Lekcja 7. Pętle cz. I

Cel lekcji

Celem lekcji jest wprowadzenie do tematu pętli. Poznamy pętlę while oraz omówimy jej zastosowanie przy użyciu przykładów.

Informacja dla nauczyciela

Temat pętli przewidziany jest na 2 spotkania - przygotowane są 2 konspekty.

Przebieg zajęć

1. Pytania wstępne z poprzednich zajęć
2. Wprowadzenie do zagadnienia pętli
3. Pętla while
4. Podsumowanie oraz pytania powtórzeniowe

1. **Przykładowe pytania z poprzednich zajęć**

- Jakie warianty instrukcji warunkowych poznaliśmy?
- Co stawiamy na koniec instrukcji warunkowej? (Dwukropek)
- Jak oznacza się kod, który ma wykonać się tylko pod określonym warunkiem? (Wcięcie, po słowie if, warunku oraz dwukropku)
- Jak działa instrukcja else? Czy wykona się za każdym razem?
- Ile instrukcji elif możemy dodać?
- Na jakiej zasadzie wybierana jest instrukcja, która zostanie wykonana w konstrukcji if-elif? (Sprawdzanie kolejnych warunków, aż znajdziemy prawdziwy lub do końca warunków)

Zadanie rozgrzewkowe:

Przygotuj program, który sprawdzi, czy wprowadzona liczba jest podzielna przez:

3, 10 oraz 77

a) osobno podzielna przez powyższe
b) jednocześnie podzielna przez powyższe

2. **Wprowadzenie do zagadnienia pętli**

Dziś poznamy jeden z ważniejszych mechanizmów w językach programowania: pętle.

Pętle pozwalają na powtarzanie pewnych instrukcji w naszym programie.

Co przez to zyskujemy?

- mniej linijek kodu do napisania:

Zamiast napisać 100 razy te same instrukcje, wystarczy napisać je raz i powiedzieć komputerowi aby wykonał je 100 razy.

- elastyczny kod:

W pewnych sytuacjach nie możemy z góry określić ile razy ma wykonać się pewna czynność. Pętle pozwalają rozwiązać ten problem. Odpowiednio stworzona pętla wykona się tyle razy ile potrzeba.

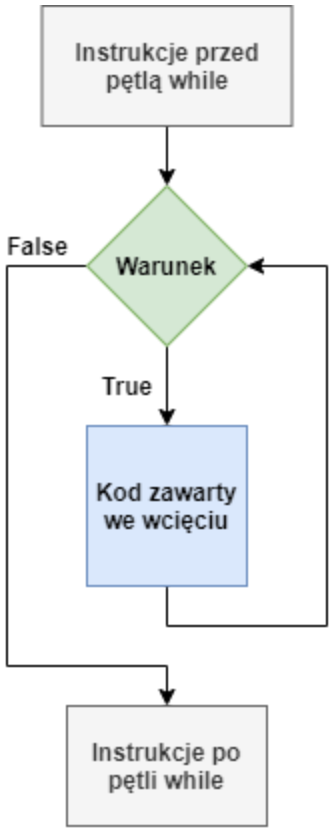
- mniej linijek kodu to również bardziej przejrzysty kod.

Przykłady pojawią się za chwilę - po omówieniu pętli while.

3. Pętla while

Pierwszy rodzaj pętli, który jest dostępny w Pythonie to pętla **while**.

Schemat działania pętli while:



Każde wykonanie instrukcji zawartych w pętli (wcięty kod) zaczyna się od sprawdzenia warunku.

Jeżeli warunek jest prawdziwy kod jest wykonywany i następnie ponownie sprawdzamy warunek.

Jeśli warunek jest fałszywy kod nie jest wykonywany i przechodzimy do kolejnych instrukcji za pętlą **while**.

Składnia pętli **while** prezentuje się następująco:

```
while warunek:
  # Kod, który może być powtarzany
  pass
```

Słowo kluczowe **while** oznacza z jaką pętlą mamy do czynienia. Po nim zapisujemy warunek, który może przyjmować dokładnie takie same postaci jak w instrukcji warunkowej **if**. Na koniec linijki zapisujemy **dwukropek**.

Tak jak w instrukcji warunkowej brak dwukropka jest błędem.

Kod, który będzie (chociaż nie musi) powtarzany zapisujemy we wcięciu, aby odróżnić go od pozostałych linijek.

Pytanie do uczniów: Czy kod zawarty w pętli while może nie wykonać się ani razu?

Odp: Tak, może się zdarzyć, że podczas pierwszego sprawdzenia warunku okaże się on fałszywy i pętla się nie wykona ani razu. Może też dojść do sytuacji, w której świadomie lub nie pętla będzie się wykonywać w nieskończoność, ponieważ warunek zawsze będzie prawdziwy. O pętli nieskończonej za chwilę.

Przykład

Nasza pierwsza pętla while, pokaz korzyści płynących z używania pętli.

1) Mniej linii do zapisu

Zadanie Napiszmy program, który 10-krotnie wyświetli w konsoli napis "*Cześć Gigancie!*" (bez użycia pętli):

```
print("Cześć Gigancie!")
# [...]
print("Cześć Gigancie!")
```

a teraz to samo tylko, że za pomocą pętli **while** (*Nie usuwamy poprzedniego przykładu*)

```
i = 0
while i < 10:
  print("Cześć Gigancie!")
  i += 1
  pass
```

Pełne omówienie zmiennej pomocniczej *i* oraz działania pętli.

Dlaczego zmienna *i* jest tutaj potrzebna? Dlaczego zmiana zmiennej wewnątrz pętli jest ważna? **Rozpisać każdą iterację pętli.** Dlaczego instrukcja **print** wykonuje się dokładnie 10 razy?

2) Elastyczny kod

Niech każdy uczeń zmieni teraz liczbę wyświetlanych komunikatów z 10 na 25.

Co łatwiej zmienić? Liczbę instrukcji w pierwszej wersji, czy liczbę w warunku pętli while?

3) Czytelność kodu

Pytanie do uczniów: Co jest czytelniejsze: 25 linijek instrukcji **print**, czy 5 linijek pętli **while** wraz z wymaganą zmienną pomocniczą?

Zadanie

Napisz program, który wczyta od użytkownika liczbę całkowitą i wyświetli na ekranie dokładnie tyle komunikatów "*Giganci Programowania*". Należy zaprezentować kilka wersji dostępnych rozwiązań. Omówić kolejne iteracje pętli.

```
liczba = int(input("Wprowadź liczbę: "))

while liczba > 0:
  print("Giganci Programowania")
  liczba -= 1
  pass
```

```
liczba = int(input("Wprowadź liczbę: "))
i = 0
while i < liczba:
  print("Giganci Programowania")
  i += 1
  pass
```

Zadanie

Timer bomby. Przekształć poprzednie rozwiązanie tak aby po wpisaniu liczby przez użytkownika program wypisywał kolejne liczby, aż do zera. Po tym w konsoli powinien pojawić się napis "BOOM!". Należy krótko omówić moduł **time** oraz metodę **time.sleep**. Po prawej wersja rozszerzona, gdzie licznik pojawia się w jednej linijce - ciekawostka dla uczniów.

```
import time
liczba = int(input("Wprowadź liczbę: "))

while liczba > 0:
  print(liczba)
  liczba -= 1
  time.sleep(1)
  pass

print("BOOM!")
```

```
import time
liczba = int(input("Wprowadź liczbę: "))

while liczba > 0:
  print('\r' + str(liczba), end=' ')
  liczba -= 1
  time.sleep(1)
  pass

print('\r' + "BOOM!")
```

Ćwiczenie

Zgadywanie liczby wylosowanej przez komputer. Program losuje liczbę, zadaniem użytkownika jest odgadnąć ją. Komputer odpowiada “za mało”, “za dużo” lub w przypadku trafienia wyświetla informację o wygranej i liczbie tur potrzebnych do wygranej.

Zaczynamy od importu moduły **random**, dzięki któremu komputer będzie mógł wylosować liczbę. Następnie przygotowujemy zmienne określające minimalną i maksymalną losowaną liczbę oraz zmienną przechowującą wylosowaną liczbę.

```
# Import modułu, który pozwala na losowanie wartości
import random

# Zakres w którym losujemy wartość
MINIMUM = 0
MAXIMUM = 100

# Wylosowanie wartości i zapisanie jej do zmiennej
wylosowana_liczba = random.randint(MINIMUM, MAXIMUM)

# Wyświetlenie wylosowanej wartości
# Tylko w celach sprawdzenia działania
print(wylosowana_liczba)
```

Każde kolejne uruchomienie programu zwraca wylosowaną liczbę.

W kolejnym etapie tworzymy zmienną, do której będziemy wpisywać wartości podawane przez użytkownika oraz zmienną pomocniczą *licznik_tur*, która zlicza liczbę tur potrzebnych do wygranej. **None** oznacza wartość pustą.

```
# Zmienne używane do kontroli działania programu
odpowiedz = None
licznik_tur = 0
```

Następnie przygotowujemy pętle **while**, której warunek musi zagwarantować nam powtarzanie kodu póki toczy się rozgrywka - czyli póki nie podano poprawnej odpowiedzi. Wewnątrz pętli należy powtarzać następujące czynności:

- odczytanie odpowiedzi użytkownika - próba odgadnięcia wartości
- zwiększenie licznika tur
- odpowiedź komputera w zależności od skuteczności strzału użytkownika

Pełne rozwiązanie:

```
# Import modułu, który pozwala na losowanie wartości
import random

# Zakres w którym losujemy wartość
MINIMUM = 0
MAXIMUM = 100

# Wylosowanie wartości i zapisanie jej do zmiennej
wylosowana_liczba = random.randint(MINIMUM, MAXIMUM)

# Wyświetlenie wylosowanej wartości
# Tylko w celach sprawdzenia działania
print(wylosowana_liczba)

# Zmienne używane do kontroli działania programu
odpowiedz = None
licznik_tur = 0

# Pętla działająca tak długo, aż odpowiedź użytkownika jest różna od wylosowanej liczby
while odpowiedz != wylosowana_liczba:
  # Odczytanie odpowiedzi użytkownika
  odpowiedz = int(input("Podaj liczbę: "))
  # Zwiększenie licznika tur, aby śledzić która to tura
  licznik_tur += 1

  # Dla niepoprawnych odpowiedzi wyświetli się podpowiedź
  if odpowiedz < wylosowana_liczba:
    print("za mało :p")
  elif odpowiedz > wylosowana_liczba:
    print("za dużo :b")
    pass
  pass

# Jeśli opuściliśmy pętle to oznacza, że udało się uzyskać poprawną odpowiedź
print("Gratulacje udało Ci się odgadnąć wylosowaną liczbę!")
print("Liczba: " + str(wylosowana_liczba))
print(f"Tura: {licznik_tur}")
```

Jeśli podamy poprawną odpowiedź pętla **nie** wykona się kolejny raz i przejdziemy do dalszych instrukcji, czyli gratulacji i podsumowania rozgrywki.

Należy usunąć linijkę wypisującą wylosowaną liczbę, jeśli program został przetestowany. Pytanie do uczniów: jaka jest najlepsza strategia do tej gry?

Odp: dzielenie przedziału na pół z każdą odpowiedzią.

Pętla nieskończona

Bardzo ważnym zagadnieniem jest pętla nieskończona. Jeżeli warunek pętli zawsze będzie prawdziwy to pętla będzie wykonywać się w nieskończoność. Składnia:

```
while True:
  print("You can't stop me now!")
  pass
```

```
import time

while True:
  print("You can't stop me now!")
  time.sleep(1)
  pass
```

Taka pętla może być przydatna, jeśli naszym planem jest aby program działał w nieskończoność, np. stacja monitorująca pogodę.

Break

Są jednak sposoby, aby zakończyć dowolną pętle, nawet tę "*nieskończoną*". Da się tego dokonać przy wykorzystaniu słowa kluczowego **break**, który działa w bardzo prosty sposób. W momencie wykonania tej instrukcji opuszczamy pętle i przechodzimy dalej.

```
import time

while True:
  print("You can't stop me now!")
  time.sleep(1)
  break
  pass

print("Haa! Maybe I - the programmer - can!")
```

W tym przykładzie zaprezentowaliśmy "suche" działanie **break**. Pętla wykona tylko jedną iterację / okrążenie po czym program wyjdzie z niej, dzięki **break**. (To samo można by osiągnąć poprzez zapisanie kodu bez żadnej pętli, również wykonałby się tylko raz).

Prawdziwa siła **break** to wychodzenie z pętli pod pewnym warunkiem - wykorzystanie instrukcji warunkowych wewnątrz pętli.

Przykład

Program echo, który kończy się dopiero wtedy kiedy napiszemy *“exit”*.

```
print("Będę Twoim echo")
while True:
  odp = input("Napisz coś: ")
  print("Echo: " + odp)
  pass
```

```
print("Będę Twoim echo")
while True:
  odp = input("Napisz coś: ")

  if odp == "exit":
    break

  print("Echo: " + odp)
  pass

print("Do zobaczenia!")
```

Słowo kluczowe **break** może być wykorzystywane w każdej pętli i nie ma ograniczeń ile razy jest wykorzystane. Tzn. możemy przyjąć wiele warunków kończących wykonywanie pętli i każdy z nich będzie działał tak samo.

Continue

Kolejnym słowem kluczowym powiązanym z pętlami jest **continue**, które pozwala na pominięcie dalszej części i sprawdzenie warunku kolejny raz, aby określić czy pętla powinna się jeszcze wykonać.

Przykład

Logowanie za pomocą pinu, roku urodzenia oraz hasła bez limitu niepoprawnych podejść. Błędna informacja na którykolwiek z etapów sprawia, że musimy rozpocząć jeszcze raz. Podanie poprawnych danych powoduje wyjście z pętli za pomocą **break**.

```
PIN = "1234"
ROK_URODZENIA = "1998"
HASLO = "qwerty"

while True:
  input_pin = input("PIN: ")
  if input_pin != PIN:
    print("Odmowa dostępu!")
    continue

  input_rok_urodzenia = input("Rok urodzenia: ")
  if input_rok_urodzenia != ROK_URODZENIA:
```

```
    print("Odmowa dostępu! Zaloguj się ponownie")
    continue

  input_haslo = input("Hasło: ")
  if input_haslo != HASLO:
    print("Odmowa dostępu! Zaloguj się ponownie")
    continue

  print("Zalogowano poprawnie!")
  break
  pass

print("Super tajne treści znajdują się tutaj")
```

4. **Podsumowanie oraz pytania powtórzeniowe**

Czym jest pętla w programowaniu?

Jak działa pętla while?

Czy możemy opuścić pętle while w dowolnym momencie, czy musimy czekać aż wykona się cały kod z jej wnętrza?

Jak dokładnie działa break i continue?