Funkcje wbudowane i słowa kluczowe

Słowa kluczowe

Słowa kluczowe (keywords) to wyrazy zarezerwoane do użytku Pythona. Nie można użyć słowa kluczowego jako nazwy zmiennej, nazwy funkcji czy innego identyfikatora. Każdy ze słów kluczowych odgrywa ważną rolę w tym języku. Lista słów kluczowych może być uzyskana wpisując:

```
1 import keyword
2 print(keyword.kwlist)
```

`import`

Biblioteki w Pythonie są pogrupowane w moduły. Słowo kluczowe `import` służy do importowania modułów do naszej przetrzeni nazw. Przestrzeń nazw (namespace) zawiera wszystkie zmienne (również funkcje i klasy), które zadeklarowaliśmy w trakcie działania programu. `` `import `` to Pythonowy odpowiednik np. dyrektywy `#include<nazwa_biblioteki>` z C++.

```
1 import module
```

Wykonanie powyższego kodu spowoduje dodanie do aktualnej przestrzeni nazw modułu `module`. Każdy moduł może zawierać submoduły, funkcje, klasy, itp. Możemy dodatkowo wskazać, którą klasę lub metodę chcemy zaimportować z modułu. Dodatkowo, wykorzystując słowo kluczowe `as` możemy nadać zaimportowanemu modułowi czy funkcji nową nazwę.

```
1 from module import submodule
2 from module.submodule import function as alias
3 from module import submodule as alias
```

Aby wykorzystać funkcję z danego modułu, musimy najpierw wskazać, z którego modułu chcemy skorzystać a następnie podać nazwę funkcji czy zmiennej do której chcemy się odwołać. Korzystając z przykładu powyżej:

```
1 import keyword
2 print(keyword.kwlist)
```

W pierwszej linijce importujemy moduł `keyword`. W drugiej linijce wypisujemy zawartość zmiennej `kwlist` z modułu `keyword`. Moglibyśmy uzyskać podobny efekt wykonując:

```
1 from keyword import kwlist
2 print(kwlist)
```

W tym przykładzie, z modułu `keyword` importujemy jedynie zmienną `kwlist`. Przy takiej składni warto wspomnieć, że zmniejsza ona czytelność, nie podnosząc wcale efektywności kodu. Interpreter i tak wczyta najpierw całą zawartość modułu, następnie stworzy nową zmienną `kwlist`, której przypisze odpowiednią wartość. Taki zapis zmniejsza czytelność kodu i zwiększa prawdopodobieństwo błędu. Używając zapisu `import module` i następnie `module.variable` jendoznacznie wskazujemy z jakiego modułu korzystamy.

```
from . import module
from .. import module

from .module import submodule
from ..module import submodule
```

`pass`

Python domyślnie oczekuje wcięcia po dwukropku. Jeżeli chcemy zostawić klasę czy funkcję pustą, korzystamy wtedy ze słowa kluczowego `pass`.

```
class User:
    pass
```

`continue`

Słowo kluczowe `continue` powoduje przerwanie aktualnie wykonywanej pętli i przejście do kolejnej iteracji. Przydatne podczas debugowania i testowania kodu.

```
>>> for number in range(0, 30):
...     if number % 5:
...         continue
...     print(number)
0
5
10
15
20
25
```

```
for i in range(1, 30):
    print(i)
    continue

    if not i % 4:
        print('podzielny przez 4')
    else:
        print('asdasd')
```

`break`

Słowo kluczowe `break` przerywa aktualnie wykonywaną pętlę.

```
>>> for number in range(0, 30):
...     if number % 5:
...         break
...     print(number)
0
```

`return`

Słowo kluczowe `return` wskazuje funkcji jaką wartość ma dana funkcja zwrócić. Wykonanie linii ze słowem kluczowym `return` kończy wykonywanie funkcji.

```
>>> def sum(a, b):
...     return a + b
...
>>> sum(2, 3)
5
```

```
>>> def sum(a, b):
...     return a + b
...     print('Total is', a + b)
...
>>> sum(2, 3)
5
```

`__file__`

```
>>> print(__file__)
```

`__name__`

Zmienna `__name__` pozwala między innymi ustalić czy dany plik jest wykonywany czy importowany. Jeżeli dany plik jest wykonywany, zmienna `__name__` ustawiana jest na `'__main__'`, jeżeli dany plik jest importowany jako moduł, zmienna `__name__` ustawiana jest na nazwę modułu. Jest to przydatne na przykład przy testowaniu modułów. Dodanie do modułu poniższej linijki:

```
if __name__ == '__main__':
    print('hello world')
```

Sprawi, że wypisane na konsoli zostanie `'hello world!'` jeżeli dany plik jest wykonywany jako główny. Powyższy kod nie wykona się natomiast jeżeli plik zaimportujemy jako moduł w innym pliku.

```
import logging

log = logging.getLogger(__name__)
```

Funkcje wbudowane

Funkcje wbudowane to funkcje dostępne domyślnie w języku Python.

`print()`

```
print('ehlo world')
print('ehlo', 'world')
print('ehlo', 'world', sep=';')
```

Wyświetla argument jako tekst w wierszu poleceń.

W Pytonie2, print jest słowem kluczowym - nie wymaga użycia nawiasów.

`sorted()` i `sort()`

Sortują elementy listy.

`sorted()` to operator niemutowalny (nie zmienia kolejności elementów w liście).

`sorted()` to funkcja, która jako argument przyjmuje listę.

```
>>> numbers = [3, 1, 2]
>>> sorted(numbers)
[1, 2, 3]
>>> print(numbers)
[3, 1, 2]
```

`.sort()` to operator zmieniający listę (mutujący).

`sort()` to metoda klasy lista.

```
>>> numbers = [3, 1, 2]
>>> numbers.sort()
>>> print(numbers)
[1, 2, 3]
```

`range()`

Tworzy iterator, który zwraca liczby w sekwencji. Jedna z rzeczy, która uległa zmianie od Pythona2, w którym range zwracał sekwencję liczb zamiast iteratora.

```
>>> numbers_generator = range(0, 5)
>>> print(numbers_generator)
range(0, 5)

>>> numbers = list(numbers_generator)
>>> print(numbers)
[0, 1, 2, 3, 4]
```

`isinstance()`

Sprawdza czy dany obiekt jest instancją danej klasy.

```
>>> isinstance(10, int)
True

>>> isinstance(10, float)
False

>>> isinstance(10, (int, float))
True
```

`min()`

Wartość minimalna z listy.

```
>>> numbers = [1, 2, 3, 4, 5]
>>> min(numbers)
1
>>> min(3, 1, 5)
1
```

`max()`

Wartość maksymalna z listy.

```
>>> numbers = [1, 2, 3, 4, 5]
>>> max(numbers)
5
>>> max(3, 1, 5)
5
```

`len()`

Długość listy.

```
>>> numbers = [1, 2, 3, 4, 5]
>>> len(numbers)
5
```

`input()`

Pozwala użytkownikowi wpisać tekst.

```
>>> name = input()
Ivan
>>> print(name)
'Ivan'
```

Pamiętaj o dodaniu dwukropka i spacji, aby tekst się nie zlewał.

```
>>> name = input('Type your name: ')
Type your name: José
>>> print(name)
'José'
```

Czasami trzeba oczyścić dane, np. usuwając zbędne spacje na początku i końcu ciągu znaków podanego przez użytkownika.

```
>>> name = input('Type your name: ')
Type your name:         Ivan
>>> print(name.strip())
'Ivan'
```

`bin()`

Konwertuje liczbę na binarną.

```
>>> bin(3)
'0b11'

>>> bin(-3)
'-0b11'
```

`hex()`

Konwertuje liczbę na hex.

```
>>> hex(99)
'0x63'
```

`oct()`

Konwertuje liczbę na oct.

```
>>> oct(23)
'0o27'
```

`ord()`

Zwraca kod jednoznakowego stringa.

```
>>> ord('a')
97
```

`chr()`

Konwertuje kod na znak Unicode.

```
>>> chr(97)
'a'
```

Wszystkie funkcje wbudowane

abs()	**all()**	**any()**	**ascii()**	**bin()**
bool()	bytearray()	bytes()	callable()	chr()
classmethod()	compile()	complex()	dict()	dir()
divmod()	enumerate()	eval()	exec()	filter()
float()	format()	frozenset()	getattr()	globals()
hasattr()	help()	hex()	id()	input()
int()	isinstance()	issubclass()	iter()	len()
list()	locals()	map()	max()	min()
next()	object()	oct()	ord()	pow()
print()	property()	range()	repr()	reversed()
round()	setattr()	slice()	sorted()	staticmethod()
vars()	zip()	open()	**import**	