

Etap 3. Lista + pętla for + instrukcja if

Przechodzimy do zapisu algorytmu w języku Python, łącząc znane już elementy: listę, pętlę for oraz instrukcję if / else.

Przykładowa struktura programu:

```
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    motor = Motor(Port.A)

    times = [200, 600, 400, 800]

    for t in times:
        print("ELEMENT LISTY:", t)
        if t > 500:
            print("DECYZJA: dlugi czas – dzwiek")
            hub.speaker.beep()
        else:
            print("DECYZJA: krotki czas – ruch")
            motor.run_time(200, t)
        wait(300)

    print("KONIEC PROGRAMU")


if __name__ == "__main__":
    main()
```

Każda decyzja jest opisana komunikatem tekstowym w terminalu, co umożliwia śledzenie działania algorytmu bez obserwacji wizualnej robota.

Etap 4. Eksperymentowanie z danymi

Uczniowie modyfikują wyłącznie wartości w liście, nie zmieniając struktury programu. Przed uruchomieniem przewidują, które elementy spełnią warunek i jakie działania zostaną wykonane.

Zwracamy uwagę, że **zmiana danych** prowadzi do zmiany zachowania programu bez modyfikacji algorytmu.

Etap 5. Zaplanowany przypadek – wartość graniczna

Do listy wprowadzamy wartość równą progowi warunku (np. 500). Analizujemy wspólnie, która gałąź instrukcji warunkowej zostanie wykonana.

Rozmawiamy o precyzji warunków i o tym, że nawet niewielka zmiana operatora porównania wpływa na działanie programu.

Etap 6. Podsumowanie – algorytm sterowany danymi

Lekcję kończymy podsumowaniem, w którym porządkujemy schemat: *lista* → *pętla* → *warunek* → *działanie*. Podkreślamy, że algorytm może mieć stałą strukturę, a jego zachowanie zależy od danych wejściowych.

Zapowiadamy, że w kolejnych zajęciach pojawi się bardziej złożona praca na danych.

14.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację pracy uczniów w parach,

- poprawność połączenia listy, pętli i instrukcji warunkowej,
- umiejętność przewidywania działania programu na podstawie danych,
- świadome korzystanie z komunikatów w terminalu.

14.5. Wskazówki metodyczne

W tej lekcji kluczowe jest utrzymanie jasnego schematu przetwarzania danych. Warto wielokrotnie wracać do pytania: *co w tym programie jest stałe, a co się zmienia*.

Nie wprowadzamy jeszcze indeksów ani modyfikowania list. Skupienie się na odczycie danych i reagowaniu na nie pozwala ograniczyć obciążenie poznawcze.

Na co uważać poznawczo:

- uczniowie mogą chcieć jednocześnie zmieniać dane i kod – warto te działania rozdzielać,
- analiza działania programu powinna zawsze poprzedzać jego uruchomienie,
- w razie trudności należy ograniczyć listę do dwóch lub trzech elementów.

Rozdział 15. Lekcja 13 – Decyzje wielowariantowe na danych

15.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

15.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń stosuje konstrukcję if / elif / else do podejmowania decyzji wielowariantowych,
- uczeń analizuje dane z listy i przypisuje im różne reakcje programu,
- uczeń rozumie znaczenie kolejności sprawdzania warunków,
- uczeń przewiduje działanie algorytmu przed jego uruchomieniem.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów zawierających złożone decyzje,
- II.1 – tworzenie programów przetwarzających dane,
- III – analiza i modyfikacja działania programu.

15.3. Przebieg lekcji

Etap 1. Wprowadzenie – jedna decyzja to czasem za mało

Lekcję rozpoczynamy od nawiązania do poprzednich zajęć, w których algorytm podejmował decyzję pomiędzy dwoma wariantami działania. Zwracamy uwagę, że w praktyce często potrzebujemy więcej niż dwóch możliwych reakcji.

Stawiamy pytanie, jak zapisać algorytm, który dla różnych danych wykona **różne działania**, nie ograniczając się do prostego „jeśli – w przeciwnym razie".

Etap 2. Algorytm wielowariantowy w języku naturalnym

W parach opisujemy algorytm słownie:

- przejdź po wszystkich elementach listy,
- jeśli wartość jest bardzo duża – wykonaj działanie A,
- jeśli wartość jest średnia – wykonaj działanie B,
- w przeciwnym razie – wykonaj działanie C.

Zwracamy uwagę, że warunki muszą być sprawdzane w określonej kolejności.

Etap 3. Konstrukcja if / elif / else w Pythonie

Przechodzimy do zapisu algorytmu w języku Python. Wprowadzamy konstrukcję if / elif / else jako naturalne rozszerzenie znanej instrukcji if.

Przykładowa struktura programu:

```
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    motor = Motor(Port.A)

    values = [200, 450, 700, 900]

    for v in values:
        print("ELEMENT:", v)
        if v > 800:
            print("DECYZJA: bardzo duza wartosc – dwa dzwieki")
            hub.speaker.beep()
            hub.speaker.beep()
        elif v > 500:
            print("DECYZJA: srednia wartosc – ruch")
            motor.run_time(200, v)
        else:
            print("DECYZJA: mala wartosc – jeden dzwiek")
            hub.speaker.beep()
        wait(300)

    print("KONIEC PROGRAMU")


if __name__ == "__main__":
    main()
```

Każdy wariant działania jest jednoznacznie opisany komunikatem tekstowym w terminalu.

Etap 4. Kolejność warunków – analiza i eksperyment

Uczniowie analizują kod przed uruchomieniem programu i przewidują, która gałąź instrukcji zostanie wykonana dla poszczególnych wartości.

Następnie eksperymentują ze zmianą kolejności warunków `if` i `elif`, obserwując, jak wpływa to na zachowanie algorytmu.

Etap 5. Zaplanowany problem – nakładające się warunki

Celowo proponujemy warunki, które częściowo się pokrywają (np. `v > 500` przed `v > 800`). Uczniowie obserwują, że algorytm zawsze wybiera pierwszy pasujący warunek.

Na tej podstawie podkreślamy znaczenie kolejności sprawdzania warunków w konstrukcji `if / elif / else`.

Etap 6. Podsumowanie – decyzja jako sekwencja pytań

Lekcję kończymy podsumowaniem, w którym porządkujemy rozumienie decyzji wielowariantowej jako **sekwencji pytań zadawanych danym**.

Zapowiadamy, że w kolejnych zajęciach decyzje te będą wykonywane wielokrotnie w bardziej złożonych algorytmach.

15.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację pracy uczniów w parach,
- poprawność zapisu konstrukcji `if / elif / else`,

- umiejętność przewidywania działania algorytmu,
- świadome analizowanie kolejności warunków.

15.5. Wskazówki metodyczne

W tej lekcji warto wielokrotnie wracać do idei, że algorytm **nie wybiera najlepszego warunku**, lecz pierwszy spełniony. Pomaga to uniknąć błędów logicznych w dalszej pracy.

Nie należy rozszerzać liczby wariantów ponad trzy – celem lekcji jest zrozumienie mechanizmu, a nie rozbudowa funkcjonalności.

Na co uważać poznawczo:

- uczniowie mogą traktować elif jako osobny warunek niezależny od if,
- analiza kilku wariantów jednocześnie może być obciążająca,
- w razie trudności warto wrócić do zapisu algorytmu w języku naturalnym.

Rozdział 16. Lekcja 14 – Algorytmy iteracyjne z danymi

16.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut

Forma pracy: praca w parach

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- kabel USB / połączenie do uruchamiania programu na robocie.

16.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie algorytm iteracyjny jako powtarzalne przetwarzanie danych,
- uczeń łączy pętlę `for` z decyzjami wielowariantowymi,
- uczeń analizuje działanie algorytmu krok po kroku,
- uczeń przewiduje zachowanie programu na podstawie danych wejściowych.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – projektowanie algorytmów iteracyjnych operujących na danych,
- II.1 – tworzenie programów przetwarzających dane i zawierających decyzje,
- III – analiza i modyfikacja działania programu.

16.3. Przebieg lekcji

Etap 1. Wprowadzenie – jeden schemat, wiele danych

Lekcję rozpoczynamy od przypomnienia schematu z poprzednich zajęć: lista → pętla → warunek → działanie. Zwracamy uwagę, że w tym schemacie algorytm wykonuje **ten sam zestaw kroków** dla kolejnych danych.

Stawiamy pytanie, jak opisać algorytm, który przetwarza dane seriami, a nie pojedynczym przypadkiem.

Etap 2. Algorytm iteracyjny w języku naturalnym

W parach zapisujemy algorytm słownie:

- weź listę danych,
- dla każdego elementu wykonaj te same kroki,
- w zależności od wartości podejmij decyzję,
- powtórz algorytm aż do przetworzenia wszystkich danych.

Podkreślamy, że struktura algorytmu się nie zmienia – zmieniają się tylko dane.

Etap 3. Pętla for + decyzja wielowariantowa

Przechodzimy do zapisu algorytmu w języku Python, łącząc elementy znane z poprzednich lekcji: listę, pętlę for oraz konstrukcję if / elif / else.

Przykładowa struktura programu:

```
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait


def main() -> None:
    hub = PrimeHub()
    motor = Motor(Port.A)

    values = [150, 350, 550, 750]

    for v in values:
        print("PRZETWARZAM:", v)
        if v > 600:
            print("WYNIK: bardzo duza wartosc")
            hub.speaker.beep()
            hub.speaker.beep()
        elif v > 300:
            print("WYNIK: srednia wartosc")
            motor.run_time(200, v)
        else:
            print("WYNIK: mala wartosc")
            hub.speaker.beep()
        wait(300)

    print("KONIEC ALGORYTMU")


if __name__ == "__main__":
    main()
```

Każdy krok algorytmu jest sygnalizowany komunikatem tekstowym w terminalu, co umożliwia analizę przebiegu bez potrzeby obserwacji wizualnej robota.

Etap 4. Analiza przebiegu algorytmu

Uczniowie przed uruchomieniem programu analizują kod i przewidują:

- ile razy wykona się pętla,
- które warianty decyzji zostaną wybrane,
- jaka będzie kolejność reakcji robota.

Po uruchomieniu programu porównują przewidywania z rzeczywistym przebiegiem algorytmu.

Etap 5. Zaplanowany problem – zmiana danych, ten sam algorytm

Uczniowie modyfikują wyłącznie zawartość listy `values`, nie zmieniając struktury programu. Analizują, jak zmiana danych wpływa na liczbę i kolejność decyzji.

Zwracamy uwagę, że algorytm pozostaje taki sam, a jego zachowanie zależy wyłącznie od danych wejściowych.

Etap 6. Podsumowanie – algorytm jako proces

Lekcję kończymy podsumowaniem, w którym porządkujemy pojęcie algorytmu iteracyjnego jako procesu przetwarzającego dane krok po kroku.

Zapowiadamy, że kolejne zajęcia będą przygotowywać uczniów do samodzielnej pracy projektowej.

16.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację pracy uczniów w parach,
- poprawność połączenia pętli i decyzji wielowariantowych,

- umiejętność analizy przebiegu algorytmu,
- świadome przewidywanie działania programu.

16.5. Wskazówki metodyczne

W tej lekcji warto traktować cały program jako **jeden algorytm**, a nie zbiór niezależnych instrukcji. Pomaga to uczniom budować myślenie procesowe.

Nie należy wprowadzać nowych konstrukcji składniowych – celem jest utrwalenie znanych elementów w nowej konfiguracji.

Na co uważać poznawczo:

- uczniowie mogą koncentrować się na pojedynczym przypadku zamiast na całym procesie,
- jednoczesna analiza pętli i decyzji może być obciążająca,
- w razie trudności warto wrócić do zapisu algorytmu w języku naturalnym.

Rozdział 17. Lekcja 15 – Przygotowanie do pracy projektowej

17.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 45 minut
Forma pracy: praca w parach oraz praca na forum klasy
Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik),
- tablica lub edytor tekstu dostępny dla uczniów (opis algorytmu w języku naturalnym).

17.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozumie różnicę między zadaniem ćwiczeniowyma projektem programistycznym,
- uczeń potrafi zaplanować algorytm w postaci wersji minimalnej (MVP),
- uczeń świadomie upraszcza rozwiązanie przed jego rozbudową,
- uczeń przygotowuje algorytm do implementacji i testowania.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.1 – analiza problemu i planowanie rozwiązania,
- I.2 – projektowanie algorytmów,
- II.1 – przygotowanie programu do realizacji,
- IV – współpraca w parach i komunikacja w zespole.

17.3. Przebieg lekcji

Etap 1. Wprowadzenie – czym projekt różni się od zadania

Lekcję rozpoczynamy od rozmowy podsumowującej dotychczasową pracę. Zwracamy uwagę, że wcześniejsze lekcje polegały na realizacji jasno określonych poleceń z jednym poprawnym rozwiązaniem.

Wprowadzamy pojęcie projektu jako zadania:

- bardziej otwartego,
- realizowanego etapami,
- dopuszczającego różne poprawne rozwiązania.

Podkreślamy, że w projekcie ważny jest **proces dochodzenia do rozwiązania**, a nie tylko efekt końcowy.

Etap 2. Wersja minimalna algorytmu (MVP)

Wspólnie z uczniami omawiamy ideę wersji minimalnej programu – takiej, która:

- działa,
- realizuje podstawowy cel,
- jest możliwie prosta.

Na przykładzie znanego już schematu (lista → pętla → decyzja → działanie) pokazujemy, jak ograniczyć liczbę danych i wariantów, aby uzyskać działający program.

Etap 3. Planowanie projektu w języku naturalnym

Uczniowie w parach opisują algorytm planowanego projektu w języku naturalnym. Opis powinien zawierać:

- jakie dane będą przetwarzane,
- jakie decyzje będą podejmowane,
- jakie reakcje robota są planowane.

Na tym etapie nie zapisujemy jeszcze kodu – skupiamy się na **logice działania programu**.

Etap 4. Analiza wykonalności i uproszczenia

Każda para krótko omawia swój pomysł. Wspólnie zastanawiamy się:

- które elementy są konieczne,
- które można pominąć lub odłożyć na później,
- jak uprościć algorytm, zachowując jego sens.

Podkreślamy, że uproszczenie algorytmu jest świadomą decyzją projektową, a nie porażką.

Etap 5. Przygotowanie do implementacji

Na zakończenie lekcji uczniowie porządkują plan projektu:

- zapisują ostateczną wersję algorytmu w języku naturalnym,
- wskazują, które znane konstrukcje programistyczne zostaną użyte,
- przygotowują się do zapisania kodu w kolejnych zajęciach.

Etap 6. Podsumowanie – projekt jako proces

Lekcję kończymy podsumowaniem, w którym akcentujemy, że projekt to proces składający się z planowania, implementacji, testowania i modyfikacji.

Zapowiadamy, że kolejne zajęcia będą poświęcone realizacji mini-projektów.

17.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację pracy uczniów w parach,
- poprawność i kompletność opisu algorytmu w języku naturalnym,
- umiejętność świadomego upraszczania rozwiązania,
- udział w dyskusji nad wykonalnością projektów.

17.5. Wskazówki metodyczne

W tej lekcji szczególnie ważne jest, aby nie przyspieszać przejścia do kodowania. Czas poświęcony na planowanie znacząco ułatwia późniejszą pracę projektową.

Warto konsekwentnie wracać do pojęcia wersji minimalnej i przypominać, że każdy projekt można rozwijać etapami.

Na co uważać poznawczo:

- uczniowie mogą chcieć od razu tworzyć rozbudowane rozwiązania,

- planowanie bez kodu może być dla części uczniów trudniejsze niż samo programowanie,
- w razie trudności warto ograniczyć projekt do jednego typu danych i jednej decyzji.

17.6. Pomysły na projekty

Poniżej znajduje się kilka ogólnych pomysłów na projekty, na wypadek kryzysu kreatywności uczniowskiej. Jeżeli uczniowie mają trudność z wyborem tematu:

- warto narzucić jeden temat dla całej grupy,
- pozwolić na indywidualne warianty w danych lub reakcjach,
- przypominać, że celem jest działająca wersja minimalna, nie oryginalność.

1. Robot – wykonawca sekwencji poleceń

Opis:
Robot wykonuje zaprogramowaną sekwencję działań (dźwięk, ruch), sterowaną listą danych.

Wersja minimalna:

- lista liczb,
- pętla `for`,
- jedna decyzja (`if / else`),
- reakcja robota + komunikat w terminalu.

Możliwe rozszerzenia:

- różne reakcje dla różnych zakresów wartości,
- zmiana danych bez zmiany algorytmu.

---

2. Robot – reaguje na dane

Opis:

Robot analizuje kolejne dane i dla każdego podejmuje decyzję o sposobie działania.

Wersja minimalna:

- lista danych,
- `if / elif / else`,
- co najmniej dwa warianty reakcji.

Możliwe rozszerzenia:

- trzeci wariant reakcji,
- komunikaty diagnostyczne w terminalu.

---

3. Robot – sygnalizator zdarzeń

Opis:

Robot pełni rolę sygnalizatora, który informuje o stanie systemu za pomocą dźwięku i ruchu.

Wersja minimalna:

- dane opisujące „stan”,
- warunek decydujący o reakcji,
- jednoznaczne komunikaty tekstowe.

Możliwe rozszerzenia:

- zmiana progów decyzyjnych,
- różne sekwencje sygnałów.

---

4. Robot – wykonawca harmonogramu

Opis:

Robot realizuje prosty harmonogram działań zapisany w liście.

Wersja minimalna:

- lista czasów lub poleceń,
- pętla `for`,
- `wait()` między działaniami.

Możliwe rozszerzenia:

- różne typy działań w harmonogramie,
- modyfikacja kolejności lub liczby elementów.

---

5. Robot – tester algorytmu

Opis:

Robot służy do sprawdzania poprawności działania algorytmu poprzez przewidywalne reakcje.

Wersja minimalna:

- zaplanowane przypadki testowe,
- komunikaty w terminalu,
- jedna decyzja algorytmiczna.

Możliwe rozszerzenia:

- wartości graniczne,
- analiza błędów logicznych.

---

6. Robot – interpretator poleceń

Opis:

Robot interpretuje uproszczone polecenia zapisane w liście (np. liczby oznaczające akcje).

Wersja minimalna:

- lista „poleceń”,
- `if / elif / else`,
- różne reakcje robota.

Możliwe rozszerzenia:

- więcej typów poleceń,
- komunikaty diagnostyczne.

---

7. Robot – symulator procesu

Opis:

Robot symuluje prosty proces (np. kolejne etapy), sygnalizując je dźwiękiem i ruchem.

Wersja minimalna:

- lista etapów,
- iteracja,
- różne reakcje dla etapów.

Możliwe rozszerzenia:

- zmiana przebiegu procesu przez dane,
- dodanie warunków zatrzymania.

Rozdział 18. Lekcja 16 – Mini-projekt: wersja minimalna

18.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 2 × 45 minut

Forma pracy: praca w parach (z możliwością łączenia par w małe zespoły projektowe)

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik; czujniki opcjonalnie),
- notatki z planem algorytmu przygotowanym w lekcji 15.

18.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń realizuje zaplanowany algorytm w postaci działającego programu,
- uczeń świadomie ogranicza zakres projektu do wersji minimalnej (MVP),
- uczeń testuje działanie programu na robocie,
- uczeń modyfikuje program na podstawie obserwacji jego działania,
- uczeń dokumentuje postęp pracy projektowej.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.1 – analiza problemu i realizacja zaplanowanego rozwiązania,
- I.2 – implementacja algorytmu,
- II.1 – tworzenie, testowanie i modyfikowanie programów,
- IV – współpraca w parach i zespołach projektowych.

18.3. Przebieg lekcji

Etap 1. Przypomnienie założeń projektu

Lekcję rozpoczynamy od krótkiego przypomnienia, że celem jest realizacja **wersji minimalnej projektu**, a nie pełnej, rozbudowanej funkcjonalności.

Wspólnie porządkujemy kryteria wersji minimalnej:

- program się uruchamia,
- wykonuje zaplanowany algorytm,
- reaguje na dane w przewidywalny sposób,
- daje jednoznaczną informację zwrotną (terminal, dźwięk, ruch).

Etap 2. Implementacja algorytmu

Uczniowie w parach przystępują do implementacji algorytmu zaplanowanego w poprzedniej lekcji. Pracują w oparciu o znane konstrukcje programistyczne:

- listy danych,
- pętle (`for` lub `while`),
- instrukcje warunkowe,
- komunikaty tekstowe w terminalu.

Nauczyciel pełni rolę konsultanta – wspiera w podejmowaniu decyzji, ale nie narzuca rozwiązań.

Etap 3. Testowanie wersji minimalnej

Po uzyskaniu pierwszej działającej wersji programu uczniowie testują go na robocie. Testowanie obejmuje:

- uruchomienie programu dla podstawowych danych,
- sprawdzenie, czy zachowanie robota jest zgodne z planem algorytmu,
- analizę komunikatów w terminalu.

Zachęcamy uczniów do świadomego przewidywania działania programu przed jego uruchomieniem.

Etap 4. Korekty i uproszczenia

Na podstawie testów uczniowie wprowadzają drobne poprawki do programu. Jeśli pojawiają się trudności, priorytetem jest **upraszczanie**, a nie rozbudowa.

Podkreślamy, że działający, prosty program jest ważniejszy niż ambitny, ale niedokończony projekt.

Etap 5. Zatrzymanie i zapis stanu projektu

Na zakończenie lekcji uczniowie:

- zapisują aktualną wersję kodu,
- krótko opisują, co działa poprawnie,
- wskazują elementy, które mogłyby zostać rozwinięte w kolejnych zajęciach.

Ten etap stanowi punkt wyjścia do dalszej pracy projektowej.

18.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację współpracy w parach,
- osiągnięcie działającej wersji minimalnej programu,
- umiejętność testowania i wprowadzania poprawek,
- jakość refleksji nad aktualnym stanem projektu.

18.5. Wskazówki metodyczne

W tej lekcji kluczowe jest konsekwentne pilnowanie zakresu projektu. Warto często wracać do pytania: *czy to jest jeszcze wersja minimalna*.

Nie należy przyspieszać ani porównywać postępów zespołów. Projekty mogą rozwijać się w różnym tempie.

Na co uważać poznawczo w projektach:

- skłonność do nadmiernej rozbudowy kosztem stabilności programu,
- trudność w jednoczesnym myśleniu o algorytmie, kodzie i zachowaniu robota,
- zmęczenie poznawcze przy długiej pracy – warto planować krótkie przerwy,
- potrzeba jasnych kryteriów „wystarczająco dobrze", aby domknąć etap MVP.

Rozdział 19. Lekcja 17 – Mini-projekt: rozbudowa i warianty

19.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 2 × 45 minut

Forma pracy: praca w parach lub w małych zespołach projektowych

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (kostka inteligentna + jeden silnik; czujniki opcjonalnie),
- kod wersji minimalnej projektu z lekcji 16,
- notatki projektowe zespołów.

19.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń rozbudowuje istniejący program w sposób kontrolowany,

- uczeń wprowadza warianty działania algorytmu bez zmiany jego podstawowej struktury,
- uczeń świadomie modyfikuje dane lub decyzje w algorytmie,
- uczeń testuje wpływ wprowadzonych zmian na działanie programu,
- uczeń dokumentuje decyzje projektowe.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.2 – modyfikowanie i rozwijanie algorytmów,
- II.1 – rozwijanie i testowanie programów,
- IV – współpraca w zespołach projektowych i komunikacja.

19.3. Przebieg lekcji

Etap 1. Punkt wyjścia – działający projekt

Lekcję rozpoczynamy od krótkiego przypomnienia, że każdy zespół posiada działającą wersję minimalną projektu. Podkreślamy, że **rozbudowa zawsze zaczyna się od programu, który działa**.

Wspólnie ustalamy, że celem tej lekcji nie jest napisanie projektu od nowa, lecz jego **kontrolowane rozwinięcie**.

Etap 2. Wybór kierunku rozbudowy

Zespoły decydują, w jaki sposób chcą rozbudować projekt. Proponujemy przykładowe kierunki:

- zwiększenie liczby danych (np. dłuższa lista),
- dodanie nowego wariantu decyzji,
- zmiana reakcji robota na istniejące dane,
- uporządkowanie i uczytelnienie kodu.

Zachęcamy, aby zespoły wybrały **jeden** kierunek rozbudowy.

Etap 3. Planowanie zmiany przed kodowaniem

Przed wprowadzeniem zmian uczniowie opisują je krótko w języku naturalnym:

- co dokładnie zostanie zmienione,
- które elementy algorytmu pozostaną bez zmian,
- jaki efekt ma przynieść modyfikacja.

Dopiero po takim opisie zespoły przechodzą do edycji kodu.

Etap 4. Implementacja wariantów

Uczniowie modyfikują kod projektu, zachowując jego podstawową strukturę. Nauczyciel wspiera zespoły pytaniami naprowadzającymi, zamiast gotowych rozwiązań.

W trakcie pracy zachęcamy do częstego uruchamiania programu i sprawdzania efektów zmian.

Etap 5. Testowanie i porównanie wersji

Po wprowadzeniu zmian zespoły porównują działanie nowej wersji programu z wersją minimalną:

- co się zmieniło,
- co pozostało bez zmian,
- czy program nadal działa stabilnie.

Zwracamy uwagę, że każda rozbudowa powinna zachować poprawność działania programu.

Etap 6. Zatrzymanie projektu – decyzje projektowe

Na zakończenie lekcji zespoły zapisują aktualną wersję projektu oraz krótką notatkę:

- jakie zmiany zostały wprowadzone,
- dlaczego wybrano taki wariant rozbudowy,
- co mogłoby być kolejnym krokiem.

Ten etap przygotowuje projekt do finalizacji w kolejnej lekcji.

19.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- obserwację współpracy w zespołach,
- sensowność i spójność wprowadzonych zmian,
- umiejętność porównania wersji projektu,
- świadome uzasadnianie decyzji projektowych.

19.5. Wskazówki metodyczne

W tej lekcji kluczowe jest pilnowanie, aby zespoły nie wprowadzały zbyt wielu zmian naraz. Jedna świadoma modyfikacja jest znacznie bardziej wartościowa niż wiele chaotycznych poprawek.

Warto wzmacniać nawyk zapisywania decyzji projektowych – pomaga to uczniom zobaczyć rozwój projektu jako proces.

Na co uważać poznawczo w projektach:

- ryzyko utraty działającej wersji programu przy zbyt dużych zmianach,
- przeciążenie poznawcze przy jednoczesnej modyfikacji algorytmu i kodu,
- frustracja wynikająca z nieoczekiwanych błędów,
- potrzeba jasnego momentu zatrzymania prac nad daną wersją projektu.

Rozdział 20. Lekcja 18 – Testowanie, poprawki i prezentacja projektu

20.1. Informacje organizacyjne (czas, narzędzia)

Czas realizacji: 2 × 45 minut

Forma pracy: praca w zespołach projektowych oraz praca na forum klasy

Narzędzia:

- komputer z edytorem kodu Python i terminalem (NVDA, powiększenie ekranu),
- robot SPIKE Prime (zestaw używany w projekcie),
- kod projektu po rozbudowie z lekcji 17,
- notatki projektowe zespołów.

20.2. Cele lekcji i korelacja z podstawą programową

Cele dydaktyczne:

- uczeń testuje działanie projektu w różnych scenariuszach,
- uczeń wprowadza poprawki zwiększające stabilność programu,
- uczeń porządkuje kod projektu przed jego prezentacją,
- uczeń potrafi opisać działanie algorytmu i uzasadnić decyzje projektowe,
- uczeń prezentuje działający program.

Korelacja z podstawą programową (LO – zakres podstawowy):

- I.1 – analiza i prezentacja rozwiązania problemu,
- II.1 – testowanie, poprawianie i dokumentowanie programów,
- III – świadome korzystanie z narzędzi informatycznych,
- IV – komunikacja i współpraca w zespole.

20.3. Przebieg lekcji

Etap 1. Wprowadzenie – po co testujemy projekt

Lekcję rozpoczynamy od krótkiego przypomnienia, że projekt uznajemy za gotowy dopiero wtedy, gdy **działa przewidywalnie**, a nie tylko w jednym, wybranym przypadku.

Podkreślamy, że celem tej lekcji nie jest dodawanie nowych funkcji, lecz **sprawdzenie, poprawienie i domknięcie** projektu.

Etap 2. Przygotowanie przypadków testowych

Zespoły przygotowują krótką listę przypadków testowych, odpowiadając na pytania:

- jakie dane mogą pojawić się w projekcie,
- które sytuacje są typowe,
- które mogą sprawiać trudność algorytmowi.

Zachęcamy do zapisywania przewidywanego zachowania programu przed jego uruchomieniem.

Etap 3. Testowanie projektu

Uczniowie uruchamiają projekt dla przygotowanych przypadków testowych i analizują:

- zachowanie robota,
- komunikaty w terminalu,
- zgodność działania z planem algorytmu.

Wszelkie rozbieżności traktujemy jako informację, a nie jako błąd ucznia.

Etap 4. Poprawki i porządkowanie kodu

Na podstawie testów zespoły wprowadzają drobne poprawki:

- korekty warunków,
- uproszczenia struktury kodu,
- doprecyzowanie komunikatów w terminalu.

Nie dopuszczamy już istotnej rozbudowy funkcjonalności – skupiamy się na stabilności i czytelności.

Etap 5. Przygotowanie prezentacji projektu

Zespoły przygotowują krótką prezentację swojego projektu, obejmującą:

- opis celu projektu,
- schemat algorytmu w języku naturalnym,
- demonstrację działania programu,
- omówienie jednej decyzji projektowej.

Forma prezentacji powinna być dostosowana do potrzeb uczniów (opis słowny, terminal, dźwięk robota).

Etap 6. Prezentacja i refleksja

Zespoły prezentują swoje projekty na forum klasy. Po każdej prezentacji poświęcamy chwilę na krótką refleksję:

- co działało szczególnie dobrze,
- jakie rozwiązanie było ciekawe lub nietypowe,
- czego można się nauczyć z danego projektu.

20.4. Narzędzia ewaluacji

Ewaluacja ma charakter kształtujący i obejmuje:

- osiągnięcie działającego projektu,
- jakość testowania i wprowadzonych poprawek,

- umiejętność opisu algorytmu i decyzji projektowych,
- zaangażowanie w prezentację i refleksję.

20.5. Wskazówki metodyczne

W tej lekcji warto zadbać o spokojną atmosferę i podkreślać, że prezentacja projektu nie jest egzaminem, lecz elementem procesu uczenia się.

Dobrze sprawdza się docenianie **stabilności i czytelności**, a nie stopnia skomplikowania projektu.

Rozszerzona ramka – na co uważać poznawczo w projektach:

- zmęczenie poznawcze po długim cyklu pracy projektowej,
- obawa przed prezentacją działania programu,
- tendencja do wprowadzania zmian w ostatniej chwili,
- trudność w jednoczesnym mówieniu o algorytmie i obsłudze robota.

W razie potrzeby warto skrócić część prezentacyjną na rzecz spokojnej refleksji nad procesem pracy.

Rozdział 21. Rozszerzenia i samodzielność (lekcje 19–22)

21.1. Założenia ogólne bloku

Rozdział 21 stanowi pomost pomiędzy mini-projektami a projektem końcowym. Jego celem jest stopniowe zwiększanie samodzielności uczniów przy jednoczesnym zachowaniu wyraźnych ram organizacyjnych. Uczniowie pracują na **dostarczonym kodzie wyjściowym**, który następnie porządkują, analizują i modyfikują. Robot pozostaje **aktywnym kontekstem** każdej lekcji – każda zmiana w kodzie powinna mieć zauważalny efekt w działaniu robota.

Blok realizowany jest jako **ciąg czterech lekcji z wyraźnymi checkpointami**, przy czym nie oceniamy efektu końcowego, lecz proces pracy, podejmowane decyzje i poziom samodzielności.

21.2. Cele dydaktyczne (wspólne dla lekcji 19–22)

- rozwijanie umiejętności czytania i rozumienia cudzego kodu,
- świadome porządkowanie struktury programu bez zmiany jego działania,
- planowanie algorytmu wychodząc od opisu problemu,
- samodzielne modyfikowanie parametrów i zachowania programu,
- współpraca zespołowa i komunikacja wokół kodu.

21.3. Organizacja pracy i checkpointy

- **Forma pracy:** praca w parach, z elementami pracy zespołowej
- **Punkt wyjścia:** wspólny kod bazowy dostarczony przez nauczyciela
- **Robot:** aktywny element każdej lekcji (dźwięk, ruch, reakcja na dane)
- **Checkpoint:** na końcu każdej lekcji krótki zapis stanu prac i wniosków

Lekcja 19 – Refaktoryzacja i porządkowanie kodu

Intencja lekcji

Uczniowie uczą się czytać i porządkować istniejący program bez zmiany jego funkcjonalności. Skupiamy się wyłącznie na **czytelności kodu**, a nie na jego rozbudowie.

Zakres działań

- analiza struktury dostarczonego programu,
- poprawa nazw zmiennych,
- uporządkowanie kolejności instrukcji,
- dodanie krótkich komentarzy opisujących działanie fragmentów kodu,
- zachowanie identycznego działania robota przed i po refaktoryzacji.

Checkpoint

- program działa tak samo jak przed zmianami,
- kod jest czytelniejszy i łatwiejszy do omówienia.

Lekcja 20 – Projektowanie algorytmu od problemu

Intencja lekcji

Uczniowie przechodzą od opisu problemu do planu algorytmu, bez natychmiastowego kodowania. Celem jest pokazanie, że **algorytm poprzedza kod**.

Zakres działań

- analiza krótkiego opisu problemu (scenariusza działania robota),
- zapis algorytmu w języku naturalnym,
- wskazanie danych, decyzji i reakcji robota,
- odniesienie planu do istniejącego kodu bazowego.

Checkpoint

- zapisany algorytm w języku naturalnym,
- wskazane miejsca w kodzie, które będą modyfikowane.

Lekcja 21 – Samodzielne modyfikacje programów

Intencja lekcji

Uczniowie samodzielnie wprowadzają zmiany do programu, pracując w jasno określonych ramach. Zmiany dotyczą **parametrów i zachowania**, nie struktury językowej.

Zakres działań

- modyfikacja wartości w listach lub zmiennych sterujących,
- zmiana progów decyzji w instrukcjach warunkowych,
- dostosowanie reakcji robota (dźwięk, ruch),
- testowanie wpływu zmian na działanie programu.

Checkpoint

- program reaguje inaczej niż wersja bazowa,
- uczniowie potrafią wyjaśnić, co zostało zmienione i dlaczego.

Lekcja 22 – Praca zespołowa nad kodem

Intencja lekcji

Lekcja poświęcona jest współpracy i komunikacji wokół kodu. Uczniowie uczą się dzielić odpowiedzialność i omawiać decyzje programistyczne.

Zakres działań

- podział ról w parze lub zespole (analiza, modyfikacja, testowanie),
- wspólne omawianie zmian w kodzie,
- scalanie pomysłów w jedną wersję programu,
- przygotowanie kodu jako punktu wyjścia do projektu końcowego.

Checkpoint

- jedna, wspólna wersja programu,
- krótki opis decyzji zespołowych.

21.4. Ewaluacja w bloku

Ewaluacja ma charakter wyłącznie kształtujący. Nie oceniamy poziomu skomplikowania programu ani efektu końcowego. Zwracamy uwagę na:

- samodzielność podejmowanych decyzji,
- umiejętność czytania i omawiania kodu,
- sensowność wprowadzanych zmian,
- współpracę i komunikację w zespole.

21.5. Na co uważać poznawczo

- przeciążenie wynikające z pracy na cudzym kodzie,
- pokusa jednoczesnej refaktoryzacji i rozbudowy funkcjonalności,
- trudność w werbalizacji zmian wprowadzonych w programie,
- zmęczenie poznawcze po intensywnych mini-projektach.

W razie trudności warto cofnąć się do ostatniego działającego checkpointu i pracować na mniejszym zakresie zmian.

Rozdział 22. Projekt końcowy (lekcje 23–28)

22.1. Założenia ogólne bloku

Rozdział 22 stanowi kulminację całego cyklu kształcenia. Uczniowie realizują **projekt końcowy**, w którym samodzielnie planują, implementują, testują i prezentują działający program sterujący robotem. Projekt ma charakter **otwarty**, jednak jego zakres jest świadomie ograniczony, tak aby umożliwić ukończenie pracy wszystkim zespołom.

Praca projektowa przebiega w **trzech etapach**, realizowanych w formie lekcji podwójnych (2 × 45 minut). w całym bloku kluczowe znaczenie ma proces: planowanie, iteracyjne poprawki oraz refleksja nad rozwiązaniem.

22.2. Cele dydaktyczne (wspólne dla lekcji 23–28)

- samodzielne zaprojektowanie algorytmu rozwiązującego określony problem,
- implementacja programu w języku Python z wykorzystaniem robota SPIKE Prime,
- testowanie i poprawianie programu na podstawie przypadków testowych,
- dokumentowanie kolejnych wersji projektu,
- prezentowanie i omawianie własnego rozwiązania.

22.3. Organizacja pracy

- **Forma pracy:** zespoły projektowe (2–4 osoby),
- **Punkt wyjścia:** pusta struktura programu lub kod bazowy zaproponowany przez zespół,
- **Robot:** aktywny element projektu (ruch, dźwięk, reakcja na dane),
- **Dokumentacja:** kod źródłowy + krótki opis algorytmu w języku naturalnym,
- **Ewaluacja:** kształtująca, bez oceniania stopnia złożoności rozwiązania.

Lekcje 23–24 – Projekt końcowy: planowanie i pierwsza wersja

Intencja etapu

Celem pierwszego etapu jest zaplanowanie projektu oraz przygotowanie **pierwszej działającej wersji programu**. Akcentujemy znaczenie wersji minimalnej (MVP).

Zakres działań

- wybór problemu lub scenariusza działania robota,
- zapis algorytmu w języku naturalnym,
- określenie danych wejściowych i reakcji robota,
- implementacja pierwszej wersji programu,
- uruchomienie i wstępne testy.

Checkpoint

- istnieje działająca wersja programu,
- algorytm jest zapisany i możliwy do omówienia,
- robot reaguje w przewidywalny sposób.

Lekcje 25–26 – Projekt końcowy: testowanie i poprawki

Intencja etapu

Celem drugiego etapu jest **stabilizacja projektu**. Uczniowie uczą się systematycznego testowania i poprawiania programu.

Zakres działań

- przygotowanie przypadków testowych,
- testowanie programu w różnych scenariuszach,
- analiza błędów logicznych i nieprzewidzianych zachowań,
- wprowadzanie poprawek i uproszczeń,
- porządkowanie kodu.

Checkpoint

- program działa stabilnie w kilku scenariuszach,
- zespoły potrafią wyjaśnić wprowadzone poprawki.

Lekcje 27–28 – Projekt końcowy: wersja finalna

Intencja etapu

Ostatni etap służy **domknięciu projektu** oraz jego prezentacji. Skupiamy się na czytelności, stabilności i komunikacji rozwiązania.

Zakres działań

- ostatnie testy programu,
- przygotowanie krótkiej prezentacji projektu,
- demonstracja działania robota,
- omówienie algorytmu i decyzji projektowych,
- refleksja nad procesem pracy.

Checkpoint

- działający program,
- przygotowana prezentacja,
- gotowość do omówienia projektu.

22.4. Ewaluacja w bloku

Ewaluacja ma charakter kształtujący i koncentruje się na:

- samodzielności zespołów,
- spójności algorytmu i jego realizacji,
- umiejętności testowania i poprawiania programu,
- jakości komunikacji podczas prezentacji.

Nie oceniamy poziomu skomplikowania projektu ani porównawczo efektów między zespołami.

22.5. Rozszerzona ramka – na co uważać poznawczo

- przeciążenie wynikające z długiego czasu pracy nad jednym zadaniem,
- presja „skończenia projektu" kosztem rozumienia algorytmu,
- trudność w prezentowaniu własnego kodu,
- zmęczenie końcową fazą pracy.

W razie potrzeby warto skrócić część prezentacyjną i położyć większy nacisk na spokojną refleksję nad procesem uczenia się.

Rozdział 23. Podsumowanie i ewaluacja (lekcje 29–30)

23.1. Założenia ogólne bloku

Ten rozdział zamyka cały cykl dydaktyczny i służy **świadomemu podsumowaniu procesu uczenia się**, a nie tylko zaprezentowaniu efektów pracy. Jego celem jest umożliwienie uczniom refleksji nad tym, czego się nauczyli, jak pracowali oraz jakie kompetencje rozwinęli w trakcie kursu.

Lekcje 29–30 mają charakter **podsumowująco-refleksyjny**. Nie wprowadzamy nowych treści programistycznych ani nie rozszerzamy projektów. Akcentujemy analizę algorytmów, doświadczenie pracy projektowej oraz informację zwrotną.

23.2. Cele dydaktyczne (wspólne dla lekcji 29–30)

- uczeń potrafi opisać działanie własnego programu i użytego algorytmu,
- uczeń dokonuje refleksji nad procesem uczenia się i pracy projektowej,
- uczeń identyfikuje swoje mocne strony i obszary do dalszego rozwoju,
- uczeń odbiera i formułuje informację zwrotną,
- uczeń porządkuje efekty pracy w postaci prostego portfolio.

23.3. Organizacja pracy

- **Forma pracy:** praca zespołowa oraz praca na forum klasy
- **Punkt wyjścia:** projekty końcowe z bloku H
- **Robot:** wykorzystywany jako element demonstracji, nie jako główny cel aktywności
- **Ewaluacja:** wyłącznie kształtująca, bez ocen sumujących

Lekcja 29 – Prezentacja rozwiązań i omówienie algorytmów

Intencja lekcji

Celem lekcji jest umożliwienie uczniom zaprezentowania swoich rozwiązań oraz **werbalnego opisania algorytmu**, który został zastosowany w projekcie. Skupiamy się na rozumieniu, a nie na atrakcyjności efektu.

Zakres działań

- prezentacja projektu przez zespoły,
- opis algorytmu w języku naturalnym,
- omówienie użytych danych, decyzji i iteracji,
- demonstracja działania programu (robot, terminal),
- pytania i komentarze od innych uczniów.

Rola nauczyciela

Nauczyciel moderuje prezentacje, dbając o spokojne tempo i jasność wypowiedzi.
w razie potrzeby pomaga uczniom doprecyzować opis algorytmu pytaniami naprowadzającymi.

Punkty kontrolne

- każdy zespół zaprezentował projekt,
- algorytm został nazwany i opisany słownie.

Lekcja 30 – Podsumowanie kursu i refleksja uczniowska

Intencja lekcji

Ostatnia lekcja służy uporządkowaniu doświadczeń z całego kursu. Jej celem jest **uświadomienie uczniom, czego się nauczyli**, a nie sprawdzanie wiedzy.

Zakres działań

- rozmowa podsumowująca cały kurs,
- refleksja nad trudnościami i sukcesami,
- identyfikacja poznanych pojęć i umiejętności,
- omówienie pracy zespołowej,
- zebranie informacji zwrotnej od uczniów.

Proponujemy pytania refleksyjne:

- czego nowego się nauczyliśmy,
- co było najtrudniejsze,
- co okazało się zaskakująco proste,
- w jakim momencie poczuliśmy samodzielność.

23.4. Ewaluacja w bloku I

Ewaluacja ma charakter podsumowujący i kształtujący. Obejmuje:

- jakość opisu algorytmu,
- umiejętność refleksji nad własną pracą,
- zaangażowanie w rozmowę i informację zwrotną,
- kompletność prostego portfolio (kod + opis).

Nie stosujemy ocen punktowych ani porównań między uczniami.

23.5. Wskazówki metodyczne

W bloku i szczególnie ważne jest stworzenie bezpiecznej atmosfery. Warto unikać presji czasu i oceniania porównawczego.

Dobrze sprawdza się podkreślanie postępu uczniów w stosunku do punktu wyjścia z początku kursu.

Na co uważać poznawczo:

- zmęczenie końcówką semestru lub roku szkolnego,
- trudność w werbalizacji własnych kompetencji,
- obawa przed wystąpieniem publicznym,
- tendencja do deprecjonowania własnych osiągnięć.

W razie potrzeby część refleksyjna może mieć formę rozmowy kierowanej lub krótkich wypowiedzi pisemnych.

Rozdział 24. Podsumowanie kursu i rekomendacje metodyczne

24.1. Charakter kursu i jego specyfika

Kurs algorytmiki i programowania opisany w niniejszym podręczniku został zaprojektowany jako **proces stopniowego budowania samodzielności uczniów**, a nie jako sekwencja izolowanych tematów programistycznych. Od pierwszych lekcji akcentowaliśmy myślenie algorytmiczne, jednoznaczność opisu działania oraz przewidywanie skutków działania programu.

Istotną cechą kursu jest wykorzystanie robota jako **kontekstu motywującego**, a nie celu samego w sobie. Robot pełni rolę wykonawcy algorytmu i źródła informacji zwrotnej, natomiast główny ciężar pracy dydaktycznej spoczywa na analizie danych, decyzji i iteracji.

24.2. Spiralny model uczenia się

Cały kurs opiera się na spiralnym modelu uczenia się:

- te same konstrukcje (sekwencja, warunek, pętla, dane) pojawiają się wielokrotnie,
- za każdym razem są używane w nowym kontekście,
- nie wprowadzamy nowych pojęć bez wcześniejszego osadzenia ich w znanym schemacie.

Taki układ szczególnie dobrze sprawdza się w pracy z uczniami niewidomymi i niedowidzącymi, ponieważ ogranicza przeciążenie poznawcze i wzmacnia poczucie kontroli nad kodem.

24.3. Rola nauczyciela

W trakcie kursu rola nauczyciela stopniowo się zmienia:

- na początku dominuje **prowadzenie i modelowanie sposobu myślenia**,
- w środkowej części kursu nauczyciel pełni rolę **konsultanta i moderatora**,
- w blokach projektowych nauczyciel staje się **opiekunem procesu**, a nie dostawcą rozwiązań.

Ważne jest, aby nauczyciel świadomie powstrzymywał się od nadmiernego instruowania, pozostawiając uczniom przestrzeń do podejmowania decyzji i popełniania kontrolowanych błędów.

24.4. Dostępność i praca z różnorodną grupą

Projekt kursu zakłada pracę w grupach mieszanych, w których uczniowie niewidomi i niedowidzący współpracują ze sobą. Kluczowe elementy wspierające dostępność to:

- konsekwentne używanie terminala jako głównego kanału informacji zwrotnej,

- werbalizacja działania algorytmu przed jego uruchomieniem,
- czytelny standard kodu i stała struktura programu,
- ograniczenie komunikacji wizualnej robota do minimum.

Takie podejście sprzyja nie tylko dostępności, ale również rozwija precyzję myślenia algorytmicznego u wszystkich uczniów.

24.5. Projekty jako narzędzie uczenia się

Projekty w kursie pełnią rolę **narzędzia dydaktycznego**, a nie celu rywalizacyjnego. Ich zadaniem jest:

- integracja poznanych wcześniej pojęć,
- nauka planowania i iteracyjnej pracy nad rozwiązaniem,
- rozwijanie umiejętności komunikacji i współpracy.

Świadome ograniczanie zakresu projektów oraz akcent na wersję minimalną pozwalają wszystkim uczniom osiągnąć sukces edukacyjny.

24.6. Ewaluacja i informacja zwrotna

W całym kursie dominuje **ocenianie kształtujące**. Informacja zwrotna dotyczy przede wszystkim:

- sposobu myślenia algorytmicznego,
- umiejętności testowania i analizowania programu,
- samodzielności i odpowiedzialności za kod.

Ocena efektu końcowego ma znaczenie drugorzędne wobec procesu uczenia się.

24.7. Możliwe modyfikacje i adaptacje kursu

Kurs został zaprojektowany jako elastyczny i może być modyfikowany w zależności od:

- tempa pracy grupy,
- dostępnego czasu,

- liczby zestawów robotycznych,
- wcześniejszych doświadczeń uczniów.

Dopuszczalne jest wydłużanie bloków projektowych kosztem części ćwiczeniowej lub łączenie wybranych lekcji, o ile zachowana zostaje logika spiralnego rozwoju kompetencji.

24.8. Efekty kształcenia

Po ukończeniu kursu uczniowie:

- rozumieją algorytm jako opis procesu,
- potrafią zaprojektować i zapisać prosty algorytm,
- implementują programy w języku Python,
- testują i modyfikują swoje rozwiązania,
- pracują zespołowo nad kodem,
- potrafią opisać działanie własnego programu.

24.9. Zakończenie

Niniejszy podręcznik ma charakter roboczy i otwarty. Jego celem nie jest narzucenie jednego sposobu prowadzenia zajęć, lecz dostarczenie **spójnych ram metodycznych**, które nauczyciel może adaptować do własnego stylu pracy i potrzeb uczniów.

Najważniejszym rezultatem kursu nie jest liczba poznanych konstrukcji językowych, lecz **świadome, spokojne i dostępne myślenie algorytmiczne**, które uczniowie mogą rozwijać w dalszej edukacji.

Rozdział 25. Spis kompetencji ucznia po kursie algorytmiki i programowania

Dokument przedstawia zestaw kompetencji, które uczeń rozwija i osiąga po ukończeniu kursu algorytmiki i programowania realizowanego z wykorzystaniem języka Python oraz robotów SPIKE Prime. Spis ma charakter opisowy i funkcjonalny – może

być wykorzystywany jako materiał informacyjny dla uczniów, nauczycieli, rodziców lub dyrekcji szkoły.

25.1. Kompetencje algorytmiczne

Po ukończeniu kursu uczeń:

- rozumie algorytm jako jednoznaczny opis procesu prowadzącego do rozwiązania problemu,
- potrafi zapisać algorytm w języku naturalnym przed jego implementacją,
- analizuje algorytm krok po kroku i przewiduje jego działanie,
- rozróżnia sekwencję, decyzję i iterację jako podstawowe struktury sterujące,
- świadomie dobiera strukturę algorytmu do rodzaju problemu.

25.2. Kompetencje programistyczne (Python)

Po ukończeniu kursu uczeń:

- tworzy proste programy w języku Python zgodnie z ustalonym standardem kodu,
- korzysta ze zmiennych do przechowywania i przetwarzania danych,
- stosuje instrukcje warunkowe (`if` / `elif` / `else`),
- wykorzystuje pętle (`for`, `while`) do realizacji algorytmów iteracyjnych,
- pracuje na listach jako podstawowej strukturze danych,
- uruchamia, testuje i modyfikuje programy.

25.3. Kompetencje związane z testowaniem i analizą

Po ukończeniu kursu uczeń:

- przewiduje działanie programu przed jego uruchomieniem,
- przygotowuje proste przypadki testowe,

- rozróżnia błędy składniowe i logiczne,
- wykorzystuje komunikaty w terminalu do analizy działania programu,
- wprowadza poprawki na podstawie obserwacji i testów.

25.4. Kompetencje projektowe

Po ukończeniu kursu uczeń:

- planuje pracę projektową etapami,
- rozumie pojęcie wersji minimalnej (MVP),
- rozwija projekt w sposób iteracyjny,
- dokumentuje kolejne etapy pracy,
- potrafi zatrzymać i domknąć projekt na danym etapie.

25.5. Kompetencje związane z czytelnością i jakością kodu

Po ukończeniu kursu uczeń:

- czyta i rozumie cudzy kod na podstawowym poziomie,
- dba o czytelność nazw zmiennych,
- porządkuje strukturę programu bez zmiany jego działania,
- stosuje komentarze jako wsparcie dla czytelnika kodu,
- rozumie znaczenie czytelności w pracy zespołowej.

25.6. Kompetencje pracy zespołowej i komunikacji

Po ukończeniu kursu uczeń:

- pracuje w parze lub zespole nad wspólnym kodem,
- komunikuje swoje pomysły i decyzje algorytmiczne,
- słucha i uwzględnia propozycje innych,
- bierze odpowiedzialność za przydzielone zadania,

- uczestniczy w prezentacji i omówieniu projektu.

25.7. Kompetencje metapoznawcze

Po ukończeniu kursu uczeń:

- reflektuje nad własnym procesem uczenia się,
- rozpoznaje swoje mocne strony i trudności,
- akceptuje błędy jako element nauki,
- stopniowo buduje samodzielność w rozwiązywaniu problemów,
- rozwija poczucie sprawczości w pracy z technologią.

25.8. Kompetencje związane z dostępnością i narzędziami

Po ukończeniu kursu uczeń:

- korzysta z terminala jako podstawowego źródła informacji zwrotnej,
- pracuje z kodem przy użyciu narzędzi wspomagających (NVDA, powiększenie ekranu),
- rozumie znaczenie werbalnego opisu algorytmu i programu,
- funkcjonuje w środowisku edukacyjnym uwzględniającym różnorodne potrzeby.

25.9. Podsumowanie

Spis kompetencji pokazuje, że efektem kursu nie jest jedynie opanowanie składni języka Python, lecz rozwój szerokiego zestawu umiejętności algorytmicznych, projektowych i społecznych. Kompetencje te stanowią solidną podstawę do dalszej nauki informatyki oraz świadomego korzystania z technologii.