

Разработка web сервисов для научных и прикладных задач

Введение. Знакомство с языком
Python.

Суханов А.Я.

Идея и цели дисциплины.

- Изучение современных средств разработки ПО (языка программирования) для дальнейшего использования при изучении других дисциплин и реализации своих академических исследований.
- Изучение современных технологий для решения научных и прикладных задач.
- Изучение технологий для предоставления сервисов пользователям, технологий высоконагруженных систем.

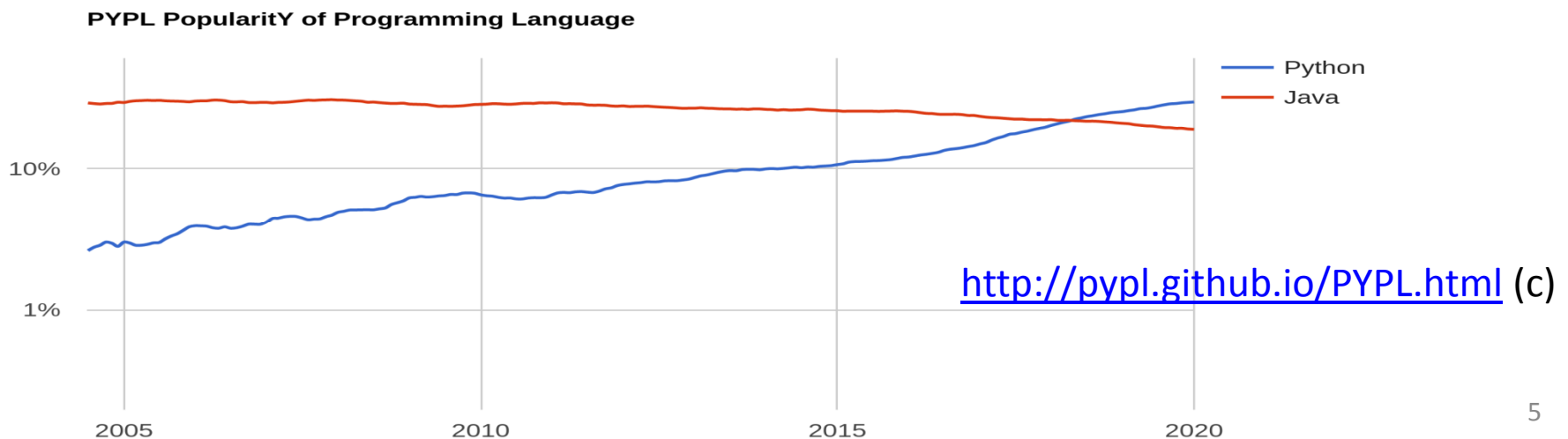
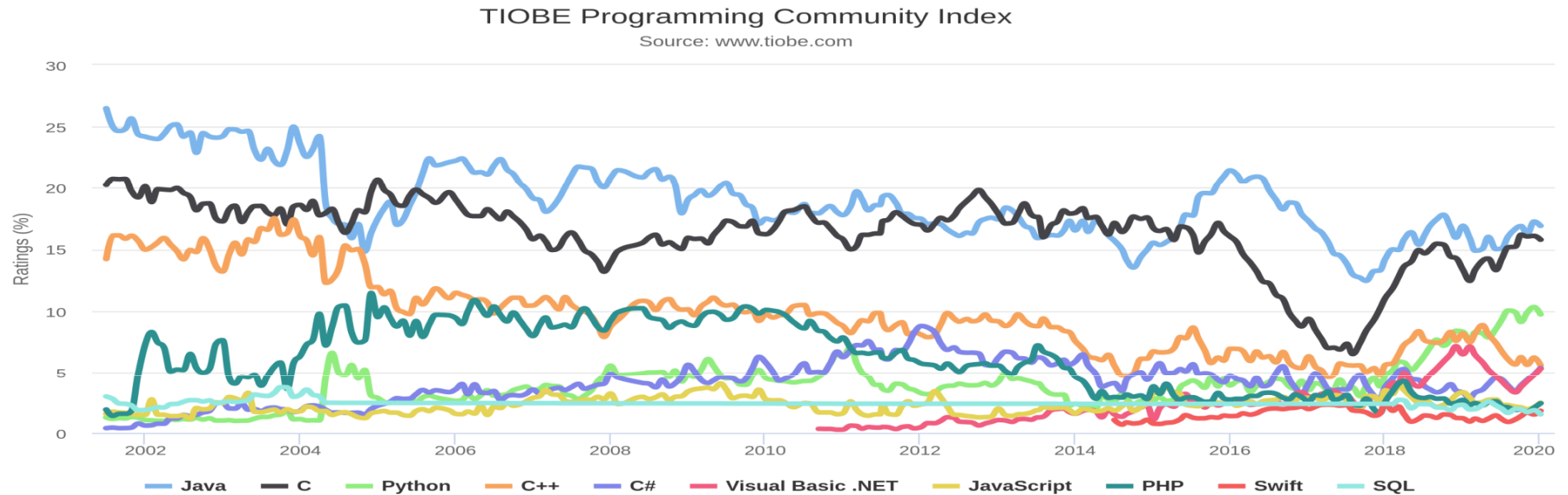
To be or not to be

- <https://porfirevich.ru/>
- Сети трансформеры
- GPT-2 нейросеть от OpenAI для NLP (Nature Language Prosessing)
- <https://sketch2code.azurewebsites.net>
- А что если? Транслировать с естественного языка в куски кода
- https://www.youtube.com/watch?v=lssDX_zOwOo
- <https://www.youtube.com/watch?v=W5wD5mleKws>

Изучение средств разработки. Python. Почему Python.

- Why not?
- Язык подходит для быстрой реализации какого-либо проекта, осуществления какой-либо идеи, быстрой оценки ее работоспособности. Высокая динамичность языка.

Популярность Python. Один из самых популярных языков программирования в настоящее время, второй после Java и Си (можно похливарить).



Python лаконичный.

- Получается короткий код. Интуитивно понятный синтаксис. Просто изучать.
- Нет кучи фигурных скобочек с точками запятой. ?!??!)

```

// A Java program for splitting a string
// using split()
import java.io.*;
public class Test
{
    public static void main(String args[])
    {
        String Str = new String("Geeks-for-Geeks");

        // Split above string in at-most two strings
        for (String val: Str.split("-", 2))
            System.out.println(val);

        System.out.println("");

        // Splits Str into all possible tokens
        for (String val: Str.split("-"))
            System.out.println(val);
    }
}

```

Python 2.x

```

line = "Geek1 \nGeek2 \nGeek3";
print line.split()
print line.split(' ', 1)

```

Что можно делать с Python?

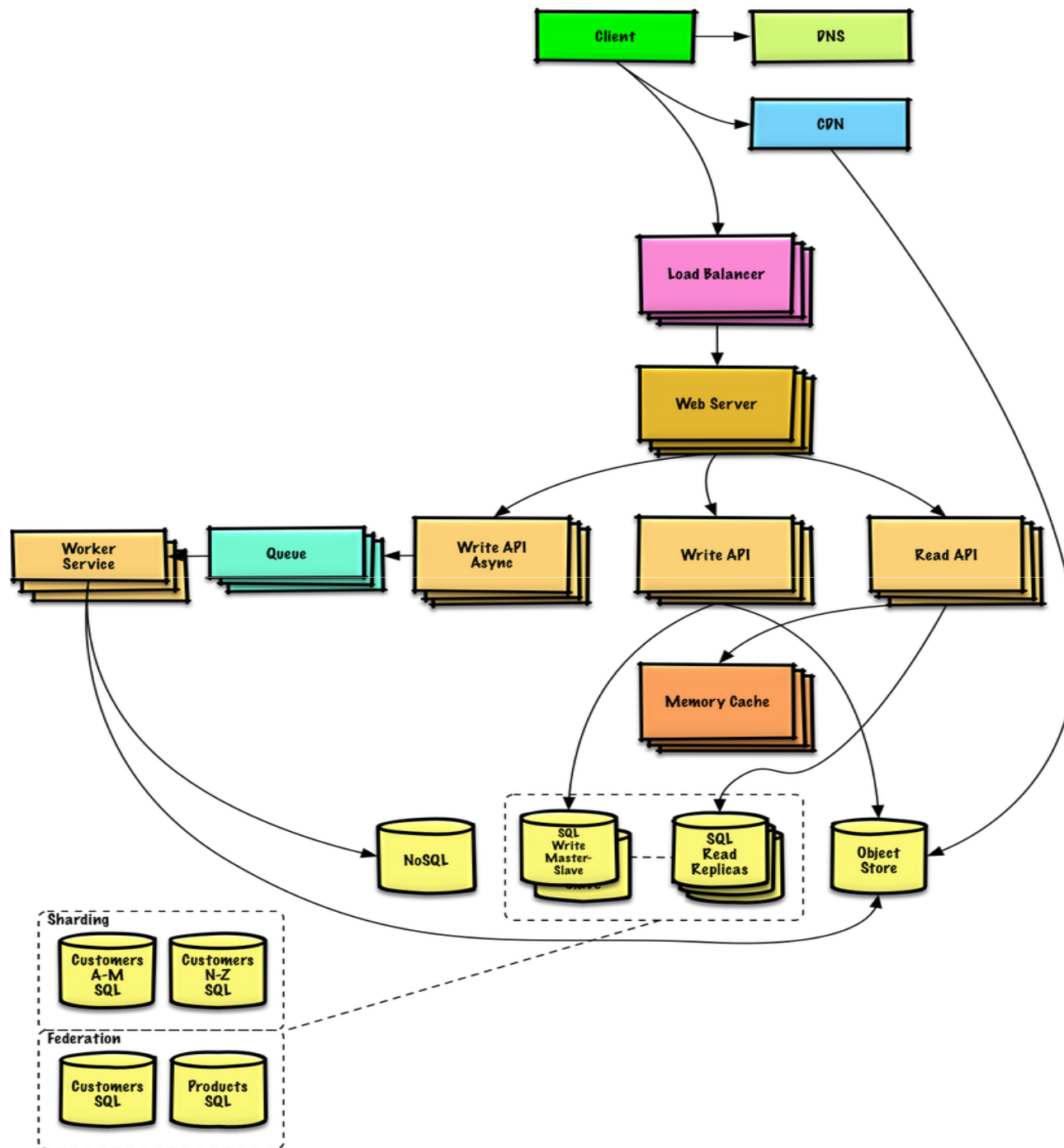
- Множество реализованных библиотек для решения научных задач, машинного обучения, нейронных сетей, обработки данных, больших данных, библиотеки и фреймворки реализации сетевых сервисов, распределенных вычислений и тензорных вычислений. Возможности обращения к библиотекам Си, различным СУБД. Есть свой JIT-компилятор PyPy.

Что реализовано на Python?

- Все эти ваши [Keras](#), [Tensorflow](#), [Flask](#), [Django](#), [Theano](#) и т.д.
- [Skikit-Learn](#)
- Больше можно посмотреть на
- <https://proglib.io/p/50-python-projects/>
- Интересные
- Python [Google Images Download](#)
- Утилита командной строки, которая позволяет искать изображения в Google Images по ключевым словам или фразам и загружать их на компьютер. Скрипт также можно запускать из любого python-файла.
- Библиотека [Mask R-CNN](#) предназначена для обнаружения объектов и сегментации изображений.
- [Detectron](#) – так же для сегментации.

System Design Primer

Коллекция ресурсов для создания масштабируемых систем



Важность умения писать красивый и правильный код.

- Думать о красоте кода.
- Рефлексии !?!?)

История

- Гвидо ван Россум задумал Python в 1980-х годах, а приступил к его созданию в декабре 1989 года в центре математики и информатики в Нидерландах. Язык Python был задуман как потомок языка программирования ABC (для обучения вместо Basic), способный к обработке исключений и взаимодействию с операционной системой Амёба (решил создать язык для создания прикладных и инструментальных средств). Несколько раз его увольняли, но он продолжал оставаться фанатом своего языка до конца)



Python's first and foremost influence was ABC, a language designed in the early 1980s by Lambert Meertens, Leo Geurts and others at CWI. ABC was meant to be a teaching language, a replacement for BASIC, and a language and environment for personal computing. It was designed by first doing a task analysis of the programming task and then doing several iterations that included serious user testing. My own role in the ABC group was mainly that of implementing the language and its integrated editing environment. Бритва Оккама. Похвала двоеточиям после блоков (для начинающих пользователей это было понятнее просто отступов). Далее автор указывает на неудобство работы с табличными данными в ABC, которые сортировались по ключу и представлялись в виде В дерева, была произведена замена на массивы списки, не было взаимодействия с ОС и файловой системой.

Влияние

- ABC
- Modula-3
- Взята обработка исключений и система модулей.
- Си
- Возможность взаимодействовать с Си и с ОС.

Философия Python или голландца Гвидо!? По началу Гвидо следовал следующим правилам, философский дзен Python можете найти на википедии

- Заимствовать идеи от куда угодно, когда это имеет смысл.
- «Все должно быть как можно проще, но не проще». (Эйнштейн)
- Делай одну вещь хорошо («философия UNIX»).
- Не беспокойтесь о производительности - планируйте оптимизацию позже, когда это необходимо.
- Не боритесь с окружающей средой и плывите по течению.
- Не пытайтесь достичь совершенства, потому что «достаточно хорошо» часто им и является.
- (Следовательно) иногда можно срезать углы, особенно если вы можете сделать что-то позже.

PEP

- Развитие языка Python происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов PEP. PEP - Python Enhancement Proposal - это предложения по развитию питона <https://www.python.org/dev/peps/> Процесс PEP является основным механизмом для предложения новых возможностей и для документирования проектных решений, которые прошли в Python.
-
- Самым известным PEP является PEP8 - это свод рекомендаций в оформлении кода. Рекомендации написаны кровью из глаз программистов. <https://www.python.org/dev/peps/pep-0008/>
- Новости по теме:
-
- PEP 471 (Python 3.5) добавил в модуль os новую функцию - scandir.
-

Добавление новых возможностей в python

- До 2018 года решение о добавлении новых возможностей рассматривал лично Гвидо.
- С некоторой точки зрения это было хорошо.
- 2008 Python3. Не совместим частично с Python2.
- В Python2 накопилось много проблем и появился Python3.

Пример кода. Читаемость.

```
def magic(top):  
    acc = []  
    for entry in os.scandir(top):  
        if entry.is_file() and entry.name.endswith(".py"):  
            acc.append(entry.path)  
    return acc
```

Python интерпретатор, транслирующий в байт-код и затем интерпретирующий этот байт код. Используется код для стековых виртуальных машин, он медленнее чем для регистровых на десятки процентов и подвержен уязвимостям, но благодаря переносимости и быстрой разработки все равно python получил широкое распространение, кроме того есть варианты с jit-компиляторами. В современных Jit-компиляторах используется регистровые виртуальные машины.

```
>>> print("Hello, World!")  
Hello, World!
```

Байт-код:

```
>>> import dis #импортируем модуль "dis" - Disassembler of Python byte code into mnemonics.  
>>> dis.dis('print("Hello, World!")')  
1          0 LOAD_NAME                0 (print)  
          2 LOAD_CONST              0 ('Hello, World!')  
          4 CALL_FUNCTION             1  
          6 RETURN_VALUE
```

Примеры реализаций.

```
❖ ❖ print("Hello World")  
Hello World  
❖ █
```

- **CPython** (PVM) – основная оригинальная реализация Python (*.py->*.pyc->выполнение PVM)
- **JPython** – трансляция(компиляция) в байт код Java, исполняемый JVM. Поддержка Java компонент, классов. Пример того же – Kotlin (Другой язык).
- **IronPython** – транслятор для .Net или Mono. Соответственно байт код CIL исполняется CLR.
- **Psyco** – транслирует часть байт кода в машинный, что ускоряет исполнение порой в 4-100 раз.
- **Shedskin C++** - транслирует Python на язык C++, потом можно скомпилировать.
- **Py2exe, freeze, PyInstaller** – создает исполнимый exe модуль или исполнимый модуль для Linux, который тащит с собой PVM.
- **PyPy** – Jit-компилятор python, сразу с языка в машинный на лету, не поддерживает некоторые библиотеки или частично, например numpy не все, поддерживает Flask, Django. Намного быстрее Cpython при сравнении напрямую для численных задач без использования numpy (реализовать самостоятельно если работу с массивами и т.д.) В 2017 отказался от бэкендов CIL, JVM, JavaScript.

PyPy-stm (Software transaction memory)

- Позволяет решить проблему стандартного Python интерпретатора глобальной блокировки интерпретатора (GIL, global interpreter lock), не позволяющей обеспечить параллельное выполнение нескольких нитей кода на языке Python на разных ядрах.
- Используется программная транзакционная память (оптимистичная). Оптимистична в том плане, что поток завершает изменения памяти независимо от действий других потоков. При записи не проверяется влияет ли это на другие потоки (нет блокировки как обычно), проверка осуществляется считывающим устройством при завершении транзакции. Если транзакция не может быть выполнена из-за конфликтов записей, она прерывается и выполняется заново.

Возможности. Интроспекция.

- Высокая динамичность. Можно получить и изменить данные о программном объекте во время исполнения.
-
- Необходимые для интроспекции данные хранятся в специальных атрибутах. Так, например, получить все пользовательские атрибуты большинства объектов можно из специального атрибута — словаря (или другого объекта, предоставляющего интерфейс dict) `__dict__`

Примеры кода

```
>>> class x(object):pass
.....
>>> f = x()
>>> f.attr = 12
>>> print(f.__dict__)
{'attr': 12}
>>> print(x.__dict__)      # т.к. классы тоже являются экземплярами объекта type
                           # то и они поддерживают этот тип интроспекции
{'__dict__': <attribute '__dict__' of 'x' objects>, '__module__':.....}
```

```
1  def f(x):pass
2  print(f.__code__)
3  print(f.__code__.co_code)
```

Получение кода и байт кода функции.

Получение интроспекции на функцию. Модуль inspect.

```
1 import inspect
2 def function(x, y = 10):
3     return x**2
4 val = inspect.getfullargspec(function)
5 val1 = inspect.signature(function)
6 print(f'{val}')
7 print(f'signature {val1}')
```

```
FullArgSpec(args=['x', 'y'], varargs=None, varkw=None, defaults=(10,), kwoptions=[],
, kwoptionsdefaults=None, annotations={})
signature (x, y=10)
```



```

import inspect
def fun(x,y=10, *unnamed, par1=4, par2=3, **named)->int:
    for val in unnamed:
        print(f'unnamed value {val}')
    for key,val in named.items():
        print(f'named value {key} = {val}')

    return x+y
print(inspect.getfullargspec(fun))
print(fun(2,2,'u1',1,2,par1 = 2,par2=2,name1=1,name2='s'))

```

```

FullArgSpec(args=['x', 'y'], varargs='unnamed', varkw='named',
  defaults=(10,), kwonlyargs=['par1', 'par2'], kwonlydefaults={
'par1': 4, 'par2': 3}, annotations={'return': <class 'int'>})
unnamed value u1
unnamed value 1
unnamed value 2
named value name1 = 1
named value name2 = s
4

```

Аннотации типов

- Нужно для анализа кода, контроля типов, несмотря на динамическую типизацию, например с помощью mypy.
- `python -m mypy test_type.py`
- Три способа аннотации
- `var = value # type: annotation`
- `var: annotation; var = value`
- `var: annotation = value`
- Пример:
- `name = "John" # type: str`
- `name: str; name = "John"`
- `name: str = "John"`
- Для функций `def func()->int`

Проверка типов

- *#Типы данных, int, float, bool, str, complex*
- `>>> type(int)`
- `<class 'type'>`
- `>>> type(2+2)`
- `<class 'int'>`
- `>>> str1 = "Hello"`
- `>>> print(str1)`
- `Hello`
- `>>> type(str1)`
- `<class 'str'>`
- `>>> d = (1+2j)`
- `>>> type(d)`
- `<class 'complex'>`
- `>>> complex(2.0)`
- `(2+0j)`

Особенности

- Все является объектом
- Все является выражением (даже, например определение функции это выражение)
- Поддерживается ООП
- Поддерживается функциональная парадигма

None

```
λ python3
Python 3.6.6 (default, Jun 27 2018, 05:47:41)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> None # аналог null, но полноценный объект
>>> res = print(None) # любая функция возвращает значение
None
>>> res == None # не делайте так
True
>>> res is None # делайте лучше так Сравнение id - идентификаторов
True
>>> id(res) # в CPython -- адресс
140503861261072
>>> id(None)
140503861261072
>>>
```

bool

```
>>> to_be = False
>>> to_be or not to_be      # слова вместо значков
True
>>> x = 1
>>> y = 2
>>> x**2 + y**2 < 5 == True # True синглтон, не делайте так
False
>>> x**2 + y**2 < 5 is True # но и так тоже не делайте
False
>>> x**2 + y**2 < 5
False
```

bool

```
>>> False and print('also')      # short-circuiting!
False
>>> res = True and print('also')
also
>>> assert res is None, "print should return None"
>>> False or 92                  # работает с любым значением
92
```

#Операции

```
>>> flag = True
```

```
>>> flag
```

```
True
```

```
>>> flag+True
```

```
2
```

```
>>> flag and False
```

```
False
```

```
>>> flag+False
```

```
1
```


Assert применяется для отлавливания багов,
вызывая исключение если какое-то условие не
выполнено

```
def apply_discount(product, discount):  
    price = int(product['цена'] * (1.0 - discount))  
    assert 0 <= price <= product['цена']  
    return price
```

```
>>> apply_discount(shoes, 0.25)  
11175
```

200% скидки вызвало ошибку исполнения.

```
>>> apply_discount(shoes, 2.0)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    apply_discount(prod, 2.0)
  File "<input>", line 4, in apply_discount
    assert 0 <= price <= product['price']
AssertionError
```

Numbers

Возведение в степень

```
>>> x = 2
```

```
>>> x**10
```

```
1024
```

Деление нацело

```
>>> x = 15//2
```

```
>>> x
```

```
7
```

```
>>> x = 15/2
```

```
>>> x
```

```
7.5
```

Остаток от деления

```
>>> x = 15%2
```

```
>>> x
```

```
1
```

```
>>> x = 2*2
```

```
>>> id(x)
```

```
94632840975328
```

```
>>> id(4)
```

```
94632840975328
```

```
>>> id(2*2)
```

```
94632840975328
```

```
>>> id(2**28)
```

```
140182829614736
```

```
>>> y = 2**28
```

```
>>> id(y)
```

```
140182829614896
```

```
>>> 2**28
```

```
268435456
```

```
>>> 2**128
```

```
340282366920938463463374607431768211456
```

Number

```
>>> 3 / 2      # вещественное деление
1.5
>>> 4 / 2
2.0
>>> 4 // 2     # деление с остатком
2
>>> 3 // 2
1
>>> -3 // 2    # как в алгебре!
-2
>>> -1 % 3     # C/Java/Rust скажут -1
2
```

$$x = a//b$$

$$a = b*x+r$$

$$R \geq 0 \text{ and } < b$$

$$-9//2 = -5$$

$$-9\%2 = 1$$

$$-5*2+1 = -9$$

Number

```
>>> x = 10
>>> 0 <= x and x < 100
True
>>> 0 <= x < 100
True
```

list

```
>>> []  
[]  
>>> xs = [1, 2, 3, ]  
>>> len(xs)  
3  
>>> xs[0]  
1  
>>> xs[0] = 0  
>>> xs  
[0, 2, 3]
```

list

```
>>> xs = [1, 2] * 3
>>> xs
[1, 2, 1, 2, 1, 2]
>>> xs = [[0] * 3] * 3 # не делайте так
>>> xs[0][0] = 1
>>> xs
[[1, 0, 0], [1, 0, 0], [1, 0, 0]] # :-(
```

list

```
>>> [1] + [2, 3] + [4] # O(?)
[1, 2, 3, 4]
>>> xs = [1, 2, 3]
>>> xs.append(4)          # O(?)
>>> xs
[1, 2, 3, 4]
>>> xs.pop()              # O(?)
4
>>> xs
[1, 2, 3]
>>> xs.pop(0)             # O(?)
1
>>> xs
[2, 3]
>>> xs.insert(0, 92)      # O(?)
>>> xs
[92, 2, 3]
>>> xs += [1]             # так делать не стоит
>>> xs
[92, 2, 3, 1]
```

Добавляет в конец
расширяемого массива (в
конце есть буфер)

Создает новый список
объединяя с другим списком
(выполняется дольше). Не так
уж и интуитивно.

slices

```
>>> xs = list(range(5))
>>> xs
[0, 1, 2, 3, 4]
>>> xs[len(xs) - 1]
4
>>> xs[-1]
4
>>> xs[2:4]
[2, 3]
>>> xs[:-2]
[0, 1, 2]
>>> xs[::2]
[0, 2, 4]
>>> xs[:]
[0, 1, 2, 3, 4]
```

```
>>> x = range(15,1,-1)
>>> list(x)
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]
>>> x = range(15,-1,-1)
>>> list(x)
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

```
>>> y[0:-1:3]
[15, 12, 9, 6, 3]
```

```
>>> y[0:-1:3] = [3]*5
>>> y
[3, 14, 13, 3, 11, 10, 3, 8, 7, 3, 5, 4, 3, 2, 1, 0]
5 – рассчитать, не угадал, вызывается
исключение)
```

slices

```
>>> y = xs[:] # ИЛИ y = list(xs)
>>> y[0] = 92
>>> y
[92, 1, 2, 3, 4]
>>> xs
[0, 1, 2, 3, 4]
```

Интересно

```
y[0:2] = x[0:5]
```

```
y[0:3] = x[0:2]
```

X[100] вызовет исключение

Копия y=x[:]

```
>>> x = [0,1,2,3,4]
```

```
>>> y = x
```

```
>>> id(y)
```

```
140182830601672
```

```
>>> id(x)
```

```
140182830601672
```

```
>>> y = x[:]
```

```
>>> id(y)
```

```
140182808094536
```

```
>>> id(x)
```

```
140182830601672
```

```
>>> step2 = slice(None,None,2)
```

```
>>> y[step2]
```

```
[3, 13, 11, 3, 7, 5, 3, 1]
```

#Ввод строк

```
>>> str=input("Input any value: ")
```

```
Input any value: 45
```

```
>>> val=int(input("Input any value: "))
```

```
Input any value: 45
```

```
>>> str
```

```
'45'
```

```
>>> val
```

```
45
```

str

```
>>> s = "hello" # 'hello'
>>> len(s)
5
>>> s[:-1]
'hell'
>>> s[0]
'h'
```

```
>>> s[1] = "i" # в отличие от списков, строки не изменяемы
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> "<" + "." * 8 + "<"
'<.....<'
```

```
>>> str = str[0:2]+'e'+str[3:5]
```

```
>>> str
```

```
'heelo'
```

```
>>> str = str[0:2]+'e'+str[3:]
```

```
>>> str
```

```
'heelo'
```

```
>>> str = str[0:2]+'e'+str[3:-1]
```

```
>>> str
```

```
'heel'
```

Лучше str_ :)

Всегда создается новая строка.

str

```
>>> "hello \nworld".splitlines() # разбить на строчки
['hello ', 'world']
>>> "a b c".split() # на слова
['a', 'b', 'c']
>>> "\thello ".strip() # убрать пробелы
'hello'
>>> ", ".join(["a", "b", "c"]) # соединить список строк через разделитель
'a, b, c'
>>> str(42)
'42'
>>> x = 92
>>> f"2 * x = {2 * x}" # интерполяция
'2 * x = 184'
```

*#Форматный вывод (Оператор % для форматирования)
строка с форматом % значения*

```
>>> print("%03d:%10s %04d \n" % (flag, str, val))
```

```
001:      45 0045
```

```
>>> print("%s:%15s %04d" % (flag, str, val))
```

```
True:      45 0045
```

```
>>> c=41
```

```
>>> f=56.12
```

15 или 20 в примере — сколько всего отводится под вывод вещественного числа

```
>>> print("%c %15.3f %020.5e" % (c, f, f))  
)      56.120 00000000005.61200e+01
```

```
>>> str = "Hello world"
```

```
>>> len(str)
```

```
11
```

Метод формат

```
>>> s = "{}".format(str)
```

```
>>> s
```

```
'Hello world'
```

```
>>> s = "{0} {1} {2}".format(str, 14.24, True)
```

```
>>> s
```

```
'Hello world 14.24 True'
```

```
>>> print("{1:10.2f} {2} {0}".format(str, 14.24, True))
```

```
14.24 True Hello world
```

```
print("{0:^15}\n".format("*"), "{0:^14}\n".format("****"), "{0:^14}\n".format("*****"))
```

```
*
```

```
***
```

```
*****
```

```
>>> print("{0:^15}\n{1:^15}\n{2:^15}".format("*", "****", "*****"))
```

```
*
```

```
***
```

```
*****
```

```
>>> print("{0:^15}\n{1:^15}\n{2:^15}".format("*", "*" * 3, "*" * 5))
```

```
*
```

```
***
```

```
*****
```

➤ - выравнивание по правому краю

< выравниванием по левому краю

^ выравнивание по центру

f- строки Python 3.6+

join — склеивание списка в строку

split разбиение строки на элементы списка

```
>>> print("".join([f"{' '* (i*2+1):^15}\n" for i in range(5)]))
```

```
    *
```

```
    ***
```

```
    *****
```

```
    *********
```

```
    ***********
```

```
>>> x = 9
```

```
>>> print("".join([f"{' '* (i*2+1):^{x*2-1}}\n" for i in range(x)]))
```

```
>>> treg = lambda x,y: "" if x== -1 else treg(x-1,y)+f"{' '* (x*2+1):^{y}}\n"
print(treg(7,7*2+1))
```

```
exec('treg = lambda x,y,c: "" if x==-1 else treg(x-  
1,y,c)+f"{(c)*(x*2+1):^{y}}\\n"\\nprint(treg(6,6*2+1,"x"))')  
#Использование Y — комбинатора лямбда исчисления  
print((lambda a:lambda v:a(a,v))(lambda s,x:"" if x==0 else s(s,x-1)+f"'*(2*x-1):^15}\\n")(8))
```

tuple

```
>>> date = ("September", 2018)
>>> len(date)                # API, как у списка
2
>>> date[1] = 2019           # но менять нельзя
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> xs = ([], [])
>>> xs[0].extend([1, 2, 3])
>>> xs
([1, 2, 3], [])
```

tuple

```
>>> ()  
()  
>>> 1  
1  
>>> (1, )  
(1,)  
>>> date = "September", 2018  
>>> date  
( 'September', 2018)
```

пустой кортеж

кортеж из одного элемента

скобки опциональны

tuple

```
>>> def div_mod(x, y):  
...     return x // y, x % y  
...  
>>> d, m = div_mod(10, 3)  
>>> assert (d, m) == (3, 1)
```

set

```
>>> xs = {1, 2, 3} # множество (hash set)
>>> 1 in xs
True
>>> 92 not in xs
True
>>> xs.add(1)
>>> xs.add(92)
>>> xs
{1, 2, 3, 92} # в множестве нет повторений
>>> set() # для пустого множества нет литерала
set()
```

set

```
>>> xs = {1, 2, 3} # множество (hash set)
>>> 1 in xs
True
>>> 92 not in xs
True
>>> xs.add(1)
>>> xs.add(92)
>>> xs
{1, 2, 3, 92} # в множестве нет повторений
>>> set() # для пустого множества нет литерала
set()

>>> 1 in [1, 2, 3] # O(?)
True
>>> "world" in "hello, world"
True
```

set

```
>>> {1, 2, 3}.union({3, 4, 5}) # или |
{1, 2, 3, 4, 5}
>>> {1, 2, 3} & {3, 4, 5}      # или .intersection
{3}
>>> {1, 2, 3} ^ {3, 4, 5}      # или .symmetric_difference
{1, 2, 4, 5}
>>> xs = {1, 2, 3}
>>> xs.discard(2) # в реальной жизни операция удаления -- редкая
>>> xs
{1, 3}
```


set

```
>>> xs = set()
>>> xs.add([])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

dict

```
>>> date = {"year": 2018, "month": "September"}
>>> len(date)
2
>>> date["year"]           # KeyError если ключа нет, в отличие от Java
2018
>>> date.get("day", 14)    # Значение по умолчанию
14
>>> date["day"] = 14
>>> date.pop("year")
2018
```

dict

```
>>> date.keys()
dict_keys(['month', 'day'])      # set
>>> date.values()
dict_values(['September', 14])  # He set
>>> date.items()                # set!
dict_items([('month', 'September'), ('day', 14)])
```

dict

```
>>> date.items() | {1: 2}.items()
{('day', 14), ('month', 'September'), (1, 2)}
>>> date.items() | {1: []}.items()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

dict

```
>>> 'day' in date.keys() # ΠΛΟΧΟ!  
True  
>>> 'day' in date  
True
```

dict

```
>>> d = {}
>>> d["a"] = 1
>>> d["b"] = 2
>>> d["c"] = 3
>>> list(d.keys())
['a', 'b', 'c']
```

порядок гарантируется

```
>>> d["a"]=1
>>> d["b"]=2
>>> d["z"]=3
>>> d["c"]=4
>>> list(d.keys())
['a', 'z', 'b', 'c']
>>> d["c"]=3
>>> d["c"]=2
>>> list(d.keys())
['a', 'z', 'b', 'c']
>>> list(d.values())
[1, 3, 2, 2]
```

if

```
if 0 <= n and n < len(xs):  
    print(xs[n])
```

ternary if

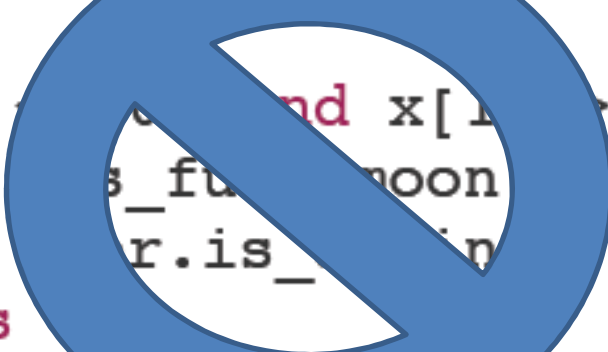
```
x = 50  
y = 25  
small = x if x < y else y  
# int small = x < y ? x : y;
```


if

```
# :( :( :(
if x[0] < 100 and x[1] > 100 and (is_full_moon() or not is_thursday()) and user.is_admin
    pass
```

if

```
# >:(  
if x[0] < 0 and x[1] > 100  
    and is_fridaynoon or not is_thursday()  
    and r.is_...  
    pass
```



if

```
# :( :( :(
if x[0] < 100 and x[1] > 100 \
    and (is_full_moon() or not is_thursday()) \
    and user.is_admin:
    pass
```

if

```
# :( :(
if (x[0] < 100 and x[1] > 100
    and (is_full_moon() or not is_thursday()))
    and user.is_admin):
    pass
```

if

```
# :|  
value_in_range = x[0] < 100 and x[1] > 100  
good_date = is_full_moon() or not is_thursday()  
if value_in_range and good_date and user.is_admin:  
    pass
```

while

```
i = 0
while i < 4:
    i += 1
i
```

Truthy/Falsy

```
>>> bool(True)
True
>>> bool(0)
False
>>> bool(1)
True
>>> bool([])
False
>>> bool([0])
True
```

Truthy/Falsy

```
(False,          # Falsy!  
None,  
0, 0.0, 0j,  
"",  
,  
[], (), set(), {})
```


Truthy/Falsy

```
if len(xs) == 0: # ΠΛΟΧΟ!  
    pass
```

```
if xs:  
    pass
```

```
if not xs:  
    pass
```

main.py



saved

```
1  x = 1
2  if x>4:
3      print("x>4")
4  elif x<0:
5      print("x<0")
6  elif x>2:
7      print("x > 2 and x<=4")
8  else: # x>=2 and x<=4, better not to comment
9      print("x>=0 and x<=2")
```

FOR

```
for x in range(10):  
    print(x)
```

```
for ch in "hello world": print(ch)
```

```
for i in reversed([1,2,3]): print(i)
```

```
for i in range(9,-1,-1): print(i)
```

break

```
target = 92
for item in items:
    if item == target:
        print("Found!", item)
        break
```

break

```
target = 92
for item in items:
    if item == target:
        print("Found!", item)
        break
print("Not found") # :(
```

break

```
target = 92
found = False # :( :( :(
for item in items:
    if item == target:
        print("Found!", item)
        found = True
        break
if found:
    print("Not found")
```

break

```
target = 92
for item in items:
    if item == target:
        print("Found!", item)
        break
else:
    print("Not found")
```

continue

```
target = 92
res = []
for item in items:
    if item != target:
        continue
    res.append(item)
```