

Мова програмування Python

Практичне заняття 5



Вбудовані функції Python

У python є велика бібліотека функцій. Ці функції називаються вбудованими функціями Python. Вони вже визначені і готові до використання безпосередньо відповідно до потреби. Вбудовані функції повинні використовуватися, як визначено. Вони не повинні бути перезаписані розробником.

Існує безліч вбудованих в мову програмування функцій. З деякими такими в Python ми вже стикалися. Це `print()`, `input()`, `int()`, `float()`, `str()`, `type()`. Код їх тіла нам не видно, він десь "захований всередині мови". Нам же надається тільки інтерфейс - ім'я функції.

abs(x)

Повертає абсолютне значення числа. У разі передачі складного числа величина цього числа повертається. Це те саме, що відстань від початку точки на графіку x .

```
number1 = 15  
number2 = -5  
number3 = 3 + 4j  
number4 = 1.25  
print(abs(number1))  
print(abs(number2))  
print(abs(number3))  
print(abs(number4))
```

```
15  
5  
5.0  
1.25
```

isinstance(x, y)

`isinstance(x, y)` – перевіряє, чи є `x` в значеннях типу `y`.

```
print(isinstance('string', str))  
a = ['q']  
print(isinstance(a, str))  
print(isinstance(a, list))
```

```
True  
False  
True
```

sorted(x)

sorted(x) – повертає список - відсортовану послідовність (рядок або список), яка складається з елементів x.

```
print(sorted('akfabdgc'))  
print(sorted([4, 2, 5, 1, 3]))
```

```
['a', 'a', 'b', 'c', 'd', 'f', 'g', 'k']  
[1, 2, 3, 4, 5]
```

all(x)

Те саме, що логічний оператор. Це означає, що вона поверне `true`, якщо всі змінні в ітераторі є істинними. Тут ітерабельні об'єкти посилаються на кортежі, списки, словники.

```
tuple = (0, True, False)
x = all(tuple)
print(x)
```

False

```
my_dict = {'key1' : 'Apple', 'key2' : 'Orange', 'key3' : 'Pear'}
my_var = all(my_dict)
print(my_var)
```

True

any(x)

x - ітеріруемый об'єкт (список, кортеж, словник).
Функція **any()** повертає True, якщо який-небудь (будь-який) елемент в ітеріруемом об'єкті є True, в іншому випадку **any()** повертає значення False.
Якщо послідовність порожня, то функція **any()** повертає False.

```
print(any([0, 6, 3, 0, 1]))  
print(any([0, False, '']))
```

```
True  
False
```

or і any()

В основному функція **any()** застосовується в поєднанні з оператором розгалуження програми **if ... else**. Роботу функції **any()** можна порівняти з оператором **or** в Python, тільки **any()** працює з послідовностями:

```
print(False or True or False)
print(any([False, True, False]))

True
True
```

Але між **or** і **any()** в Python є два основні відмінності:

- Синтаксис.
- Значення, що повертається.

Функція **any()** завжди повертає False або True.

Оператор **or** повертає ПЕРШЕ справжнє значення, а якщо всі значення False, то ОСТАННЄ помилкове значення.

```
print(1 or 0 or 0 or 5 or '')
print(0 or '' or 5)

print(bool(1 or 0 or 0 or 5 or ''))
print(bool(0 or '' or 5))

1
5
True
True
```

```
num = [2, 2.0, 1.24, 4, 0.19, 5, 1.1]
# використання вбудованих функцій на прикладі 'isdigit()'
print([str(x).isdigit() for x in num])

#використання операцій порівняння
print([x > 4 for x in num])

#використання оператора входження in
print(['.' in str(x) for x in num])

#використання оператора ідентичності in
print([type(x) is int for x in num])
```

```
[True, False, False, True, False, True, False]
[False, False, False, False, False, True, False]
[False, True, True, False, True, False, True]
[True, False, False, True, False, True, False]
```

Приклад з числовою послідовністю. Необхідно дізнатися, чи є в послідовності числа більше певного значення.

```
number1 = range(1, 27, 2)
number2 = range(0, 10, 3)
print(any([x > 10 for x in number1]))
print(any([x > 10 for x in number2]))
```

```
True
False
```

Можна перевіряти рядок на наявність, певних символів.

```
symbols = ['(', ')', ',', '-', '']
my_string1 = "функція any() - повертає 'False', 'True'"
my_string2 = "функція any завжди повертає bool"
print(any([x in my_string1 for x in symbols]))
print(any([x in my_string2 for x in symbols]))
```

```
True
False
```

complex(real, imag)

параметри:

real - число або рядок. Якщо число, то воно являє дійсну частину комплексного числа.

imag - тільки число, уявна частина комплексного числа. За замовчуванням `imag = 0`.

Значення, що повертається:

комплексне число

опис:

Клас **complex ()** перетворює рядок із записом комплексного числа в комплексне число або поверне комплексне число зі значенням переданих аргументів: дійсною і уявною частиною. Якщо перший параметр є рядком, наприклад `'3 + 5j'`, то виконається перетворення рядка в комплексне число, в цьому випадку другий параметр `imag` повинен бути опущений.

Другий параметр `imag` ніколи не може бути рядком. Кожен аргумент може бути будь-якого числового типу (`int` або `float`). Якщо `imag` опущений, то за умовчанням використовується `0`. Якщо обидва аргументи опущені, повертається `0j`.

При перетворенні з рядка, рядок не повинен містити пропусків навколо + або - оператора. Наприклад, `complex('1+2j')` - добре, але при записі рядка комплексного числа з пробілами `complex('1 + 2j')` буде помилка `ValueError`.

```
print(complex())  
print(complex(1))  
print(complex(2.4))  
print(complex(2, 4))  
print(complex(1, 2e-2))
```

```
0j  
(1+0j)  
(2.4+0j)  
(2+4j)  
(1+0.02j)
```

```
print(complex('1+2j'))  
print(complex('0.1+2.0j'))  
print(complex('.1+2.j'))  
print(complex('1e3+2e-3j'))
```

```
(1+2j)  
(0.1+2j)  
(0.1+2j)  
(1000+0.002j)
```

len(s)

параметри: s - послідовність або колекція.

Значення, що повертається: ціле число.

Функція **len()** повертає довжину (кількість елементів) в об'єкті.

Аргумент може бути послідовністю, такою як рядок, кортеж, список або діапазон або колекцією (такою як словник, множина або незмінна множина).

Коли об'єкт є рядком, функція len() повертає кількість символів в рядку.

```
my_list = ['яблуко', 'груша', 'банан', 'слива']  
print(len(my_list))
```

```
my_string = 'Привіт'  
print(len(my_string))
```

```
my_dictionary = {'яблуко': 'apple', 'груша': 'pear', 'вишня': 'cherry', 'слива': 'plum'}  
keys = list(my_dictionary)  
print(len(keys))
```

4

6

4

sum(iterable, /, start=0)

Функція **sum()** починає підсумовування елементів послідовності `iterable` з початкового значення `start`, якщо вони зазначені, складання відбувається з ліва на право і в результаті повертає їх суму. Функція **sum()** очікує, що елементи `iterable` є числами, а початкове значення `start` не може бути рядком. Якщо аргумент `iterable` порожній, то функція **sum()** повертає початкове значення `start`, якщо воно вказане.

```
x = []
print(sum(x))

# Сума чисел, переданих в якості аргументів (кожна окремо)
print(sum(x, 10))

# сума списку цілих чисел і 20
x = [5, 15, 10, 25]
print(sum(x, 20))

# сума списку цілих чисел
x = [2, 4, 6, 8, 2, 6]
print(sum(x))

# сума списку цілих и десяткових чисел
x = [1.1, 2.2, 3.3, 4.256]
print(sum(x, 1))

--
0
10
75
28
11.856000000000002
```

max(x, key)

Ця функція використовується для знаходження «максимального» значення в послідовності, об'єкті що ітерується і так далі. У параметрах можна змінювати спосіб обчислення максимального значення.

```
print(max('a', 'A'))
```

```
a = ['3', '59', '3', '98', '2']  
print(max(a))
```

```
b = [4, 6, 57, 52, 24, 7]  
print(max(b))
```

```
x = ["Яблоко", "Апельсин", "Автомобиль"]  
print(max(x, key = len))
```

```
a  
98  
57  
Автомобиль
```

min(x,key)

Ця функція використовується для знаходження «мінімального» значення в послідовності, об'єкті що ітерується і так далі. У параметрах можна змінювати спосіб обчислення мінімального значення.

Аргумент key - функція подібна до тієї, яка використовується в додатковому методі списків list.sort (). Функція приймає один аргумент і використовується для впорядкування елементів.

```
x = ['4', '9', '10', '37', '67']  
# функція `min` порівнює числа як рядки  
print(min(x))  
  
print(min([3, 7, 9, 1, 3], [2, 5], key=sum))  
  
10  
[2, 5]
```

reversed()

Ця функція надає простий і швидкий спосіб розгорнути порядок елементів в послідовності. Як параметр вона приймає послідовність, наприклад список, а повертає об'єкт що ітерується.

```
x = [40, 3, 127]
b = list(reversed(x))      [127, 3, 40]
print(b)
```

Перевернемо рядок (реверс рядку) за допомогою reversed():

```
x = 'forest'

for i in reversed(x):
    # вивід символів рядка 'x' по одному в зворотному порядку
    print(i, end='')

print('\n' + '-'*len(x))
print(x)
```

tserof

forest

Завдання

Завдання 1

Напишіть функцію `my_sum_range (start, end)`, яка підсумовує всі цілі числа від значення «start» до величини «end» включно.

Якщо користувач введе перше число більше ніж друге, просто поміняйте їх місцями.

При вирішенні зручно скористатися вбудованою функцією `sum()`.

Завдання 2

Микола створив 3 функції (представлені нижче). Спробуйте вгадати, як поведе себе кожна з них при запуску (чи виникнуть помилки? що повернеться?).

```
def func1():  
    param = 4
```

```
    def inner():  
        param += 1
```

```
    return param
```

```
def func2():  
    param = 4
```

```
    def inner(var):  
        var += 1
```

```
        inner(param)  
        return param
```

```
def func3():  
    param = 4
```

```
    def inner(var):  
        var += 1  
        return var
```

```
    param = inner(param)  
    return param
```

Напишіть функцію `season_events (n_month)`, яка приймає номер місяця вашого народження і в залежності від сезону друкує на виході наступне:
«Ви народилися в <НАЗВА_МІСЯЦА>. <ОПИС_ПОДІЙ> ».

В якості ОПИСУ_ПОДІЙ буде характеристика сезону:

- для зими «За вікном падав білий сніг»,
- для весни «Птахи співали прекрасні пісні»,
- для літа «Сонце світило яскравіше ніж будь-коли»,
- для осені «Урожай був неймовірним».

Важливо врахувати, що користувачі можуть ввести будь-який тип даних в якості аргументу (не попадіться на цьому і попередьте про те, що «Потрібно ввести реальний номер місяця»).

Для виведення на друк кожного місяця зручно користуватися словником. Можна, звичайно, використовувати `if / else` конструкцію, але в цьому випадку вона нерозумно збільшує обсяг коду. А ось для чотирьох сезонів цілком годиться умовний вираз.