Funkcje

- *Funkcja* jest narzędziem grupującym zbiór instrukcji w taki sposób, by mogły one być wykonane w programie więcej niż jeden raz.
- Funkcje są najważniejszą strukturą programu w Pythonie, służą do *maksymalizacji możliwości ponownego wykorzystania kodu* i *minimalizowania powtarzalności kodu*
- Funkcje są również narzędziem projektowym pozwalającym na rozbicie złożonych systemów, którymi łatwiej jest zarządzać – proceduralny podział na części
- Funkcje w Pythonie zapisywane są za pomocą nowej instrukcji *def* . W przeciwieństwie do funkcji z języków kompilowalnych *def* jest instrukcją wykonywalną – funkcja w Pythonie nie istnieje, dopóki Python nie dotrze do jej kodu i nie wykona instrukcji *def*
- Instrukcja *def* tworzy obiekt i przypisuje go do nazwy

Funkcje

- *Zalety użycia funkcji*
- Umieszczenie kodu w funkcji sprawia, że uzyskujemy narzędzie, które można wykonać tyle razy ile będzie to potrzebne
- Ponieważ kod wywołujący może przekazywać elementy dowolnego typu, funkcje mogą być uniwersalne i działać na różnych obiektach
- Kiedy chcemy zmienić sposób działania, wystarczy wprowadzić modyfikację w jednym miejscu
- Umieszczenie funkcji w pliku modułu sprawia, że można ją zaimportować i użyć jej ponownie w dowolnym programie

Funkcje

- Ogólny format instrukcji *def*

def *<nazwa>**(**arg1,arg2,…,argN**):***

<instrukcje>

- Wiersz nagłówka *def* określa nazwę funkcji przypisywaną do obiektu funkcji, a także listę *argumentów* (*parametrów*). Nazwy argumentów w nagłówku przypisywane są do obiektów przekazanych w nawiasach w momencie wywołania funkcji
- Ciało funkcji często zawiera instrukcję *return*

def *<nazwa>**(**arg1,arg2,…,argN**):***

return *<wartość>*

Instrukcja *return* może pojawić się w dowolnym miejscu ciała funkcji , kończy ona wywołanie funkcji i odsyła wyniki z powrotem do wywołującego. Jest ona opcjonalna, kiedy nie pojawia się, funkcja kończy się kiedy sterowanie wychodzi poza ciało funkcji

Funkcje

- Na funkcję składają się dwa elementy – *definicja* (instrukcja def tworząca funkcję) oraz *wywołanie* (wyrażenie nakazujące Pythonowi wykonanie ciała funkcji)
- Definicja funkcji o nazwie *times*, która zwraca iloczyn dwóch argumentów

```
>>> def times(x,y):
        return x*y
```

- Kiedy Python trafi w kodzie na def i wykona tę instrukcję, utworzy nowy obiekt funkcji zawierający jej kod i przypisze ten obiekt do nazwy times
- Po wykonaniu instrukcji def możemy wywołać (wykonać) funkcję w programie

```
>>> times(2,4)
8
>>> times(3.14,2)
6.28
>>> x=times('Ni',2)
>>> x
'NiNi'
```

Funkcje

- Przykład funkcji zbierającej elementy wspólne dla dwóch łańcuchów znaków

```
def intersect(seq1,seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res
s1 = ['aaa', 111, (4,5), 2.12]
s2 = [(4,5), 3.14]
s = intersect(s1,s2)
print(s)
s1 = 'mielonka'
s2 = 'biedronka'
s = intersect(s1,s2)
print(s)
```

```
[(4, 5)]
['i', 'e', 'o', 'n', 'k', 'a']
>>>
```

Funkcje

- *Zmienne lokalne*. W poprzednim przykładzie zmienna *res* jest zmienną lokalną czyli nazwą widoczną jedynie dla kodu wewnątrz definicji funkcji i istniejącą jedynie w czasie wykonania tej funkcji.
- Wszystkie nazwy przypisane wewnątrz funkcji a więc *seq1*, *seq2*, *res* i *x* są zmiennymi lokalnymi

```
def intersect(seq1,seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res
```

- Zmienne lokalne pojawiają się w momencie wywołania funkcji i znikają kiedy funkcja kończy swoje działanie, instrukcja return odsyła obiekt wynikowy jednak nazwa res znika

Funkcje – zakresy nazw

- Kiedy w programie użyjemy jakiejś nazwy, Python tworzy, zmienia lub wyszukuje nazwę w czymś co nazywamy *przestrzenią nazw* (*namespace*)
- Miejsce przypisania nazwy w kodzie określa zakres (*scope*) w jakim nazwa ta jest w kodzie widoczna
- Funkcje dodają do programu dodatkową warstwę przestrzeni nazw, oznacza to że do nazw zdefiniowanych wewnątrz instrukcji def nie możemy się odnosić poza tą funkcją
- Zmienne można przypisywać w trzech różnych miejscach, odpowiadającym trzem różnym zakresom
- Wewnątrz def – *zmienna lokalna*
- Wewnątrz def zawierającą inną funkcję – *zmienna nielokalna*
- Poza def – *zmienna globalna*

Funkcje – zakresy nazw

- Przykład

```
X = 99  #Zakres globalny
def  fun():
    #Zakres lokalny
    X = 88
    return X
print(fun())
print(X)
```

- *Pomimo że obie zmienne noszą nazwę X ich zakresy sprawiają że są one innymi obiektami*
- Funkcje definiują zakres lokalny natomiast moduły zakres globalny
- Te dwa zakresy wiążą się ze sobą w następujący sposób:

✓ Moduł zawierający funkcję jest zakresem globalnym

✓ Zakres globalny rozciąga się jedynie na jeden plik

✓ Każde wywołanie funkcji tworzy nowy zakres lokalny

✓ Przypisane nazwy są lokalne, o ile nie zostaną zadeklarowane jako globalne lub nielokalne

Funkcje – zakresy nazw

- Przykład

```
X = 99  #Zakres globalny
def  fun(Y):
    #Zakres lokalny
    Z = X + Y
    return Z
print(fun(1))
print(X)
```

Nazwy globalne *X* i *fun* zdefiniowane na najwyższym poziomie pliku modułu. Do zmiennej X można się odnieść ze środka funkcji bez deklarowania jej jako globalnej

Nazwy lokalne *Z i Y* zdefiniowane na poziomie funkcji, do obu przypisane są wartości, *Z* za pomocą instrukcji =, *Y* zostało przekazane jako argument funkcji a argumenty zawsze przekazywane są przez przypisanie.

Funkcje – zakresy nazw

- Instrukcje *global*, *nonlocal* są deklaracjami przestrzeni nazw
- Instrukcja *global* składa się ze słowa kluczowego *global*, po którym następuje jedna lub większa liczba nazw rozdzielonych przecinkami. Wszystkie wymienione nazwy po global zostaną odwzorowane na zakres modułu zawierającego funkcję
- W pythonie 3.0 wprowadzono nową instrukcję *nonlocal* która odnosi się do zakresu nazw wewnątrz funkcji (ma ona znaczenie w przypadku definiowania funkcji zagnieżdżonych)
- Przykład

```
X = 99  #Zakres globalny
def  fun(Y):
    #Zakres lokalny
    global X
    X = 88
fun()
print(X)
```