Занятие 7

Функции

Модули

Начнем с ...

- 1. Разминка (Что напечатает?)
- 2. Обсуждение домашнего задания

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, MXMLX1 = 1961, MM = 2000, MDXXXLXII = 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’

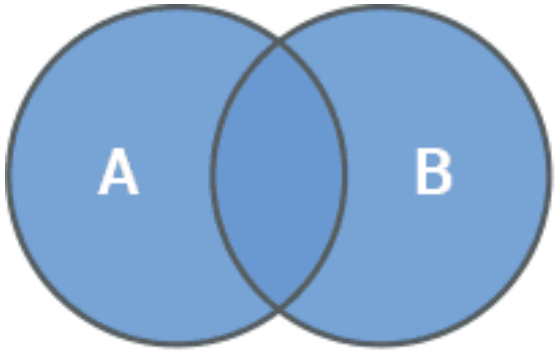
Коллекции

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}**

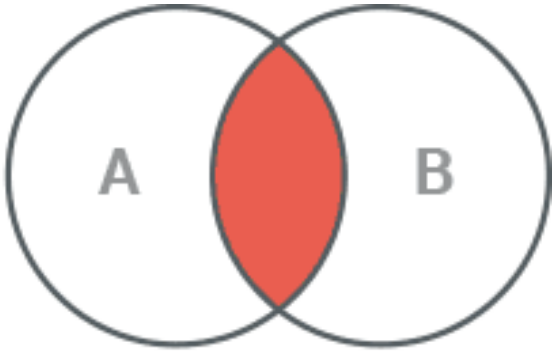
Важные методы

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}

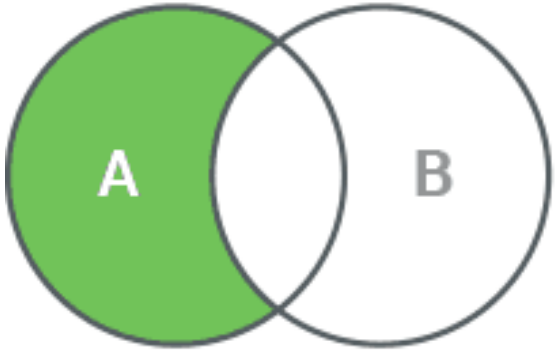
Из теории множеств



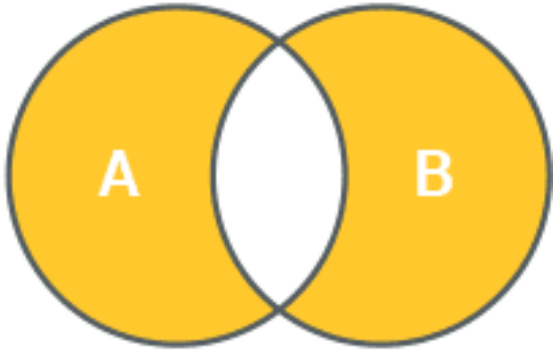
Union



Intersection



Difference



Symmetric Difference

10 вопросов по множествам в Питоне

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

Функции

Пример функции

```
def convert_to_celsius(temp = 32):
    result = (5 / 9) * (temp - 32)
    return result

x = convert_to_celsius(32)
print(x)
```

Аргументы функций

- ***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}

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 или лямбда-выражения.

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

Модули

- Система модулей позволяет вам логически организовать ваш код на Python.
- Группирование кода в модули значительно облегчает процесс написания и понимания программы.
- Модуль в Python это **просто файл**, содержащий код на Python.
- Каждый модуль в Python может содержать **переменные**, **объявления классов и функций**.
- Кроме того, в модуле может находиться **исполняемый код**.

Команда **import**

- Вы можете использовать любой питоновский файл как модуль в другом файле, выполнив в нем команду import. Команда import в Python обладает следующим синтаксисом:

- **import** *module_1*[, module_2[,... module_N]
- Когда интерпретатор Python встречает команду import, он импортирует этот модуль, если он присутствует в пути поиска Python. Путь поиска Python это список директорий, в которых интерпретатор производит поиск перед попыткой загрузить модуль.

math

- Например, чтобы использовать модуль math следует написать:

- **import** math

- # Используем функцию sqrt из модуля math
- print(math.sqrt(9))
- # Печатаем значение переменной pi, определенной в math
- print(math.pi)

- **Важно знать!** Модуль загружается лишь однажды, независимо от того, сколько раз он был импортирован. Это препятствует цикличному выполнению содержимого модуля.

from *module* import *var*

- Выражение from ... import ... не импортирует весь модуль, а только предоставляет доступ к конкретным объектам, которые мы указали.

- # Импортируем из модуля math функцию sqrt
- **from** math **import** sqrt
- # Выводим результат выполнения функции sqrt.
- # **Обратите внимание**, что нам больше незачем указывать имя модуля
- print (sqrt(144))
- # Но мы уже не можем получить из модуля то, что не импортировали !!!
- print (pi) # Выдаст ошибку

from *module* import *var, func, class*

- Импортировать из модуля объекты можно через запятую.

- from **math** import pi, sqrt
- print(**sqrt**(121))
- print(pi)
- print(e)

from *module* import *

- В Python так же возможно импортировать всё (переменные, функции, классы) за раз из модуля, для этого используется конструкция **from** ... **import** *
- **from** math **import** *
- # Теперь у нас есть доступ ко всем функция и переменным, определенным в модуле math
- print(sqrt(121))
- print(pi)
- print(e)

- Не импортируются объекты с именами, начинающимися с ‘_’ (***_var)***
- Повторяющиеся названия перезаписываются. Такое поведение нужно отслеживать при импорте нескольких модулей.

Некоторые функции из math

Тригонометрия: acos acosh asin asinh atan atan2 atanh cos cosh sin sinh tan tanh degrees radians

Округления: ceil floor trunc

Экспонента и логарифмы: exp log log10 log1p log2

Арифметика: dist remainder sqrt factorial gcd isqrt pow prod fsum

Великие постоянные: e pi inf nan isinf isnan isfinite

Задание

- Напишите функцию, которая рассчитывает НОК (наименьшее общее кратное) двух чисел.
- Подсказка. Попробуйте использовать функцию math.gcd

import *module_1* [,module_2]

- За один раз можно импортировать сразу несколько модулей, для этого их нужно перечислить через запятую после слова import

- **import** math, os
- print(math.sqrt(121))
- print(os.env)

import *module* as *my_alias*

- Если вы хотите задать псевдоним для модуля в вашей программе, можно воспользоваться вот таким синтаксисом

- **import** math **as** matan
- print(matan.sqrt(121))

Получение списка всех модулей Python установленных на компьютере

- Для того, чтобы получить список всех модулей, установленных на вашем компьютере достаточно выполнить команду:

- >>>help("modules")

- Через несколько секунд вы получите список всех доступных модулей.

Создание своего модуля в Python

- Чтобы создать свой модуль в Python достаточно сохранить ваш скрипт с расширением .py
- Теперь он доступен в любом другом файле.
- Например, создадим два файла: module_1.py и module_2.py и сохраним их в одной директории.

- # В первом запишем:
- # module_1.py
- def hello():
- print("Hello from module_1")

- # А во втором вызовем эту функцию:
- # module_2.py
- **from** module_1 **import** hello
- hello()
- #Убедитесь, что это работает, вызовите функцию одного модуля из другого.

Площадь круга

- Напишите функцию, которая рассчитывает площадь круга.
- Кто помнит формулу?

Пакеты модулей в Python

- Отдельные файлы-модули с кодом на Python могут объединяться в пакеты модулей. Пакет это директория (папка), содержащая несколько отдельных файлов-скриптов.
- Например, имеем следующую структуру:
- **my_file.py**
- **my_package**
- **__init__.py**
- **inside_file.py**
- В файле inside_file.py определена некая функция foo. Тогда чтобы получить доступ к функции foo, в файле my_file следует выполнить следующий код:
- **from** my_package**.**inside_file **import** foo

Задание

Отсортируйте список целых чисел по возрастанию последней цифры.

Для этого надо воспользоваться операцией sorted и написать правильную функцию для key=.

Более сложный вариант, отсортировать по последней цифре, внутри группы с одинаковыми цифрами – по возрастанию самих чисел.

Задача 7-1

Напишите программу, которая рассчитывает НОК 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]