Программирование на Python

Базовый курс. Часть 1

Андрей Скороходов

udv|group 2023

Python. Введение

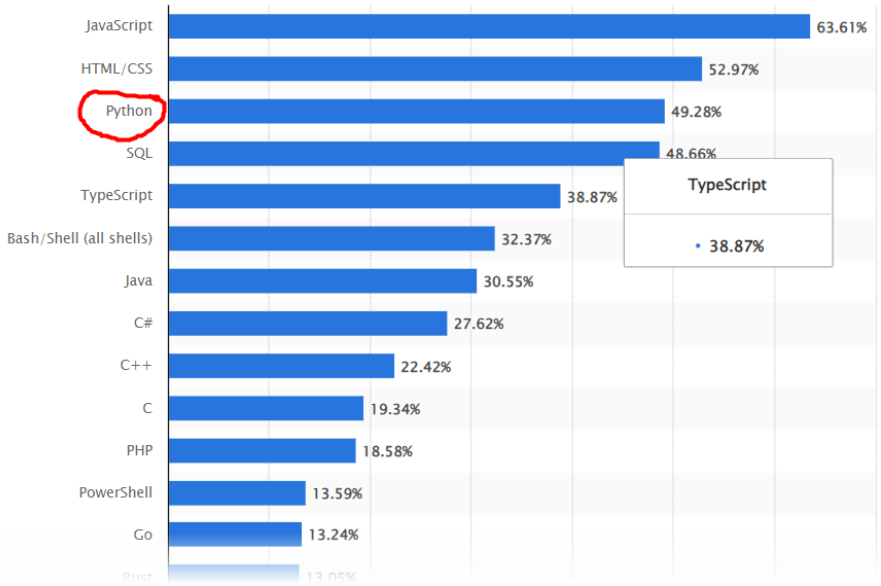
Guido van Rossum
(Netherlands, 31.01.1956…)

- **1991** – первая публикация Python (0.9.0)
- **1994** – Python 1.0
- **2000** – Python 2.0
- **2008** – Python 3.0
- **2023** – Python 3.11

Преимущества Python:

- Простой базовый синтаксис -> низкий порог вхождения
- Расширяемость и гибкость
- Интерпретируемость
- Кроссплатформенность
- Стандарт написания кода (PEP8)
- Широта применения
- OpenSource, большое сообщество

Наиболее популярные ЯП в 2023 году:

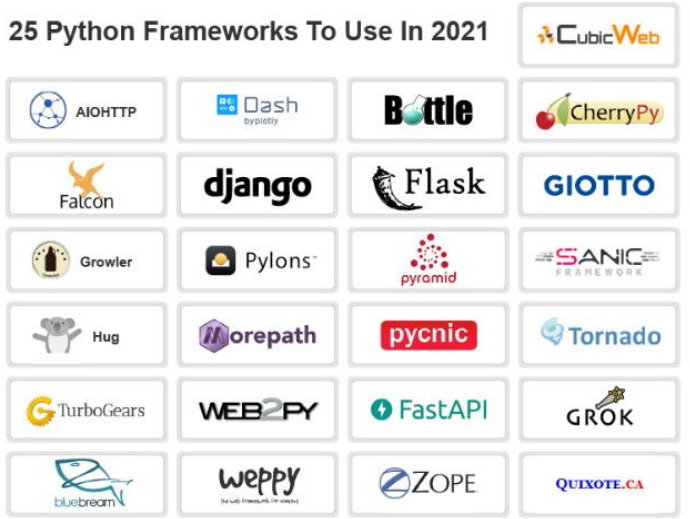


https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/

Python. Применение

- Веб-разработка
- Data Science и машинное обучение
- Написание скриптов
- Тестирование
- Прототипирование
- Системное программирование
- Приложения баз данных
- Графические интерфейсы
- Бизнес-приложения
- …

https://brainstation.io/career-guides/who-uses-python-today



https://www.trio.dev/python/resources/python-framework

Python. Пример кода

```python
import random

class Person:

    def __init__(
            self,
            name,
            gender,
            age,
            favorite_things,
            happyness_level=50
            ):

        self.name = name
        self.gender = gender
        self.age = age
        self.favorite_things = favorite_things
        self.happyness_level = happyness_level

    def make_happier(self):
        if self.favorite_things:
            thing = random.choice(self.favorite_things)
            self.buy_thing(thing)

            self.happyness_level = self.happyness_level + 10


            if self.gender == 'М':
                verb = "стал"
            else:
                verb = "стала"

            print(f"{self.name} {verb} счастливее, уровень счастья: {self.happyness_level}")

    def buy_thing(self, thing):
        if self.gender == 'М':
            verb = "купил"
        else:
            verb = "купила"

        print(f"{self.name} {verb} {thing}")
```

```
person1 = Person('Алена', 'Ж', '30', ["духи", "цветы", "шоколад"])
✓ 0.0s

person1.make_happier()
✓ 0.0s
```

Задание: предположите, что делает данный код и какой будет результат выполнения строчки person1.make_happier()?

Python. Пример кода

```python
import random

class Person:

    def __init__(
            self,
            name,
            gender,
            age,
            favorite_things,
            happyness_level=50
            ):

        self.name = name
        self.gender = gender
        self.age = age
        self.favorite_things = favorite_things
        self.happyness_level = happyness_level

    def make_happier(self):
        if self.favorite_things:
            thing = random.choice(self.favorite_things)
            self.buy_thing(thing)

            self.happyness_level = self.happyness_level + 10


            if self.gender == 'М':
                verb = "стал"
            else:
                verb = "стала"

            print(f"{self.name} {verb} счастливее, уровень счастья: {self.happyness_level}")

    def buy_thing(self, thing):
        if self.gender == 'М':
            verb = "купил"
        else:
            verb = "купила"

        print(f"{self.name} {verb} {thing}")
```

```
person1 = Person('Алена', 'Ж', '30', ["духи", "цветы", "шоколад"])
✓ 0.0s

person1.make_happier()
✓ 0.0s
Алена купила шоколад
Алена стала счастливее, уровень счастья: 60
```

Строка	Значение
import random	Импорт библиотеки random
Class Person	Объявление класса Person
def __init__(…)	Задание начальных значений для объекта класса
def make_happier(…)	Объявление метода, который делает объект класса Person счастливее
random.choice(…)	Выбор произвольного элемента из списка
self.happyness_level = self.happyness_level + 10	Увеличение уровня счастья объекта класса на 10 пунктов
person1 = Person(…)	Объявление объекта person1 класса Person
person1.make_happier()	Вызов метода, который сделает person1 счастливее

Python. PEP

PEP (Python Enhanced Proposal) – набор документов, описывающих особенности языка:

- PEP1: Purpose and Guidelines
- PEP2: Procedure for Adding New Modules
- PEP4: Deprecation of Standard Modules
- PEP5: Guidelines for Language Evolution
- PEP6: Bug Fix Releases
- PEP7: Style Guide for C Code
- PEP8: Style Guide for Python Code
- …

https://peps.python.org/pep-0000/

Python. PEP20 (The Zen of Python)

```
>>> import this
```

• Beautiful is better than ugly.	• Красивое лучше, чем уродливое.
• Explicit is better than implicit.	• Явное лучше, чем неявное.
• Simple is better than complex.	• Простое лучше, чем сложное.
• Complex is better than complicated.	• Сложное лучше, чем запутанное.
• Flat is better than nested.	• Плоское лучше, чем вложенное.
• Sparse is better than dense.	• Разреженное лучше, чем плотное.
• Readability counts.	• Читаемость имеет значение.
• Special cases aren't special enough to break the rules.	• Особые случаи не настолько особые, чтобы нарушать правила.
• Although practicality beats purity.	• При этом практичность важнее безупречности.
• Errors should never pass silently.	• Ошибки никогда не должны замалчиваться.
• Unless explicitly silenced.	• Если они не замалчиваются явно.
• In the face of ambiguity, refuse the temptation to guess.	• Встретив двусмысленность, отбрось искушение угадать.
• There should be one-- and preferably only one --obvious way to do it.	• Должен существовать один и, желательно, только один очевидный способ сделать это.
• Although that way may not be obvious at first unless you're Dutch.	• Хотя он поначалу может быть и не очевиден, если вы не голландец.
• Now is better than never.	• Сейчас лучше, чем никогда.
• Although never is often better than *right* now.	• Хотя никогда зачастую лучше, чем прямо сейчас.
• If the implementation is hard to explain, it's a bad idea.	• Если реализацию сложно объяснить — идея плоха.
• If the implementation is easy to explain, it may be a good idea.	• Если реализацию легко объяснить — идея, возможно, хороша.
• Namespaces are one honking great idea -- let's do more	• Пространства имён — отличная штука! Будем делать их больше!

https://tyapk.ru/blog/post/the-zen-of-python

Python. PEP8 (Style Guide for Python Code)

PEP8 - документ, описывающий общепринятый стиль написания кода на языке Python

https://peps.python.org/pep-0008/

https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html

«Код читается намного больше раз, чем пишется»

- Правильное использование отступов.
- Правильное использование пробелов внутри строки.
- Правильное использование пустых строк.
- Длину строки рекомендуется ограничить 79 символами.
- Каждый импорт должен быть на отдельной строке.
- Импорты всегда помещаются в начало файла в порядке: стандартные библиотеки -> сторонние -> модули проекта
- Комментарии, противоречащие коду, хуже, чем отсутствие комментариев. Всегда исправляйте комментарии, если меняете код
- Соглашения по именованию.
- Другие рекомендации (исключения, кодировка, контроль версий и т.п.)

Python. PEP8. Отступы

- 4 пробела на каждый уровень отступа.
- Выравнивание длинных строк с использованием висячего отступа или неявной линии внутри скобок.
- Закрывающие скобки могут находиться или под первым непробельным символом или под первым символом строки
- Для переноса строки можно использовать “\” или скобки вокруг предложения
- При переносе строк лучше использовать перенос после оператора

```
def long_function_name(
        var_one, var_two, var_three,
    var_four):
    print(var_one)

def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)

foo = long_function_name(var_one, var_two,
                         var_three, var_four)

foo = long_function_name(
    var_one = 1,
    var_two = 2,
    var_three = 3,
    var_four = 4
    )

my_list = [
    1, 2, 3,
    4, 5, 6,
    ]

with open('/path/to/some/file/you/want/to/read') as file_1, \
    open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
```

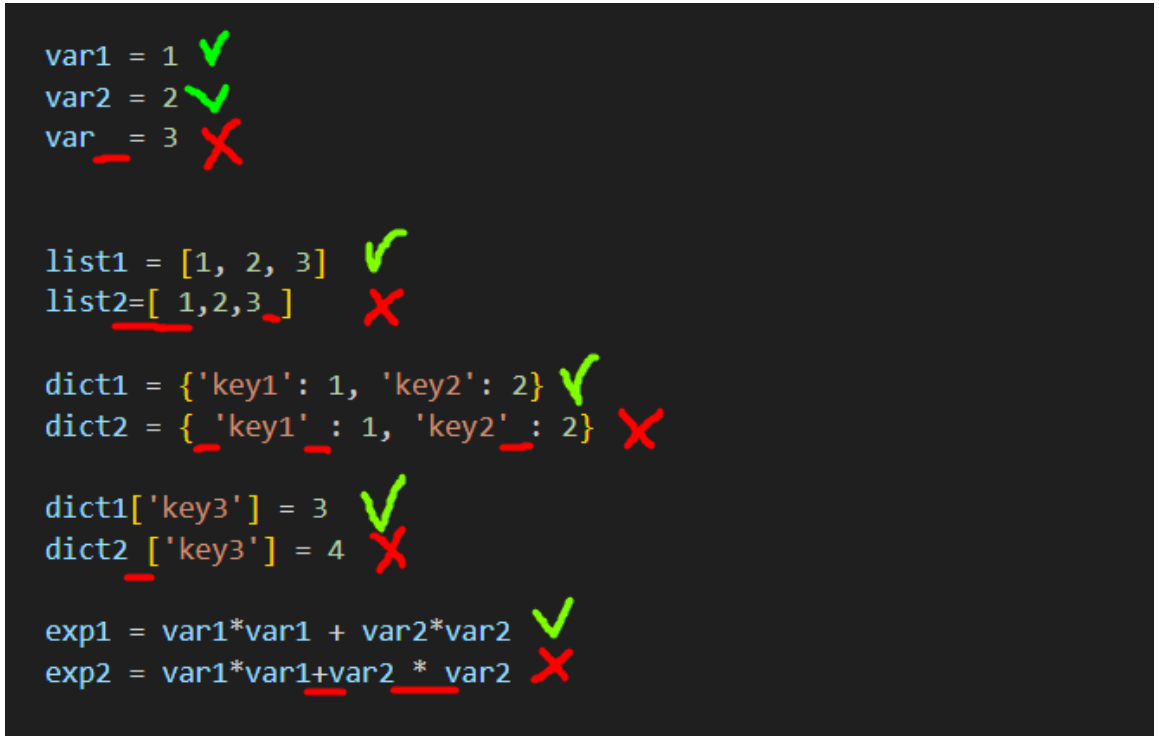
Python. PEP8. Пробелы

Не используются:

- После открывающей или перед закрывающей скобкой.
- Перед двоеточием, запятой, точкой с запятой.
- Перед открывающей скобкой после вызова функции, обращения к массиву, списку и т.п.
- Более одного пробела вокруг операторов.
- Вокруг знака присваивания для именованных аргументов функции.

Используются:

- Для выделения операторов (=, +, - и т.п.).
- После двоеточия, запятой, точкой с запятой.
- Для выделения приоритета операторов внутри выражения



Python. PEP8. Пустые строки

Используются:

- Для отделения функций верхнего уровня и определения классов – 2 строки.
- Определения методов внутри классов.
- Разделения различных групп внутри функций.
- Для разделения логических разделов.

```python
class Car:

    def __init__(self, model, color):
        self.model = model
        self.color = color

    def get_color(self):
        return self.color


class Moto:

    def __init__(self, model, color):
        self.model = model
        self.color = color
```

```python
def _extract_data_from_forecast(self, forecast_raw: pd.DataFrame) -> N
    data = pd.DataFrame()
    data['temperature'] = forecast_raw['Температура, °C']
    data['wind_speed'] = forecast_raw['Ветер: скорость, м/с']
    data['wind_direction'] = forecast_raw['направление']
    data['pressure'] = forecast_raw['Давление, мм рт. ст.']

    data_len = data.shape[0] # Для контроля равенства длин основого и дополнительного прогнозов

    conditions, cloudiness = self.__get_cloudiness()
    data['conditions'] = conditions[-data_len:]
    data['cloudiness'] = cloudiness[-data_len:]

    precipitation, precipitation_info = self.__get_precipitation()
    data['precipitation'] = precipitation[-data_len:]
    data['precipitation_info'] = precipitation_info[-data_len:]
    data['humidity'] = self.__get_humidity()[-data_len:]
```

Python. PEP8. Рекомендации по именованию

- Имена переменных:

lowercase, lower_case_with_underscores

- Имена модулей и пакетов:

lowercase, lower_case_with_underscores

- Имена классов:

CapitalizedWords.

- Исключения (Exceptions):

CapitalizedWords.

- Имена функций:

lower_case_with_underscores.

- Константы:

UPPERCASE_WITH_UNDERSCORES

```python
class ForecastRumeteo(Forecast):

    def __init__(self, **kwargs) -> None:
        super().__init__('rumeteo', URL = "https://ru-meteo.ru/ekaterinburg/hour", **kwargs)

    @reconnect()
    def _get_data_from_source(self) -> pd.DataFrame:
        forecast_table = pd.read_html(self.URL, encoding="UTF-8", header=0)
        forecast = self.__forecast_from_table(forecast_table[0])

        for i, table in enumerate(forecast_table[1:]):
            # данные по разным дням находятся в разных таблицах
            next_forecast = self.__forecast_from_table(table, i)
            forecast = pd.concat([forecast, next_forecast])

        return forecast
```

Общие правила:

- Следует избегать использования O, l, i в качестве одиночных идентификаторов
- Названия не должны начинаться с цифры
- Названия не должны совпадать со служебными словами
- Названия не должны содержать специальных символов, пробелов, дефисов и т.п.
- **Названия должны быть информативны!**

Python. Встроенные библиотеки

datetime – операции со временем, датой

```
import datetime

print('Текущее время: ', datetime.datetime.now())
print('Год: ', datetime.datetime.now().year)
print('Месяц: ', datetime.datetime.now().month)
print('День: ', datetime.datetime.now().day)
print('10 дней назад: ', datetime.datetime.now() - datetime.timedelta(days=10))
```

✓ 0.0s

```
Текущее время:  2023-09-03 17:49:33.764543
Год:  2023
Месяц:  9
День:  3
10 дней назад:  2023-08-24 17:49:33.764543
```

math – математические операции

```
import math

print('cos: ', math.cos(0.5))
print('exp: ', math.exp(2))
print('sqrt: ', math.sqrt(121))
print('e: ', math.e)
```

✓ 0.0s

```
cos:  0.8775825618903728
exp:  7.38905609893065
sqrt:  11.0
e:  2.718281828459045
```

random – генерация случайных значений

```
import random
print('randint: ', random.randint(1, 10))
print('choice: ', random.choice([10, 21, 32, 41, 5, 6, 7]))
print('random: ', random.random())
print('uniform: ', random.uniform(100,200))
```

✓ 0.0s

```
randint:  6
choice:  41
random:  0.41935291602161207
uniform:  195.12661159265596
```

Еще примеры:

- re – операции с регулярными выражениями
- json – операции с форматом JSON
- logging – модуль логгирования
- os – модуль взаимодействия с ОС
- pathlib – модуль взаимодействия с файловыми путями
- …

Python. Сторонние библиотеки

- matplotlib – отрисовка графиков
- pandas – работа с табличными данными
- numpy – работа с матрицами
- requests – работа с http запросами
- PyTorch – работа с нейронными сетями
- …

Для использования библиотеки достаточно ее импортировать, если она уже установлена в системе. Если нет – тогда необходимо ее установить командой pip install <имя библиотеки>.

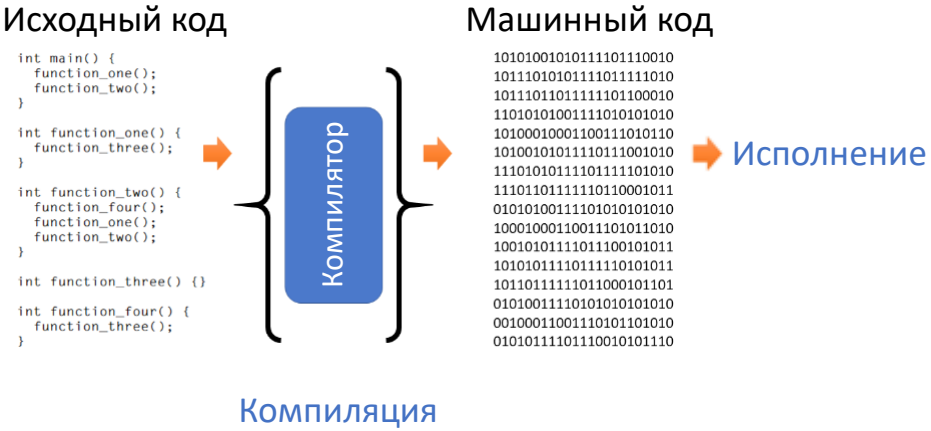
```
import pandas

data = {
    "Ограничение": [60, 90, 40, 20],
    "Локация": ['Город', 'Трасса', 'Ямы', 'Лежачий полицейский']
}

df = pandas.DataFrame(data)
df
```

✓ 0.0s

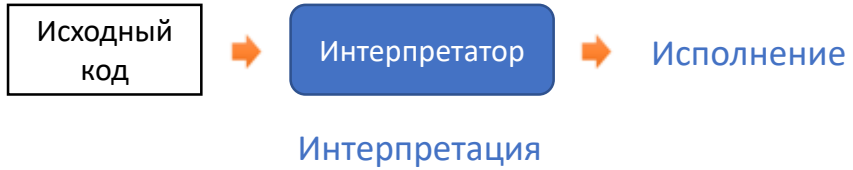
	Ограничение	Локация
0	60	Город
1	90	Трасса
2	40	Ямы
3	20	Лежачий полицейский

Python. Интерпретируемый язык

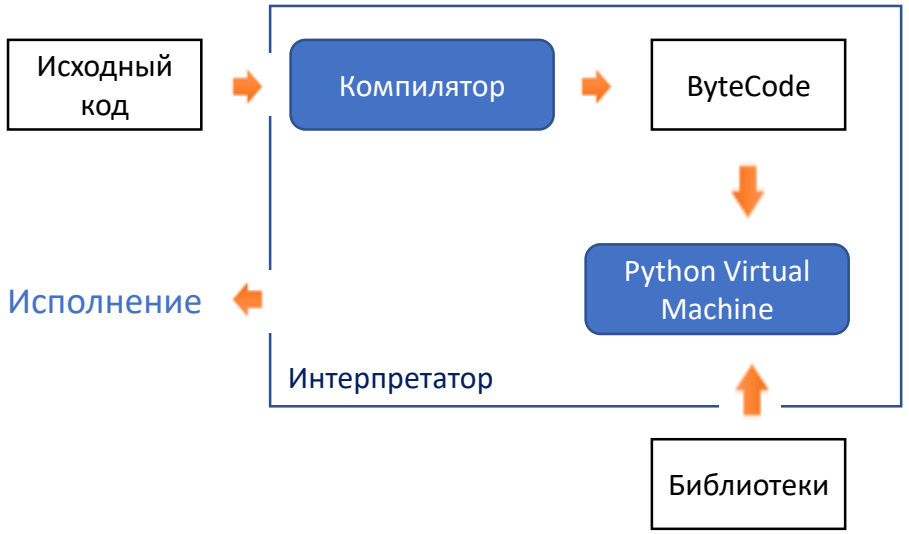
Компилируемые языки (C, .NET, Go, Java…)



Интерпретируемые языки (Python, Perl, JavaScript…)



Реализация Python



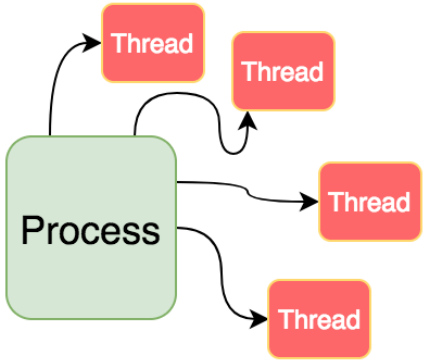
Компилируемые языки	Интерпретируемые языки
Высокая скорость исполнения	Низкая скорость исполнения
Сложность при отладке	Простота отладки
Платформо-зависимые	Кроссплатформенность

Python. Процессы и потоки

Процесс (Process) – часть виртуальной памяти и ресурсов, которую ОС выделяет для выполнения программы.

Поток (Thread) – наименьшая единица выполнения операций с независимым набором инструкций, часть процесса.

Многопоточность (Multithreading) – параллельное (без предписанного порядка во времени) выполнение потоков процесса, порожденного в ОС.



Процессы	Потоки
Каждое приложение имеет минимум один процесс	Каждый процесс имеет минимум один поток
Требуют выделения отдельного места в памяти	Используют память, выделенную под процесс -> создаются и завершаются быстрее
Оперируют только со своими данными. Обмен данными с другими процессами через межпроцессорное взаимодействие	Имеют прямой доступ к ресурсам других потоков процесса
Процесс можно прервать	Поток прервать нельзя

Python. GIL

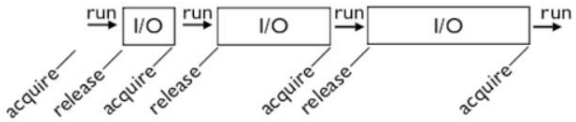
Global Interpreter Lock – Глобальная блокировка интерпретатора
www.dabeaz.com/python/GIL.pdf
https://habr.com/ru/articles/84629/ (перевод)

В конкретный момент времени может выполняться только один поток Python. За это отвечает GIL.

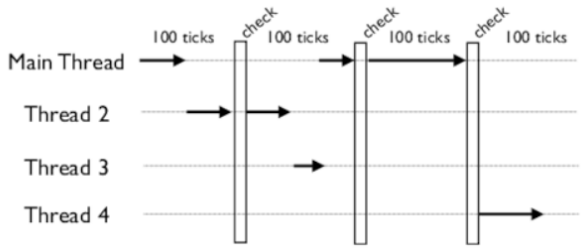
Задачи в потоке:

- CPU-bound – преимущественно использующие процессор (напр., математические вычисления)
- IO-bound – работающие с вводом-выводом (напр., с диском)

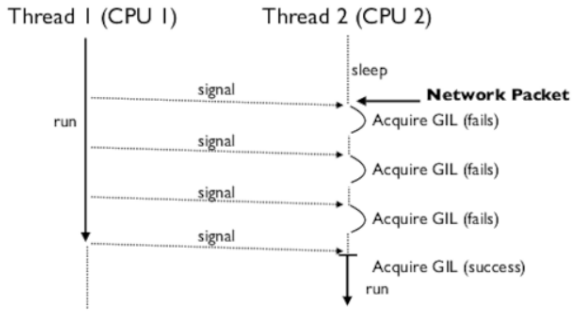
Потоки удерживают GIL, пока выполняются, но освобождают при задачах ввода-вывода. В этот момент могут запуститься другие потоки.



Поток прерывается на время выполнения операций ввода-вывода



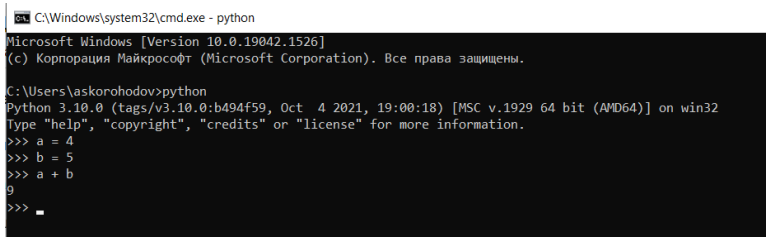
В процессе всегда есть главный поток (main thread), в котором производятся обработчики сигналов, интерпретатор периодически проверяет наличие операций ввода-вывода

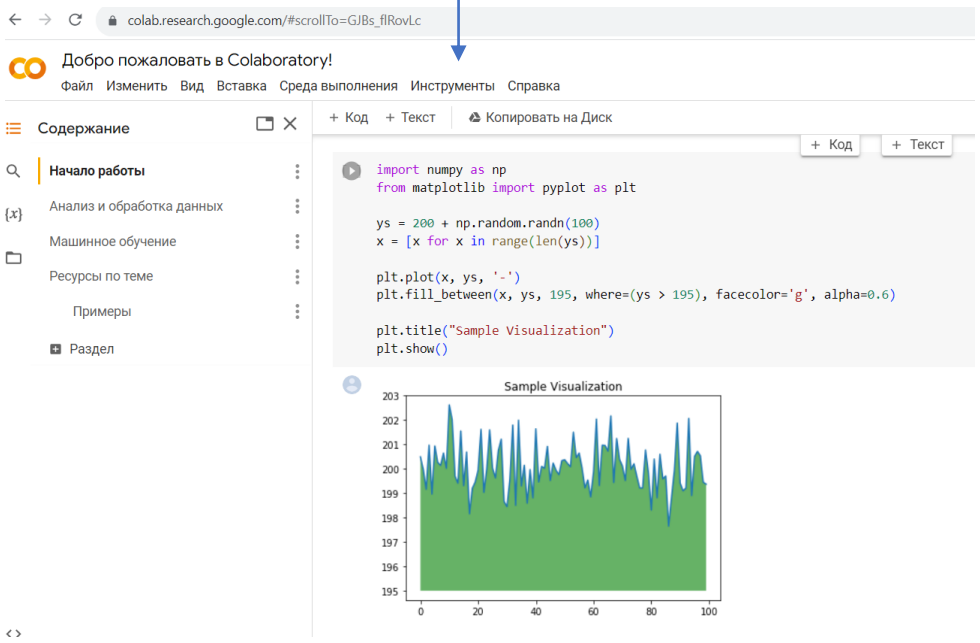


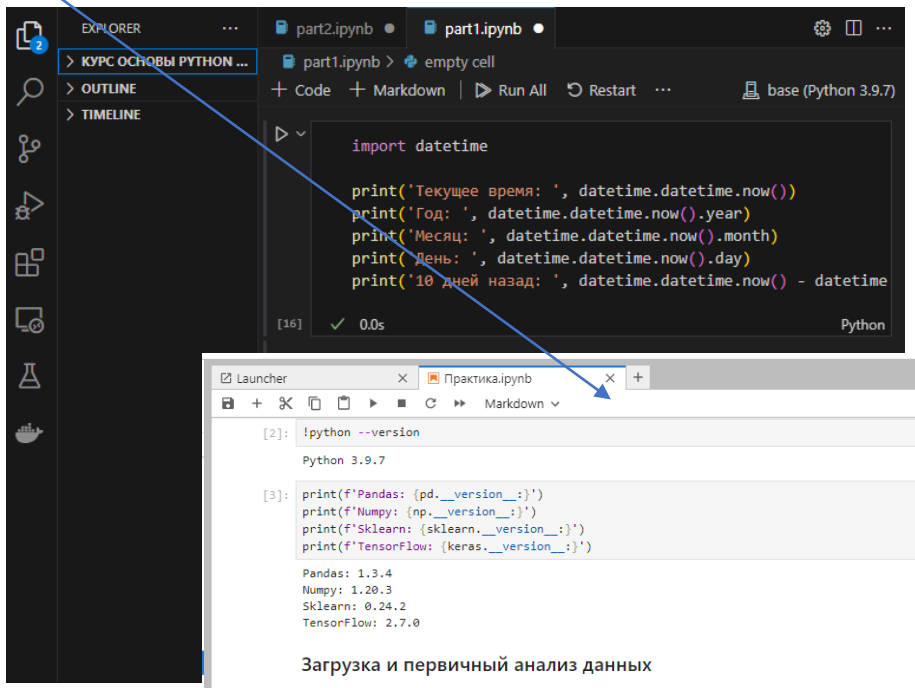
Один CPU-bound поток может привести к увеличению времени отклика IO-bound потока

Python. Как пользоваться?

- Интерпретатор Python в ОС
- Jupyter Notebook (Jupyter Hub, Jupyter Lab, …)
- IDE (VS Code, PyCharm, …)
- Облачные ресурсы (Google Colab, Kaggle, …)

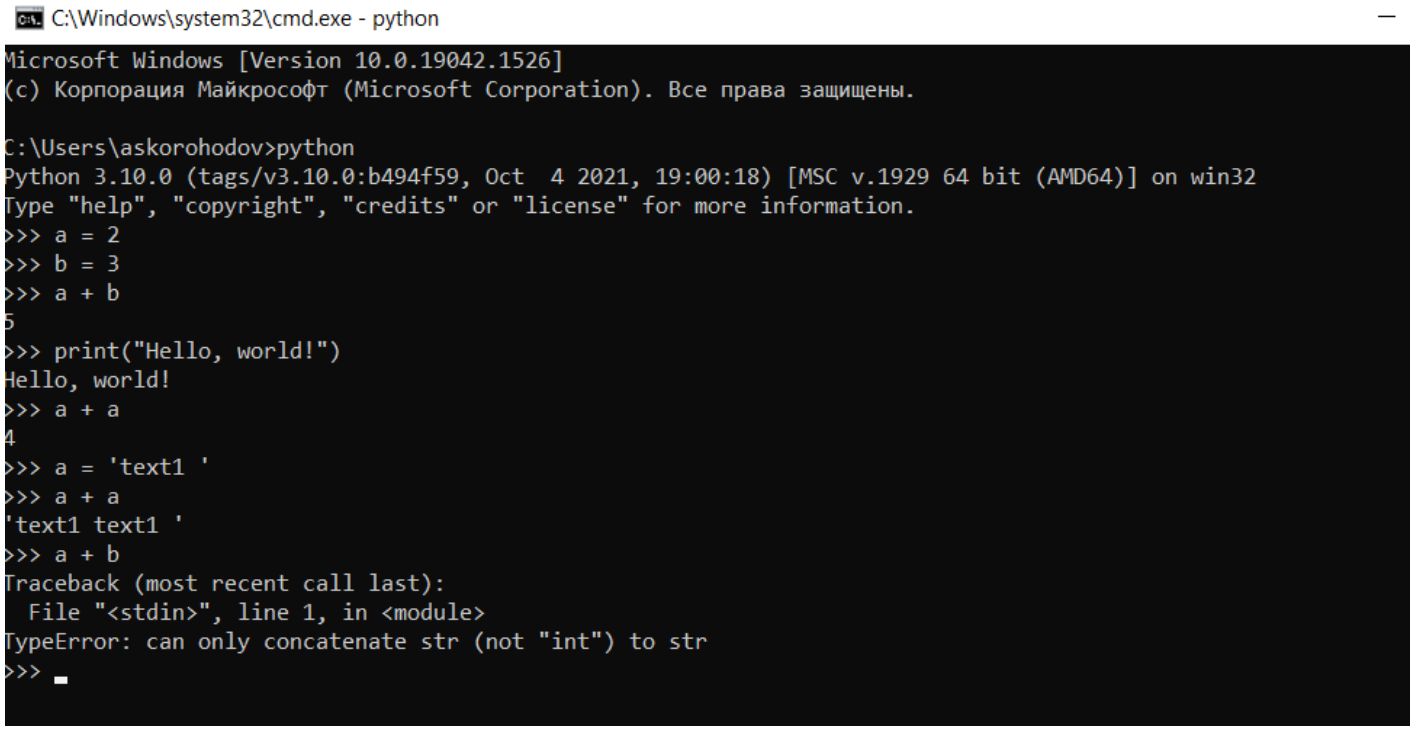






Python. Задание 1

1. Запустите в консоли windows (*cmd*) интерпретатор Python (*python*)
2. Узнайте версию python, которая установлена на вашем ПК
3. Вычислите сумму двух переменных (объявите вначале переменные, например, *a=2*)
4. Выведите в консоль классическое *“Hello, world!”*
5. Попробуйте сложить текст и текст
6. Попробуйте сложить текст и цифру



Python. Задание 2

1. Зайдите на https://colab.research.google.com/
2. Нажмите «Создать блокнот»
3. Выполните все действия из предыдущего задания
4. Для того, чтобы узнать версию Python наберите *!python --version*

Python. Типы данных и динамическая типизация

Типы данных:

- Встроенные: int, float, complex, bool, str, None.
- Специализированные – требуют загрузки определенного модуля (например, datetime)

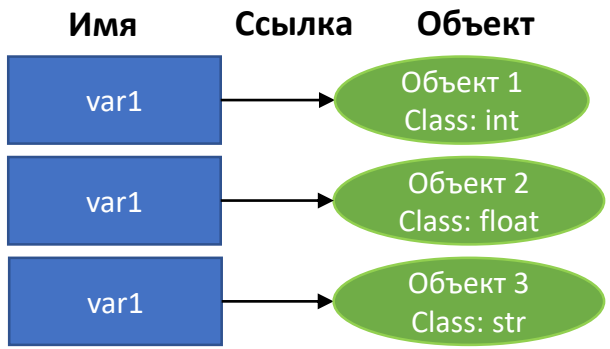
Все типы данных в Python являются классами, а значения – объектами классов.

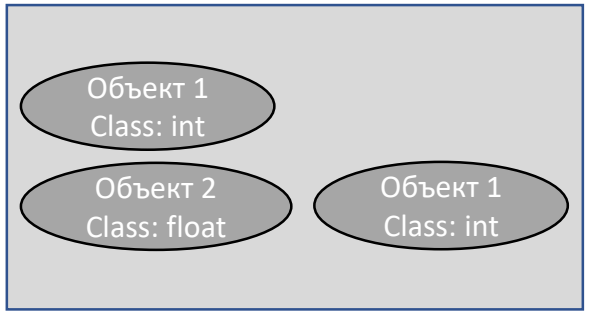
Для проверки типа данных следует использовать функцию ***type()*** (например, *type(var1)*)

Определение типов производится автоматически в ходе выполнения программы.

```
var1 = 13
print(var1, type(var1), hex(id(var1)))
var1 = 2.4
print(var1, type(var1), hex(id(var1)))
var1 = 'some string'
print(var1, type(var1), hex(id(var1)))
```

```
13 <class 'int'> 0x7ff98cff94a8
2.4 <class 'float'> 0x19f878fed70
some string <class 'str'> 0x19f8800aff0
```





Python. Арифметические операции

Приоритет	Оператор Python	Операция	Пример	Результат
1	**	Возведение в степень	5 ** 5	3125
2	%	Деление по модулю (получение остатка)	16 % 7	2
3	//	Целочисленное деление (дробная часть отбрасывается)	13 // 3	4
4	/	Деление	39 / 2	19.5
5	*	Умножение	123 * 321	39483
6	-	Вычитание	999 – 135	864
7	+	Сложение	478 + 32	510

https://proglib.io/p/samouchitel-po-python-dlya-nachinayushchih-chast-3-tipy-dannyh-preobrazovanie-i-bazovye-operacii-2022-10-14

Python. Преобразования типов

В арифметических операциях Python пытается автоматически выполнить преобразование операндов к одному типу:

- Если один из операндов **complex**, то делается попытка преобразовать другой в **complex**.
- Аналогичная ситуация с **float**.

Если автоматически преобразовать операнды не получается, необходимо применять явное преобразование типов: **int()**, **float()**, **str()**

Не все операнды могут быть преобразованы в другие типы, например, не получится преобразовать в int строки, содержащие что-либо, кроме цифр.

Также если число складывается со строкой, необходимо выполнить преобразование операндов к одному типу, т.к. интерпретатор не знает, что нужно нам: сложить строки или числа.

```
a = 2
b = 3.1
c = a + b
print(c, type(c))
```

```
5.1 <class 'float'>
```

```
b = '3.1'
c = a + b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[14], line 2
      1 b = '3.1'
----> 2 c = a + b

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
b = float(3.1)
c = a + b
print(c, type(c))
```

```
5.1 <class 'float'>
```

```
b = int('3.1x')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 1
----> 1 b = int('3.1x')
      2 c = a + b
      3 print(c, type(c))

ValueError: invalid literal for int() with base 10: '3.1x'
```

```
print(int(True))
print(int(False))
print(float(True))
print(float(False))
```

```
1
0
1.0
0.0
```

```
print("a" + str(2))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[11], line 1
----> 1 print("a" + 2)

TypeError: can only concatenate str (not "int") to str
```

Python. Задание 3

1. Создайте целочисленную переменную *a*, равную десяти, выведите в консоль ее значение и тип.
2. Создайте переменную с плавающей запятой *b*, также равную десяти, выведите в консоль ее значение и тип.
3. Создайте строковую переменную *c*, равную восьми, выведите в консоль ее значение и тип.
4. Возведите в квадрат переменную *a* и сложите ее с переменной *b*, и из полученного значения извлеките квадратный корень. Присвойте значение этого выражения переменной *x*.
5. Сложите переменную *x* с переменной *c*, найдите остаток от деления на 2 и присвойте это значение переменной *y*.
6. Выведите идентификаторы полученных переменных (*id()*) и сравните их
7. Выполните шаги 1..6, но предварительно преобразуйте переменную *b* в *int*.

Python. Структуры данных

Структура	Класс Python	Пример	Использование
Список	list	[1, 2, 3] [‘cat’, ‘dog’, ‘duck’, ‘dog’]	l = [1, 2, 3] l = list((‘cat’, ‘dog’, ‘duck’, ‘dog’))
Кортеж	tuple	(1, 2, 3) (‘cat’, ‘dog’, ‘duck’, ‘dog’)	t = (1, 2, 3) t = tuple((‘cat’, ‘dog’, ‘duck’, ‘dog’))
Множество	set	{1, 2, 3} {‘cat’, ‘dog’, ‘duck’}	s = (1, 2, 3) s = set((‘cat’, ‘dog’, ‘duck’, ‘dog’))
Словарь	dict	{‘cat’: 1, ‘dog’: 2, ‘duck’: 3}	d = dict(‘cat’=1, ‘dog’=2, ‘duck’=3)
Неизменяемое множество	frozenset	{1, 2, 3} {‘cat’, ‘dog’, ‘duck’}	s = (1, 2, 3) s = frozenset((‘cat’, ‘dog’, ‘duck’, ‘dog’))
Диапазон	range	1, 3, 5, 7, 9	r = range(1, 10, 2)

Все структуры данных в Python также являются классами, а значения – объектами классов.

Для проверки типа структуры данных следует также использовать функцию **type()** (например, *type(var1)*)

Определение типов структур производится автоматически в ходе выполнения программы.

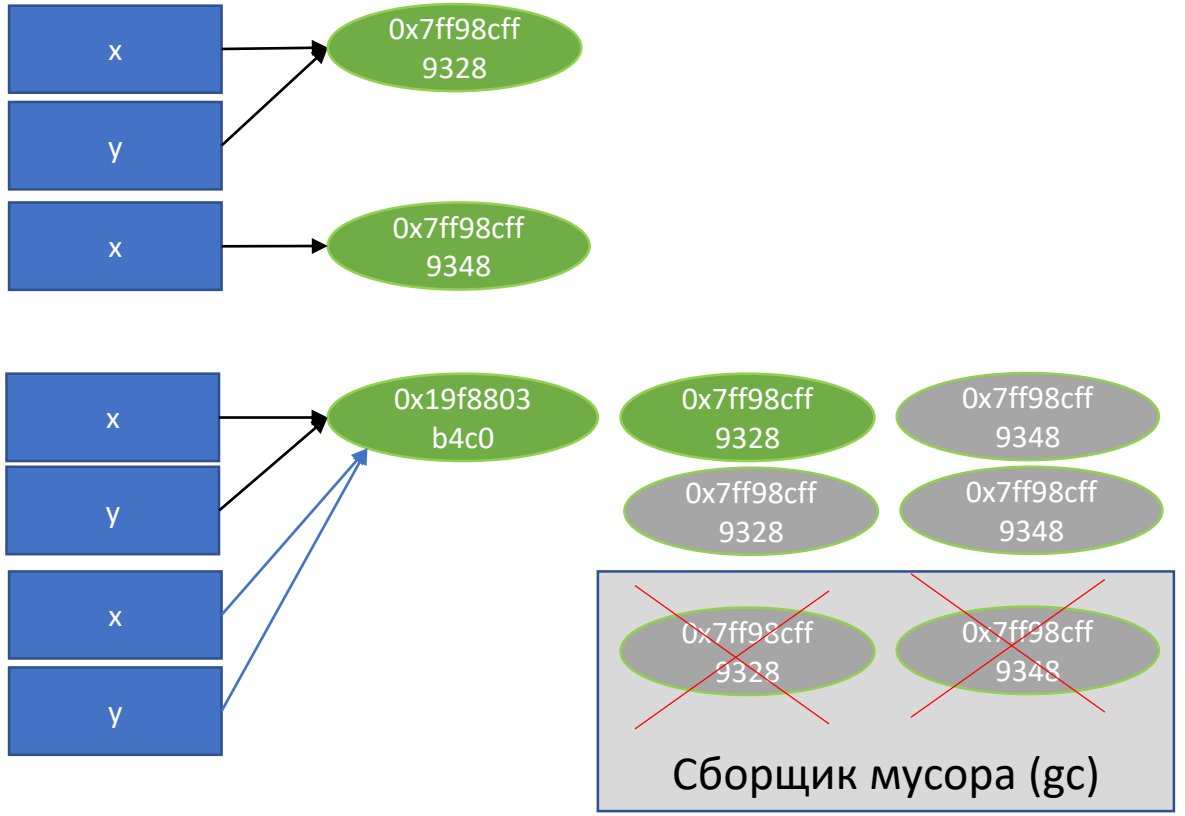
Python. Изменяемые и неизменяемые объекты

- **Неизменяемые (immutable):** int, bool, tuple, float, string, frozenset
- **Изменяемые (mutable):** list, set, dict

Доступ к неизменяемым объектам осуществляется быстрее.

```
print('== immutable ==')
x = 1
y = x
print(x, hex(id(x)), y, hex(id(y)))
x = x+1
print(x, hex(id(x)), y, hex(id(y)))
print(hex(id(2)), hex(id(1)))

print('== mutable ==')
x = list([1, 2, 3])
y = x
print(x, hex(id(x)), y, hex(id(y)))
x.append(4)
print(x, hex(id(x)), y, hex(id(y)))
```

```
== immutable ==
1 0x7ff98cff9328 1 0x7ff98cff9328
2 0x7ff98cff9348 1 0x7ff98cff9328
0x7ff98cff9348 0x7ff98cff9328
== mutable ==
[1, 2, 3] 0x19f8803b4c0 [1, 2, 3] 0x19f8803b4c0
[1, 2, 3, 4] 0x19f8803b4c0 [1, 2, 3, 4] 0x19f8803b4c0
```



Python. Списки (List)

Список – упорядоченная* изменяемая коллекция объектов произвольных типов

```
lst = [1, 2, 'one', 'two', ['three', 'four', 'five']]
print(f'lst[0]: {lst[0]}')
print(f'lst[3]: {lst[3]}')
print(f'lst[1:]: {lst[1:]}')
print(f'lst[-1:]: {lst[-1:]}')
print(f'lst[2:4]: {lst[2:4]}')
print(f'lst[::2]: {lst[::2]}')
print(f'lst[::-1]: {lst[::-1]}')
```

```
lst[0]: 1
lst[3]: two
lst[1:]: [2, 'one', 'two', ['three', 'four', 'five']]
lst[-1:]: [['three', 'four', 'five']]
lst[2:4]: ['one', 'two']
lst[::2]: [1, 'one', ['three', 'four', 'five']]
lst[::-1]: [['three', 'four', 'five'], 'two', 'one', 2, 1]
```

***Упорядоченная коллекция** – коллекция, значения которой представляют собой упорядоченное множество, с каждым элементом которого связан его порядковый номер. Любое значение, кроме первого, имеет предшественника, и любое значение, кроме последнего, имеет последователя, определяемых отношением порядка данного типа.

Метод	Описание
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x.
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список
list.copy()	Копия списка
list.clear()	Очистка списка

https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html

Python. Кортежи (tuple)

Кортеж – упорядоченная неизменяемая коллекция объектов произвольных типов = неизменяемый список

```
tpl = (1, 2, 'one', 'two', ['three', 'four', 'five'])
print(f'tpl[1]: {tpl[1]}')
print(f'tpl*2: {tpl*2}')
print(f'tpl+2: {tpl+tpl}')
tpl[-1][0] = 111111111
print(f'tpl[-1][0]: {tpl[-1][0]}')
tpl[-1] = 5
```

```
tpl[1]: 2
tpl*2: (1, 2, 'one', 'two', ['three', 'four', 'five'], 1, 2, 'one', 'two', ['three', 'four', 'five'])
tpl+2: (1, 2, 'one', 'two', ['three', 'four', 'five'], 1, 2, 'one', 'two', ['three', 'four', 'five'])
tpl[-1][0]: 111111111
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[68], line 7
      5 tpl[-1][0] = 111111111
      6 print(f'tpl[-1][0]: {tpl[-1][0]}')
----> 7 tpl[-1] = 5

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

К кортежам применимы все операции, не изменяющие кортеж. Кортежи занимают меньше места в памяти.

Python. Задание 4

1. Создайте список *employees*, в котором будут содержаться кортежи сотрудников в формате *(имя, фамилия, возраст, должность).* Инициализируйте его произвольными пятью сотрудниками.
2. Добавьте в список нового сотрудника
3. Выведите в консоль должность нового сотрудника
4. Измените должность нового сотрудника на «руководитель»
5. Выведите в консоль данные произвольного сотрудника (модуль *random*)

Python. Операции над строками

Примеры операций со строками

```
s = "1. Пример строки"
print(type(s))
print(s)
s = '2. Можно использовать одинарные кавычки'
print(s)
s = '''3. Если текст длинный, можно использовать
многострочный вариант с тремя кавычками'''
print(s)
```

```
<class 'str'>
1. Пример строки
2. Можно использовать одинарные кавычки
3. Если текст длинный, можно использовать
многострочный вариант с тремя кавычками
```

```
print(f"4. Можно вычислять длину строки - len(s): {len(s)}")
print('5. Строки ' + 'можно ' + 'складывать')
print('6. И даже ' + 'умножать '*3)
```

```
4. Можно вычислять длину строки - len(s): 81
5. Строки можно складывать
6. И даже умножать умножать умножать
```

```
print('7. Можно искать вхождение подстроки: ', 'Если' in s, 'числа' in s)
print('8. Строки можно разбивать в списки: ', s.split())
print('9. А потом соединять: ', ' '.join(s.split()))
```

```
7. Можно искать вхождение подстроки:  True False
8. Строки можно разбивать в списки:  ['3.', 'Если', 'текст', 'длинный,', 'можно', 'ис
9. А потом соединять:  3. Если текст длинный, можно использовать многострочный вариан
```

```
print('10. Можно все сделать заглавными буквами: ', s.upper())
print('11.', 'или сделать как в предложении'.capitalize())
```

```
10. Можно все сделать заглавными буквами:  3. ЕСЛИ ТЕКСТ ДЛИННЫЙ, МОЖНО ИСПОЛЬЗОВАТЬ
МНОГОСТРОЧНЫЙ ВАРИАНТ С ТРЕМЯ КАВЫЧКАМИ
11. Или сделать как в предложении
```

https://proglib.io/p/samouchitel-po-python-dlya-nachinayushchih-chast-4-metody-raboty-so-strokami-2022-10-24

https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html

Строка – итерируемый объект

```
print('Исходная строка: ', s, '\n')
print('Элемент строки: ', s[3])
print('Несколько последовательных элементов: ', s[:10])
print('То же самое через 1 символ: ', s[::2])
```

```
Исходная строка:  3. Если текст длинный, можно использовать
многострочный вариант с тремя кавычками

Элемент строки:  Е
Несколько последовательных элементов:  3. Если те
то же самое через 1 символ:  3 ситктдинй он сьюаьмоотонйвратстеяквчаи
```

```
for i in s[3:8]:
    print(i)
```

```
Е
с
л
и
```

И еще методы работы со строками:

- **replace()** – замена символа.
- **isalnum(), isalpha(), isdigit()** – определение, состоит строка только из букв/цифр, только букв или только цифр соответственно.
- **find(), rfind()** – поиск искомой подстроки в строке, возвращает индекс первого вхождения слева или справа соответственно.
- **startswith()** – возвращает True, если строка начинается с заданного символа.
- **strip(), lstrip()** и **rstrip()** – удаление пробелов в начале и конце строки, только слева или только справа соответственно

Python. Задание 5

1. Присвойте строковой переменной *s* любой текст из нескольких (>2) предложений длиной не менее 20 символов.
2. Выведите на экран длину переменной *s*.
3. Приведите весь текст к нижнему регистру, выведите на экран.
4. Замените все пробелы в переменной на знак @, выведите на экран.
5. Разбейте ваш текст на предложения и выведите их на экран в виде списка
6. Проверьте, состоит ли ваше первое предложение только из букв. Выведите результат на экран.
7. Выведите на экран второе слово первого предложения вашего текста в обратном порядке.
8. Разбейте ваше второе предложение и выведите на экран слова с первого по пятое включительно, через одно.

Python. Множества (set)

Множество – неупорядоченная изменяемая коллекция неповторяющихся объектов произвольных типов

```
st1 = {1, 2, 3, 4}
st2 = {4, 5, 6, 7, 8}
print(f"Длина st1: {len(st1)}")
print(f"Элемент 7 есть в st2: {7 in st2}")
print(f"Объединение st1 и st2: {st1 | st2}") #set.union
print(f"Пересечение st1 и st2: {st1 & st2}") #set.intersection
print(f"Есть в st1, но нет в st2: {st1 - st2}") #set.difference
print(f"Есть только в st1 или в st2: {st1 ^ st2}") #set.symmetric_difference
st1.add(10)
print(f"st1: {st1}")
st1[0]
```

```
Длина st1: 4
Элемент 7 есть в st2: True
Объединение st1 и st2: {1, 2, 3, 4, 5, 6, 7, 8}
Пересечение st1 и st2: {4}
Есть в st1, но нет в st2: {1, 2, 3}
Есть только в st1 или в st2: {1, 2, 3, 5, 6, 7, 8}
st1: {1, 2, 3, 4, 10}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[83], line 11
      9 st1.add(10)
     10 print(f"st1: {st1}")
---> 11 st1[0]

TypeError: 'set' object is not subscriptable
```

Python. Словари (dict)

Словарь (ассоциативный массив, хеш-таблица) – неупорядоченная изменяемая коллекция произвольных объектов с доступом по ключу.

```
d = {'Вася': 5, 'Алена': 10, 'Петя': 28}
print(f"d.items(): {d.items()}")
print(f"d.keys(): {d.keys()}")
print(f"d.keys(): {d.keys()}")
petya = d.pop('Петя')
print(f"petya: {petya}, d: {d}")
d['Марина'] = 29
print(f"d: {d}")
d.update([('Вася', 10), ('Петя', 20)])
print(f"d: {d}")
print(f"d['Вася']: {d['Вася']}")
print(f"d.get('Неизвестный чел', 'я такого не знаю'): {d.get('Неизвестный чел', 'я такого не знаю')}")
print(f"d['Неизвестный чел']: {d['Неизвестный чел']}")
```

```
d.items(): dict_items([('Вася', 5), ('Алена', 10), ('Петя', 28)])
d.keys(): dict_keys(['Вася', 'Алена', 'Петя'])
d.keys(): dict_keys(['Вася', 'Алена', 'Петя'])
petya: 28, d: {'Вася': 5, 'Алена': 10}
d: {'Вася': 5, 'Алена': 10, 'Марина': 29}
d: {'Вася': 10, 'Алена': 10, 'Марина': 29, 'Петя': 20}
d['Вася']: 10
d.get('Неизвестный чел', 'я такого не знаю'): я такого не знаю
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[90], line 13
     11 print(f"d['Вася']: {d['Вася']}")
     12 print(f"d.get('Неизвестный чел', 'я такого не знаю'): {d.get('Неизвестный чел', 'я такого не знаю')}")
---> 13 print(f"d['Неизвестный чел']: {d['Неизвестный чел']}")

KeyError: 'Неизвестный чел'
```

Ключами в словаре могут выступать только неизменяемые объекты!

Метод	Описание
dict.fromkeys(seq[, value])	Создает словарь с ключами из seq и значением value (по умолчанию None).
dict.get(key[, default])	Возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).
dict.items()	Возвращает пары (ключ, значение).
dict.keys()	Возвращает ключи в словаре.
dict.pop(key[, default])	Удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).
dict.popitem()	Удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError.
dict.setdefault(key[, default])	Возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).
dict.update([other])	Обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются
dict.values()	Возвращает значения в словаре.
dict.copy()	Поверхностная копия словаря
dict.clear()	Очистка словаря

Python. Задание 6

1. Преобразуйте (можно вручную) ваш список *employees* из задания 4 в словарь, где ключом будет выступать кортеж из имени и фамилии сотрудника, а значениями – словарь из возраста (‘*age*’) и должности (‘*position*’)
2. Добавьте еще одного сотрудника в словарь
3. Выведите на экран запрос несуществующего сотрудника, чтобы запрос вернул (*None*)
4. Выведите на экран всех сотрудников в формате <имя> <фамилия>

Python. Условные выражения и операторы

```
if <условие1>:
    <действие1>
elif <условие2>:
    <действие2>
elif <условие3>:
    <действие3>
else:
    <действие4>
```

<условие1> – выражение, возвращающее объект типа bool:

- Результат операций сравнения: **>**, **<**, **>=**, **<=**, **==**, **!=**
- Результат вычисления логических операций: **and**, **or**, **not**
- Результат работы оператора **in** /**not in**: наличие/отсутствие значения в наборе значений
- Результат работы оператора **is**: идентичность объектов

Python >= 3.10

```
match <переменная>:
    case <значение1>:
        <действие1>
    case <значение2>:
        <действие2>
    case _:
        <действие по умолчанию>
```

<переменная> – имя переменной, по которой будет идти сравнение

<значение1> – значение переменной, при котором будет выполняться ***<действие1>***, аналогично конструкции:

```
if <переменная> == <значение1>:
    <действие1>
```

- Любое число, не равное 0, или непустой объект – истина (**True**).
- Числа, равные 0, пустые объекты и значение **None** – ложь (**False**)
- Операции сравнения применяются к структурам данных рекурсивно
- Операции сравнения возвращают **True** или **False**
- Логические операторы **and** и **or** возвращают истинный или ложный объект-операнд

```
print(1 == 1)                    # True
print(1 == '1')                  # False
print(bool('asf'))               # True
print(bool(0))                   # False
print(bool([]))                  # False
print(bool([0, 0]))              # True
print('a' in ['a', 'b', 'c'])    # True
a = 1; b = 2; print(a is b)      # False
a = 2; b = 2; print(a is b)      # True
```

Python. Циклы

```
for <элемент> in <объект>:
    <действие1>
    break
    <действие2>
    continue
    <действие3>
else:
    <действие4>
```

```
while <условие>:
    <действие1>
    break
    <действие2>
    continue
    <действие3>
else:
    <действие4>
```

<объект> – итерируемый объект (списки, строки, кортежи, словари)

<элемент> – переменная, которой будут поочередно присваиваться значения итерируемого объекта

<условие> – условие, которое будет проверятся перед каждым выполнение цикла

else (опционально) – добавление действия, которое выполнится после прохождения всего цикла (или исключения StopIteration)
break (опционально) – прерывание цикла (else игнорируется)
continue (опционально) – переход к следующему итерируемому объекту без выполнения оставшегося кода.

- Если цикл **for** не сложный для ускорения обработки лучше использовать функцию **map()** или генераторы.
- По завершению цикла **for** переменная, ***<элемент>*** остаётся доступным, равным последнему значению итерируемого объекта.
- Изменение последовательности ***<объект>*** в ходе итерирования может приводить к ошибкам и пропускам элементов.
- Цикл **while** используется, когда точное число повторений неизвестно и может изменяться в зависимости от поведения переменной в теле цикла.
- В других случаях предпочтительнее использовать цикл **for**, т.к. он быстрее **while**.
- Цикл **while** может стать бесконечным, необходимо предусматривать механизмы защиты.

Python. Примеры циклов

1. Минималистичный for

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

2. while для той же задачи выглядит не так изящно

```
i = 0

while i < 5:
    print(i)
    i += 1
```

```
0
1
2
3
4
```

3. Зато while подойдет для задач с условием. Но если переменная error никогда не изменится, цикл станет бесконечным

```
error = False

while not error:
    execute_some_function()
```

4. Работа операторов break и continue

```
for word in ['dog', 'table', 'cat', 'horse', 'tiger']:
    if word == 'table':
        print('table - лишнее слово')
        continue

    if word == 'horse':
        print('лошадь мы не ожидали')
        break

    print(word)
```

```
dog
table - лишнее слово
cat
лошадь мы не ожидали
```

5. Можно перебирать итерируемые объекты с порядковым номеров (enumerate)

```
for i, word in enumerate(['dog', 'table', 'cat',
                          'horse', 'tiger']):
    print(i, word)
```

```
0 dog
1 table
2 cat
3 horse
4 tiger
```

6. Можно перебирать сразу несколько объектов (zip)

```
A = range(5)
print('A = ', A)
B = range(10,15)
print('B = ', A)

for a, b in zip(A, B):
    print(f"a = {a}, b = {b}")
```

```
A =  range(0, 5)
B =  range(0, 5)
a = 0, b = 10
a = 1, b = 11
a = 2, b = 12
a = 3, b = 13
a = 4, b = 14
```

7. Можно использовать несколько уровней вложений циклов (вложенные циклы)

```
A = [1, 2]
B = ['один', 'два']
C = ['one', 'two']

for a in A:
    for b in B:
        for c in C:
            print(f"a = {a}, b = {b}, c = {c}")
```

```
a = 1, b = один, c = one
a = 1, b = один, c = two
a = 1, b = два, c = one
a = 1, b = два, c = two
a = 2, b = один, c = one
a = 2, b = один, c = two
a = 2, b = два, c = one
a = 2, b = два, c = two
```

Python. Задание 7

1. Используйте ваш словарь *employees* из задания 6, добавьте каждому сотруднику в словарь его параметров параметр «зарплата» (‘*salary*’) с произвольными целочисленными значениями.
2. Найдите (*for*) самого низкооплачиваемого и высокооплачиваемого сотрудника, выведите на экран (например, “Петров получает 100000, а Сидоров 10000”). Для форматирования можно воспользоваться f-string:

```
>>> name = "Eric"
>>> age = 74
>>> f"Hello, {name}. You are {age}."
'Hello, Eric. You are 74.'
```

```
print(f"{key_max[1]} получает {employees[key_max]['salary']}р, а {key_min[1]} - {employees[key_min]['salary']}р")
```

```
Иванов получает 60866р, а Сидоров - 33825р
```

3. Повышайте зарплату самому низкооплачиваемому сотруднику до тех пор, пока (*while*) она не сравняется или не станет выше зарплаты самого высокооплачиваемого сотрудника. Шаг повышения зарплаты должен быть равен одной десятой разницы между зарплатами указанных сотрудников. После каждого повышения выводите на экран новую зарплату, а в конце – увольте сотрудника путем удаления его из словаря.

Python. Генераторы словарей и списков

Создание списка

```
%%time
lst_1 = []

for i in range(10_000_000):
    if i%2 == 0:
        lst_1.append(i)
```

```
Wall time: 1.22 s
```

```
%%time
lst_2 = [i for i in range(10_000_000) if i%2==0]
```

```
Wall time: 694 ms
```

List comprehension

Создание словаря

```
%%time
dict_1 = {}

for i in range(10_000_000):
    if i%2 == 0:
        dict_1[str(i)] = i
```

```
Wall time: 3.17 s
```

```
%%time
dict_2 = {str(i): i for i in range(10_000_000) if i%2==0}
```

```
Wall time: 2.63 s
```

Dict comprehension

```
lst_1 = []

for i in range(10):
    if i%2 == 0:
        lst_1.append(i)

lst_1
```

```
[0, 2, 4, 6, 8]
```

```
lst_2 = [i for i in range(10) if i%2==0]
lst_2
```

```
[0, 2, 4, 6, 8]
```

```
%%time
dict_1 = {}

for i in range(10_000_000):
    if i%2 == 0:
        dict_1[str(i)] = i
```

```
Wall time: 3.17 s
```

```
%%time
dict_2 = {str(i): i for i in range(10_000_000) if i%2==0}
```

```
Wall time: 2.63 s
```

Python. Задание 8

1. При помощи *list comprehension* создайте список из целочисленных значений от 1 до 20.
2. При помощи *list comprehension* создайте список из целочисленных значений от 1 до 20, но только нечетные числа.
3. При помощи *dict comprehension* создайте новый словарь из словаря *employees*, где ключом будет фамилия, а значением – должность сотрудника.
4. При помощи *dict comprehension* создайте новый словарь из словаря *employees*, где ключом будет фамилия, а значением – должность сотрудника, если его зарплата больше 50000.