Wrocław University of Science and Technology

Wstęp do programowania

INP003203L

Semestr zimowy 2020/2021

Poniedziałek, 7:30 - 9:00

sala wirtualna
– zajęcia online

Sylwia Majchrowska

sylwia.majchrowska@pwr.edu.pl

https://majsylw.netlify.app/teaching/

pokój 213, budynek L-1

Wrocław University of Science and Technology

Plan na dziś

1. Złożone operatory przypisania
2. Wprowadzenie do pętli
   - Pętla while
   - Pętla for
3. Pętle zagnieżdżone

Wrocław University of Science and Technology

Operatory przypisania

w Pythonie

Operatory przypisania w Pythonie używane są do przypisania wartości do zmiennej.
Najprostszym rodzajem przypisania, jest taki zapis x = 3, czyli do zmiennej x została przypisana wartości 3. Możemy jednak ten zapis trochę skomplikować. W poniższej tabeli druga kolumna pokazuje „skrócony” zapis działania z wykorzystaniem operatora przypisania. Trzecia kolumna pokazuje jak można to samo działanie zapisać w formie pełnej.

Operator złożony	Przykład działania	Równoważny zapis działania
+=	x += 2	x = x + 2
-=	x -= 2	x = x - 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
//=	x //= 2	x = x // 2
%=	x %= 2	x = x % 2
**=	x **= 2	x = x ** 2

Wrocław University of Science and Technology

Wprowadzenie do pętli

Chcemy napisać program, który będzie umiał obliczyć średnią niezależnie od liczby ocen, jaką będzie chciał podać użytkownik. Nie chcemy kodu na sztywno zawsze obliczającego jej z 3 czy 5 liczb, ale z n liczb. Wiąże się z tym zagadnieniem problem powtarzania kodu: możemy napisać 5 razy pod rząd kod, który prosi o podanie liczby przez użytkownika, a następnie dodaje ją do sumy. Jednak jest to dokładnie tym, czego chcemy uniknąć - tutaj z pomocą przychodzą nam pętle. W języku Python mamy dwa rodzaje pętli: **for** oraz **while**.

```
print(0)
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
```

Typowym zadaniem wykonywanym przez programy komputerowe jest wielokrotne powtarzanie tej samej sekwencji poleceń.

Ciało pętli jest wcięte: indentacja (wcięcia) jest sposobem na grupowanie instrukcji. W trybie interaktywnym trzeba wprowadzić znak(i) spacji lub tabulacji, aby wciąć wiersz.

Wrocław University of Science and Technology

Pętla while

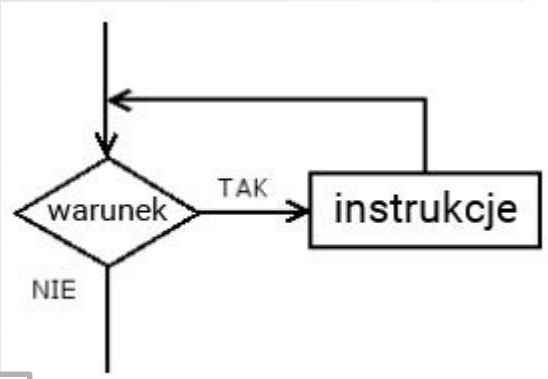
```
x = 2
while x < 5:
    print(x > 5)
    x += 1
print(x)
print("Koniec")
```

słowo kluczowe

dwukropek

warunek

wnętrze instrukcji while (wcięcie)




```
while true_or_not:
    wykonuj_gdy_true
```

- Instrukcja warunkowa **if** wykona się raz gdy warunek jest spełniony, natomiast instrukcja while będzie wykonywała zawarty w niej kod tak długo aż warunek postawiony przy niej będzie spełniony.
- Wnętrze pętli wydzielamy wcięciem (ciało pętli).
- Jeśli warunek od samego początku będzie fałszywy to ciało pętli nie wykona się ani razu.
- Dla tego typu pętli najczęściej zmieniamy wartość pewnej zmiennej wewnątrz pętli i w ten sposób w pewnym momencie warunek przestaje być spełniony

Wrocław University of Science and Technology

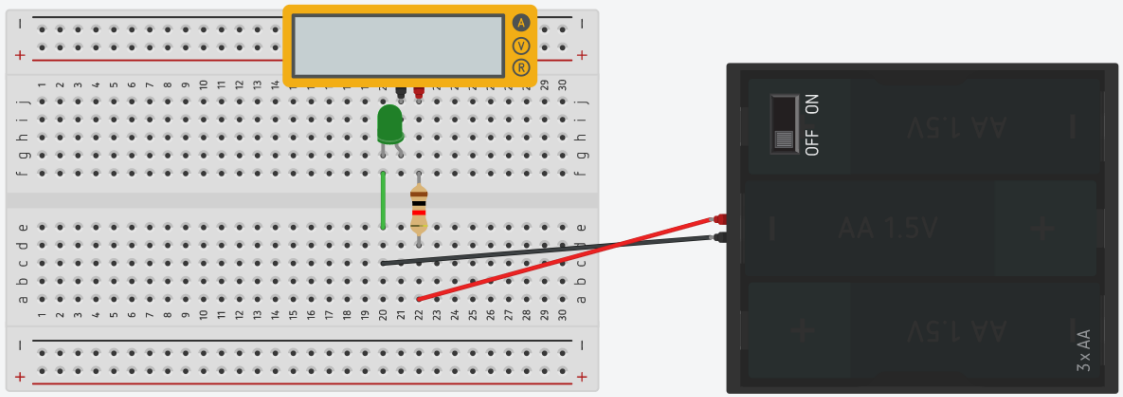
Pętla while – pętla nieskończona

Może się zdarzyć, że nasza pętla będzie nieskończona (np. przez pomyłkę nie zmodyfikujemy warunku, lub tak ma właśnie być).

```
while True:
    print("Utknąłem w pietli!"
```

Ten fragment spowoduje wypisywanie w nieskończoność napisu "Utknąłem w pętli!" – jak to zatrzymać? Ctrl-C

KeyboardInterrupt



Wrocław University of Science and Technology

Pętla while

Przykład 9.1 - Jak zakończyć odliczanie?

Załóżmy, że chcemy napisać program, który zlicza ile parzystych i nieparzystych liczb dodatnich podał użytkownik. Jak się za to zabierzesz?

Pytamy o liczbę liczb

```
# Kończymy gdy dostajemy 0
odd_numbers = 0
even_numbers = 0

# zapytaj o liczbę liczb
n = int(input("n: "))

# iterator do odliczania
i = 0
# wpisanie n liczb kończy działanie
while i < n:
  # odczyt kolejnej liczby
  number = int(input("Podaj liczbę: "))
  # sprawdź nieparzystość
  if number % 2 == 1:
      # zwiększamy licznik nieparzysty
      odd_numbers += 1
  else:
      # zwiększamy licznik parzysty
      even_numbers += 1
  # zwiększamy iterator o 1
  i += 1


# Pokaż wynik
print("Nieparzyste:", odd_numbers)
print("Parzyste:", even_numbers)
```

Pytamy po każdej iteracji

```
# Kończymy gdy dostajemy 0
odd_numbers = 0
even_numbers = 0

# odpowiedź do sprawdzenia
odp = True
# wpisanie n liczb kończy działanie
while odp:
  # odczyt liczby
  number = int(input("Podaj liczbę: "))
  # sprawdź nieparzystość
  if number % 2 == 1:
      # zwiększamy licznik nieparzysty
      odd_numbers += 1
  else:
      # zwiększamy licznik
parzysty
      even_numbers += 1
  # pytamy czy koniec
  odp = int(input(„Czy kończymy? "))

# Pokaż wynik
print("Nieparzyste:", odd_numbers)
print("Parzyste:", even_numbers)
```

Wartownik

```
# Kończymy gdy dostajemy 0

odd_numbers = 0
even_numbers = 0

# odczytaj pierwszą liczbę
number = int(input("Liczba: "))

# 0 kończy działanie
while number != 0:
  # sprawdź nieparzystość
  if number % 2 == 1:
      # zwiększamy licznik nieparzysty
      odd_numbers += 1
  else:
      # zwiększamy licznik parzysty
      even_numbers += 1
  # odczyt kolejnej liczby
  number = int(input("Podaj liczbę lub 0: "))

# Pokaż wynik
print("Nieparzyste:", odd_numbers)
print("Parzyste:", even_numbers)
```

Wrocław University of Science and Technology

Pętla while – jak zakończyć odliczanie?

Do wczytywania danych o różnych długościach możemy:

- na koniec każdej iteracji zapytać użytkownika, czy chce kontynuować,
- na początku programu pytamy użytkownika, o długość danych.

Dla długich list obie techniki są niewygodne:

- w pierwszej trzeba wielokrotnie potwierdzać kontynuację,
- w drugiej użytkownik musi sam przeliczyć elementy.

Rozwiązaniem może być wykorzystanie zmiennej o specjalnej wartości - wartownika.

- Wartownik to zmienna, która przyjmuje specjalną (umówioną) wartość, która wskazuje na koniec listy elementów.
- Przykładowo: w programie obliczającym średnią ocen – wprowadzenie wartości 0 (która nie jest poprawną oceną) może oznaczać koniec danych.

Wrocław University of Science and Technology

Pętla for

```
x = 2
for i in [1, 2]:
    print(x+i)
print(x)
print("Koniec")
```

słowa kluczowe

dwukropek

wnętrze instrukcji for (wcięcie)

Możemy iterować po:
- dowolnej kolekcji
- napisie

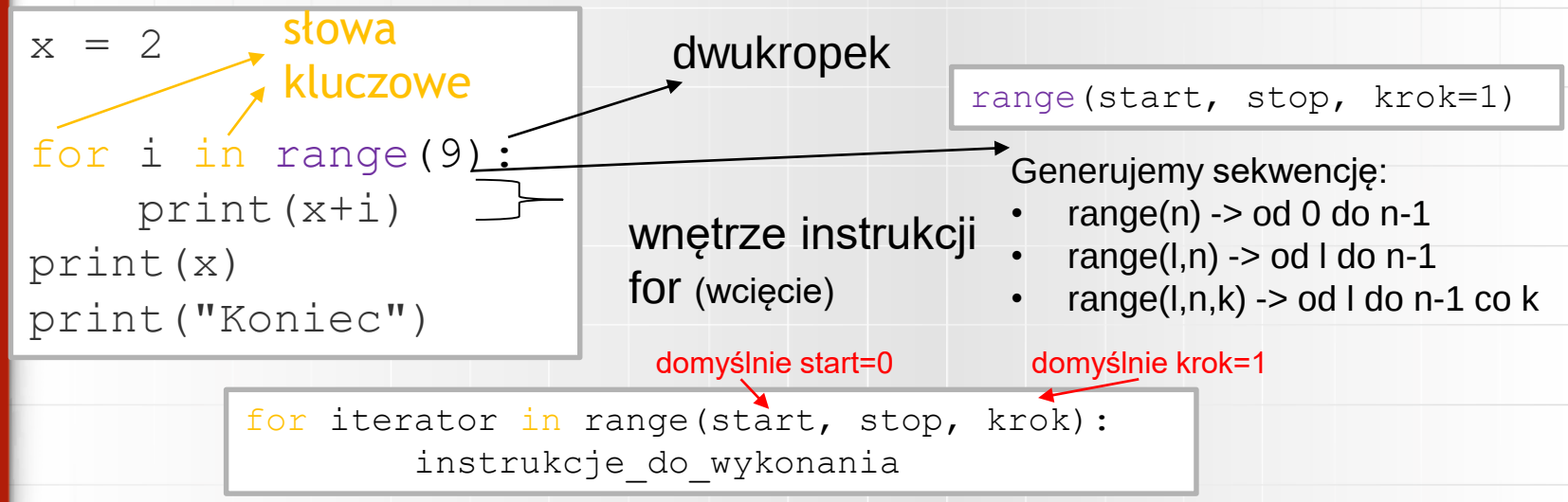
```
for i in "Ala":
    print(x)
print(x)
print("Koniec")
```

```
for iterator in sekwencja[lista,napis,tablica,słownik]:
        instrukcje_do_wykonania
```

- W przypadku pętli **for** z góry powinniśmy wiedzieć, ile razy instrukcje mają się wykonać.
- Zmienna i w powyższym przykładzie to liczba nazywana zmienną sterującą pętlą (albo iteratorem pętli). Automatycznie przyjmuje ona kolejne wartości z kolekcji/napisu pojawiającego się przed znakiem : (operator **in**).
- Jeśli warunek od samego początku będzie fałszywy to ciało pętli nie wykona się ani razu.

Wrocław University of Science and Technology

Generator sekwencji range()



- **range()** jako argumenty przyjmuje jedynie liczby całkowite: jedną (tylko końcową), dwie (początek i koniec), trzy (początek, koniec i krok).
- generator **range()** tworzy sekwencję, którą można zrzutować na sekwencję - listę

https://docs.python.org/3/library/stdtypes.html#typesseq-range

Wrocław University of Science and Technology

Pętla for

Przykład 9.2 – Iterowanie po napisie

Sprawdź ile znaków w podanym przez użytkownika słowie jest samogłoską.

```
def main():
    napis = input("Podaj swój wyraz: ")
    counter = 0
    for l in napis.lower():
        if l in ['a','o','i','y','e','u']:
            counter += 1
    print("Liczba samogłosek:", counter)

main()
```

Wrocław University of Science and Technology

Pętla for

Przykład 9.3 – Iterowanie po range()

Wypisz kolejne potęgi 2 (do 10).

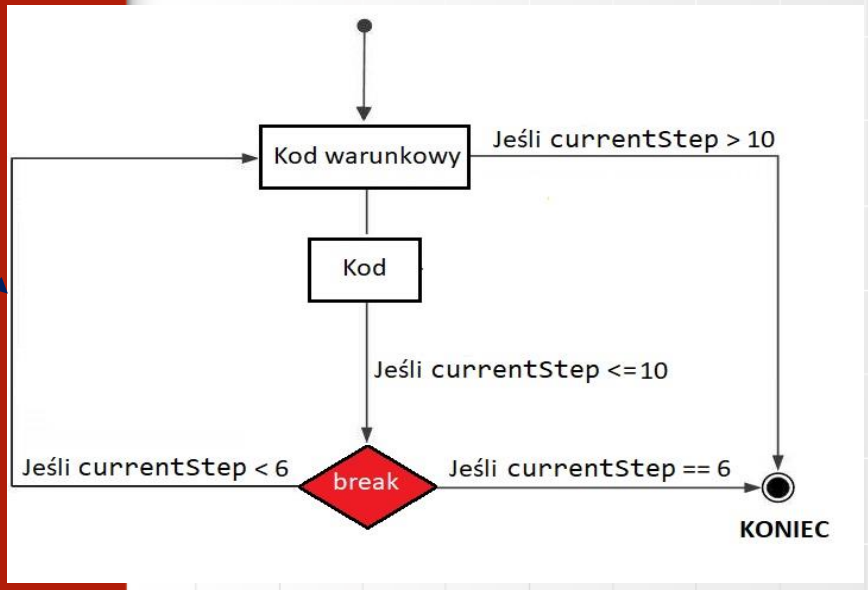
```
def main():
    pow = 1
    for exp in range(11):
        print("2 do potęgi", exp, "to", pow)
        pow *= 2
main()
```

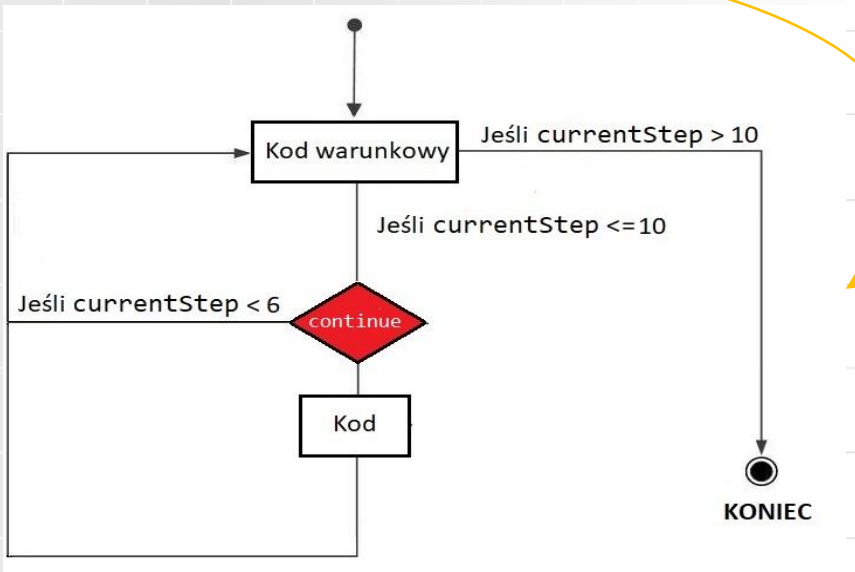
Wrocław University of Science and Technology

Instrukcje "*break*" i "*continue*"

Do tej pory traktowaliśmy całą pętlę jako niepodzielną i nierozdzielną sekwencję instrukcji, które są wykonywane na każdym kroku pętli. Jednak jako programista możesz stanąć przed następującymi wyborami:

- wydaje się, że nie ma potrzeby kontynuowania pętli jako całości; powinieneś powstrzymać się od dalszego wykonywania ciała pętli i przejść do wykonania kodu za nią; -> break
- wygląda na to, że musisz rozpocząć następną iterację pętli bez kończenia wykonywania bieżącej tury. -> continue





https://mateuszrus.pl/struktury-danych-w-python/

Wrocław University of Science and Technology

Instrukcje "break" i "continue"

- przykład 9.4 – zatrzymywanie wykonania pętli

Za pomocą pętli wypisz wszystkie liczby parzyste z tablicy liczby naturalnych mniejszych od 6 w takiej samej kolejności, w jakiej zostały w niej zapisane (lista = [5, 0, 1, 2, 4, 3]). Nie wypisuj żadnej liczby, która w tej tablicy znajduje się za liczbą 2.

```
def main():
    lista = [5, 0, 1, 2, 4, 3]
    for i in lista:
        if i == 2:
            break
        if not (i % 2):
            print(i)
main()
```

Wrocław University of Science and Technology

Pętle i instrukcja else

Obie pętle, **while** i **for**, mają jeden interesujący (i rzadko używany) dodatek.

Pętle mogą mieć również gałąź **else**, podobnie jak instrukcje warunkowe **if**.

Gałąź **else** przy pętli jest zawsze wykonywana raz, tuż po zakończeniu wykonywania się ciała pętli – chyba że działanie pętli zostało przerwane instrukcją **break**.

```
i = 1
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)
```

```
for i in range(5):
    print(i)
else:
    print("else:", i)
```

```
i = 111
for i in range(2, 1):
    print(i)
else:
    print("else:", i)
```

Wrocław University of Science and Technology

Zagnieżdżanie pętli

```
for h in range(60):
    for m in range(60):
        for s in range(60):
            print(h, m, s, sep=':')
```

wnętrze zewnętrznej pętli (wcięcie)

wnętrze kolejnej pętli wewnętrznej (wcięcie)

wnętrze kolejnej pętli wewnętrznej (wcięcie)

Wewnątrz pętli można także umieszczać pętle, a taka pętla nazywana jest pętlą zagnieżdżoną. Przykład pętli zagnieżdżonej – zegar: odlicza w pętli godziny, ale na każdą godzinę przypada pętla odliczająca minuty, a w każdej minucie jest pętla odliczająca sekundy.

Zmienna sterująca pętlą wewnętrzną może zależeć od wartości zmiennej kontrolującej pętlę zewnętrzną

```
# Trójkąt prostokątny z *
def jedna_linijka(n):
    for i in range(n):
        print('*', end="")

def main():
    h = int(input("Podaj wysokosc: "))
    for i in range(h):
        jedna_linijka(i + 1)
        print("")

main()
```

Wrocław University of Science and Technology

Pętle zagnieżdżone

Przykład 9.5 – Jak sprawdzić poprawność danych?

Załóżmy, że chcemy napisać program, który liczy średnią ocen wprowadzanych przez użytkownika. Zauważ, że oceny to liczby, które należą do zbioru {2, 3, 3.5, 4, 4.5, 5, 5.5} . Nie dopuść do wprowadzenia złej liczby.

```
def main():
    n = int(input("Podaj liczbę ocen: "))
    ocena = 0
    suma = 0
    for i in range(n):
        while ocena not in [2, 3, 3.5, 4, 4.5, 5, 5.5]:
            ocena = float(input(f"Podaj ocenę {i+1}.: "))
            if ocena not in [2, 3, 3.5, 4, 4.5, 5, 5.5]:
                print("Niewłaściwa liczba!")
        suma += ocena
        ocena = 0
    print("Twoja średnia:", format(suma/n, '.1f'))

main()
```

Wrocław University of Science and Technology

Instrukcje "*break*" i "*continue*"

Przykład 9.6 – Zjadanie liter

Załóżmy, że bawimy się w grę pomidor, ale z trochę zmodyfikowanymi zasadami. Napisz program, który ciągle pyta „A co jeśli?” i użytkownik powinien odpowiedzieć „pomidor”, co wyświetlamy na ekranie. Gra się toczy tak długo aż użytkownik nie wypowie innego słowa, wtedy komputer powinien je wyświetlić bez samogłosek.

```
def main():
    while True:
        odp = input("A co jeśli? ")
        if odp.lower() != "pomidor":
            for letter in odp:
                if letter in ['a','o','u','e','i','y']:
                    continue
                print(letter, end="")
            break

main()
```

Wrocław University of Science and Technology

Rzeczy do zapamiętania!

1. W języku python mamy dwa rodzaje pętli:
   - pętle **while**, która wykonuje się tak długo jak spełniony jest postawiony dla niej warunek
   - pętle **for**, która wykonuje się zadaną wcześniej liczbę razy, często jest wykorzystywana do przejścia po sekwencji lub napisie lub wykorzystując wbudowany generator **range**().
2. Sterowanie wykonania instrukcji zawartych w pętli może dodatkowo być wsparte poprzez dwa polecenia:
   - **break** jest używany do zakończenia pętli **for** lub **while**,
   - podczas gdy **continue** pozwala opuścić blok instrukcji poniżej niej i wrócić do nagłówka pętli (omijamy kontynuację danej iteracji).
3. W pythonie pętle **while** i for mogą dodatkowo posiadać wyrażenie **else**. Instrukcje zamieszczone w ciele **else** wykonają się zaraz po zakończeniu przebiegu pętli, o ile sama ona nie została przerwana instrukcją **break**.
4. Generator **range**() generuje sekwencję liczb – przyjmuje jako argumenty jedynie liczby całkowite i zwraca obiekt typu **range** (ale możemy go zrzutować na listę) – wywołanie wygląda następująco **range**(start, stop, krok).