

Funkcje

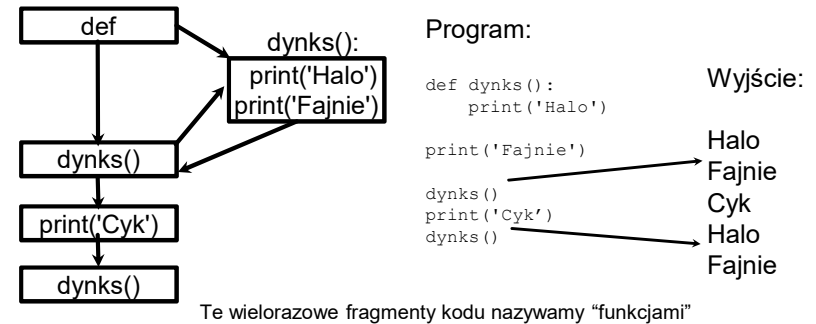
Rozdział 4



Python dla wszystkich
www.py4e.pl



Zapisane (i ponownie użyte) kroki

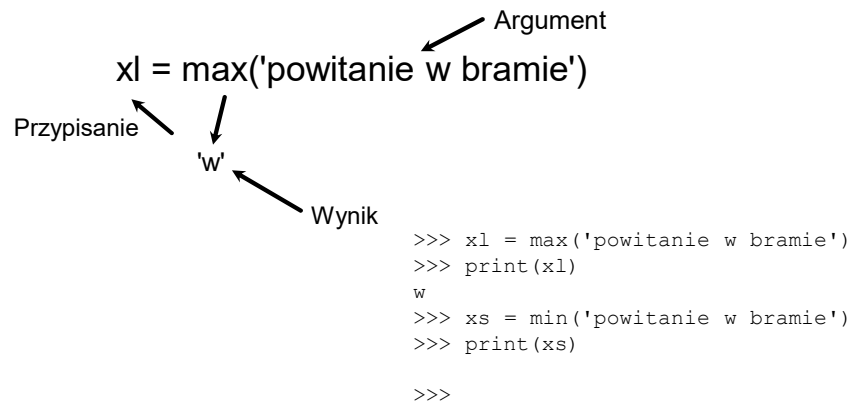


Funkcje Pythona

- W Pythonie są dwa rodzaje funkcji:
 - Wbudowane, które są częścią Pythona – `print()`, `input()`, `type()`, `float()`, `int()`, ...
 - Funkcje, które definiujemy samodzielnie, a potem używamy
- Traktujemy nazwy wbudowanych funkcji jak "nowe" słowa zastrzeżone (czyli nie używamy ich jako nazw zmiennych)

Definicja funkcji

- W Pythonie funkcja to kod wielokrotnego użytku, który przyjmuje argument(y) jako wartości wejściowe, wykonuje jakieś obliczenia i zwraca wynik lub wyniki
- Definiujemy funkcje, używając zastrzeżonego słowa `def`
- Wywołujemy funkcję wyrażeniem złożonym z jej nazwy, nawiasów i argumentów



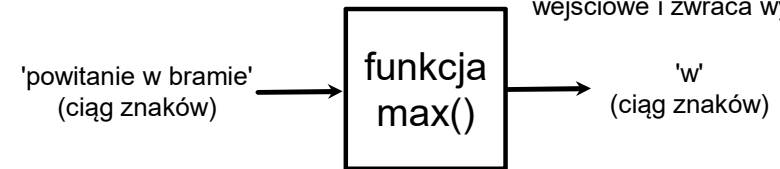
Funkcja max()

```

>>> xl = max('powitanie w bramie')
>>> print(xl)
w

```

Funkcja to wcześniej zapisany kod, którego używamy. Funkcja przyjmuje wartości wejściowe i zwraca wyniki.



Ten kod napisał Guido

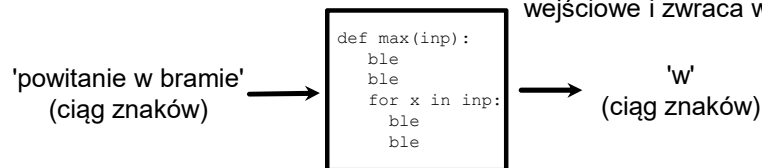
Funkcja max()

```

>>> xl = max('powitanie w bramie')
>>> print(xl)
w

```

Funkcja to wcześniej zapisany kod, którego używamy. Funkcja przyjmuje wartości wejściowe i zwraca wyniki.



Ten kod napisał Guido

Konwersje typów

- Jeśli w wyrażeniu znajdzie się liczba zmiennoprzecinkowa i całkowita, Python automatycznie skonwertuje całkowitą na zmiennoprzecinkową
- Możesz też kontrolować konwersje wbudowanymi funkcjami `int()` i `float()`

```

>>> print(float(99) / 100)
0.99
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
>>> print(1 + 2 * float(3) / 4 - 5)
-2.5
>>>

```

Konwersje ciągów znaków

- Możesz też użyć `int()` i `float()` do konwersji pomiędzy ciągiem a liczbą całkowitą
- Otrzymasz błąd, jeśli ciąg nie zawiera cyfr

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str'
and 'int'
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hej bob'
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

Nasze własne funkcje...

Tworzenie własnych funkcji

- Tworzymy nowe funkcje, używając słowa kluczowego `def`, a następnie opcjonalnych parametrów w nawiasach
- Ciało funkcji zapisujemy z wcięciem
- Tak definiujemy funkcję, ale nie wykonujemy instrukcji z jej ciała

```
def print_lyrics():
    print("jestem sobie drwal i równy chłop.")
    print('pracuję w dzień i śpię całą noc.')
```

`print_lyrics():`

```
print("jestem sobie drwal i równy chłop.")
print('pracuję w dzień i śpię całą noc.')
```

```
x = 5
print('Halo')
```

```
def print_lyrics():
    print("jestem sobie drwal i równy chłop.")
    print('pracuję w dzień i śpię całą noc.')
```

```
print('Yo')
x = x + 2
print(x)
```

Halo
Yo
7

Definiowanie i używanie


- Po zdefiniowaniu funkcji możemy ją wywołać (czyli uruchomić) tyle razy, ile chcemy
- To jest wzorzec zapisz i użyj ponownie

```
x = 5
print('Halo')
```

```
def print_lyrics():
    print("Jestem sobie drwal i równy chłop.")
    print('pracuję w dzień i śpię całą noc.')
```

```
print('Yo')
print_lyrics()
x = x + 2
print(x)
```

Halo
Yo
Jestem sobie drwal i równy chłop.
Pracuję w dzień i śpię całą noc.
7




Argumenty

- Argument to wartość, którą przekazujemy do funkcji przy jej wywołaniu
- Argumenty służą do dawania funkcji innych zadań, kiedy wywołujemy ją w innych okolicznościach
- Argumenty są umieszczane w nawiasach po nazwie funkcji

```
xl = max('powitanie w bramie')
```

Argument



Parametry

Parametr jest zmienną używaną w definicji funkcji. To “uchwyt”, dzięki któremu kod w funkcji ma dostęp do argumentów w konkretnym wywołaniu funkcji.

```
>>> def greet(lang):
...     if lang == 'es':
...         print('Hola')
...     elif lang == 'fr':
...         print('Bonjour')
...     else:
...         print('Witaj')
...
>>> greet('pl')
Witaj
>>> greet('es')
Hola
>>> greet('fr')
Bonjour
>>>
```

Wartości zwracane

Zazwyczaj funkcja przyjmuje argumenty, wykonuje obliczenia i zwraca wartość do wykorzystania przez wywołujące ją wyrażenie. Służy do tego słowo kluczowe `return`.

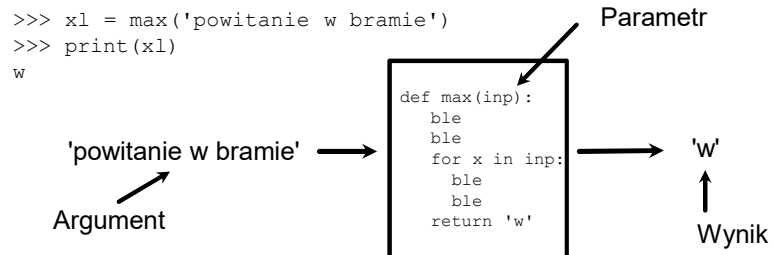
```
def greet():  
    return "Witaj"           Witaj Glenn  
                               Witaj Sally  
  
print(greet(), "Glenn")  
print(greet(), "Sally")
```

Wartość zwracana

- “Owocne” funkcje zwracają wyniki (czyli wartość zwracaną)
- Instrukcja `return` kończy wykonywanie funkcji i “odsyła” wynik funkcji

```
>>> def greet(lang):  
...     if lang == 'es':  
...         return 'Hola'  
...     elif lang == 'fr':  
...         return 'Bonjour'  
...     else:  
...         return 'Witaj'  
...  
>>> print(greet('pl'), 'Glenn')  
Witaj Glenn  
>>> print(greet('es'), 'Sally')  
Hola Sally  
>>> print(greet('fr'), 'Michael')  
Bonjour Michael  
>>>
```

Argumenty, parametry i wyniki



Wiele parametrów / argumentów

- Możemy zdefiniować więcej niż jeden parametr w definicji funkcji
- Dodajemy po prostu więcej argumentów, wywołując funkcję
- Dopasowujemy liczbę i kolejność argumentów i parametrów

```
def addtwo(a, b):  
    added = a + b  
    return added  
  
x = addtwo(3, 5)  
print(x)  
  
8
```

Puste (bezowocne) funkcje

- Kiedy funkcja nie zwraca wartości, nazywamy ją “pustą” funkcją
- Funkcje zwracające wartości to “owocne” funkcje
- Puste funkcje nie są “owocne”

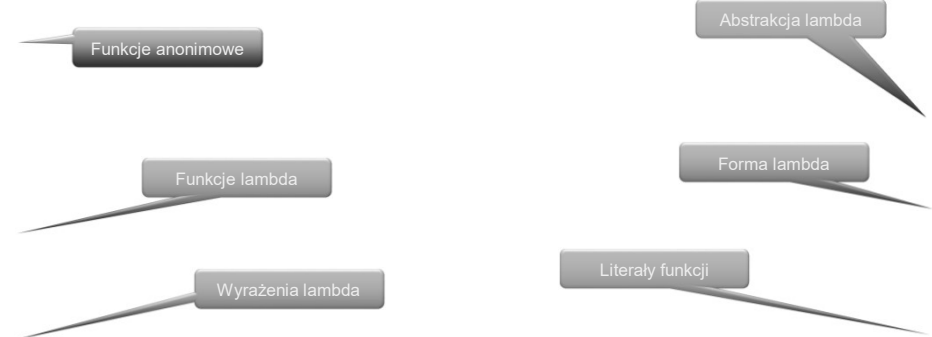
Być (funkcją) albo nie być...

- Dziel swój kod na “akapity” – zapisz całą myśl i “nazwij ją”
- Nie powtarzaj się – zrób coś raz i używaj ponownie
- Jeśli myśl robi się zbyt długa i złożona, podziel ją na logiczne kawałki i umieść je w osobnych funkcjach
- Stwórz bibliotekę rzeczy, które robisz często, może udostępnisz ją znajomym...

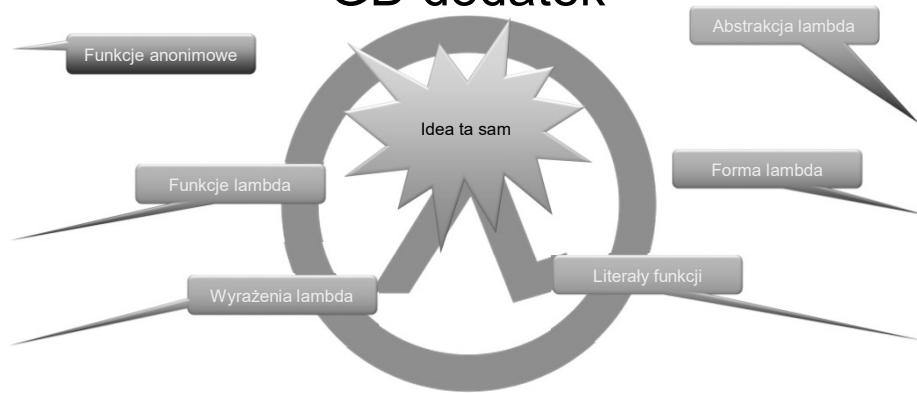
GD dodatek



GD dodatek



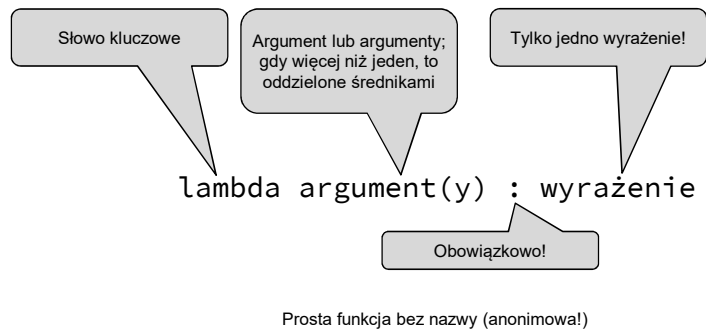
GD dodatek



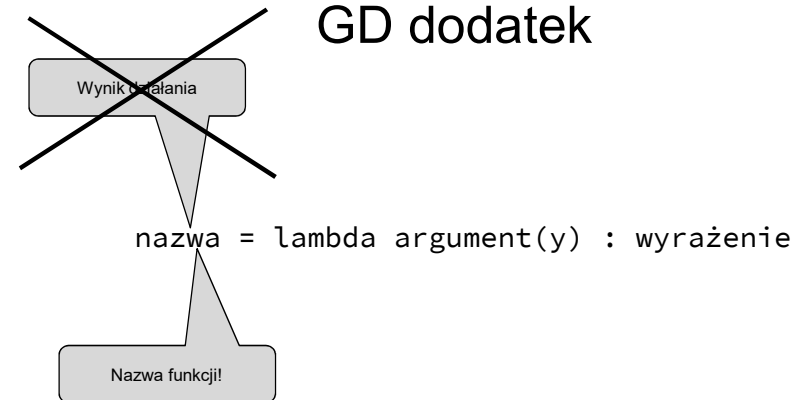
GD dodatek

`lambda argument(y) : wyrażenie`

GD dodatek



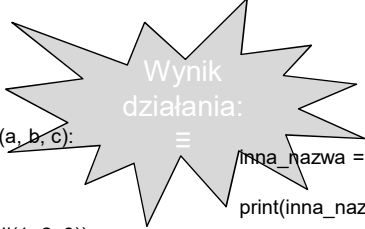
GD dodatek



GD dodatek

```
def nazwa_funkcji(a, b, c):  
    return a + b + c  
print(nazwa_funkcji(1, 2, 3))  
  
inna_nazwa = lambda a, b, c: a + b + c  
print(inna_nazwa(1, 2, 3))
```

GD dodatek



```
def nazwa_funkcji(a, b, c):  
    return a + b + c  
print(nazwa_funkcji(1, 2, 3))  
  
inna_nazwa = lambda a, b, c: a + b + c  
print(inna_nazwa(1, 2, 3))
```

Podsumowanie

- Funkcje
- Funkcje wbudowane
- Konwersje typów (int, float)
- Konwersje ciągów znaków
- Parametry
- Argumenty
- Wyniki (funkcje owocne)
- Puste (bezowocne) funkcje
- Dlaczego używać funkcji?

Ćwiczenie

Przepisz ponownie swoje obliczenie wynagrodzenia z dodatkiem za nadgodziny i stwórz funkcję o nazwie `compute_pay()`, która przyjmuje dwa parametry (hours i rate).

Podaj liczbę godzin: 45
Podaj stawkę godzinową: 10

Wynagrodzenie: 475.0

$$475 = 40 * 10 + 5 * 15$$



Podziękowania dla współpracowników



Copyright slajdów 2010 - Charles R. Severance
(www.dr-chuck.com) University of Michigan School of Information
i open.umich.edu dostępne na licencji Creative Commons
Attribution 4.0. Aby zachować zgodność z wymaganiami licencji
należy pozostawić ten slajd na końcu każdej kopii tego
dokumentu. Po dokonaniu zmian, przy ponownej publikacji tych
materiałów można dodać swoje nazwisko i nazwę organizacji do
listy 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

... wstaw tu nowych współpracowników i tłumaczy