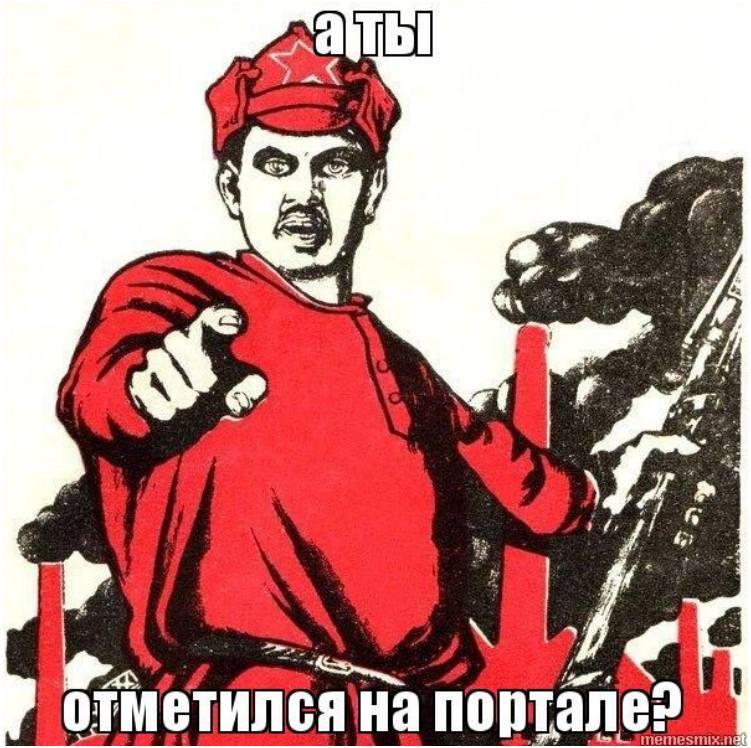
Python

Лекция 1: Основы Python

Отметься на портале!



План занятия

1. О курсе
2. Оценка сложности алгоритмов
3. Интерпретируемые и компилируемые языки
4. Типы данных
5. Управляющие конструкции
6. Функции
7. Классы
8. Домашнее задание

О курсе. Контакты

Амир Сафиуллин

(Программист-исследователь Mail.ru, МИФИ, ВШЭ)

- Telegram: @SafiAmir
- Вконтакте
- email: a.safiullin@corp.mail.ru

Миша Баранов

(Программист-исследователь Mail.ru, МИФИ, ВШЭ, Sberbank)

- Telegram: @Kinetikm
- Вконтакте
- email: mikhail.baranov@corp.mail.ru

и, конечно, беседа в ВКонтакте

О курсе. Структура

9 недель занятий

- Программирование на Python
- Основы работы с данными (парсинг, первичная обработка)
- Обработка данных простейшими алгоритмами машинного обучения
- Выбор способа хранения данных в боевой системе
- Реализация веб-сервиса (основы web на Python)
- Проект, сочетающий практически все, изученного за курс
- Возможность самостоятельно усложнять структуру проекта (полет фантазии)

Программа курса

О курсе. Правила

- Знаешь? умеешь? - помоги!
- Не понимаешь? - спроси!
- Не знаешь? - погугли! не удалось найти? - спроси!
- Не получается? - спроси!
- Нагуглил решение? - попробуй повторить сам! или разберись и модифицируй!
- Мы все друзья, мы должны дойти до конца курса! (Максимальный враг - это препод)
- Дедлайн по ДЗ - 2 недели, сдача после срока - 50% за неделю

Самый полезный сайт

О курсе. Подготовимся к занятию

- Repo: https://github.com/Kinetikm/PrPythonAtom.git
- Делаем fork репозитория на githube
- клонируем к себе
- переходим в папку репозитория, делаем:

  git remote add upstream https://github.com/Kinetikm/PrPythonAtom.git

Оценка сложности алгоритмов

Определение

Пусть $f, g: \mathbb{N} \to \mathbb{R}_{>0}$. Говорим, что f растёт не быстрее g и пишем $f(n) = O(g(n))$ или $f \preceq g$, если существует такая константа $c > 0$, что $f(n) \leq c \cdot g(n)$ для всех $n \in \mathbb{N}$.

Определение

Пусть $f, g : \mathbb{N} \to \mathbb{R}_{>0}$.

- $f(n) = \Omega(g(n))$ и $f \succeq g$, если существует положительная константа c, для которой $f(n) \geq c \cdot g(n)$ (f растёт не медленнее g)
- $f(n) = \Theta(g(n))$ и $f \asymp g$, если $f = O(g)$ и $f = \Omega(g)$ (f и g имеют одинаковую скорость роста)
- $f(n) = o(g(n))$ и $f \prec g$, если $f(n)/g(n) \to 0$ при $n \to \infty$ (f растёт медленнее g).

Постоянные множители можно опускать:

$$7n^3 = \Theta(n^3),\ \tfrac{n^2}{3} = \Theta(n^2)$$

Многочлен более высокой степени растёт быстрее:

$$n^a \prec n^b \text{ при } a < b$$
$$n = O(n^2),\ n^2 = O(n^4),\ \sqrt{n} = O(n)$$

Экспонента растёт быстрее многочлена:

$$n^a \prec b^n\ (a > 0, b > 1):$$
$$n^5 = O(\sqrt{2}^n),\ n^2 = O(3^n),\ n^{100} = O(1.1^n)$$

Многочлен растёт быстрее логарифма:

$$(\log n)^a \prec n^b\ (a, b > 0):$$
$$(\log n)^3 = O(\sqrt{n}),\ n \log n = O(n^2)$$

Медленнее растущие слагаемые можно опускать :

$$(f + g = O(\max(f, g)))$$

Оценка сложности алгоритмов. Задание

1) Разбиться на 8 команд. Ответить на вопросы. Спросим ***любого*** из команды (объяснить почему!)
2) Вопросы:
   a) У вас есть массив. Какова сложность алгоритма распечатки всех элементов массива?
   b) У вас есть статический массив. Какова сложность вставки элемента в массив по индексу?
   c) У вас есть динамический массив. Какова сложность вставки элемента в конец массива?
   d) У вас есть односвязный список. Какова сложность удаления элемента по значению?
   e) У вас есть односвязный список. Какова сложность вставки элемента?
   f) Чем отличается очередь от стека? Чем отличается их сложность?
   g) У вас есть список односвязный. Как определить, что он содержит цикл?

У вас есть 5 минут!

Интерпретируемые и компилируемые языки

Компиляция:

- исходный код -> низкоуровневый код
- Например:
  - в C++: исходный код -> машинный код
    - машинный код м.б. выполнен непосредственно процессором
  - в Java: исходный код -> байт код
    - байт код интерпретируется* JVM

Интерпретация (Python*):

1. прочитать инструкцию;
2. проанализировать инструкцию и определить соответствующие действия;
3. выполнить соответствующие действия;
4. если не достигнуто условие завершения программы, прочитать следующую инструкцию и перейти к пункту 2.

Типы данных. Числа

Все - объект!

Числа
- int ограничены только доступной памятью
- float ограниченная точность
- complex

```
a = 5
b = 5.0
print("Type of a is {}".format(type(a)))
print("Type of b is {}".format(type(b)))
```

```
Type of a is <class 'int'>
Type of b is <class 'float'>
```

```
5 + 5 =
5 / 5 =
5 - 5 =
5 * 5 =
5 ** 5 =
5 // 5 =
5 % 5 =
```

Идем в ноутбук!

Типы данных. Строки

Все - объект!

Неизменяемые !

Строка представляет собой последовательность символов. Мы можем использовать одинарные или двойные кавычки для создания строки.

```
s = "строка1"
print("Для строки {}, адрес: {}".format(s, id(s)))
s = s.replace("с", "С")
print("Для строки {}, адрес: {}".format(s, id(s)))
```

```
Для строки строка1, адрес: 140368660461024
Для строки Строка1, адрес: 140368660410768
```

Идем в ноутбук!

```
s = "строка1"
s[0] = "С"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-0d4e365a2eba> in <module>()
      1 s = "строка1"
----> 2 s[0] = "С"

TypeError: 'str' object does not support item assignment
```

Типы данных. Булевые

Все - объект!

True, False

```
a = True
b = True
print(id(a))
print(id(b))
```

```
94842126869248
94842126869248
```

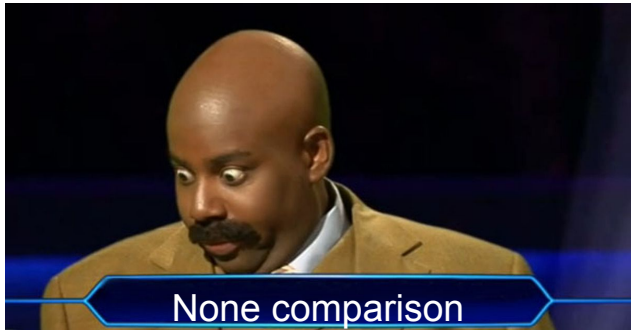
```
print(True == False)
print(True == True)
print(False == False)
```

```
False
True
True
```

and or not

```
True and True or False
```

```
True
```




```
if a is None:
    a = 10
```

```
a = 5
b = 5
a is b
```

```
True
```

```
a = 1000
b = 1000
a is b
```

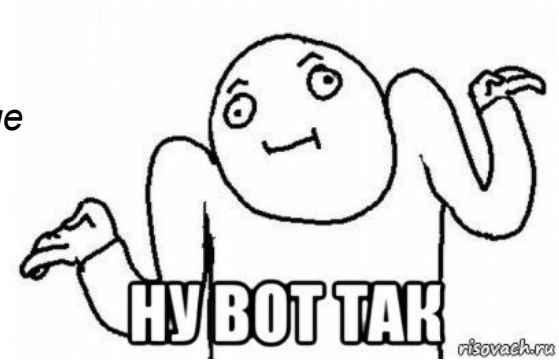
```
False
```

Почему???

Типы данных. Массивы

А массивов-то и нет!

Есть списки, *разнородные*




```
typedef struct {
    PyObject_VAR_HEAD
    PyObject **ob_item;
    Py_ssize_t allocated;
} PyListObject;
```

“Список в Python - динамический массив указателей”

Создание списка:

```
PyList_New(Py_ssize_t size)
{
    // Вычисляется реальный размер
необходимой памяти
    nbytes = size * sizeof(PyObject *);

    // Инициализируется ob_item
    if (size <= 0)
        op->ob_item = NULL;
    else {
        op->ob_item = (PyObject **)
PyMem_MALLOC(nbytes);
        memset(op->ob_item, 0, nbytes);
    }

    // Сохраняется количество выделенных
ячеек
    op->allocated = size;

    return (PyObject *) op;
}
```

Типы данных. Массивы

Все - объект!

Созданием массива:

```
a = []
a = list()
a = [1, 2, 'яблоко']
a = [i for i in range(10)]
a = [[i for i in range(10)] for _ in range(5)]
a = [0] * 5
```

```
a = [0, 1]
a.append("яблоко")
print(a)
```

```
[0, 1, 'яблоко']
```

```
a = [0, 1]
a.append(a)
print(a)
```

```
[0, 1, [...]]
```

```
a = [0, 1]
a.insert(1, 200)
print(a)
```

```
[0, 200, 1]
```

```
a = [0, 1, 2]
print(a.pop())
print(a)
```

```
2
[0, 1]
```

```
a = [0, 1, 2, 3]
print(" ".join(str(i) for i in a))
```

```
0 1 2 3
```

```
a = [0, 1, 2, 3]
b = [4, 5, 6]
a.append(b)
print(a)
```

```
[0, 1, 2, 3, [4, 5, 6]]
```

```
a = [0, 1, 2, 3]
b = [4, 5, 6]
a += b
print(a)
```

```
[0, 1, 2, 3, 4, 5, 6]
```

Идем в ноутбук!

Типы данных. Tuple

Кортеж - неизменяемый! набор упорядоченных данных
Неизменяемость позволяет быть ключом словаря (рассмотрим позже)

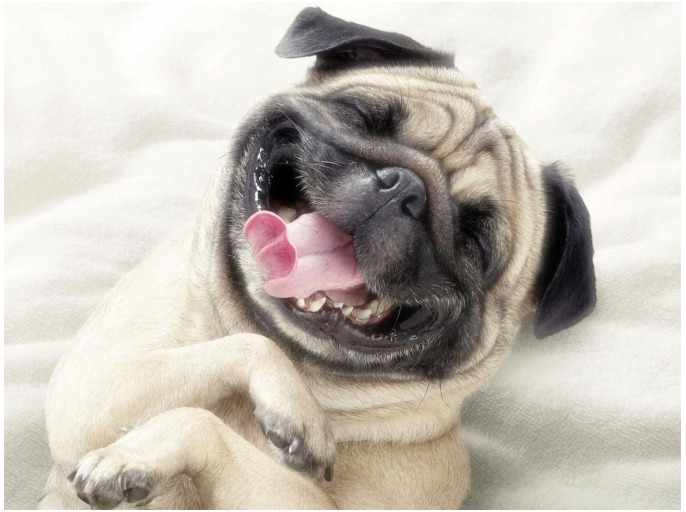
```
a = tuple()
print(a)
a = (1, 2)
print(a)
```

```
()
(1, 2)
```

```
a = 3
b = 5
a, b = b, a
print(a)
print(b)
```

```
5
3
```

В целом довольно просто =)



```
t = tuple([[1,2,3], ['a','b','c']])
```
executed in 11ms, finished 12:13:05 2018-09-23

```
type(t)
```
executed in 4ms, finished 12:13:06 2018-09-23

```
tuple
```

```
t
```
executed in 4ms, finished 12:13:06 2018-09-23

```
([1, 2, 3], ['a', 'b', 'c'])
```

```
t[0].append(12)
```
executed in 11ms, finished 12:13:07 2018-09-23

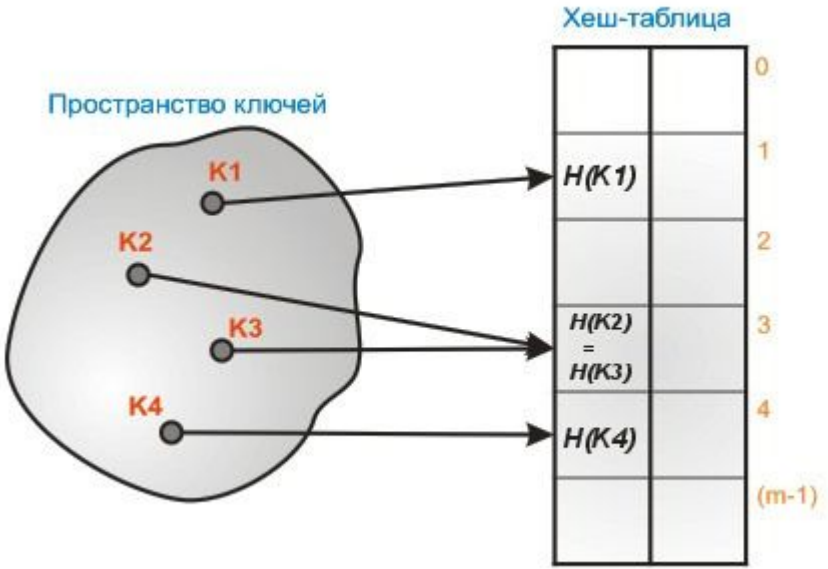
```
t
```
executed in 9ms, finished 12:13:07 2018-09-23

```
([1, 2, 3, 12], ['a', 'b', 'c'])
```

неизменяемый - сам tuple, но не содержащиеся в нем объекты

Хеширование. Открытое, закрытое

По сути хеширование - это отображения множества ключей на множество значений хеш-функции



Основная идея базовой структуры при открытом (внешнем) хешировании заключается в том, что потенциальное множество (возможно, бесконечное) разбивается на конечное число классов. Для B классов, пронумерованных от 0 до B-1, строится хеш-функция h(x)такая, что для любого элемента x исходного множества функция h(x) принимает целочисленное значение из интервала 0,1,...,B-1, соответствующее классу, которому принадлежит элемент x.

Если исходное множество состоит из N элементов, тогда средняя длина списков будет N/B элементов.

Честная копипаста

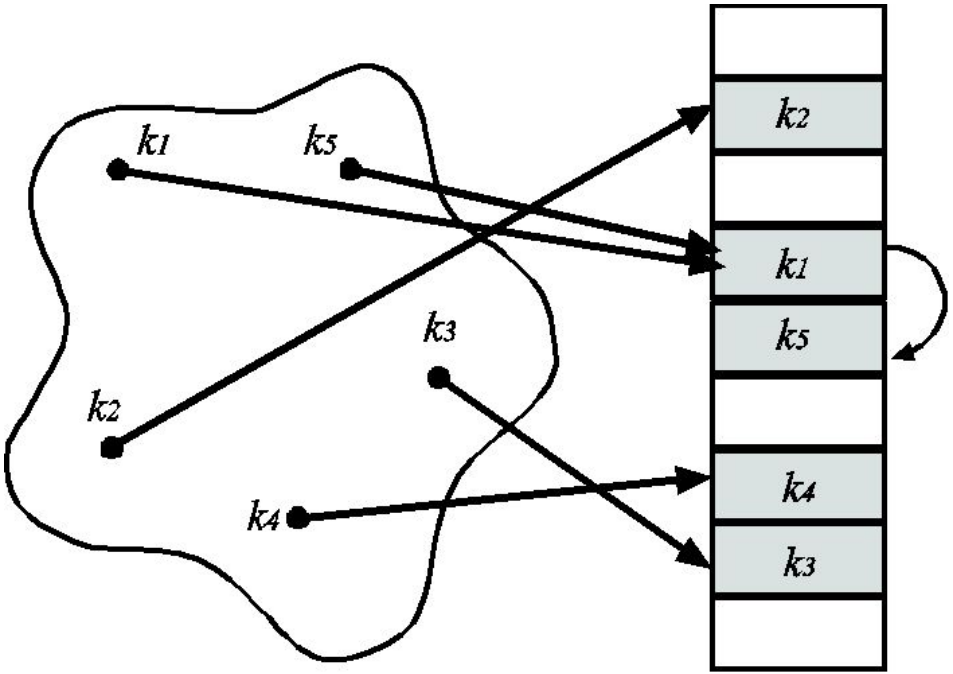
При закрытом (внутреннем) хешировании в хеш-таблице хранятся непосредственно сами элементы, а не заголовки списков элементов. Поэтому в каждой записи (сегменте) может храниться только один элемент. При закрытом хешировании применяется методика *повторного хеширования*. Если осуществляется попытка поместить элемент x в сегмент с номером h(x), который уже занят другим элементом (коллизия), то в соответствии с методикой повторного хеширования выбирается последовательность других номеров сегментов h1(x),h2(x),..., куда можно поместить элемент x. Каждое из этих местоположений последовательно проверяется, пока не будет найдено свободное. Если свободных сегментов нет, то, следовательно, таблица заполнена, и элемент x добавить нельзя. При поиске элемента x необходимо просмотреть все местоположения h(x),h1(x),h2(x),..., пока не будет найден x или пока не встретится пустой сегмент.

Честная копипаста


PAUSE

Типы данных. Dict

Словари - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами. В случае коллизий - метод открытой адресации. Коэффициент заполнения 2/3



Какой тип хеширования?

```
j = ((5 * j) + 1) % 2**i
```

Типы данных. Dict

Ключи только неизменяемые объекты!

```
d = dict(key="value")
d = {"key": "value"}
d = dict([("key_1", "value_1"), ("key_2", "value_2")])
d = dict.fromkeys(['key_1', 'key_2'], "value")
```

```
d[[1, 2]]
```

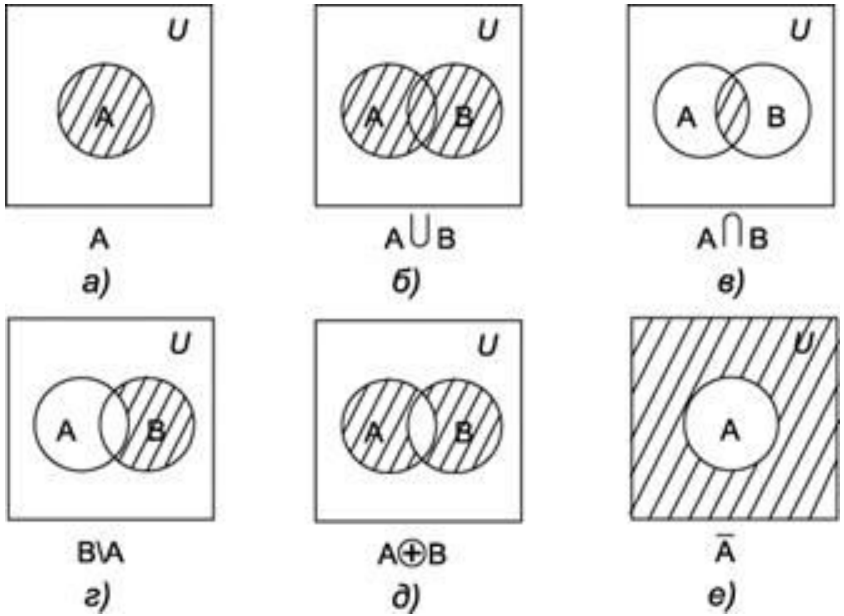
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-084c77bd943b> in <module>()
----> 1 d[[1, 2]]

TypeError: unhashable type: 'list'
```

```
d[(1, 2)] = "э"
```

Типы данных. Set

Множество - это структура данных, содержащая элементы в случайном порядке. С точки зрения реализации - это dict, у которого вместо value установлены заглушки.



Идем в ноутбук!

Управляющие конструкции. Условия

1) Управление вложенностью через табуляцию
2) If … elif … else
3) Тернарный оператор

```
a = 10
if a > 23:
    print("23")
elif a > 5:
    print("5")
else:
    print("other")
```

```
a = 10 if 5 > 4 else 3
```

```
a = "собака"
a or None
```

```
'собака'
```

```
a = "собака"
if a:
    print("б")
```

```
б
```

Операторы вхождения: in, not in

Операторы отношения: is, is not

Идем в ноутбук!

Генераторы и итераторы

Итератор — это объект-абстракция, который позволяет брать из источника, будь это stdin или, скажем, какой-то большой контейнер, элемент за элементом, при этом итератор знает только о том объекте, на котором он в текущий момент остановился.

```
class SimpleIterator:
    def __iter__(self):
        return self
    def __init__(self, limit):
        self.limit = limit
        self.counter = 0
    def __next__(self):
        if self.counter < self.limit:
            self.counter += 1
            return 1
        else:
            raise StopIteration


s_iter2 = SimpleIterator(5)
for i in s_iter2:
    print(i)
```

Генератор – это функция, которая будучи вызванной в функции *next()* возвращает следующий объект согласно алгоритму ее работы. Вместо ключевого слова *return* в генераторе используется *yield*.

```
def simple_generator(val):
   while val > 0:
        val -= 1
        yield 1


gen_iter = simple_generator(5)
print(next(gen_iter))
print(next(gen_iter))
```

range -это ни то ни другое

Управляющие конструкции. Циклы

while

```
a = 0
while a < 3:
    print(a)
    a += 1
```

```
0
1
2
```

for (по любому итерируемому объекту)

```
for let in "слово":
    print(let)
```

```
с
л
о
в
о
```

```
for v in [0, 1, 2, 3]:
    print(v)
```

```
0
1
2
3
```

```
d = dict.fromkeys(["key_1", "key_2"], "value")
for k, v in d.items():
    print(k, v)
```

```
key_1 value
key_2 value
```

```
for i in range(2, 8, 2):
    print(i)
```

```
2
4
6
```

continue

```
for i in range(2, 8, 2):
    if i == 4:
        continue
    print(i)
```

```
2
6
```

break

```
for i in range(2, 8, 2):
    if i == 4:
        break
    print(i)
```

```
2
```

else

```
for i in range(2, 8, 2):
    if i == 5:
        break
    print(i)
else:
    print("Вышли без break")
```

```
2
4
6
Вышли без break
```

Функции

Все - объект!

```
def function_name(arg_1=default_arg_1, **kwargs):
        …
        return ...
```

```
def summair(*args):
    return sum(args)
def summair_2(a, b, c=2):
    return a + b + c
print(summair(1, 2, 4))
print(summair(1, 2))
print(summair_2(1, 2))
```

```
7
3
5
```

```
(lambda x, y, z: x * y * z)(1, 2, 3)
```

```
6
```

lambda args: function

```
def bias_value(n):
    def value_passer(x):
        return x + n
    return value_passer
func_values = bias_value(10)
func_values(40)
```

```
50
```

Идем в ноутбук!

Классы

Все - объект!

Класс — тип, описывающий устройство объектов.
Объект — это экземпляр класса. Класс можно сравнить с чертежом, по которому создаются объекты.

```
class ClassName(Ancestor):
        def __init__(self)
```

```
class Test:

    def __init__(self, a, b):
        self._a = a
        self._b = b

    def summ(self):
        return self._a + self._b
test = Test(5, 4)
test.summ()
```

```
9
```

Идем в ноутбук!

Домашнее задание

1) git fetch upstream
2) git checkout -b homework_1
3) Появится папка homework_1
4) После выполнения задания git add, commit, push **origin** и pull-request на гитхабе
5) Суть дз:
    a) Недельник - задачи на каждый день недели, задачи на 5-10 минут на python. Pull-request после первого коммита будет сам обновляться. Просрочить задачи на 2 дня - 0 баллов за задачу. (1 задача - 1 балл)
    b) На 2 недели заполнить пропуски в классе HashMap и HashSet (пишем свой dict и set на python с открытым хешированием)

Оставьте обратную связь!!!

MUST READ

Источники

1. Питон в 3 страницах
2. Операторы в Python
3. Некоторые фишки работы с jupyter
4. Цикл статей про интерпретацию
5. Реализация списков в Python
6. Реализация словарей в Python
7. Сайт с ответами на многие вопросы
8. Итераторы и генераторы