Занятие 1. Знакомство с языком программирования Python и введение в науку о данных

Что такое python?

Python -- объектно-ориентированный сценарный язык, официально опубликованный в 1991-м году. Он был разработан *Гвидо ван Россумом* из Национального исследовательского института математики и компьютерных наук в Амстердаме.

Python быстро стал одним из самых популярных языков программирования в мире. Он пользуется особой популярностью в среде образования и научных вычислений и давно уже превзошёл язык программирования R в качестве самого популярного языка обработки данных. Почему он так популярен?

- Python -- бесплатный общедоступный проект с открытым кодом, имеющий огромное сообщество пользователей.
- Он проще в изучении, чем такие языки, как C, C++, C# и Java, что позволяет быстро включиться в работу как новичкам, так и профессиональным разработчикам.
- Код Python проще читается, чем большинство других популярных языков программирования.
- Он широко применяется в образовательной области.
- Он повышает эффективность труда разработчика за счёт обширной подборки стандартных и сторонних библиотек с открытым кодом, так что программисты могут быстрее писать код и решать сложные задачи с минимумом кода.
- Python -- популярный язык веб-разработки. В его экосистеме можно найти довольно классные (и популярные) фреймворки Django, Flask, Streamlit и другие.
- Он поддерживает популярные парадигмы программирования -- процедурную, функциональную и объектно-ориентированную. Мы будем пользоваться в основном процедурной и функциональной парадигамами, не вдаваясь в подробности ООП на Python.
- Python используется для построения любых программ от простых сценариев до сложных приложений со множеством пользователей, таких как Dropbox, YouTube, Reddit и Instagram (вопросы, почему нет маркировки для этого сервиса, не задавать).
- Python -- популярный язык для задач искусственного интеллекта, а эта область в последнее время стремительно развивается (отчасти благодаря её особой связи с областью data science).
- Для программистов Python существует обширный рынок труда во многих областях, связанных не только со сферами ИИ (искусственный интеллект) и науки о данных.

Скачать python на своё компьютер можно здесь --> https://www.python.org/ (https://www.python.org/)

Библиотеки

В моём курсе мы будем по возможности пользоваться существующими библиотеками (включая стандартные библиотеки Python, библиотеки data science и некоторые сторонние библиотеки) -- их применение способствует повышению эффективности разработки программных продуктов. Так, вместо того чтобы писать большой объём исходного кода (дорогостоящий и длительный процесс, особенно на вашем **экзамене**), можно просто создать объект уже существующего библиотечного класса посредством всего одной команды Python.

Некоторые модули стандартной библиотеки Python

- collections — дополнительные структуры данных помимо списков, кортежей, словарей и множеств;
- csv — обработка файлов с данными, разделенными запятыми;
- datetime, time — операции с датой и временем;
- decimal — вычисления с фиксированной и плавающей точкой, включая финансовые вычисления;
- doctest — простое модульное тестирование с использованием проверочных тестов и ожидаемых результатов, закодированных в doc-строках;
- json — обработка формата JSON (JavaScript Object Notation) для использования с веб-сервисами и базами данных документов NoSQL;
- math — распространенные математические константы и операции;
- os — взаимодействие с операционной системой;
- queue — структура данных, работающая по принципу «первым зашел, первым вышел» (FIFO);
- random — псевдослучайные числа;
- re — регулярные выражения для поиска по шаблону;
- sqlite3 — работа с реляционной базой данных SQLite;
- statistics — функции математической статистики (среднее, медиана, дисперсия, мода и т. д.);
- string — обработка строк;
- sys — обработка аргументов командной строки; потоки стандартного ввода, стандартного вывода; стандартный поток ошибок;
- timeit — анализ быстродействия;

Jupyter

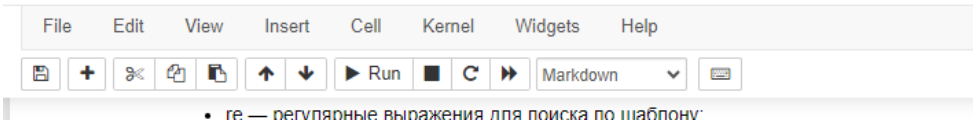
Вы уже работали в среде Google Collab, которая позволяет писать в одном документе и код, и текст, и вставлять где-то между ними разные мультимедиа (aka картинки). Среда **Jupyter Notebook** — интерактивная браузерная среда, в которой можно писать и выполнять код, а также комбинировать его с текстом, изображениями и видео, то есть то же самое, только её вы можете установить на свой компьютер и пользоваться ресурсами своей машины, а не полагаться на сервера Google.

Документы Jupyter Notebook широко применяются в сообществе data science в частности и в более широком научном сообществе в целом. Они рассматриваются как предпочтительный механизм проведения аналитических исследований данных и распространения воспроизводимых результатов. Среда Jupyter Notebook поддерживает все больше языков программирования.

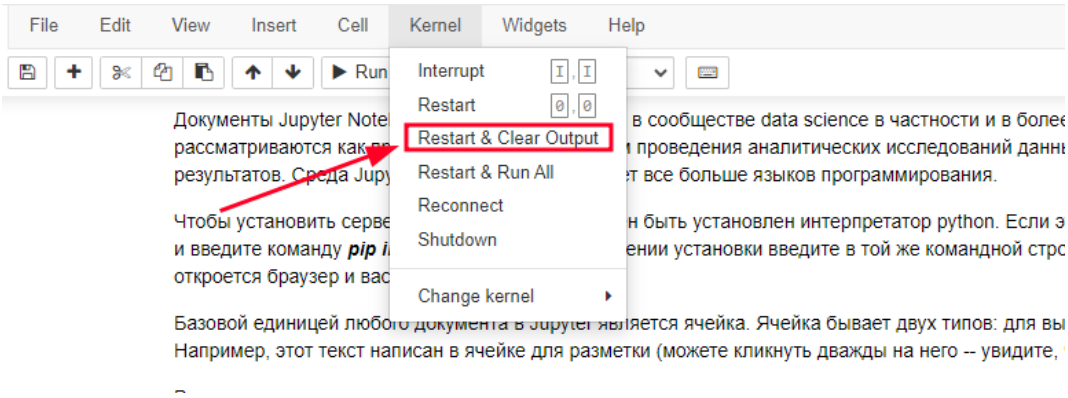
Чтобы установить сервер Jupyter, у вас уже должен быть установлен интерпретатор python. Если это условие выполнено, откройте командную строку и введите команду ***pip install notebook***. По завешении установки введите в той же командной строке команду ***jupyter notebook*** -- после этого откроется браузер и вас перебросит в него.

Базовой единицей любого документа в Jupyter является ячейка. Ячейка бывает двух типов: для выполнения кода и для какой-нибудь разметки. Например, этот текст написан в ячейке для разметки (можете кликнуть дважды на него -- увидите, что собой представляет этот текст на самом деле).

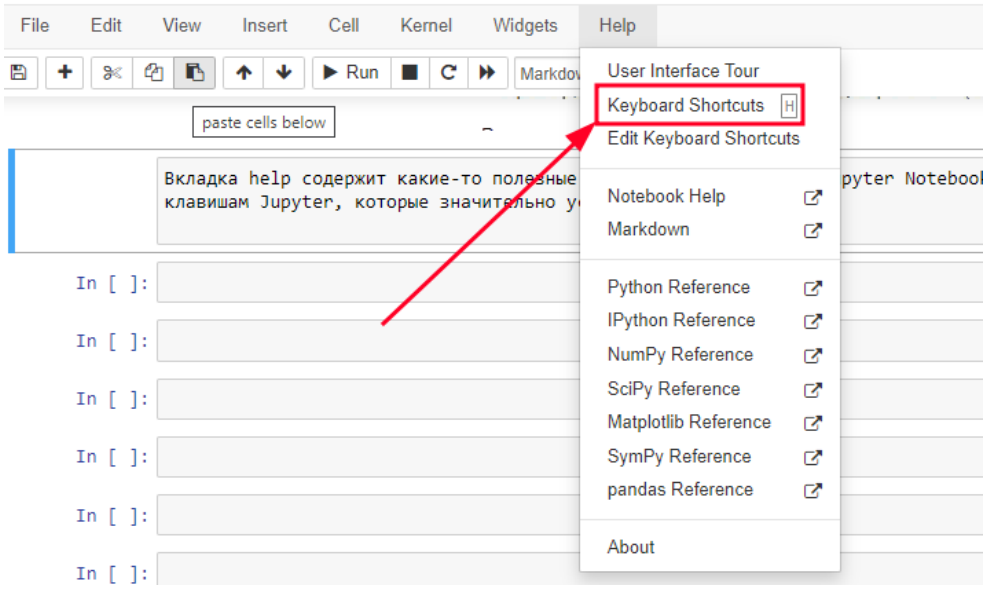
Вверху есть панель инструментов:



Панель инструментов, в принципе, должна быть вам знакома. Раздел File отвечает за действия с вашим документом: например, через него можно сохранить изменения в документе, а затем скачать к себе на компьютер в удобном для вас формате (например, html). Вкладка Kernel содержит действия для работы с ядром интерпретатора python. Например, если вы хотите сбросить нумерацию всех ячеек и отменить все изменения в данных, можно воспользоваться этой командой:



Вкладка help содержит какие-то полезные шпаргалки по работе с Jupyter Notebook. Я сам часто пользуюсь подсказкой по горячим клавишам Jupyter, которые значительно ускоряют мою работу:



Насколько велики большие данные?

Для специалистов по компьютерной теории и data science данные играют не менее важную роль, чем написание программ. По данным IBM, в мире ежедневно создаются приблизительно 2,5 квинтиллиона байт (2,5 экзабайта) данных, а 90% мировых данных были созданы за последние два года. По материалам IDC, к 2025-му году ежегодный объем глобального генерирования данных достигнет уровня 175 зеттабайт (175 триллионов гигабайт, или 175 миллиардов терабайт). Рассмотрим наиболее популярные единицы объема данных.

1. Мегабайт

Один мегабайт составляет приблизительно 1 миллион (а на самом деле 220) байт. Многие файлы, используемыми нами в повседневной работе, занимают один или несколько мегабайтов.

Примеры:

- **Аудиофайлы** в формате MP3 — минута качественного звука в формате MP3 занимает от 1 до 2,4 Мбайт;
- **Фотографии** — фотография в формате JPEG, сделанная на цифровую камеру, может занимать от 8 до 10 Мбайт;
- **Видеоданные** — на камеру смартфона можно записывать видео с разным разрешением. Каждая минута видео может занимать много мегабайтов.

2. Гигабайт

Один гигабайт составляет приблизительно 1000 мегабайт (а точнее, 230 байт). Двуслойный диск DVD способен вмещать до 8,5 Гбайт1, что соответствует:

- до 141 часа аудио в формате MP3;

- приблизительно 1000 фотографий на 16-мегапиксельную камеру;
- приблизительно 7,7 минуты видео 1080p при 30 FPS, или приблизительно 2,85 минуты видео 4K при 30 FPS.

Современные диски наивысшей емкости Ultra HD Blu-ray вмещают до 100 Гбайт видеоданных. Потоковая передача фильма в разрешении 4K может происходить со скоростью от 7 до 10 Гбайт (с высокой степенью сжатия).

3. Терабайт

Один терабайт составляет около 1000 гигабайт (а точнее, 240 байт). Ёмкость современного диска для настольных компьютеров достигает 15 Тбайт, что соответствует:

- приблизительно 28 годам аудио в формате MP3;
- приблизительно 1,68 миллиона фотографий на 16-мегапиксельную камеру;
- приблизительно 226 часов видео 1080p при 30 FPS;
- приблизительно 84 часов видео 4K при 30 FPS.

У Nimbus Data наибольший объем диска SSD составляет 100 Тбайт; он вмещает в 6,67 раза больше данных, чем приведенные выше примеры аудио-, фото- и видеоданных для 15-терабайтных дисков.

4. Петабайт, экзабайт и зеттабайт

Почти 4 миллиарда людей в интернете ежедневно создают около 2,5 квинтиллиона байт данных, то есть 2500 петабайт (1 петабайт составляет около 1000 терабайт) или 2,5 экзабайта (1 экзабайт составляет около 1000 петабайт). По материалам статьи из AnalyticsWeek за март 2016 года, через пять лет к интернету будет подключено свыше 50 миллиардов устройств (в основном через «интернет вещей»), а к 2023 году на каждого человека на планете ежесекундно будет производиться 1,7 мегабайта новых данных2. Для современного уровня населения (приблизительно 7,7 миллиарда людей) это соответствует приблизительно:

- 13 петабайт новых данных в секунду;
- 780 петабайт в минуту;
- 46 800 петабайт (46,8 экзабайт) в час;
- 1123 экзабайта в день, то есть 1,123 зеттабайта (ЗБ) в день (1 зеттабайт составляет около 1000 экзабайт).

Это эквивалентно приблизительно 5,5 миллиона часов (более 600 лет) видео в разрешении 4K в день, или приблизительно 116 миллиардам фотографий в день!

Потенциал больших данных

Вероятно, в ближайшие годы в области больших данных будет наблюдаться экспоненциальный рост. В перспективе количество вычислительных устройств достигнет 50 миллиардов, и трудно представить, сколько еще их появится за несколько ближайших десятилетий. Очень важно, чтобы всеми этими данными могли пользоваться не только бизнес, правительственные организации, военные, но и частные лица.

Интересно, что некоторых из лучших работ по поводу больших данных, data science, искусственного интеллекта и т. д. были написаны в авторитетных коммерческих организациях: J. P. Morgan, McKinsey и др. Привлекательность больших данных для большого бизнеса бесспорна, особенно если принять во внимание стремительно развивающиеся результаты. Многие компании осуществляют значительные инвестиции в области высоких технологий (большие данные, машинное обучение, глубокое обучение, обработка естественных языков, и т. д.), и добиваются ценных результатов. Это заставляет конкурентов также вкладывать средства, что стремительно повышает спрос на профессионалов с опытом работы в области больших данных и теории вычислений. По всей вероятности, эта тенденция сохранится в течение многих лет.

Анализ больших данных

Аналитическая обработка данных — зрелая, хорошо проработанная научная и профессиональная дисциплина. Сам термин «анализ данных» появился в 1962 году, хотя люди применяли статистику для анализа данных в течение тысяч лет, еще со времен Древнего Египта. Анализ больших данных следует считать более современным явлением — так, термин «большие данные» появился около 2000 года. Есть четыре основные характеристики больших данных, обычно называемые **«четырьмя V»**:

1. **Объем (Volume)** — количество данных, производимых в мире, растет с экспоненциальной скоростью.
2. **Скорость (Velocity)** — темпы производства данных, их перемещения между организациями и изменения данных стремительно растут.
3. **Разнообразие (Variety)** — некогда данные в основном были алфавитно-цифровыми (то есть состояли из символов алфавита, цифр, знаков препинания и некоторых специальных знаков). В наши дни они также включают графику, аудио, видео и данные от стремительно растущего числа датчиков «интернета вещей», установленных в жилищах, коммерческих организациях, транспортных средствах, городах и т. д.
4. **Достоверность (Veracity)** — характеристика, указывающая на то, являются ли данными полными и точными, и, как следствие, позволяющая ответить на вопросы вроде: «Можно ли на них полагаться при принятии критических решений?», «Насколько реальна информация?» и пр.

Большая часть данных сейчас создается в цифровой форме, относится к самым разным типам, достигает невероятных объемов и перемещается с потрясающей скоростью. Закон Мура (https://ru.wikipedia.org/wiki/Закон_Мура (https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%9C%D1%83%D1%80%D0%B0)) и связанные с ним наблюдения позволяют нам организовать экономичное хранение данных, ускорить их обработку и перемещение — и все это на скоростях, экспоненциально растущих со временем.

Следующая цитата Ричарда Хемминга (Richard W. Hamming), пусть и относящаяся к 1962 году, задает тон для всей этой области и для моего курса в частности: *«Целью вычислений является понимание сути, а не числа»*

Искусственный интеллект -- на пересечении компьютерной теории и data science

Когда ребенок впервые открывает глаза, «видит» ли он лица своих родителей? Имеет ли он представление о том, что такое лицо, — или хотя бы о том, что такое форма? Как бы то ни было, ребенок должен «познать» окружающий мир. Именно этим занимается искусственный интеллект (AI, Artificial Intelligence).

Он обрабатывает колоссальные объемы данных и обучается по ним. Искусственный интеллект применяется для игр, для реализации широкого спектра приложений распознавания изображений, управления автономными машинами, обучения роботов, предназначенных для выполнения новых операций, диагностики медицинских состояний, перевода речи на другие языки практически в реальном времени, создания виртуальных собеседников, способных отвечать на произвольные вопросы на основании огромных баз знаний, и многих других целей.

Кто бы мог представить всего несколько лет назад, что автономные машины с искусственным интеллектом появятся на дорогах, став вскоре обыденным явлением? А сегодня в этой области идет острая конкуренция! Конечной целью всех этих усилий является общий искусственный интеллект, то есть искусственный интеллект, способный действовать разумно на уровне человека. Многим эта мысль кажется пугающей.

В течение многих десятилетий искусственный интеллект рассматривался как область, в которой есть задачи, но нет решений. Дело в том, что после того, как конкретная задача была решена, люди начинают говорить: «Что ж, это не интеллект, а просто компьютерная программа, которая указывает компьютеру, что нужно делать». Однако благодаря методам машинного обучения и глубокого обучения мы уже не программируем компьютер для решения конкретных задач. Вместо этого мы предлагаем компьютерам решать задачи, обучаясь по данным, и обычно по очень большим объемам данных.

Для решения многих самых интересных и сложных задач применялись методы глубокого обучения. Компания Google (одна из многих подобных себе) ведет тысячи проектов глубокого обучения, причем их число быстро растет.

Первые шаги на python

Имена переменных

Имя переменной, такое как x, является идентификатором. Каждый идентификатор может состоять из букв, цифр и символов подчеркивания (_), но не может начинаться с цифры. В языке Python учитывается регистр символов, так что идентификаторы Number и number считаются различными, потому что один начинается с прописной буквы, а другой — со строчной.

Типы

Python -- язык программирования с нестрогой типизацией. Это значит, что система типов в python есть (она такая же или очень похожая на систему типов Java и C++), но каждая переменная может менять свой тип в зависимости от того значения, которое ей присваивается.

In [1]:

```
# например, вот так
my_var = 3 # здесь число
print(my_var, type(my_var))
my_var = "а здесь теперь строка" # а почему бы и да!
print(my_var, type(my_var))
```

```
3 <class 'int'>
а здесь теперь строка <class 'str'>
```

Как вы уже заметили, тип данных переменной можно получить через команду **type()**.
Данные можно преобразовывать.

In [2]:

```
int_var = "34" # пока ещё не число
int_var = int(int_var) # а вот теперь число, причём целое
int_var = str(int_var) # а теперь снова не число
```

Тип данных -- это множество каких-то значений и действий, которые с этими значениями можно выполнять.
Например, для тип данных **int** -- это множество целых чисел. А что с ними можно делать?

In [3]:

```
print(4 / 2) # например, просто делить
print(4 * 2) # просто умножать
print(4 + 4) # складывать
print(4 - 4) # вычитать
print(4 ** 5) # возводить в степень
print(4 // 3) # опечатаюсь в знаке деления и получу целую часть от результата деления
print(4 % 3) # а теперь -- остаток
```

```
2.0
8
8
0
1024
1
1
```

Тип данных float -- множество чисел с плавающей точкой (иначе говоря, дробные числа). Над ними можно выполнять те же действия, что и над множеством целых чисел.
Куда интереснее тип данных str -- строки. Со строками можно делать много всего интересного.

In [4]:

```
print("4" + "2") # складывать, или конкатенировать
print("4" * 20) # умножать на какое-то целое число
```

```
42
44444444444444444444
```

In [5]:

```
my_str = "Здесь какая-то строка для примера"
print(my_str.replace('к', '!')) # можно заменять какие-то символы в строке
print(my_str.find('для')) # искать подстроку, метод вернёт вам индекс первого вхождения, либо -1, если такой подстроки
print(my_str[6]) # можно брать отдельные символы из строки
```

```
Здесь !а!ая-то стро!а для примера
22
к
```

Обратите внимание, что в python нет типа данных char, как в других известных вам языках программирования. Поэтому при создании строк можно указывать как двойные кавычки, так и одинарные. Как удобнее.

In [6]:

```
print(type("Здесь строка"), type('И здесь тоже строка'))
```

```
<class 'str'> <class 'str'>
```

Условие представляет собой логическое выражение со значением «истина» (True) или «ложь» (False). Следующее условие сравнивает числа 7 и 4 и проверяет, какое из них больше:

In [7]:

```
7 > 4
```

Out[7]:

```
True
```

In [8]:

```
7 < 4
```

Out[8]:

```
False
```

True и False — ключевые слова Python. Использование ключевого слова в качестве идентификатора приводит к ошибке SyntaxError. Каждое из ключевых слов True и False начинается с буквы верхнего регистра.

Более того, True и False -- это всё множество типа данных **bool** -- логического типа данных.

Алгебраический оператор	Оператор Python	Пример условия	Смысл
>	>	x > y	x больше y
<	<	x < y	x меньше y
≥	>=	x >= y	x больше или равно y
≤	<=	x <= y	x меньше или равно y
=	==	x == y	x равно y
≠	!=	x != y	x не равно y

Как и в любом другом языке программирования, в python есть **оператор ветвления**, отвечающий за принятие решений. Это if.

In [9]:

```
age = 22
if age >= 18:
    print("Совершеннолетний. Проходи")
else:
    print("Малолетка! Иди домой.")
```

```
Совершеннолетний. Проходи
```

In [10]:

```
# если логика нашего кода сложнее, можно добавить третью ветку
grade = 88
if grade >= 90:
    print('A')
elif grade >= 80:
    print('B')
elif grade >= 70:
    print('C')
else:
    print('Не сдал')
```

```
B
```

Иногда необходимо выполнять какой-то блок команд до наступления выполнения какого-то условия. Вместо того, чтобы выводить 5 чисел таким образом:

In [11]:

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
1
2
3
4
5
```

Можно быть умнее и воспользоваться циклической конструкцией, например:

In [12]:

```
a = 1
while a <= 5:
    print(a)
    a+=1 # так можно сокращенно записывать увеличение переменной на какое-то число
```

```
1
2
3
4
5
```

In [13]:

```
# или так
for i in range(1, 6):
    print(i)
```

```
1
2
3
4
5
```

Всё, в глазах hr-ов вы уже выглядите серьёзным программистом. Шутка. Функция range(start, end, step) создаёт числовую последовательность от start до (end-step). Параметр step -- это шаг, с которым можно идти по числовой последовательности.

In [17]:

```
for i in range(1, 12, 2):
    print(i, end=' ') # параметр end позволяет выводить все числа в одну строку
```

```
1 3 5 7 9 11
```

In [23]:

```
# у циклов в python есть одна особенность, которой нет в других ЯП
# у циклов есть оператор else, который выполняется только в конце цикла
for i in range(0, 10):
    if i == 4:
        break # обрываем цикл, если условие выполняется
    print(i)
else:
    print("Цикл выполнился не полностью")

print('Мы здесь')
```

```
0
1
2
3
Мы здесь
```

In [24]:

```
for i in range(0, 10):
    if i == 4:
        continue # перепрыгиваем итерацию цикла, если условие выполняется
    print(i)
else:
    print("Выполнился полностью")

print("Мы здесь")
```

```
0
1
2
3
5
6
7
8
9
Выполнился полностью
Мы здесь
```

Функции

Вы уже пользовались встроенными функциями объектов python. Но у вас есть возможность писать и свои функции, пользовательские.

In [ ]:

```
def my_sum(a, b):
    return a + b
```

Определение функции начинается с ключевого слова def, за которым следует имя функции square в круглых скобках и двоеточие (:). Как и идентификаторы переменных, по соглашению имена функций должны начинаться с буквы нижнего регистра, а в именах, состоящих из нескольких слов, составляющие должны разделяться символами подчеркивания.

Снабженные отступом строки после двоеточия (:) образуют блок функции. Он состоит из необязательной doc-строки, за которой следуют команды, выполняющие операцию функции.

У функций python тоже есть одна особенность, которой нет у многих других языков программирования. Например, в Java или C++ из функции можно возвращать строго одно значение. В python такого ограничения нет, любая функция может возвращать больше одного значения.

In [25]:

```
def my_sum_and_mul(a, b):
    return a + b, a * b

s, m = my_sum_and_mul(10, 2) # такое присваивание называется деструктурирующим присваиванием
print(s, m)
```

```
12 20
```

In [26]:

```
# если нам требуется только одно значение для работы, но функция возвращает два значения,
# а переписывать не хочется -- есть лайфхак, как с этим быть

a, _ = my_sum_and_mul(10, 2) # объявляем всего одну переменную, вторую игнорируем с помощью нижнего подчёркивания
```

Локальные переменные

Хотя мы не определяем переменные в блоке my_sum, это можно сделать. Параметры функции и переменные, определенные в ее блоке, являются *локальным переменными* — они могут использоваться только внутри функции и существуют только во время ее выполнения. Попытка обратиться к локальной переменной за пределами блока функции приводит к ошибке NameError, которая означает, что переменная не определена

In [27]:

```
def my_sum(a, b):
    returned_sum = a + b
    return returned_sum

print(returned_sum)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[27], line 5
      2     returned_sum = a + b
      3     return returned_sum
----> 5 print(returned_sum)

NameError: name 'returned_sum' is not defined
```

Последовательности: списки и кортежи

Коллекции представляют собой структуры данных, состоящие из взаимосвязанных элементов данных. Примерами коллекций могут служить подборки ваших любимых песен на смартфоне, списки контактов, библиотечных книг, ваши карты в карточной игре, игроки вашей любимой спортивной команды, акции в инвестиционном портфеле, списки покупок и т. д. Встроенные коллекции Python позволяют хранить данные и обращаться к ним удобно и эффективно.

Списки в Python -- упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы внутри списка могут отличаться). Чтобы использовать списки, их нужно создать. Создать список можно несколькими способами. Например, можно обработать любой итерируемый объект (например, строку) встроенной функцией list:

In [28]:

```
list("список")
```

Out[28]:

```
['с', 'п', 'и', 'с', 'о', 'к']
```

In [29]:

```
# список можно создать и при помощи литерала
s = [] # можно s = list()
l = ['s', 'p', ['isok'], 2]
print(s, l)
```

```
[] ['s', 'p', ['isok'], 2]
```

In [30]:

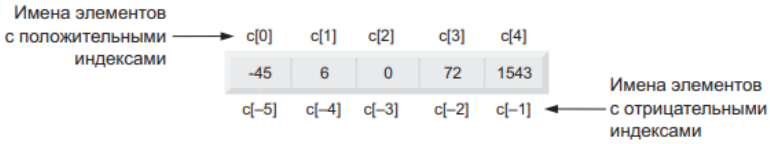
```
print(l[2]) # можно взять элемент списка по индексу
l[2] = 'something' # можно изменять отдельные элементы списков
print(l[2])
```

```
['isok']
something
```

In [31]:

```
# индексы списка должны быть целыми числами
print(l[1])
# целыми, но не всегда положительными
print(l[-1]) # вернёт последний элемент
```

```
p
2
```



In [32]:

```
# списки можно складывать
first_lst = [1, 2, 4, 10, 12]
second_lst = ['str', 'int', 'float', 'bool']
first_lst + second_lst
```

Out[32]:

```
[1, 2, 4, 10, 12, 'str', 'int', 'float', 'bool']
```

In [35]:

```python
# списки можно обходить, перебирая индексы элементов вот так
for i in range(0, len(l)):
    print(l[i], end=" ") # функция len() вычисляет длину коллекции

print()
# можно перебирать не индексы элементов, а сами элементы
for item in l:
    print(item, end=" ")
print()

# а можно делать всё вместе с помощью функции enumerate()
for idx, item in enumerate(l):
    print(f"индекс: {idx} - элемент: {item}")
```

```
s p something 2
s p something 2
индекс: 0 - элемент: s
индекс: 1 - элемент: p
индекс: 2 - элемент: something
индекс: 3 - элемент: 2
```

In [44]:

```python
# ещё один способ генерировать списки -- использовать генераторы списков
c = [i for i in range(0, 10)]
print(c)

g = [i for i in l if i in c and i % 2 == 0]
print(g)
```

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

У списков есть встроенные методы, которые позволяют их модифицировать.

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, 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. ValueError, если такого элемента не существует
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()	Поверхностная копия списка
list.clear()	Очищает список

Кортежи (tuple)в Python – это те же списки за одним исключением. Кортежи неизменяемые структуры данных. Так же как списки они могут состоять из элементов разных типов, перечисленных через запятую. Кортежи заключаются в круглые, а не квадратные скобки.

In [45]:

```python
a = (10, 2.13, "square", 89, 'C')
print(a)
```

```
(10, 2.13, 'square', 89, 'C')
```

In [46]:

```
# из кортежа можно извлекать элементы
print(a[0])
# но изменять элементы кортежа нельзя
a[0] = 11 # !!!
```

```
10

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[46], line 4
      2 print(a[0])
      3 # но изменять элементы кортежа нельзя
----> 4 a[0] = 11

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

Срезы, или сегментация последовательностей

Вы можете создавать сегменты последовательностей, чтобы сформировать новые последовательности того же типа, содержащие подмножества исходных элементов. Операции сегментации могут модифицировать изменяемые последовательности, а те, которые не изменяют последовательность, работают одинаково для списков, кортежей и строк.

In [47]:

```
# сегмент, состоящтий из элементов списка с индексами с 2 по 5
numbers = [2, 3, 5, 7, 11, 13, 17, 19]
numbers[2:6]
```

Out[47]:

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

Если начальный индекс не указан, то предполагается значение 0. Таким образом, обозначение сегмента numbers[:6] эквивалентно записи numbers[0:6].

In [48]:

```
print(numbers[:6])
print(numbers[0:6]) # явно указываем стартовый индекс
```

```
[2, 3, 5, 7, 11, 13]
[2, 3, 5, 7, 11, 13]
```

Если не указан конечный индекс, то Python предполагает, что он равен длине последовательности (8 в данном случае), поэтому сегмент из фрагмента содержит элементы numbers с индексами 6 и 7:

In [49]:

```
print(numbers[6:])
print(numbers[6:len(numbers)]) # явно указываем конечный индекс
```

```
[17, 19]
[17, 19]
```

В следующем коде используется шаг 2 для создания сегмента, содержащего каждый второй элемент numbers:

In [50]:

```
numbers[::2]
```

Out[50]:

```
[2, 5, 11, 17]
```

Отрицательное значение шага используется для сегментации в обратном порядке. Следующий код компактно создает новый список в обратном порядке:

In [51]:

```
numbers[::-1]
```

Out[51]:

```
[19, 17, 13, 11, 7, 5, 3, 2]
```

Задачи на закрепление

Задержи дыхание!

Вам предоставлен массив чисел, которые через равные промежутки времени обозначают высоту вашего персонажа над уровнем моря:

Положительные числа обозначают высоту над водой. 0 – уровень моря. Отрицательные числа обозначают глубину под поверхностью воды.

Создайте функцию, которая возвращает, выживет ли ваш персонаж после погружения без присмотра, учитывая массив целых чисел. Ваш персонаж начинает со счетчиком дыхания 10, что является максимальным значением. При погружении под воду счетчик вашего дыхания уменьшается на 2 единицы на каждый элемент в наборе*. Осторожно! Если ваше дыхание уменьшится до 0, ваш персонаж умрет!

Чтобы этого не произошло, вы можете пополнить свое дыхание на 4 единицы (максимум до 10) за каждый элемент в массиве, где вы находитесь на уровне моря или выше.

Ваша функция должна возвращать True, если ваш персонаж выжил, и False, если нет.

Примеры: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] → True

[-3, -6, -2, -6, -2] → False

[2, 1, 2, 1, -3, -4, -5, -3, -4] → False

Примечание от меня: уменьшение/увеличение запаса воздуха у персонажа происходит не через умножение порции кислорода на глубину/высоту в массиве, а по-другому: один элемент ==> +2 или -4 единицы кислорода.

Переключение по щелчку

Создайте функцию, которая всегда возвращает True для каждого элемента в заданном списке. Однако, если элементом является слово «flick», переключитесь на возврат противоположного логического значения.

["data science", "flick", "code", "python"] → [True, False, False, False]

['flick', 'chocolate', 'adventure', 'sunshine'] →[False, False, False, False]

['bicycle', 'jarmony', 'flick', 'sheep', 'flick'] → [True, True, False, False, True]

In [ ]:

```
1
```

In [ ]:

```
1
```

Удаление дубликатов

Определите функцию, которая удаляет дубликаты из массива неотрицательных чисел и возвращает их в результате. Порядок последовательности должен оставаться прежним.

Input -> Output [1, 1, 2] -> [1, 2] [1, 2, 1, 1, 3, 2] -> [1, 2, 3]

In [ ]:

```
1
```

Анализ трафика

Вы работаете в компании, которая анализирует количество трафика на определенном перекрестке в часы пик с 16:00 до 20:00. Каждый день вам предоставляется список, содержащий количество транспортных средств, проезжающих через перекресток каждые 10 минут с 16:00 до 20:00. Ожидается, что вы вернете список кортежей, каждый из которых содержит час и максимальный объем трафика за каждый час.

a1 = [23,24,34,45,43,23,57,34,65,12,19,45, 54,65,54,43,89,48,42,55,22,69,23,93]

traffic_count(a1) ==> [('4:00pm', 45), ('5:00pm', 65), ('6:00pm', 89), ('7:00pm', 93)]

In [ ]:

```
1
```