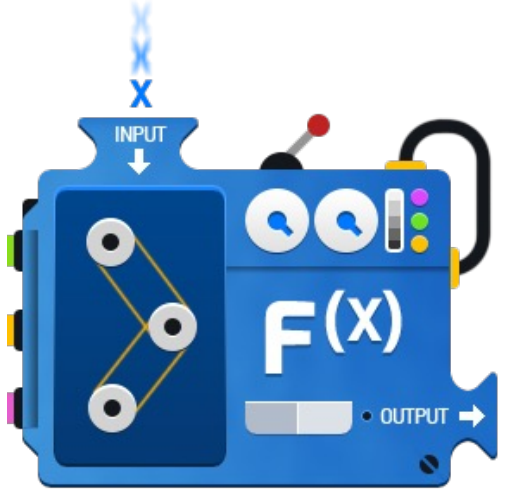
Funkcje - część 2

Podstawy programowania w języku Python



Listy i funkcje

- lista może zostać wysłana do funkcji jako argument
- lista także może być wynikiem funkcji

```python
def show_numbers(numbers):
    print(numbers)

show_numbers([1, 2, 3])
```

```python
def generate_numbers(n):
    numbers = []
    for i in range(n):
        numbers.insert(0, i)
    return numbers

print(generate_numbers(5))
```

Funkcje a zasięg zmiennych

- zmienne posiadają swój zasięg
- zmienne lokalne, tworzone wewnątrz funkcji mają zasięg funkcji
- zmienne globalne mają zasięg całego programu (modułu)
- zmienne lokalne o tej samej nazwie co zmienne globalne na obszarze funkcji mają przed nimi pierwszeństwo

```
1  def my_func():
2      x = 3 #zmienna o zasięgu lokalnym
3
4  x = 1 #zmienna o zasięgu globalnym
5  my_func()
6  print(x)
```

Instrukcja global

- zmienne zdefiniowane poza ciałem funkcji to zmienne globalne
- zmienne globalne mogą być używane zarówno wewnątrz jak i na zewnątrz funkcji
- aby umożliwić modyfikację zmiennej globalnej wewnątrz funkcji musimy zadeklarować ją z użyciem instrukcji **global**

```
1   def my_func():
2       global x #korzystając z instrukcji global
3       x = 3 #umożliwiamy nadpisanie zmiennej globalnej wewnątrz funkcji
4
5   x = 1 #zmienna o zasięgu globalnym
6   my_func()
7   print(x)
```

Przekazywanie argumentów do funkcji

- przekazywanie argumentów to sposób w jaki obiekty przesyłane są do funkcji w charakterze danych wejściowych
- argumenty przekazywane są przez automatyczne przypisanie obiektów do nazw zmiennych lokalnych
- nazwy argumentów w nagłówku funkcji stają się w czasie wykonania nowymi zmiennymi lokalnymi

```
1  def pass_value(a):
2      print(a)
3
4  pass_value(12) #przekazanie liczby
5  pass_value([1, 2, 3]) #przekazanie listy
```

Przekazywanie różnych typów argumentów

- przekazywanie argumentów do funkcji odbywa się zawsze przez wartość, przy czym ta wartość jest zawsze referencją do obiektu
- gdy do funkcji przekazujemy argumenty typów niezmiennych takich jak int, float, bool, str itp., zakładamy, że argumenty te przekazywane są przez wartość (podczas próby zmiany argumentu powstaje jego kopia)
- gdy do funkcji przekazujemy typy zmienne takie jak listy, zbiory, słowniki itp., przyjmujemy uproszczenie, że argumenty te przekazywane są przez referencję (podczas próby modyfikacji argumentu nie powstaje jego kopia)

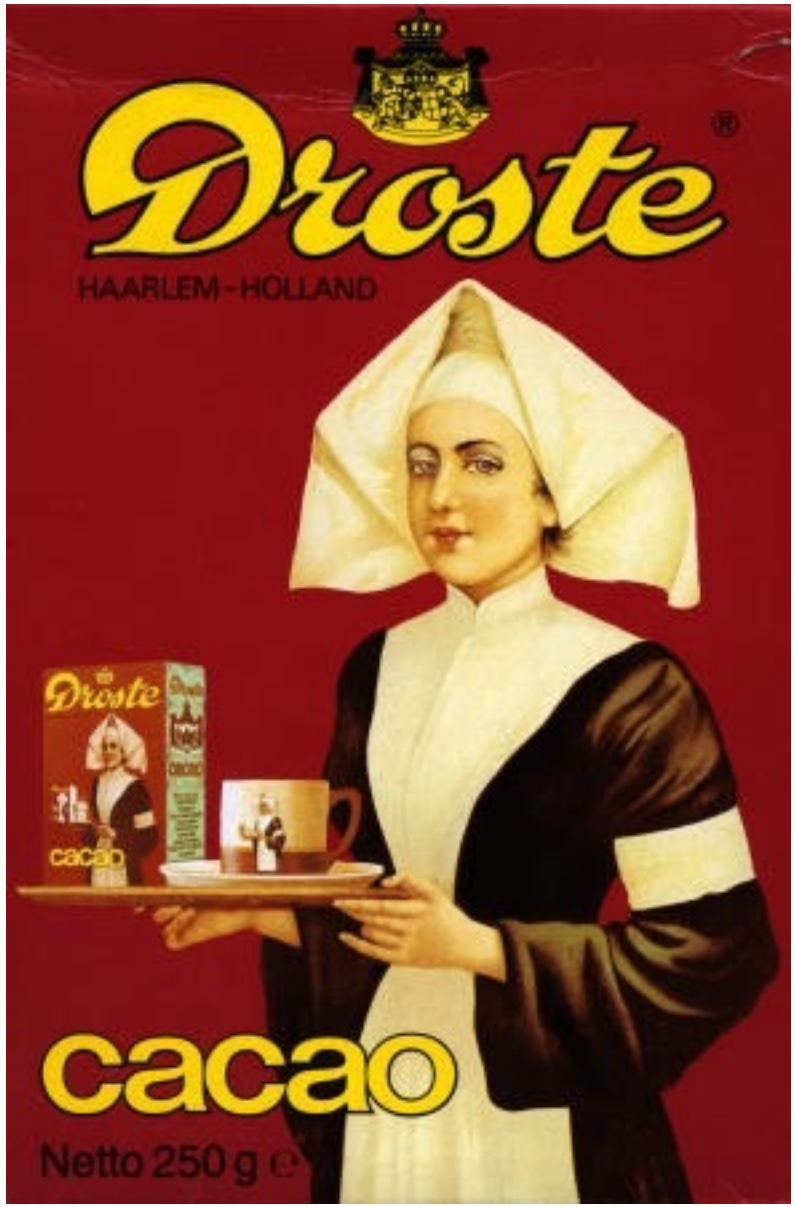
Rekurencja

- rekurencja (rekursja) to odwołanie funkcji do samej siebie

```python
def show_me_recursion(n):
    if n < 1:
        return
    print("recursion " * n)
    n -= 1
    show_me_recursion(n)

show_me_recursion(20)
```

Żeby zrozumieć rekurencję, trzeba zrozumieć rekurencję ;)



Pytanie

Czy zmienna globalna może być widoczna także w ciele funkcji?

a) nie, można ją tylko przysłonić
b) tak, ale należy zastosować instrukcję global
c) tak, zmienne globalne są dostępne lokalnie o ile nie zostaną przysłonięte
d) nie, tylko zmienna lokalna może być widoczna w ciele funkcji

Odpowiedź: c)

O ile nie przysłonimy zmiennej globalnej w ciele funkcji, a deklaracja zmiennej globalnej nastąpi przed wywołaniem funkcji to będzie ona dostępna także w funkcji.

Pytanie

Co wyświetli się na ekranie po wykonaniu poniższego skryptu?

```
a = 1

def fun():
    global b
    a = 2

fun()
print(a)
```

a) 2
b) 1 2
c) 1
d) [1, 2]
e) wystąpi błąd NameError

Odpowiedź: c)

Wywołanie funkcji nie wpływa na zmianę globalnej zmiennej a = 1.

Pytanie

Co wyświetli się na ekranie po wykonaniu poniższego skryptu?

```
a)  [1, 2, 3]
b)  []
c)  [9, 9, 9]
d)  wystąpi błąd
```

```
1  def change(numbers):
2      numbers = [9, 9, 9]
3      return numbers
4
5  numbers = [1, 2, 3]
6  change(numbers)
7  print(numbers)
```

Odpowiedź: a)

Funkcja nie dokonuje modyfikacji istniejącej listy lecz tworzy nową co nie wpływa na stan listy do której odwołujemy się przez globalną zmienną **numbers**.