Python: pętle i funkcje

Pętla - sekwencja instrukcji (blok kodu), który jest wykonywany zadaną liczbę razy lub dopóki określony warunek jest spełniony.

Przykład pętli - wykonać i przeanalizować wynik:

```
for i in range(5):
    print("Hello world")
```

W języku python mamy dwa rodzaje pętli: `for` oraz `while`.

Pętla “for”

Wykonuje zadany blok kodu dla każdego elementu sekwencji (listy). Blok określony jest za pomocą wcięcia.

Przykład - wykonać i przeanalizować:

```
numbers = [2, 3, 5, 7]
for n in numbers:
    print(n)
```

powyższy kod można by przeczytać w następujący sposób:
dla każdego elementu “n” z listy “numbers” wykonaj: …

Aby wygodnie podać sekwencję, dla której pętla ma się wykonać, można użyć funkcji `range`. Zwraca on tzw. Iterator - nie generuje całej listy, lecz po pojedyncze elementy sekwencji po kolei.

Przeanalizować działanie funkcji range:

```
for x in range(5):
    print(x)

for x in range(3, 6):
    print(x)

for x in range(3, 8, 2):
    print(x)

for x in range(100, 0, -5):
    print(x)
```

Pętla “while”

Wykonuje zadany blok kodu, dopóki określony warunek logiczny jest spełniony.

Przykład pętli - wykonać i przeanalizować wynik:

```
count = 0
while count < 5:
    print(count)
    count += 1  # This is the same as count = count + 1
```

Uwaga! Jeśli zdarzy się, że warunek pozostanie prawdziwy zawsze, pętla nigdy się nie skończy! Rozwiązaniem może być instrukcja `break` (patrz poniżej).

Wykonać i przeanalizować:

```
while 5 > 3:
    print('infinite loop')
```

co jest równoważne:

```
while True:
    print('infinite loop')
```

Program można zatrzymać kombinacją klawiszy Ctrl+C:

Instrukcje “break” oraz “continue”

Instrukcja `break` służy do natychmiastowego przerwania pętli.

Wykonać i przeanalizować:

```
count = 0
while True:
    print(count)
    count += 1
    if count >= 5:
        break
```

Instrukcja `continue` służy do zaniechania wykonywania bieżącego obiegu pętli (bloku kodu) i natychmiastowe przejście do następnego obiegu.

Wykonać i przeanalizować:

```
for x in range(10):
    # Check if x is even
    if x % 2 == 0:
        continue
    print(x)
```

Funkcje

Funkcje służą do grupowania kawałków kodu wykonującego określone zadanie i łatwego jego re-używania. Do funkcji można przekazać dane za pomocą parametrów. Funkcja może zwracać wyliczoną wartość. Funkcja zostanie wykonana jedynie wtedy, gdy zostanie “wywołana” z kodu.

Wykonać i przeanalizować:

```
def square(x):
    return x*x

def cube(x):
    return x*x*x

print(square(3))
print(cube(3))
```

W pythonie zmienna może również przechowywać wskazanie na funkcję. Możemy ją zawołać tak samo jak funkcję, używając nazwy zmiennej z nawiasami okrągłymi i ew. argumentami.

Wykonać i przeanalizować:

```
def make_double(x):
   return 2 * x


def square(x):
   return x * x
```

```
def calculate(function, x):
   return function(x)

print(calculate(make_double, 5))
print(calculate(square, 5))
```

Zadania

Szkielet kodu - dotyczy wszystkich zadań:

```
numbers = [15, 21, 40, 6, 18, 44, 3, 33, 28, 21, 4, 26, 26, 19, 2, 38, 23, 25, 16, 30]
...
print(...)
```

Zadanie 1: znajdź najmniejszą liczbę w liście i wypisz ją na ekran - użyj pętli.

Oczekiwany wynik: 2

Zadanie 2: zmodyfikuj **zadanie 1** aby również podawało pozycję najmniejszej liczby (**nie** używaj funkcji index)

Oczekiwany wynik: 2 (pozycja 14.)

Zadanie 3: policz średnią **arytmetyczną** wszystkich liczb w liście i wypisz ją - użyj pętli.

Oczekiwany wynik: 21.9

Zadanie 4: policz średnią **geometryczną** wszystkich liczb w liście i wypisz ją - użyj pętli.

Oczekiwany wynik: 3.1009506106059677

Zadanie 5: przetwórz listę liczb w taki sposób, że za każdą z liczb x podstaw wartość funkcji **f(x) = -x^3 + 20x^2 - 13x + 78** - użyj funkcji oraz pętli.

Oczekiwany wynik: [1008, -636, -32442, 504, 492, -46958, 192, -14508, -6558, -636, 282, -4316, -4316, 192, 124, -26408, -1808, -3372, 894, -9312]

Zadanie 6: posortuj liczby w liście i wypisz posortowaną listę - użyj dwóch zagnieżdżonych pętli.

Oczekiwany wynik: [2, 3, 4, 6, 15, 16, 18, 19, 21, 21, 23, 25, 26, 26, 28, 30, 33, 38, 40, 44]

Zadanie 7: wydziel logikę porównywania liczb z zadania 6 do funkcji `compare(a, b)`, która zwraca `True` jeśli liczby są w odpowiedniej kolejności (`a` powinno być przed `b`), lub `False` jeśli nie.

Zadanie 8: przerób program z zadania 7 tak, aby sortował malejąco

Zadanie 9: Przerób program z zadania 7 tak, aby logika sortowania była zamknięta w funkcji `sort`, która przyjmuje listę do posortowania i funkcję porównującą, oraz nie zwraca nic (*pytanie: dlaczego funkcja nie musi nic zwracać?*). Zaimplementuj funkcje `is_lower` i `is_greater`, służące do porównywania. Wywołaj swoje funkcje i wypisz wyniki w następujący sposób:

```
sort(numbers, is_lower)
print(numbers)
sort(numbers, is_greater)
print(numbers)
```

Zadanie 10: Zaimplementuj funkcję `factorial` zwracającą silnię z liczby całkowitej. Wytestuj swój program:

```
print(factorial(0))
print(factorial(1))
print(factorial(2))
print(factorial(3))
print(factorial(5))
print(factorial(6))
```

Oczekiwany wynik: 1, 1, 2, 6, 120, 720

Zadanie 11: Przerób zadanie 10 aby używało **rekurencji**.

Zadanie 12: Zaimplementuj rekurencyjną funkcję `fibonacci`, wyliczającą wartość danego elementu ciągu fibonacciego: https://pl.wikipedia.org/wiki/Ci%C4%85g_Fibonacciego. Wytestuj swoją funkcję podobnie jak silnię.

Zadanie 13: Zaimplementuj dwa różne sposoby sortowania, spośród możliwości: bąbelkowe, przez wybieranie, przez wstawianie. Sprawdź czas działania swoich funkcji sortujących dla dużego zbioru liczb - wypełnij szkielet programu implementacją funkcji sortowania:

```
import time
import random

...

numbers = [random.randint(0, 100000000) for _ in range(1000000)]
numbers_copy = numbers[:]

start = time.time()
bubble_sort(numbers, is_lower)
end = time.time()
print('Bubble: {0:.3f} seconds'.format(end - start))

start = time.time()
selection_sort(numbers_copy, is_lower)
end = time.time()
print('Selection: {0:.3f} seconds'.format(end - start))
```

Zadania dodatkowe: https://www.practicepython.org/