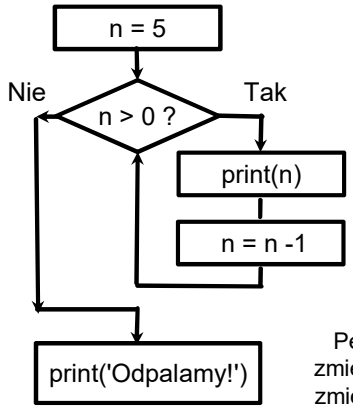
Pętle i iteracje

Rozdział 5

UMSI

Python dla wszystkich
www.py4e.pl



Powtarzane kroki

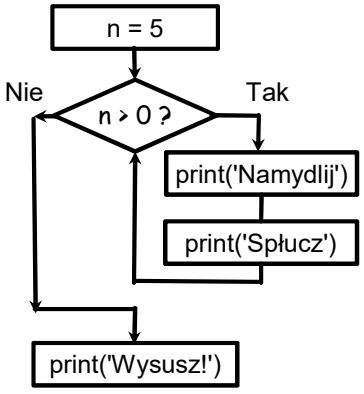
Program:

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Odpalamy!')
print(n)
```

Wyjście:

```
5
4
3
2
1
Odpalamy!
0
```

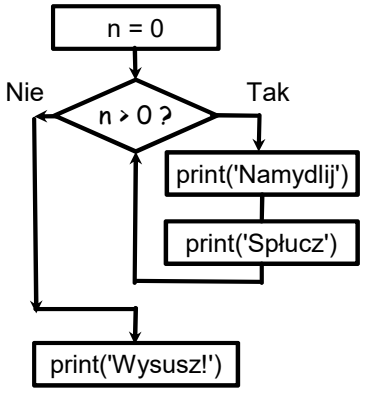
Pętle (powtarzane kroki) mają zmienne sterujące zmieniające się z każdym przejściem pętli. Często te zmienne sterujące przyjmują wartości kolejnych cyfr.



Pętla nieskończona

```
n = 5
while n > 0 :
    print('Namydlij')
    print('Spłucz')
print('Wysusz!')
```

Co jest nie tak w tej pętli?



Inna pętla

```
n = 0
while n > 0 :
    print('Namydlij')
    print('Spłucz')
print('Wysusz!')
```

Co robi ta pętla?

Wyskakiwanie z pętli

- Instrukcja break kończy bieżącą pętlę i wyskakuje do instrukcji znajdującej się bezpośrednio za pętlą
- Przypomina to test pętli, który można zastosować w dowolnym miejscu jej ciała

```
while True:
    line = input('> ')
    if line == 'zrobione' :
        break
    print(line)
print('Zrobione!')
```

```
> no hejka
no hejka
>
zakończone
zakończone
> zrobione
Zrobione!
```

Wyskakiwanie z pętli

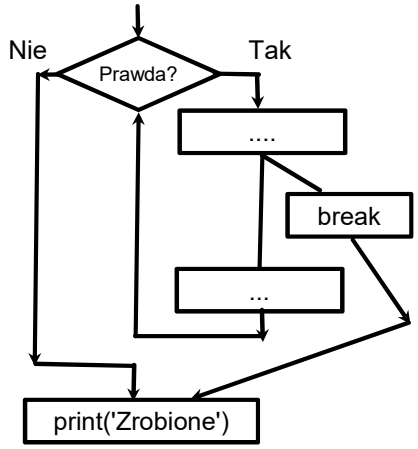
- Instrukcja break kończy bieżącą pętlę i wyskakuje do instrukcji znajdującej się bezpośrednio za pętlą
- Przypomina to test pętli, który można zastosować w dowolnym miejscu jej ciała

```
while True:
    line = input('> ')
    if line == 'zrobione' :
        break
    print(line)
print('Zrobione!')
```

```
> no hejka
no hejka
>
zakończone
zakończone
> zrobione
Zrobione!
```

```
while True:
    line = input('> ')
    if line == 'zrobione' :
        break
    print(line)
print('Zrobione!')
```

https://pl.wikipedia.org/wiki/Transporter_(Star_Trek)



Kończenie iteracji przy pomocy continue

Instrukcja continue kończy bieżącą iterację, przeskakuje do początku pętli i zaczyna kolejną iterację

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'zrobione' :
        break
    print(line)
print('Zrobione!')
```

```
> no hejka
no hejka
> # nie wypisuj tego
> wypisz to!
wypisz to!
> zrobione
Zrobione!
```

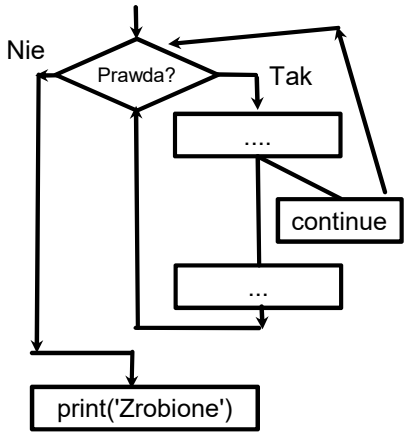
Kończenie iteracji przy pomocy continue

Instrukcja continue kończy bieżącą iterację, przeskakuje do początku pętli i zaczyna kolejną iteracje

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'zrobione' :
        break
    print(line)
print('Zrobione!')
```

```
> no hejka
no hejka
> # nie wypisuj tego
> wypisz to!
wypisz to!
> zrobione
Zrobione!
```

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'zrobione' :
        break
    print(line)
print('Zrobione!')
```



Nieokreślone pętle

- Pętle 'while' nazywamy “pętlami nieokreślonymi”, ponieważ powtarzają się tak długo, aż określony warunek logiczny stanie się fałszywy
- W pętlach, które widzieliśmy do tej pory, dość łatwo stwierdzić, czy się zakończą, czy będą “pętlami nieskończonymi”
- Czasami jest nieco trudniej stwierdzić, czy pętla się zakończy

Określone pętle

Iteracja po zbiorze elementów

Określone pętle

- Często mamy listę elementów, linii w pliku – czyli skończony zbiór elementów
- Możemy napisać pętlę, która wykona jedną iterację dla każdego elementu zbioru za pomocą konstrukcji for
- Takie pętle nazywamy “określonymi”, bo są wykonywane określoną liczbę razy
- Mówimy, że “pętle określone przeprocesowują elementy zbioru”

Prosta pętla określona

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Odpalamy!')
```

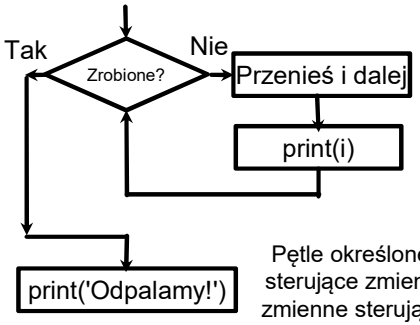
5
4
3
2
1
Odpalamy!

Pętla określona z ciągami znaków

```
friends = ['Józek', 'Gienek', 'Staszek']
for friend in friends:
    print('Szczęśliwego Nowego Roku:', friend)
print('Zrobione!')
```

Szczęśliwego Nowego Roku: Józek
Szczęśliwego Nowego Roku: Gienek
Szczęśliwego Nowego Roku: Staszek
Zrobione!

Prosta pętla określona




```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Odpalamy!')
```

5
4
3
2
1
Odpalamy!

Pętle określone (pętle for) mają zdefiniowane zmienne sterujące zmieniające się z każdym przejściem pętli. Te zmienne sterujące przechodzą po kolei przez sekwencję lub zbiór.

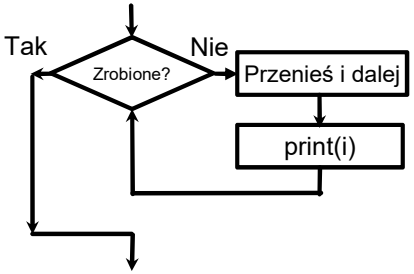
Przyjrzyjmy się 'in'...

- zmienna sterująca “przechodzi” przez sekwencję (uporządkowany zbiór)
- Blok (ciało) kodu jest wykonywany jeden raz dla każdego elementu w sekwencji
- Zmienna sterująca przechodzi przez wszystkie elementy w sekwencji

Zmienna sterująca

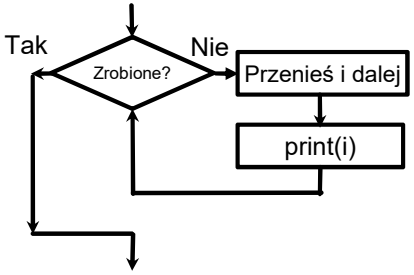
Pięcioelementowa sekwencja

```
for i in [5, 4, 3, 2, 1] :
    print(i)
```

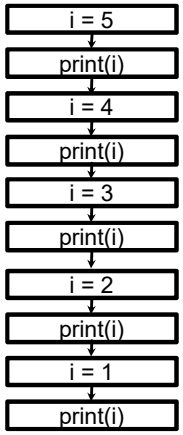



```
for i in [5, 4, 3, 2, 1] :
    print(i)
```

- Zmienna sterująca “przechodzi” przez sekwencję (uporządkowany zbiór)
- Blok (ciało) kodu jest wykonywany jeden raz dla każdego elementu w sekwencji
- Zmienna sterująca przechodzi przez wszystkie elementy w sekwencji




```
for i in [5, 4, 3, 2, 1] :
    print(i)
```



Idiomy pętli: co robimy w pętlach

Uwaga: Mimo że przykłady są proste, te same schematy stosujemy w każdym rodzaju pętli

Tworzenie “inteligentnych” pętli

Sztuczka polega na tym, żeby “wiedzieć” coś o całej pętli, kiedy musisz pisać kod, który widzi tylko jeden element naraz

Nadaj niektórym zmiennym wartości początkowe

for element in dane:

Poszukaj czegoś w każdym elemencie z osobna lub zrób z nim coś i zaktualizuj zmienną

Spójrz na zmienne

Przechodzenie pętlą przez zbiór

```
print('Przed')
for element in [9, 41, 12, 3, 74, 15] :
    print(element)
print('Po')
```

```
$ python basicloop.py
Przed
9
41
12
3
74
15
Po
```

Która liczba jest największa?

Która liczba jest największa?

9

Która liczba jest największa?

41

Która liczba jest największa?

12

Która liczba jest największa?

3

Która liczba jest największa?

74

Która liczba jest największa?

15

Która liczba jest największa?

Która liczba jest największa?

9 41 12 3 74 15

Która liczba jest największa?

largest_so_far -1

Która liczba jest największa?

9

largest_so_far 9

Która liczba jest największa?

41

largest_so_far 41

Która liczba jest największa?

12

largest_so_far 41

Która liczba jest największa?

3

largest_so_far 41

Która liczba jest największa?

74

largest_so_far | 74

Która liczba jest największa?

15

74

Która liczba jest największa?

9 41 12 3 74 15

74

Znajdowanie największej wartości

```
largest_so_far = -1
print('Przed:', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('Po:', largest_so_far)
```

```
$ python3 largest_sf.py
Przed: -1
9  9
41  41
41  12
41  3
74  74
74  15
Po: 74
```

Tworzymy zmienną, która zawiera największa widzianą wartość. Jeśli aktualna liczba, na którą patrzymy, jest większa, to staje się największą widzianą wartością.

Więcej schematów pętli...

Liczenie w pętli

```
zork = 0
print('Przed:', zork)
for element in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, element)
print('Po:', zork)
```

```
$ python3 countloop.py
Przed: 0
1 9
2 41
3 12
4 3
5 74
6 15
Po: 6
```

Aby policzyć, ile razy wykonaliśmy pętlę, wprowadzamy zmienną licznika, zaczynając z wartością 0 i dodajemy 1 z każdym wykonaniem pętli.

Sumowanie w pętli

```
zork = 0
print('Przed:', zork)
for element in [9, 41, 12, 3, 74, 15] :
    zork = zork + element
    print(zork, element)
print('Po:', zork)
```

```
$ python3 sumloop.py
Przed: 0
9 9
50 41
62 12
65 3
139 74
154 15
Po: 154
```

Aby dodać wartość napotkaną w pętli, wprowadzamy zmienną sumowania, zaczynając od 0 i dodajemy aktualną wartość z każdym wykonaniem pętli.

Znajdowanie średniej w pętli

```
count = 0
sum = 0
print('Przed:', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('Po:', count, sum, sum / count)
```

```
$ python3 averageloop.py
Przed: 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
Po: 6 154 25.666
```

Średnia po prostu łączy schematy zliczania i sumowania i wykonuje dzielenie po zakończeniu pętli.

Filtrowanie w pętli

```
print('Przed')
for value in [9, 41, 12, 41, 74, 15] :
    if value > 20:
        print('Duża liczba:', value)
print('Po')
```

```
$ python3 search_filter.py
Przed
Duża liczba: 41
Duża liczba: 74
Po
```

Korzystamy z instrukcji if w pętli, żeby wyłapać/ filtrować szukane wartości.

Wyszukiwanie z użyciem zmiennej logicznej

```
found = False
print('Przed:', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print(found, value)
print('Po:', found)
```

```
$ python3 find_value.py
Przed: False
False 9
False 41
False 12
True 3
True 74
True 15
Po: True
```

Jeśli chcemy wyszukać z powiadomieniem o znalezieniu, korzystamy ze zmiennej z początkową wartością False i ustawiamy ją na True, gdy tylko znajdziemy szukaną watość.

Jak znaleźć najmniejszą wartość

```
largest_so_far = -1
print('Przed:', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('Po:', largest_so_far)
```

```
$ python3 largest_sf.py
Przed: -1
9  9
41  41
41  12
41  3
74  74
74  15
Po: 74
```

Co trzeba zmienić, żeby wyszukać najmniejszą wartość na liście?

Znajdowanie najmniejszej wartości

```
smallest_so_far = -1
print('Przed:', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('Po:', smallest_so_far)
```

Zmieniliśmy nazwę zmiennej na smallest_so_far (na razie najmniejsza) i znak > na <

Znajdowanie najmniejszej wartości

```
smallest_so_far = -1
print('Przed:', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('Po:', smallest_so_far)
```

```
$ python3 smallbad.py
Przed: -1
-1 9
-1 41
-1 12
-1 3
-1 74
-1 15
Po: -1
```

Zmieniliśmy nazwę zmiennej na smallest_so_far (na razie najmniejsza) i znak > na <

Znajdowanie najmniejszej wartości

```
smallest = None
print('Przed')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)

print('Po:', smallest)
```

```
$ python3 smallest.py
Przed
9 9
9 41
9 12
3 3
3 74
3 15
Po: 3
```

Nadal mamy zmienną, która jest najmniejsza na razie. W pierwszej iteracji pętli smallest ma wartość None, więc przypisujemy jej pierwszą wartość jako najmniejszą.

Operatory is i is not

```
smallest = None
print('Przed')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)

print('Po', smallest)
```

- Python ma operator is (jest), który można stosować w wyrażeniach logicznych
- Oznacza tyle, co “jest tym samym, co”
- Podobnie, ale dobitniej niż ==
- is not (nie jest) to również operator logiczny

Podsumowanie

- Pętle while (nieokreślone)
- Nieskończone pętle
- Używanie break
- Używanie continue
- Stałe i zmienne None
- Pętle for (określone)
- Zmienne sterujące
- Idiomy pętli
- Największa i najmniejsza wartość

Podziękowania dla współpracowników

Autorstwo pierwszej wersji: Charles Severance, University of Michigan School of Information

Polska wersja powstała z inicjatywy Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu

Tłumaczenie: Agata i Krzysztof Wierzbiccy, EnglishT.eu
Poprawki: Andrzej Wójtowicz

... wstaw tu nowych współpracowników i tłumaczy

...