Занятие 7

Функции

Что напечатает?

```
print({1,2,3,(1,2,3), ‘123’, 1.0, 2.0, 3.0})

x = {1,2,3,4,5,6,1,2,3,4,5,6}
print(len(x))

print({1,2,3,4,5,’123’, (1,2,3)}.pop())

print({1,2,3,4,5,’123’, (1,2,3)}.add(1))
```

Задание 6-1

Написать функцию, которая переводит строку римских цифр в десятичное число.

Римские цифры:

I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000, IV= 4, IX = 9, XC = 90, XL = 40, CD = 400, CM = 900

Пример:

MMXXIII = 2023, MMXXIV = 2024, MCMXVII = 1917, MCMLX1 = 1961, MM = 2000, MDCCCLXII = 1862

Программа в цикле должна обращаться к этой функции с различными строками из примера.

Подсказка. Можно использовать функцию ‘abcde’.startswith(‘ab’), которая выдает True, если строка ‘abcde’ начинается с ‘ab’

Задача 6-2

Напишите программу, которая:

Получает на вход две строки, в которых перечисляются книги, прочитанные двумя учениками.

Выводит количество книг, которые прочитали **оба** ученика.

Пример ввода:

Война и мир, Над пропастью во ржи, Мастер и Маргарита, Идиот

Евгений Онегин, Идиот, Мастер и Маргарита, Война и мир

Ответ: 3

Задача 6-3

Напишите программу, которая принимает на вход строку из символов и печатает одну строку из букв, вторую из цифр, третью из прочих символов без повторений.

Например:

Ввод: ab18.,cab=561:xz:

Вывод:

a b c x z

1 8 5 6

. , = :

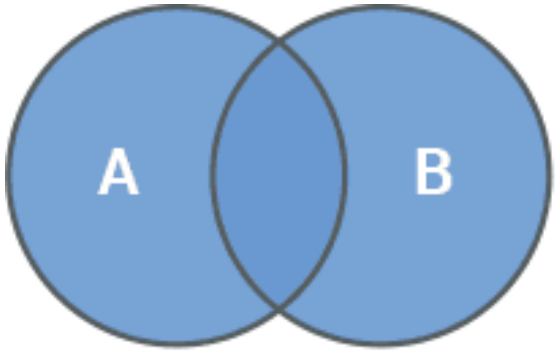
Коллекции

1. Строка (str) ‘Hello world’
2. Список (list) [1, 100, 1, ‘a’, True]
3. Кортеж (tuple) (1, 100, 1, ‘a’, True)
4. Словарь (dict) {1:1, 22:100, 123:1, 'a':'a', 5:True}
5. **Множество (set) {1, 100, 'a', True}**

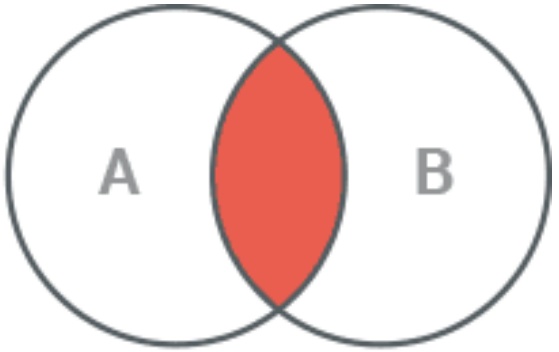
Важные методы

SET	LIST	DICT
Add	Append	Setdefault
Copy	Copy	Copy
Pop	Pop	Pop
Clear	Clear	Clear
Union	Extend	Update
Len	Len	Len
Set()	List()	Dict()
{1,2,3,}	[1,2,3, 1,2,3]	{1:11, 2:22, 3:33}

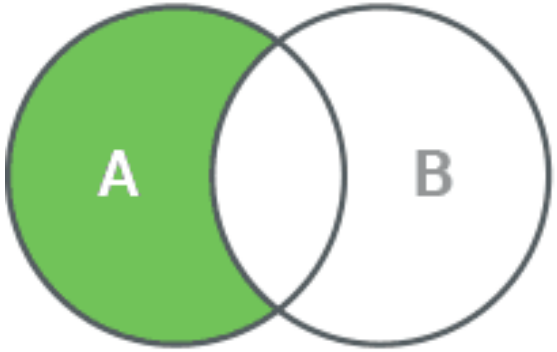
Из теории множеств



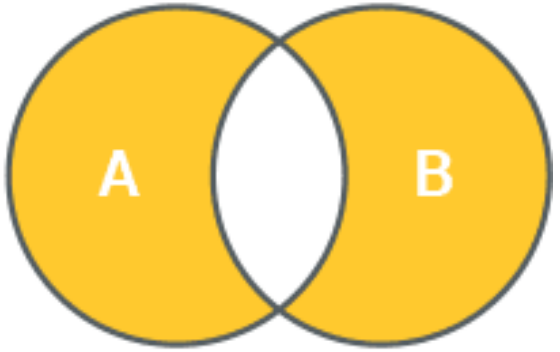
Union



Intersection



Difference



Symmetric Difference

10 вопросов по множествам в Питоне

https://pythonist.ru/test-po-mnozhestvam-v-python/

Функции

Понятие функции

Функция в **Python** - объект, принимающий аргументы, реализующий какие-то законченные действия и возвращающий результат.

Вход в функцию - это передача ей аргументов - данных, полученных во внешней части программы.

Тело функции - получив данные, функция должна их как-то обработать: выполнить некоторые действия, вычислить какое-то значение.

Выход из функции - значение, вычисленное блоком кода данной функции и передаваемое во внешнюю часть программы.

Входные данные называют параметрами, а выходные - возвращаемым значением.

Впрочем, функция может и не принимать никаких параметров.

Что принимает в качестве параметров и что возвращает функция в результате своей работы, определяет программист.

Пример:

```
#Определение функции:
def summ(x, y):
    result = x + y
    return result

#вызов функции
a = 100
b = 50
answer = summ(a, b)

print(answer)
150
```

Роль функции в программировании

1. Сокращение кода

Код, который повторяется можно перенести в функцию и использовать её тогда,

когда нужно выполнить код, который находится внутри этой функции.

2. Логическое разделение программы

Мы можем выделить определённое сложное действие (например перемножение матриц) в отдельную функцию, чтобы оно не мешалось в коде, даже если используем её один раз за всё время выполнения программы.

3. Проще тестировать

4. Более эффективная организация труда команды разработчиков

Можно разрабатывать проект большим количеством программистов, каждый из которых отвечает за свои функции

Определение функции

```
# объявление функции my_function()

def my_function([параметр1, параметр2,..]):
    # тело функции

    # возвращаемое значение
    return result  # необязательно

# вызов функции
my_function([аргумент1 ,аргумент2,...] )
```

или

```
result = my_function([аргумент1 ,аргумент2,...] )

type(my_function)
<class 'function'>   - еще один тип в Python
```

Вызов функции

Можно ли так вызвать функцию ?

```
z = summ(10, 20)
```

```
def summ(x, y):
    return x + y
```

NameError: name 'summ' is not defined

```
z = sum(10, 20)  #  а так можно
```

Что вернет функция без return:

```
#Определение функции:
def summ(x, y):
    result = x + y

#вызов функции
answer = summ(100, -50)

print(answer)  # ->None
```

Что вернет функция в отсутствии аргументов ?

```
#Определение функции:
def summ(x, y):
    result = x + y
    return result;


#вызов функции
answer = summ(100)
```

TypeError: summ() missing 1 required positional argument: 'y'

Задание

Напишите программу, которая содержит функцию вычисления подоходного налога (13%) от суммы дохода, передаваемой как параметр.

Программа в цикле вводит доход, вызывает функцию и печатает подоходный налог.

Выход из цикла - 0

Параметры по умолчанию

Для некоторых параметров в функции можно указать значение по умолчанию, таким образом, если для этого параметра не будет передано значение при вызове функции, то ему будет присвоено значение по умолчанию.

```
def premium(salary, percent=10):
    p = salary * percent / 100
    return p

result = premium(60000)    # указана только salary, percent = 10 по умолчанию

print(result) #   ->    6000.0

print(premium(60000,20)) # указаны и salary, и percent  -> 12000.0
```

Пустая функция

```
def empty(var1, var2):
    pass

result = empty(8, 10)

print(result) # -> None
```

Именованные параметры

Иногда происходит такая ситуация, что функция требует большое количество аргументов.

Пример:

```
def my_func(arg1, arg2, arg3, arg4, arg5, arg6):
    pass #  оператор, если кода нет

result = my_func(1, 2, 3, 4, 5, 6)
```

В такой ситуации код не всегда удобно читать и тяжело понять, какие переменные к каким параметрам относятся.

```
result = my_func(arg2=2, arg1=1, arg4=4, arg5=5, arg3=3,  arg6=6)
# Давайте проверим
```

Функция с неограниченным количеством позиционных аргументов

***args** - произвольное число позиционных аргументов

```
def manyargs(var1, *args):
    print(type(args))
    print(args)

result = manyargs(4, 9, 1, 3, 3, 1)
```

<class **'tuple'**>

(9, 1, 3, 3, 1)

Задание

Создайте функцию, которая принимает любое количество позиционных числовых параметров и возвращает их произведение.

Функции с неограниченным количеством именнованных аргументов

****kwargs** - произвольное число именованных аргументов.

```
def manykwargs(**kwargs):
    print(type(kwargs))
    print(kwargs)
manykwargs(name=’Piter’, age=20)
```

<class **'dict'**>

{'name': 'Piter', 'age': 20}

Можно ли мне по другому назвать параметр ****kwargs ?**

Можно, но не нужно)))

Задание

Создайте функцию, которая принимает любое количество именованных параметров и возвращает произведение целых числовых параметров и конкатенацию текстовых параметров, игнорируя остальные.

Все вместе

***args -** произвольное число позиционных аргументов

****kwargs** - произвольное число именованных аргументов

Параметр с одним префиксом * может захватывать любое количество позиционных аргументов в кортеж.

Параметр с двойным префиксом ** может захватывать любое количество ключевых аргументов в словарь.

```
def many_all(var1, *args, **kwargs):
    print(var1)
    print(args)
    print(kwargs)
many_all(10, 34, 77, name='Piter', age=20)
```

```
10
(34, 77)
{'name': 'Piter', 'age': 20}
```

Задание

Доработайте функцию, чтобы одним из аргументов ее была ставка налога (по умолчанию 13%).

1. Вызовите ее только с одним параметром – суммой дохода.

2. Вызовите ее с двумя параметрами – суммой дохода и ставкой ПН (13 или 15 или 20).

Задание

Напишите функцию, которая принимает в качестве аргумента список, состоящий из слов.

Например, def uni_let(lst):

Результатом функции получается кортеж, состоящий из двух элементов:

1. Строка из всех уникальных букв, отсортированная в алфавитном порядке
2. Количество таких уникальных букв.

Т.е. в конце функции должен стоять оператор return string, number

Проверьте ее работу на списке из нуля слов, из одного слова, из двух и т.д.

Вложенность функций

```
def func(n):
    def func_1(p):
        return p * p
    def func_2(w):
        return w + w
    if n < 10:
        res = func_1(n)
    else:
        res = func_2(n)
    return res
x = int(input())
print(func(x))
```

Задание

Напишите программу, которая для каждого из чисел от 0 до введенного n печатает «Четное» или «Нечетное».

Программа содержит функцию check2(i), которая содержит две функции chet(i) и nechet(i), которые и печатают i и одно из этих слов.

Вся программа имеет следующий вид:

```
def check2(i):
    def chet(i):
…
for i in range(int(input()) + 1):
    check2(i)
```

Scope (область видимости)

- **Область видимости** указывает интерпретатору, когда наименование

(или переменная) видима. Другими словами, область видимости определяет, когда и где вы можете использовать свои переменные, функции и т.д.

- Если вы попытаетесь использовать что-либо, что не является в вашей области видимости, вы получите ошибку NameError.

- Python содержит три разных типа области видимости:

  * Локальная область видимости
  * Глобальная область видимости
  * Нелокальная область видимости (была добавлена в Python 3)

Локальная область видимости

Локальная область видимости (local) — это блок кода или тело любой функции Python.

Эта область Python содержит имена, которые вы определяете внутри функции. В этих областях каждый разработчик может использовать одни и те же переменные, независимо друг от друга.

```
x = 100

def doubling(y):
    z = y*y


doubling(x)

print(z)
```

NameError: name 'z' is not defined

Где какая переменная s? Что будет напечатано?

```
s = 0
print('000', s)

def a():
    s = 1
    print('111', s)

    def b(): # Функция, вложенная в функцию a()
        s = 2
        print('222', s)

    b()
    print('333', s)

a()
print('444', s)
s = 3
print('555', s)
```

Глобальная область видимости (global)

В Python есть ключевое слово global, которое позволяет изменять изнутри функции значение глобальной переменной.

Оно записывается перед именем переменной, которая дальше

внутри функции будет считаться глобальной.

```
x = 100
def doubling(y):
        global x
        x = y*y
doubling(x)
print(x)
```

10000

Не используйте global без острой необходимости!

Нелокальная область видимости

В Python 3 было добавлено новое ключевое слово под названием **nonlocal**.
С его помощью мы можем добавлять переопределение области видимостей функций.

```
def counter():
    num = 0

    def incr():
        num += 1 # num = num + 1
        return num

    x = incr()
    return x
```

Если вы попробуете запустить этот код, вы получите ошибку UnboundLocalError, так как переменная num ссылается прежде, чем она будет назначена в самой внутренней функции.

Нелокальная область видимости

Добавим nonlocal в наш код:

```
def counter():
    num = 0

    def incr():
        nonlocal num
        num += 1
        return num

    x = incr()
    return x


inc = counter()
  print(inc)
1
```

Задание

Напишите программу, которая реализует нашу программу FizzBuzz.

Для чисел от 1 до введенного n печатает FIZZ, если число делится на 3, BUZZ, если делится на 5, и FIZZBUZZ, если делится на 15, и само число в противном случае.

Для этого напишите функцию fb(i), которая анализирует число i и реализует указанные действия. Функция fb(i) содержит 4 функции, fb3, fb5, fb15, fb_other, которые и печатают, что надо.

```
#       Сама программа имеет следующий вид:
def fb(i):
    def fb3(i):
…
#       Собственно основная программа
for i in range(int(input()) + 1):
    fb(i)
```

Полезные встроенные функции (built-in)

print, len, str, int, float, list, tuple, dict, set, range

bool, enumerate, zip, reversed, sum, max, min, sorted, any, all

type, filter, map, round, divmod, bin, oct, hex, abs, ord, chr, pow

Если не знаете или забыли, что за функция, то набираете

help(<имя функции>), например: help(print)

zip

```
for k in zip([1,2,3,4,5], ‘abcdef’):
        print(k)
```

(1, ‘a’)

(2, ‘b’)

(3, ‘c’)

(4, ‘d’)

(5, ‘e’)

Останавливается на наименьшем итерируемом объекте, можно указывать несколько объектов

Сделайте zip((1,2,3,4), {1:111, 2:222}, {123, 456,789})

reversed

```
for k in reversed([1, 5, 2, 4, 3]):
    print(k)
```

```
3
4
2
5
1
```

А как еще можно перевернуть список?

filter

```
print(list(filter(bool, [0, 1, 2])))
```

[1, 2]

На месте bool может быть любая функция, которая выдает True или False, например, определенная нами.

```
def chet(x):
    if x % 2 == 0: return True
    return False
```

Что напечатает?

```
print(list(filter(chet, [0, 1, 2, 3, 4, 5, 6, 7])))
```

any all

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

Что напечатает?

```
print(all([]))
print(all([[]]))
print(all([[[]]]))
```

map

```
s = map(int, input().split())
```

Вводим 1 2 3

Получаем список s = [1, 2, 3]

Что напечатает?

```
k = 123
print(sum(map(int, list(str(k)))))
```

sorted

Все превращает в отсортированный список.

print(sorted([1, -2, 3, -4, 5]))

Результат: -4, -2, 1, 3, 5

Что напечатает?

print(sorted([1, -2, 3, -4, 5], key = abs))

Можно создать свою функцию и использовать ее для сортировки.

Напишите такую функцию, чтобы список был отсортирован по убыванию.

Подсказка: используйте в вашей функции abs, но со знаком минус.

Задание

Напишите функцию, которая принимает два аргумента, lst - список чисел и x – число.

Функция возвращает список, содержащий квадраты чисел из lst, которые больше числа x.

Сделайте несколько вариантов решений:

1. Просто цикл с условием.
2. Воспользуйтесь функцией filter, для чего создайте функцию проверки числа больше x * x

Задача 7-1

Напишите программу, которая рассчитывает НОК (наименьшее общее кратное) для списка натуральных чисел.

Задача 7-2

Напишите функцию, которая шифрует строку, содержащую латинские буквы с помощью шифра Цезаря. Каждая буква сдвигается на заданное число n позиций вправо. Пробелы, знаки препинания не меняются.

Например, для n = 1.

a -> b, b -> c, p -> q, y -> z, z -> a

A -> B, B -> C, Z->A

Т.е. заголовок функции будет def code(string, n):

В качестве результата печатается сдвинутая строка.

Задача 7-3

Дан x -двумерный массив чисел в виде списка, содержащего строки в виде списков. Размер массива n – строк и m – столбцов.

Напишите функцию, которая принимает этот массив как аргумент и в качестве результата выдает отсортированный список трех самых больших чисел.

Пример: n = 2, m = 3, x = [[1,6,3], [4,5,4]]

Результат: [4, 5, 6]