DIRTY PYTHON 1.0

STONE 17TH PRESENTS

ЛЕКЦИЯ 5

ЧТО У НАС СЕГОДНЯ?

Продолжаем бодаться с функциями. Нас ждут:

- Функции как объекты
- Встроенные функции
- Лямбда-функция
- Функции высшего порядка
- Чуть-чуть алгоритмов
- Немножко рекурсии (знать что это такое надо, трогать не обязательно)

ВСПОМНИМ, ЧТО ЕСТЬ ФУНКЦИЯ?

- Функция – это кусок кода программы, которая часто повторяется, и мы ее вызываем при необходимости
- Этот кусок кода можно вынести в функцию, назвать и вызывать при необходимости.
- Зачем использовать функции?

✓Сократить код DRY – don’t repeat yourself – не повторяй сам себя

Лежит себе функция и лежит, жрать не просит

НЕМНОГО ПРО ОБЪЕКТЫ

- Функция это тоже объект

✓Можем объявить

✓Можем вызывать

✓Можем присвоить переменной

Давайте посмотрим, как это происходить.

Мы можем не только сам результат положить, но и саму функцию!!!

Теперь вызывая new_func у нас отрабатывает, та самая func

И ни что нам не мешает передавать функцию еще и как аргумент.

```
2 usages
def func(a, b):
    return a + b


print(func(4,5))

new_func = func

print(new_func(4,5))
```

ФУНКЦИИ КАК АРГУМЕНТЫ

- Импортируем тип данных функции (чисто для объявления в type hint):

from typing import Callable

- Передаем нашу функцию (только без круглых скобочек, чтоб не вызвать ее ненароком) в функцию и прочие аргументы для работы
- Вообще можно передать любую функцию, даже print (только при return он вернет None)
- И тут мы близенько подошли к декораторам – но это оставим для advanced
- Для чего это использовать? В некоторых случаях требуется выполнить однотипные действия в зависимости от того, какие данные прилетели. Тогда вполне можно вызывать общую функцию. Иногда это просто удобно))) Но вовсе не обязательно.

```python
from typing import Callable

def summa_two (a, b):
    return a + b

def minus_two (a, b):
    return a - b
# ниже создаем функцию, в которую передаем какую-то функцию
def operation(func: Callable, a, b):
    return func(a, b)
# если мы передаем функцию как аргумент, то вызывать ее не надо!
print(operation(summa_two, 10, 5))

print(operation(minus_two, 10, 5))
```

```
lection5
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Le
15
5

Process finished with exit code 0
```

Список функций

DIRTY PYTHON

- Проделываем разные действия с одним набором данных
- Сделали список функций и по нему прошлись в цикле: главное в список добавить именно названия, а не вызов функции

```python
def summa(a, b):
    return a + b

def mult(a, b):
    return a * b

def diff(a, b):
    return a - b

def div(a, b):
    return a / b

for func in [summa, mult, diff, div]:
    print(func(10, 5))
```

for func in [summa, mult, diff,...

lection5

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPyt
15
50
5
2.0
```

ВСТРОЕННЫЕ ФУНКЦИИ

- Многие вы уже знаете и даже не задумываетесь, что это функции

✓ len() – длина коллекции

✓ max() – максимум ир коллекции

✓ min() – минимум из коллекции

✓ sum() – сумма элементов коллекции

✓ sorted() – сортировка коллекции

✓ print() – печать в терминал

✓ input() – ввод через терминал

- Если хотите все посмотреть – то нужно import builtins и вызвать в help() или print(dir(builtins)), если хотите только названия
- И их прям очень много)))

О MIN, MAX, SORTED ПОПОДРОБНЕЕ

DIRTY PYTHON

- Мы можем сравнивать строки по длине, если укажем что key = len
- Более того, ключ можно передать даже более сложный
- Например, можно отсортировать по количеству букв а
- И не забываем про reverse

```
first = 'sdfsdgdrfhgd'
second = 'adsada'

print(max(first, second, key=len))
```

builtins
def len(__obj: Sized) -> int
Return the number of items in a container.
`len(__obj)` on docs.python.org

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:
sdfsdgdrfhgd
```

```
first = 'asdasdasfadfasda'
second = 'adsada'
third = 'dskjhkhjkuyjy'

print(sorted((first, second, third), key=lambda x: x.count('a')))
```

Вот здесь записана лямбда-функция, которая делает подсчет букв а в каждом элементе, а уже это значение используется в качестве ключа

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProject1\test.py
['dskjhkhjkuyjy', 'adsada', 'asdasdasfadfasda']
```

МУДРОСТЬ ОТ СТОУНА

D1RTY PYTHON

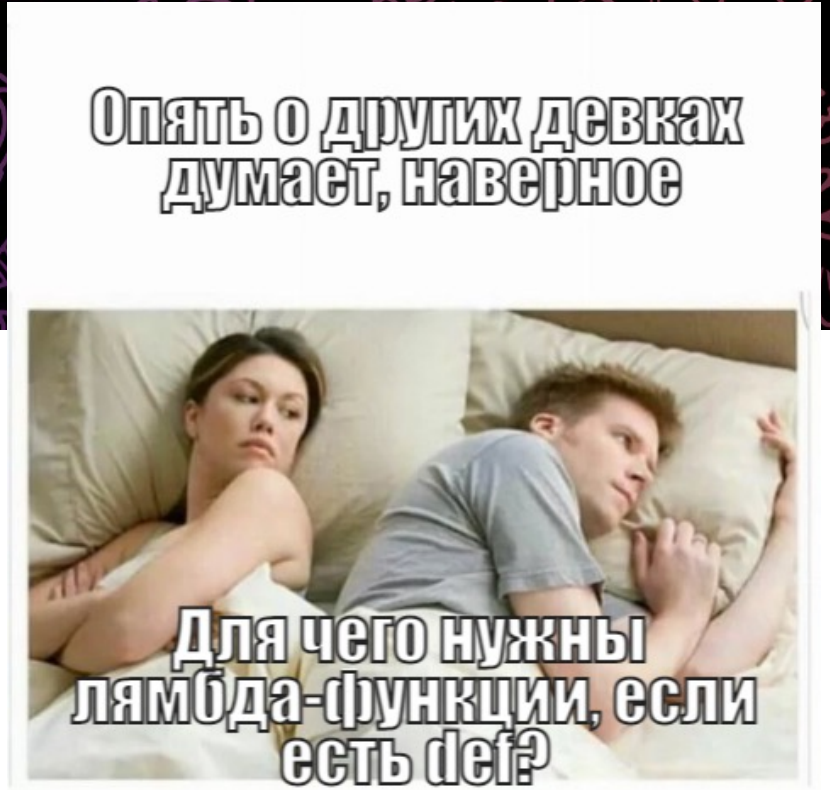
ЗНАТЬ КАК - ЭТО ПРЕКРАСНО,

НО САМОЕ ГЛАВНОЕ - ЗНАТЬ ЧТО И ПРИМЕНИТЬ В НУЖНЫЙ МОМЕНТ!

И ДА! ГУГЛИТЬ НЕ ЗАЗОРНО!

РАЗБЕРЕМСЯ С ЛЯМБДОЙ

DIRTY PYTHON



- Лямбда – анонимная однострочная функция, которая что-то возвращает
- Пишем lambda, что передаем (аргументы – сколько угодно) , двоеточие и действие.

lambda a, b: a + b

- И эту лямбду встраиваем куда угодно.
- Удобно, если функция далее встречаться не будет, чтобы не выносить ее отдельно – экономим место и память, лямбда компилируется по месту.
- Часто используют в качестве ключей и фильтров.
- Но если хотите, можете и объявить отдельно)
- Главное, чтобы при встрече с лямбдой вы не терялись.

```
32  def operation(func, a, b):
33      return func(a, b)
34  # (lambda a, b: a + b ) - это аналог функции def summa(a, b):
35  #                                                 return a + b
36  # записанный в одну строчку
37  print(operation((lambda a, b: a + b ), 5, 10))
```

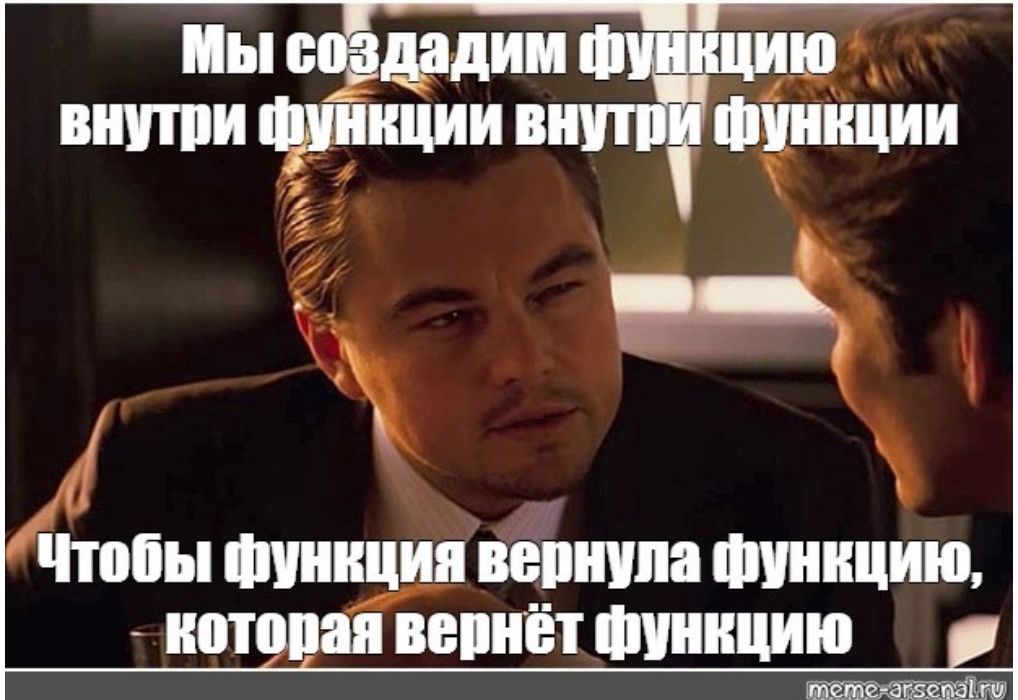
operation()

lection5

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPytho
15

Process finished with exit code 0
```

Функция внутри функции

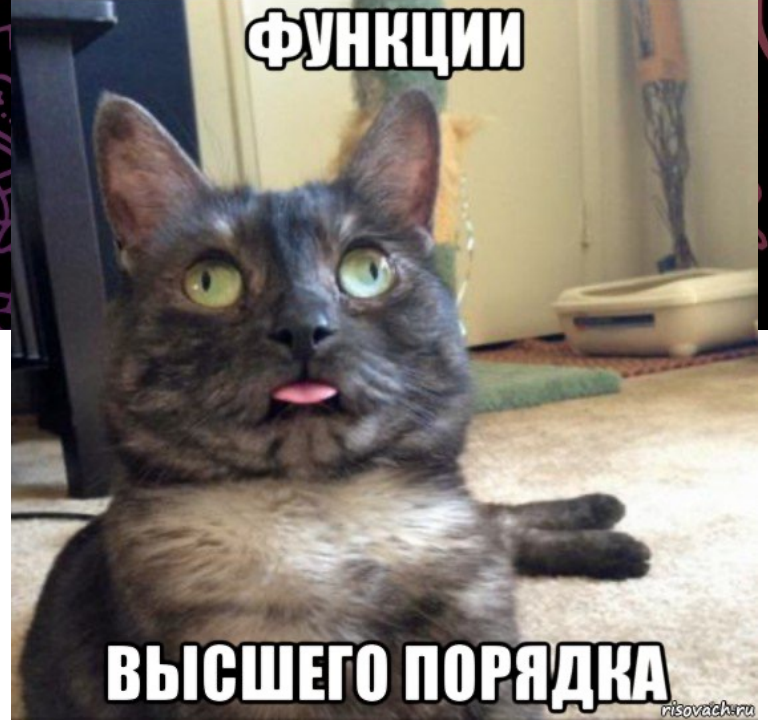
- Внутри функции можно использовать другую функцию
- И видна она может быть только внутри, а снаружи – уже нет




```
1 usage
def outter(a,b):
    def inner():
        print('Я внутри')
    inner()
    return a + b

print(outter(5,10))
inner()
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\pyt
Я внутри
15
```

Функции высшего порядка



- В принципе ничего нового нет, просто удобнее – чтобы не писать лишние циклы и т.д.
- Например, чтобы получить из строки список, но не строковых значений, а int
- При помощи map можно применить другую функцию ко всем элементам последовательность и сократить запись до одной строки
- Но чтобы получить не генератор, а именно список – оберните map например в list()
- В качестве функции можно указать ту же лямбду

```
4   string = '1234567890'
5   num_list = list(string)
6   print(num_list)
7   for i in range(len(num_list)):
8       num_list[i] = int(num_list[i])
9   print(num_list)
10
11  # for item in num_list:
12  #     item = int(item)
13
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkSho
['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

```
10
11  num_list = list(map(int, num_list))
12
13  print(num_list)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\Wo
['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

ЕЩЕ ПРИМЕРЫ С MAP

DIRTY PYTHON

- Используем лямбда-функцию, чтобы что-то добавить к каждому элементу

```python
string = '1234567890'
num_list = list(string)
print(num_list)

# for i in range(len(num_list)):
#     num_list[i] = int(num_list[i])

num_list = list(map(lambda x: x + 'x1', num_list))

print(num_list)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonPr
['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
['1x1', '2x1', '3x1', '4x1', '5x1', '6x1', '7x1', '8x1', '9x1', '0x1']
```

- Или проверку на цифру

```python
string = '1234567890'
num_list = list(string)
print(num_list)

# for i in range(len(num_list)):
#     num_list[i] = int(num_list[i])

num_list = list(map(lambda x: x.isdigit(), num_list))

print(num_list)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProject1\test.py
['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
[True, True, True, True, True, True, True, True, True, True]
```

FILTER

DIRTY PYTHON

- То же работает с коллекцией
- Нужно указать условие для фильтра
- В том числе можно использовать lambda: в данном случае мы будем получать True или False и фильтровать, что добавлять, а что нет.
- Если число четное – добавляем в список
- И точно так же можно сделать для нечетных
- Фильтр по очереди загоняет элементы в коллекцию, загоняет в функцию и если получает True – добавляет в новый список

```
# num_list = list(map(lambda x: x.isdigit(), num_list))
num_list = list(filter(lambda x: int(x)%2 == 0, num_list))

print(num_list)

# for item in num_list:
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProject1\test.py
['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
['2', '4', '6', '8', '0']
```

```
# num_list = list(map(lambda x: x.isdigit(), num_list))
num_list = list(filter(lambda x: int(x)%2 == 1, num_list))

print(num_list)

# for item in num_list:
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
['1', '3', '5', '7', '9']
```

FILTER

- Под капотом у filter происходит примерно следующее
- Фильтр перебирает элементы на соответствие условию и пропускает в список
- В принципе то же самое можно сделать и при помощи list comprehension
- И иногда map или filter использовать удобно: они хоть и старенькие, но не бесполезные

```
string = '123dsad456asd78adssa9asd0'
num_list = list(string)
print(num_list)
new_list = []
for i in range(len(num_list)):
    if num_list[i]%2 == 0:
        new_list.append(num_list[i])
```

ENUMERATE

DIRTY PYTHON

- Enumerate используется для нумерации – возвращает кортеж из индекса (и кстати, можно задать начало нумерации) и элемента
- Очень удобно во всяких циклах использовать вместо range, особенно, когда вам и индекс нужен, и элемент – например, для переприсваивания
- Опять же range занимает место в памяти

```python
n_list = [i for i in 'asdsafdsgffdgh']
print(n_list)

for i, letter in enumerate(n_list,100):
    print(i, letter)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProje
['a', 's', 'd', 's', 'a', 'f', 'd', 's', 'g', 'f', 'f', 'd', 'g', 'h']
100 a
101 s
```

```python
n_list = [i for i in 'asdsafdsgffdgh']
print(n_list)

for i, letter in enumerate(n_list):
    if letter == 's':
        n_list[i] = 'SS'
print(n_list)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProject1\test.py
['a', 's', 'd', 's', 'a', 'f', 'd', 's', 'g', 'f', 'f', 'd', 'g', 'h']
['a', 'SS', 'd', 'SS', 'a', 'f', 'd', 'SS', 'g', 'f', 'f', 'd', 'g', 'h']

Process finished with exit code 0
```

ZIP

DIRTY PYTHON

- Из нескольких коллекций делает кортеж поэлементно и заносит в новый список.
- Проще показать, чем объяснить словами)))
- В случае, если длина не совпадает – зиповать будет по минимальной длине, в данном случае получим список с одним кортежем из трех элементов.

```python
a_list = [i for i in 'asdsafdsgffdgh']
b_list = [i for i in '12321343234556']
c_list = [i for i in '*^^$&^@*(&#^#(']
print(a_list)
print(b_list)
print(c_list)

new_list = list(zip(a_list, b_list, c_list))
print(new_list)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProject1\test.py
['a', 's', 'd', 's', 'a', 'f', 'd', 's', 'g', 'f', 'f', 'd', 'g', 'h']
['1', '2', '3', '2', '1', '3', '4', '3', '2', '3', '4', '5', '5', '6']
['*', '^', '^', '$', '&', '^', '@', '*', '(', '&', '#', '^', '#', '(']
[('a', '1', '*'), ('s', '2', '^'), ('d', '3', '^'), ('s', '2', '$'), ('a', '1', '&'), ('f', '3',
```

```python
a_list = [i for i in 'asdsafdsgffdgh']
b_list = [i for i in '1']
c_list = [i for i in '*^^$&#^#(']
print(a_list)
print(b_list)
print(c_list)

new_list = list(zip(a_list, b_list, c_list))
print(new_list)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProject1\test.py
['a', 's', 'd', 's', 'a', 'f', 'd', 's', 'g', 'f', 'f', 'd', 'g', 'h']
['1']
['*', '^', '^', '$', '&', '#', '^', '#', '(']
[('a', '1', '*')]

Process finished with exit code 0
```

ZIP_LONGEST

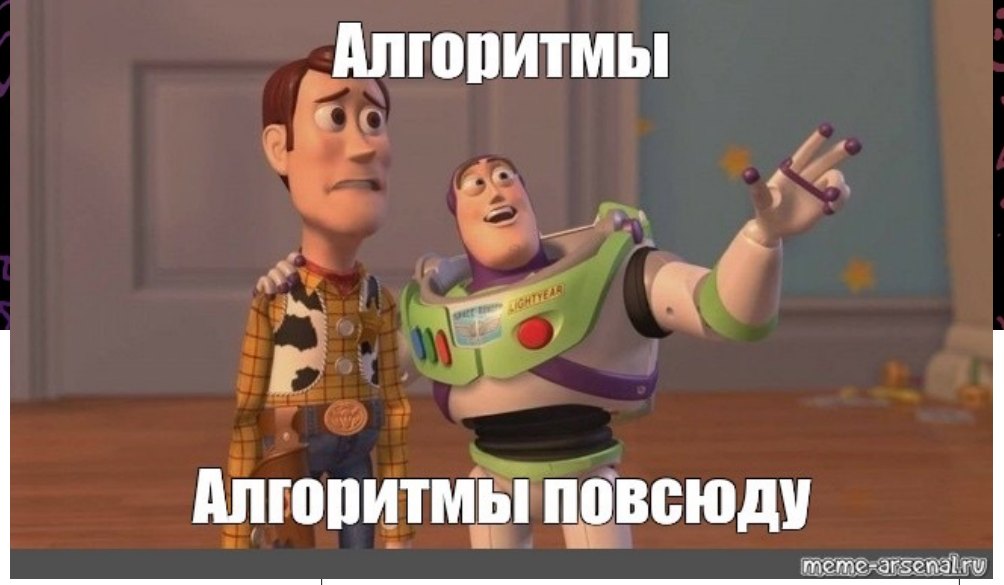
- Еще один вариант получить кортеж из коллекций разной длинны – использовать zip_longest из коллекции itertools: длина списка кортежей будет определяться по самому длинному, а недостающие элементы будут сформированы как None или можно указать заместитель через fillvalue=’какое-то значение’

```
from itertools import zip_longest

a_list = [i for i in 'asdsafdsgffdgh']
b_list = [i for i in '1']
c_list = [i for i in '*^^$&#^#(']
print(a_list)
print(b_list)
print(c_list)

new_list = list(zip_longest(a_list, b_list, c_list))
print(new_list)
```

```
D:\WorkSnopry\.my_venv_3.11.3\Scripts\python.exe D:\WorkSnopry\PythonProject1\test.py
['a', 's', 'd', 's', 'a', 'f', 'd', 's', 'g', 'f', 'f', 'd', 'g', 'h']
['1']
['*', '^', '^', '$', '&', '#', '^', '#', '(']
[('a', '1', '*'), ('s', None, '^'), ('d', None, '^'), ('s', None, '$'), ('a', None, '&'), ('f', None, '#'),

Process finished with exit code 0
```

АЛГОРИТМЫ

DIRTY PYTHON



Рекомендую «Грокаем алгоритмы» Адитья Бхаграва

Лучше не расскажешь, а пересказывать неинтересно

Там есть и про различные сортировки, бинарный поиск, обходы графов, хэш-таблицы и жадные алгоритмы…

Читаем аккуратно. Как только чувствуете, что перестали понимать – отложите, займитесь синтаксисом, покодьте немножко… Потом вернитесь)

Надо, чтобы каждая глава оставляла после себя понимание.

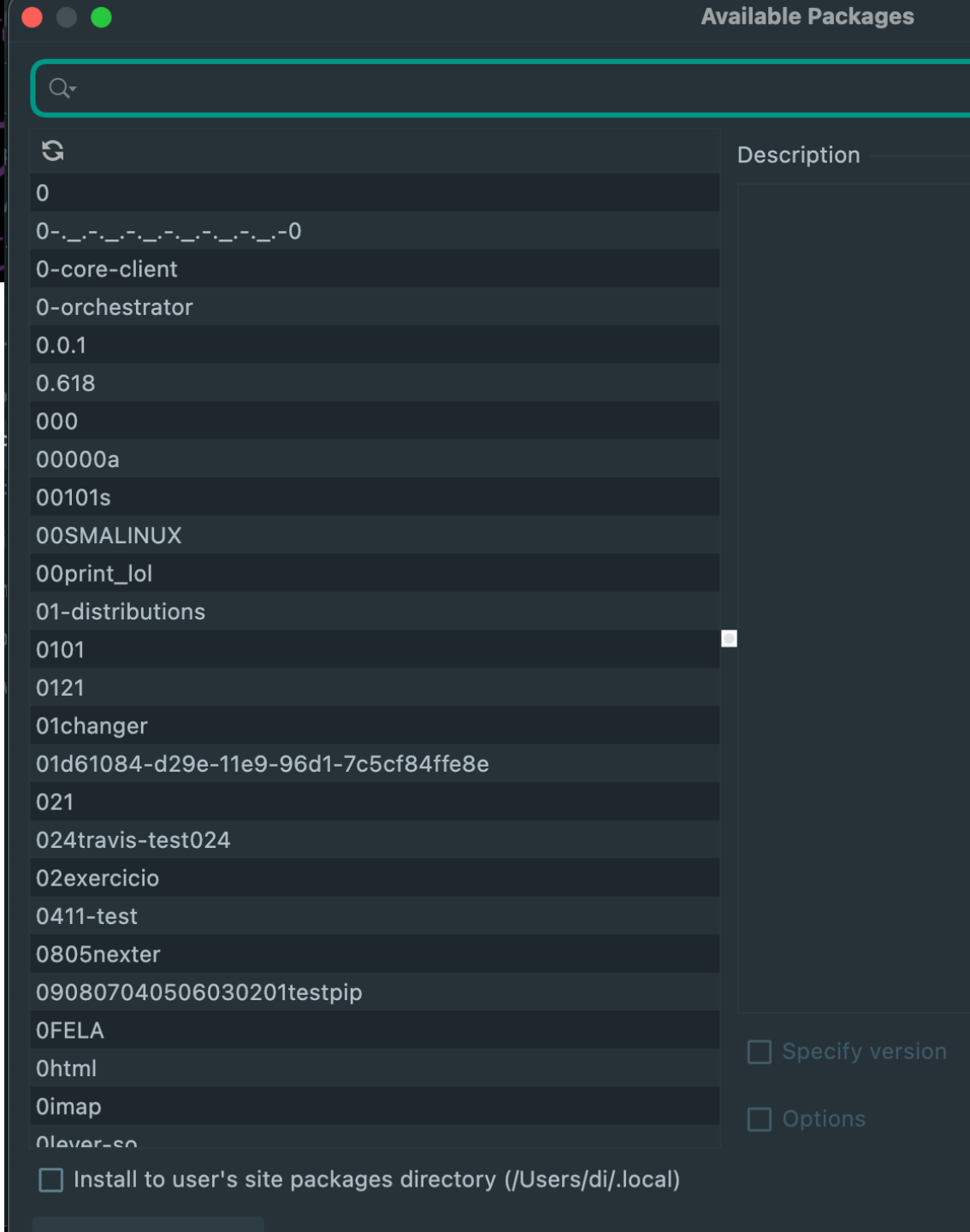
Перед написанием кода мы должны составить алгоритм – определенная конструкция, которая позволит решить ее на любом языке.

В принципе все популярные алгоритмы уже есть в библиотеках, но для лучшего понимания надо разобрать, как это работает. На готовом не рекомендую выезжать, но это не значит, что каждый раз надо выводить эти алгоритмы.

БИБЛИОТЕКИ

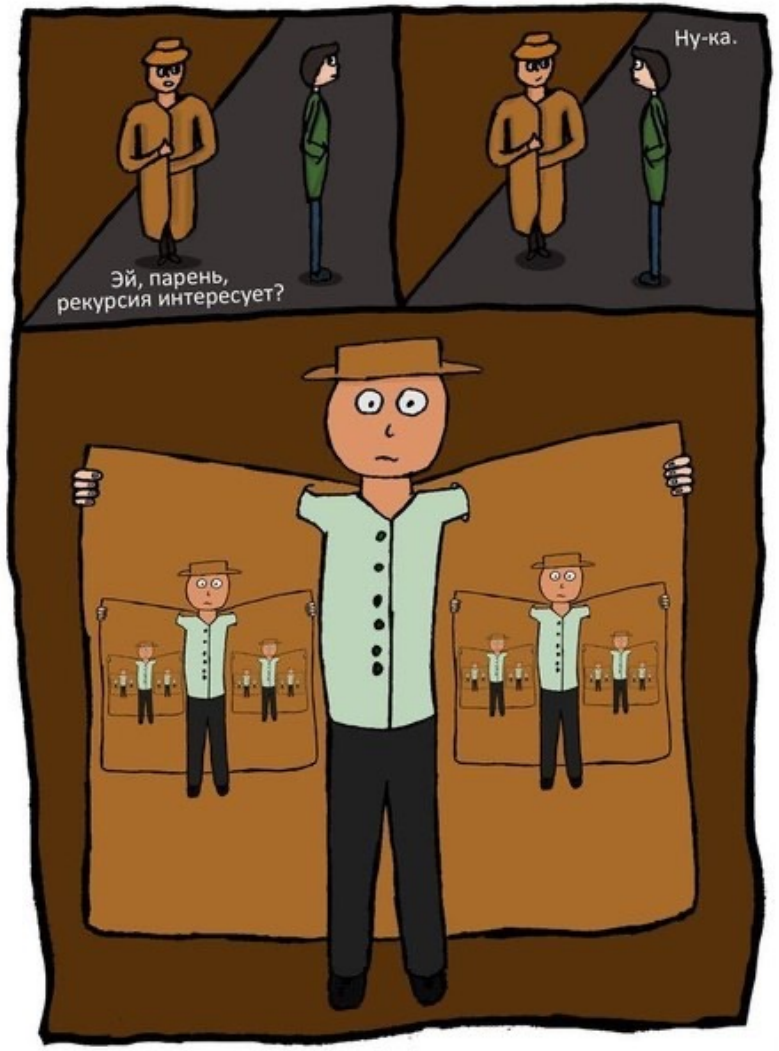
DIRTY PYTHON

- Если хотите посмотреть на библиотеки…
- File -> Settings -> Project… -> Python Interpreter тут вы увидите все подключенные к текущему окружению библиотеки
- А если вы нажмете на плюсик – то получите список всех доступных библиотек (и офигеете)))
- И это только официально известные, а на гитхабе еще народные творцы



РЕКУРСИЯ

- Функция, которая вызывает сама себя
- Если ее можно не использовать – не используйте
- Сама по себе довольно тяжело читается, а часто еще и память жрет
- Звучит вроде не сложно: обозначаем точку выхода, в противоположном случае вызываем функцию…
- Сказать проще, чем написать
- НО ЗНАТЬ НАДО (можете столкнуться с чужим кодом или на собеседовании)



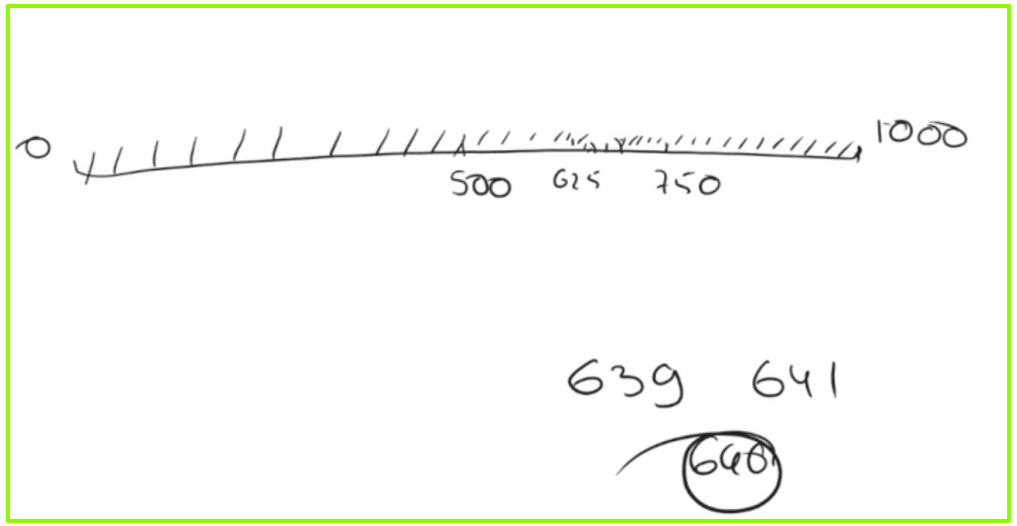
МУДРОСТЬ ОТ СТОУНА

D1RTY PYTHON

ПИШИТЕ ПОНЯТНО, ДОСТУПНО И КРАСИВО –
И ВАМ БУДЕТ СЧАСТЬЕ!

КАК РАБОТАЕТ БИНАРНЫЙ ПОИСК

- Бинарный – потому что пополам: каждый раз мы сокращаем область поиска вдвое.
- Данная история работает только с упорядоченной коллекцией



И ЧТО ДАЛЬШЕ?

D1RTY PYTHON

И ЧТО ДАЛЬШЕ?

- НА ДОМ 1 МАКСИМУМ 2 ЗАДАЧКИ, НО ОБЪЕМНЫЕ
- РАЗБЕРЕМ ДОМАШКУ В ВОСКРЕСЕНЬЕ
- НАДО КОДИТЬ ОЧЕНЬ МНОГО!
- ЗАГЛЯНИТЕ К БОТУ @DPZO_BOT ТАМ ЕСТЬ ЗАДАЧКИ - ПОРЕШАЙТЕ

D1RTY PYTHON