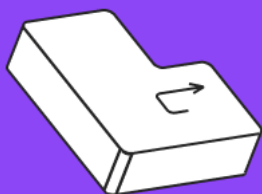




Лекция 1.

Основы Python

Погружение в Python



Оглавление

Введение	2
На этой лекции мы	2
Дополнительные материалы к лекции, которые вам пригодятся	3
Термины лекции	3
Подробный текст лекции	
1. Установка и настройка Python	
Введение	4
Установка Python	4
Работа с pip	7
Команда install	7
Команда freeze	8
Файл requirements.txt	8
Отличия командной строки ОС и интерпретатора Python	8
Работа в режиме интерпретатора	9
Использование " _".	9
Арифметические операторы в Python	9
Написание кода в интерактивном режиме	10
Задание	11
2. Повторяем основы Python	11
Работа в IDE	12
Python Style. PEP-8 (руководство по стилю) и PEP-257 (оформление документации/комментариев)	13
Переменные и требования к именам	13
Константы	15
True, False, None	15
Функция id()	15
Зарезервированные слова, keyword.kwlist	16
Ввод и вывод данных	16
Вывод, функция print()	16
Ввод, функция input()	1
Антипаттерн "магические числа"	19
Задание	19
3. Ветвление	20
Если, if	20
Отступы вместо фигурных скобок	20
Иначе, else	21

Еще если, elif	22
Выбор из вариантов, match и case	22
Логические конструкции, or, and, not	23
Ленивый if	24
Проверка на вхождение, in	24
Тернарный оператор	25
Задание	26
4. Циклы	27
Логический цикл while	27
Синтаксический сахар	27
Возврат в начало цикла, continue	28
Досрочное завершение цикла, break	29
Действие после цикла, else	29
Цикл итератор for in	31
Цикл по целым числам, он же арифметический цикл, функция range()	31
Имена переменных в цикле	32
Цикл с нумерацией элементов, функция enumerate()	32
Задание	33
5. Выводы	34

Введение

На курсе мы погрузимся в разработку на языке Python. Начнём с повторения основ, которые есть в любом языке программирования. Далее погрузимся в особенности написания программ на Python в процедурном стиле. Изучим особенности и возможности языка, такие как: функции, итераторы и генераторы, декораторы. Разберём обработку исключений и основы тестирования кода. Часть курса посвятим объектно-ориентированному программированию (ООП) на Python. В финале вас ждёт обзор стандартной библиотеки — "батареек" Python.

На этой лекции мы

1. Разберём установку и настройку Python.
2. Изучим правила создания виртуального окружения и работу с pip.
3. Повторим основы синтаксиса языка Python.
4. Познакомимся с рекомендациями по оформлению кода.
5. Изучим способы создания ветвящихся алгоритмов в Python.
6. Разберём варианты реализации циклов.
7. Узнаем про математические модули Python.

Дополнительные материалы к лекции, которые вам пригодятся

1. Инструкции по установке Python
https://drive.google.com/drive/folders/1f6J22TY_ga7ufS3Pu84rKYhskZ6Hjws2?usp=sharing
2. Википедия. Високосный год
https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%81%D0%BE%D0%BA%D0%BE%D1%81%D0%BD%D1%8B%D0%B9_%D0%B3%D0%BE%D0%B4

Термины лекции

- **UTF-8 (от англ. Unicode Transformation Format, 8-bit — «формат преобразования Юникода, 8-бит»)** — распространённый стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Юникода, используя переменное количество байт (от 1 до 4), и обеспечивающий полную обратную совместимость с 7-битной кодировкой ASCII. Кодировка UTF-8 сейчас является доминирующей в веб-пространстве. Она также нашла широкое применение в UNIX-подобных операционных системах.
- **Операционная система, сокр. ОС (англ. operating system, OS)** — комплекс управляющих и обрабатывающих программ, которые, с одной стороны, выступают как интерфейс между устройствами вычислительной системы и прикладными программами, а с другой стороны — предназначены для управления устройствами, управления вычислительными процессами, эффективного распределения вычислительных ресурсов между вычислительными процессами и организации надёжных вычислений.

- **IDE (Integrated Development Environment)** – это интегрированная, единая среда разработки, которая используется разработчиками для создания различного программного обеспечения. IDE представляет собой комплекс из нескольких инструментов, а именно: текстового редактора, компилятора или интерпретатора, средств автоматизации сборки и отладчика.
- **PIP (Package Installer for Python)** – система управления пакетами, которая используется для установки и управления программными пакетами, написанными на Python.
- **PEP (Python Enhancement Proposal, «Предложение по улучшению Python»)**. Это документы, предлагающие новые особенности языка. Они образуют официальную документацию особенности языка, принятие или отклонение которой обсуждается в сообществе Python.

Подробный текст лекции

1. Установка и настройка Python

Введение

Все примеры кода в рамках этого курса будут показаны/запущены в 64-разрядной версии Python 3.10. Для успешного выполнения заданий достаточно установить Python 3.9 или новее для вашей операционной системы. Некоторые примеры могут работать иначе или вовсе не работать, если у вас установлена более ранняя версия.


Установка Python

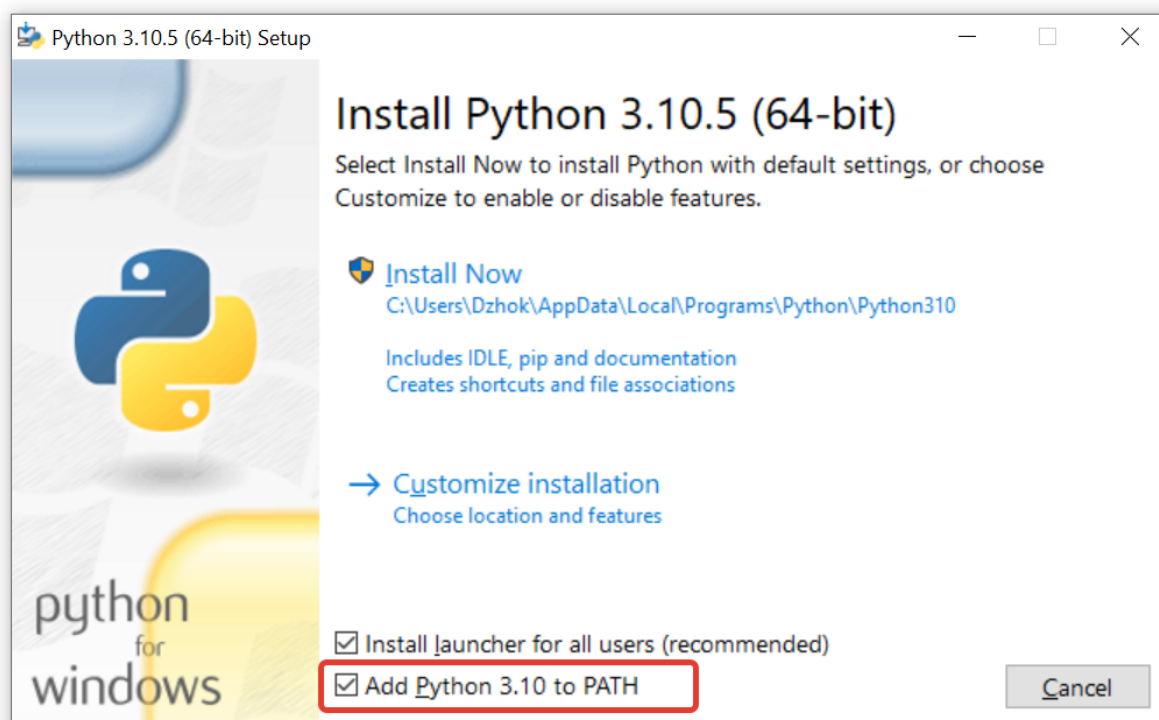
Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. О них мы будем регулярно говорить на курсе.

По умолчанию Python может быть не установлен на вашей ОС. Например, если у вас Windows. Либо у вас стоит старая версия Python, если вы используете Linux или

MacOS. Чтобы гарантированно иметь нужный интерпретатор стоит установить актуальную версию Python.

- Переходим на официальный сайт Python в раздел "Загрузки" <https://www.python.org/downloads/>
- Выбираем вашу версию операционной системы.
- Скачиваем установочный дистрибутив
- Запускаем установку, отвечаем на вопросы и ждём завершения.


 **Важно!** Если в установщике есть поле "Add Python to PATH", обязательно ставим галочку. Это упростит работу с Python из командной строки.



Чтобы убедиться в установке интерпретатора откроем командную строку вашей ОС и введем команду

```
python --version
```

В ответ ОС вернёт нам версию интерпретатора.

 **Внимание!** В некоторых ОС команда может выдать ошибку. В таком случае пробуем более точно указать версию Python. Например `python3 --version` или `python3.10 --version`

Если у вас возникли сложности с установкой, обратитесь к инструкции по установке для вашей ОС. Они приложены в материалы к уроку.

Создание виртуального окружения

Запуск простых программ возможен непосредственно из ОС. Но когда программист работает над большим проектом, используется виртуальное окружения. Так мы изолируем проект от других, хранящихся на этом же ПК. Кроме того изменения внутри проекта не влияют на "эталонную" версию Python, которую только что установили.

Для начала создайте каталог для нового проекта.



Важно! Не все устанавливаемые модули Python дружат с кириллицей, пробелами и вообще любыми символами помимо латиницы. Путь от корня диска до каталога должен быть без пробелов, а названия директорий написаны латинскими буквами. Допускается символ подчеркивания.

```
Выбрать dzhoker@DESKTOP-DI3TAQF: /mnt/c/Users/Dzhok/new_project
dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok$ mkdir new_project
dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok$ cd new_project/
dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok/new_project$
```

После создания каталога перейдите в него.

Далее создаём копию Python внутри рабочего каталога одной из команд (Windows или Unix)

```
python -m venv venv - для Windows;
python3 -m venv venv - для Linux и MacOS.
```

Будет создана папка venv. Её мы указали в конце команды. Первый venv - вызов модуля для создания окружения. В указанную папку копируется локальная версия интерпретатора.


Остаётся активировать виртуальное окружение. Это финальная команда, которая отличается в Windows и Unix системах.

```
venv\Scripts\activate - для Windows;  
source venv/bin/activate - для Linux и MacOS.
```

Если в начале командной строки вы видите приставку (venv), значит активация прошла успешно. Можем работать над проектом в изолированной среде.


Сразу уточню, что для завершения работы внутри виртуального окружения необходимо выполнить команду

```
deactivate
```

 **Важно!** Виртуальное окружение созданное в одной ОС нельзя скопировать в другую ОС. Мультиплатформенность работает с кодом на языке Python. А виртуальные окружения, IDE и т.п. индивидуальны для каждой ОС и независимо устанавливаются в начала процесса разработки.

Работа с pip

Package Installer for Python — система управления пакетами, которая используется для установки и управления программными пакетами, написанными на Python. Благодаря pip мы можем устанавливать сторонние библиотеки, фреймворки, пакеты. Например через pip устанавливаются Django, Flask, NumPy.

 **Важно!** Прежде чем устанавливать дополнения убедитесь, что вы находитесь в каталоге проекта и активировали виртуальное окружение.

➤ Команда install

Для установки используем команду `install` далее указываем название устанавливаемого дополнения.

Потренируемся с установкой и добавим в проект библиотеку requests. Requests — это простая, но элегантная библиотека для работы с HTTP.

```
pip install requests
```

"Пип" скачает и установит библиотеку requests последней версии в папку venv виртуального окружения. В основной системе и в других окружениях библиотеки не будет.

➤ Команда freeze

Убедимся, что установка прошла успешно. Выполним команду

```
pip freeze
```

Мы получили список всех дополнений внутри нашего виртуального окружения. Помимо requests были скачаны библиотеки зависимости, необходимые для работы requests. Данный процесс автоматизирован. Зависимости указывает разработчик, а pip рекурсивно их устанавливает.

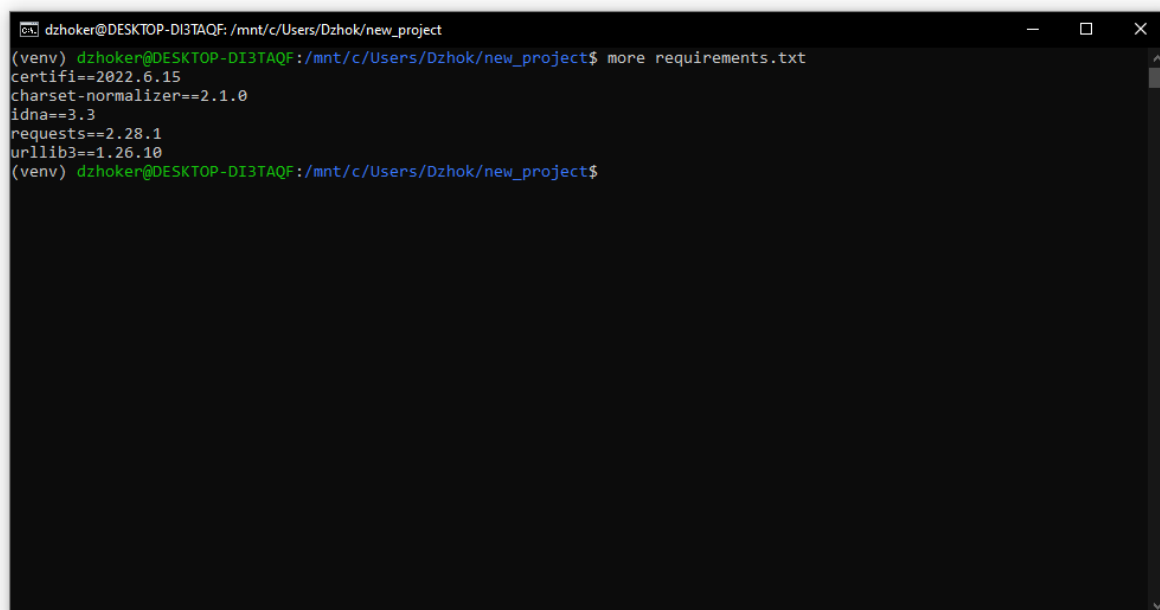
➤ Файл requirements.txt

Чтобы в будущем облегчить развертывание проекта на новом/боевом/тестовом сервере, используют файл requirements.txt. В него помещают перечень всех уже установленных дополнений. Для этого используют команду

```
pip freeze > requirements.txt
```

Откроем файл и посмотрим на его содержимое.

```
more requirements.txt
```

A screenshot of a terminal window with a dark background. The title bar shows the path /mnt/c/Users/Dzhok/new_project. The terminal shows the command (venv) dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok/new_project\$ more requirements.txt. The output lists several dependencies: certifi==2022.6.15, charset-normalizer==2.1.0, idna==3.3, requests==2.28.1, and urllib3==1.26.10. The prompt (venv) dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok/new_project\$ is visible at the bottom.

```
dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok/new_project$ more requirements.txt
certifi==2022.6.15
charset-normalizer==2.1.0
idna==3.3
requests==2.28.1
urllib3==1.26.10
(venv) dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok/new_project$
```

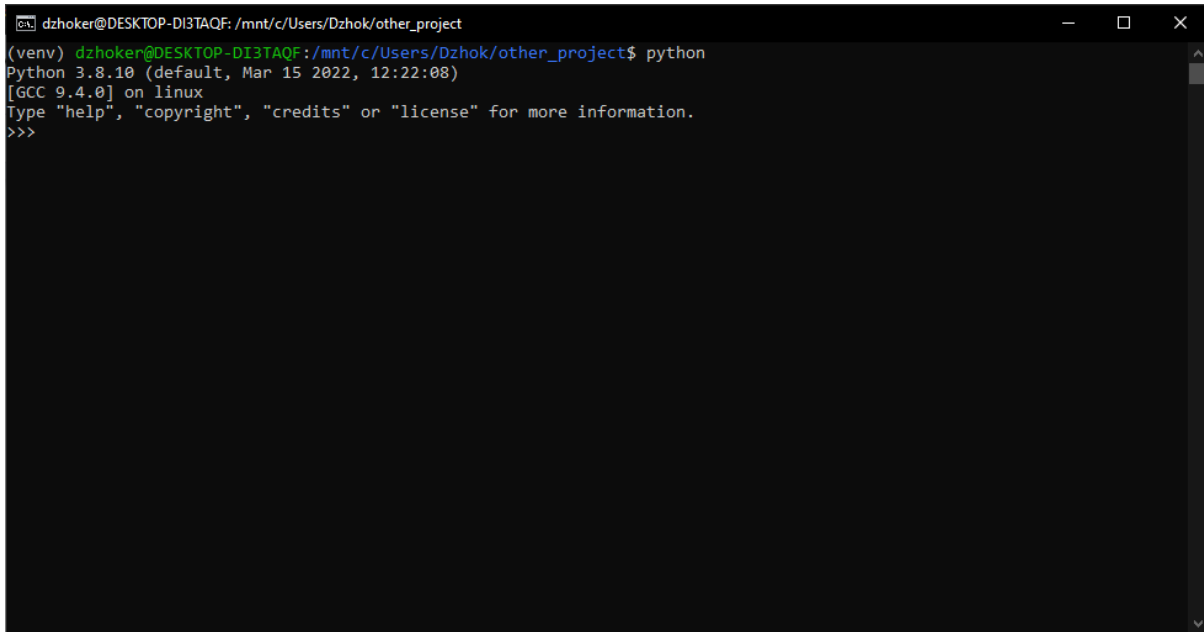
Для быстрой установки в новое окружение всего содержимого файла используется команда

```
pip install -r requirements.txt
```

➤ Отличия командной строки ОС и интерпретатора Python

Всё, что мы делали до этого делалось в командной строке операционной системы. Мы передавали ОС команды и дополнительные параметры и получали результат. Речь шла о Python, но команды получал и выполнял не он, а ОС.

Пришло время написать свою первую программу. И писать её мы будем в режиме интерпретатора. Выполним команду `python` (либо `python3`) чтобы активировать интерпретатор

A screenshot of a terminal window with a dark background. The window title is 'dzhoker@DESKTOP-DI3TAQF: /mnt/c/Users/Dzhok/other_project'. The command prompt shows '(venv) dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok/other_project\$ python'. The output of the command is: 'Python 3.8.10 (default, Mar 15 2022, 12:22:08)' followed by '[GCC 9.4.0] on linux' and 'Type "help", "copyright", "credits" or "license" for more information.' The prompt has changed to '>>>'.

```
dzhoker@DESKTOP-DI3TAQF: /mnt/c/Users/Dzhok/other_project
(venv) dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok/other_project$ python
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Обратите внимание на три треугольные стрелки. Теперь мы находимся в интерактивном сеансе Python. В нём не будут работать команды ОС, такие как `cd`, `pip` и другие.

Работа в режиме интерпретатора

Особенностью работы в режиме интерпретатора является то, что каждая строка кода выполняется сразу после нажатия клавиши Ввод. Есть и исключения, но о них через минуту. А пока воспользуемся оболочкой как калькулятором. Можно вводить любые математические операции и сразу получать ответ.

Использование `"_"`.

А если нам нужен последний вывод, можно воспользоваться переменной `"_"`. Она хранит именно финальный результат, т.е. обновляется с каждым новым выводом.

```
dzhoker@DESKTOP-DI3TAQF: /mnt/c/Users/Dzhok/other_project
(venv) dzhoker@DESKTOP-DI3TAQF:/mnt/c/Users/Dzhok/other_project$ python
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> 8*6
48
>>> 4+3*8-(2+4)
22
>>> _ + 2
24
>>>
```

➤ Арифметические операторы в Python

Основные математические операторы представлены в таблице.

Оператор	Описание	Примеры
+	Сложение	$398 + 20 = 418$
-	Вычитание	$200 - 50 = 150$
*	Умножение	$34 * 7 = 238$
/	Деление	$36 / 6 = 6.0$ $36 / 5 = 7.2$
//	Целочисленное деление	$36 // 6 = 6$ $9 // 4 = 2$ $-9 // 4 = -3$ $15 // -2 = -3$
%	Остаток от деления	$36 \% 6 = 0$ $36 \% 5 = 1$
**	Возведение в степень	$2 ** 16 = 65536$

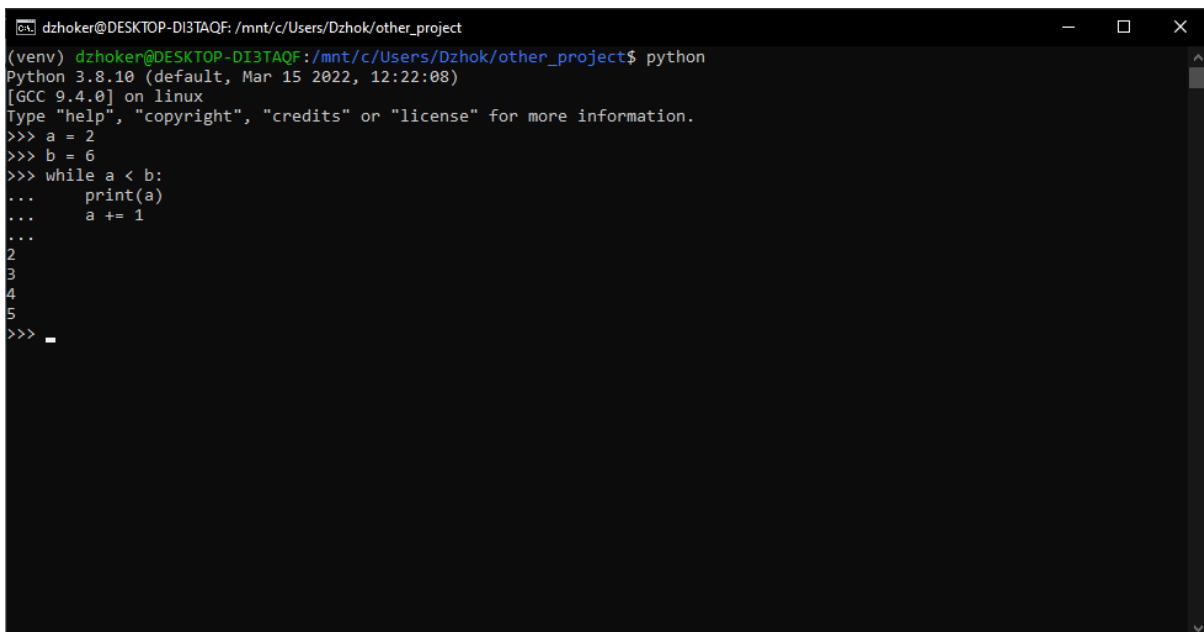
Обратите внимание на операцию целочисленного деления с участием отрицательных чисел в качестве делимого или делителя. Такой результат обусловлен тем, что целочисленное деление в Python округляет итоговое значение в меньшую сторону.

```
9 / 4 = 2.25, округляем до меньшего целого - это 2
-9 / 4 = -2.25, округляем до меньшего целого - это -3
```

➤ Написание кода в интерактивном режиме

Мы можем писать программы любой сложности, использовать переменные, ветвления, циклы внутри оболочки. При этом наличие двоеточия позволяет писать несколько строк кода без вывода результата. Python запоминает целый блок и выводит ответ после нажатия клавиши Ввод дважды.

```
>>> a = 2
>>> b = 6
>>> while a < b:
...     print(a)
...     a += 1
...
2
3
4
5
```



```
dzhoker@DESKTOP-DI3TAQF: /mnt/c/Users/Dzhok/other_project
(venv) dzhoker@DESKTOP-DI3TAQF: /mnt/c/Users/Dzhok/other_project$ python
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> b = 6
>>> while a < b:
...     print(a)
...     a += 1
...
2
3
4
5
>>> _
```

У оболочки есть один минус. После выхода мы теряем все введённые данные. Поэтому для написания большого кода используются файлы с расширением `.py`. Далее по курсу мы будем использовать их.

А пока завершим интерактивный сеанс командой

```
exit()
```

Задание

Проверим усвоение материала. Вот несколько простых вопросов. Для ответа на каждый вам даётся одна минута. Ответы напишите в чат.

1. Какая версия Python у вас установлена? Если не установлена, какую версию будете устанавливать?
2. Запомнили ли вы имя файла, в котором сохраняется перечень всех необходимых дополнений для вашего проекта. Как он называется?
3. Как получить доступ к результату последней операции в режиме интерпретатора Python?

2. Повторяем основы Python

Пришло время поговорить о базовых вещах в программировании применительно к языку Python.

Работа в IDE

Python-программы можно писать в любом текстовом редакторе, главное — сохранять текст программы в кодировке UTF-8. Свой код буду демонстрировать в IDE PyCharm. Версия Community бесплатна. Код сохраняется в файлах с расширением `py` и может быть многократно использован. Однако вы можете писать код в любой другой IDE, к которой привыкли. Стоит вспомнить, что Python является мультиплатформенным языком программирования, а значит разные операционные системы будут одинаково выполнять код. Аналогично и IDE являются лишь оберткой, которая помогает писать код разработчику, но никак не влияет на результаты выполнения Python программ. Используйте ту IDE в которой вам удобно работать.



Важно! Не давайте именам питоновских файлов имена встроенных и внешних модулей. Это приведёт к затиранию имён и отключению функционала модулей.

Python Style. PEP-8 (руководство по стилю) и PEP-257 (оформление документации/комментариев)

Одно из важнейших требований к коду Python-разработчика — следование стандарту PEP-8, который представляет собой описание рекомендованного стиля кода. Причем PEP-8 действует для основного текста программы, а для строк документации разработчику рекомендуется придерживаться положений PEP-257. Документ содержит достаточно объёмное описание стандарта. В процессе этого курса мы будем знакомиться с различными пунктами из этих "пеп". Привыкайте к верному стилю с самого начала. Ведь фразу "Превед, каг твай дила" прочитать можно. Но доверять такому автору написание шедевра уже не хочется. Далее на протяжении курса буду упоминать о правилах по написанию читаемого кода, уточняя что это "пеп8".

Переменные и требования к именам

В Python всё объект. Числа, строки, массивы и даже функции и классы являются объектом. Переменная в Python является указателем на объект. Разберём на примере.

```
a = 5
```

Переменная "a" хранит значение "5". Верно. Также верно, что Python создал объект целого типа 5 (о типах данных мы поговорим на следующих лекциях). Далее была создана переменная "a" которая является указателем на объект целого типа число пять.

Python является языком со строгой динамической типизацией. Это означает что тип объекта изменить невозможно он строго задаётся при создании объекта. При этом переменные могут ссылаться на объекты разных типов. И это не вызывает ошибки.

```
a = 5
a = "hello world"
a = 42.0 * 3.141592 / 2.71828
```



PEP-8! Обратите внимание на отступы в один пробел вокруг математических знаков "=", "*" и др. Это требование по оформлению кода, которое повышает его читаемость.



Важно! Использование одной и той же переменной для разных типов данных является плохим стилем. Предварительная инициализация переменных в языке не нужна. Вы можете использовать переменную в том месте где оно понадобилось. Даже если уже написали несколько сотен строк кода.

Что ещё нужно знать когда речь идёт о переменных и их именах:

- Python использует кодировку utf-8. Поэтому в качестве имени переменной может выступать любой набор символов, даже смайлик. Однако правильные имена это имена на латинице, т.е. английские строчные буквы, слова. Например name, age.
- Если название переменной состоит из нескольких слов такие слова записываются строчными буквами разделяются символом подчёркивания. Этот стиль называется snake_case.
- В качестве первого символа в имени переменной запрещено использовать цифры и другие знаки пунктуации в этом случае Python выдаст ошибку. Имя начинается с буквы или символа подчёркивания.
- Не используйте написание слов на транслите. Воспользуйтесь онлайн переводчиком на английский. Не zdorove, а health.

Верно	Не верно
first_name, user_1, request, _tmp_name, min_step_shift	1_name, User_1, Orequest, tmpName, minStep_shift, 😊, имя

Константы

Дополнительных команд для создания констант в языке Python нет. Есть лишь договорённость что константа — это переменная написанное прописными буквами.

Примеры констант.

```
MAX_COUNT = 1000
ZERO = 0
DATA_AFTER_DELETE = 'No data'
DAY = 60 * 60 * 24
```

Константа в программировании — способ адресации данных, изменение которых программой не предполагается или запрещается. Python не вызовет исключение, если вы измените константу внутри кода. Но подобные действия со стороны программиста являются неверными.

True, False, None

Отдельно хочу выделить три встроенные в язык Python константы. Это истина True, ложь False и ничего None. Первая буква строчная остальные прописные. Истину или ложь мы получаем в результате логических операций. О них мы поговорим чуть позже. Значение None означает "ничего". Его можно использовать для создания переменной, значение которой изначально мы не знаем. Также None возвращают некоторые функции, результат работы которых не подразумевает возврат значения.

Функция id()

Вернёмся к тому что в Python всё объект, а переменная является ссылкой на объект. Воспользуемся встроенной функцией id(), которая возвращает адрес объекта в оперативной памяти вашего компьютера.

```
a = 5
print(id(a))
a = "hello world"
print(id(a))
a = 42.0 * 3.141592 / 2.71828
print(id(a))
```

В отличие от режима интерпретатора, запуск программных файлов не выводит значения на экран. Поэтому мы используем функцию print(), внутри которой передаём вызов функции id(a). Как вы видите одна и та же переменная возвращает три разных адреса для трёх разных объектов в памяти Python.


```
1615821406576
1615826697776
1615831485936
```

Зарезервированные слова, keyword.kwlist

Существует чуть менее 40 зарезервированных слов, которые образуют базовый синтаксис языка Python. Ниже представлены они все.

```
False, None, True, and, as, assert, async, await, break, class,
continue, def, del, elif, else, except, finally, for, from,
global, if, import, in, is, lambda, nonlocal, not, or, pass,
raise, return, try, while, with, yield.
```

А также case и match начиная с версии Python 3.10.

Первую Троицу из списка мы разобрали, когда говорили о встроенных в Python константах. О остальных зарезервированных словах мы будем говорить далее на протяжении всего курса. Уверен, что некоторые слова вам знакомы по прошлым курсам и другим языкам программирования.



Важно! Запрещено использовать в качестве имён переменных зарезервированные слова. Python завершит код с ошибкой.

Ввод и вывод данных

Для ввода и вывода данных в консоль поэтому используются стандартные потоки ввода-вывода. Однако программист не работает с ними напрямую, а использует встроенные в язык функции `print()` для вывода данных и `input()` для ввода данных и сохранение в переменные.

➤ Вывод, функция `print()`

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Функция `print()` принимает один или несколько объектов разделённых запятыми и выводит их на печать. По умолчанию это вывод в консоль, т.е. стандартный поток вывода.

```
print(42)
print(1, 2, 3, 4)
print('Hello', ',', 'world', '!')
```

Вывод:

```
42
1 2 3 4
Hello , world !
```

Отдельно уточню про два ключевых аргумента `sep` и `end`.

`sep` по умолчанию хранит один пробел. Именно этим символом разделяются все объекты, перечисленные через запятую.

`end` по умолчанию хранит символ перехода на новую строку `'\n'`. Это то, что функция `print` добавляет после вывода всех объектов.

```
print(42, sep='___', end='\n(=^.^=)\n')
print(1, 2, 3, 4, sep='___', end='\n(=^.^=)\n')
print('Hello', ',', 'world', '!', sep='___', end='\n(=^.^=)\n')
```

Вывод:

```
42
(=^.^=)
1___2___3___4
(=^.^=)
Hello___,___world___!
(=^.^=)
```

Обратите внимание, что текст, заключённый в одинарные или двойные кавычки, выводится без изменения. Python воспринимает его как строковую информацию. Для вывода содержимого переменных мы указываем имя переменной без кавычек.

```
number = 42
print(number, sep='___', end='\n(=^.^=)\n')
ONE = 1
TWO = 2
print(ONE, TWO, 3, 4, sep='> <', end='>')
```

Вывод:

```
42
```

```
(=^ . ^=)
1> <2> <3> <4>
```

➤ Ввод, функция input()

Для ввода данных и сохранение их в переменной используется функция input().

```
result = input([prompt])
```

Если в качестве аргумента функции передать значение, оно будет выведено как подсказка перед вводом данных. Функция возвращает объект строкового типа, который можно сохранить в переменную.

```
name = input('Ваше имя: ')
```

Обратите внимание на следующий пример.

```
age = input('Ваш возраст: ')
```

В переменной age будет сохранено значение в строковом формате. Посчитать сколько лет назад человеку стукнуло 18 не получится.

```
how_old = age - 18
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Для приведения строки к числу используем функции int() — целый тип или float() — вещественный тип, число с плавающей запятой. Подробнее о них поговорим на следующей лекции.

```
age = float(input('Ваш возраст: '))
how_old = age - 18
print(how_old, "лет назад ты стал совершеннолетним")
```

Вывод:

```
Ваш возраст: 33
15.0 лет назад ты стал совершеннолетним
```



Важно! Если попытаться преобразовать к числу не число, получим ошибку.

```
ValueError: could not convert string to float: 'пять'
```

Способы обработки пользовательского ввода и обработка ошибок — тема одной из следующих лекций. Пока договоримся об идеальном пользователе, который вводит числа там где это необходимо и не создаёт ошибок.

Антипаттерн "магические числа"

В прошлом примере мы использовали антипаттерн — плохой стиль для написания кода. Число 18 используется в коде без пояснений. Такой антипаттерн называется "магическое число". Рекомендуется помещать числа в константы, которые хранятся в начале файла.

```
ADULT = 18
age = float(input('Ваш возраст: '))
how_old = age - ADULT
print(how_old, "лет назад ты стал совершеннолетним")
```

Плюсом такого подхода является возможность легко корректировать большие проекты. Представьте, что в вашем коде несколько тысяч строк, а число 18 использовалось несколько десятков раз.

- При развертывании проекта в стране, где совершеннолетием считается 21 год вы будете перечитывать весь код в поисках магических "18" и править их на "21". В случае с константой изменить число нужно в одном месте.
- Дополнительный сложности могут возникнуть, если в коде будет 18 как возраст совершеннолетия и 18 как коэффициент для расчёт чего-либо. Теперь править кода ещё сложнее, ведь возраст изменился, а коэффициент -нет. В случае с сохранением значений в константы мы снова меняем число в одном месте.

Задание

Очередная серия из трёх вопросов. По две минуты на ответ. Его вы пишете в чат.

1. Какое из слов лишнее и что означают остальные слова: True, NaN, False, None?
2. Какие имена переменных плохие, а какие правильные и почему: name, 1_year, _hello, id, MAX_DATE, zdorove?
3. Что вы можете рассказать про магические числа?

3. Ветвление

Ветвление в программировании — операция, применяющаяся в случаях, когда выполнение или невыполнение некоторого набора команд должно зависеть от выполнения или невыполнения некоторого условия. Ветвление — одна из трёх базовых конструкций структурного программирования.

"Некоторое условие" должно возвращать истину True или ложь False. В Python есть возможность использовать неявное преобразование типов. Так любое целое число помимо нуля считается истиной, а ноль — ложью. Любая коллекция с данными — истина, а пустая коллекция — ложь. Подробнее хорошие приёмы неявного преобразования мы рассмотрим далее на курсе. А пока стоит запомнить, что явное лучше неявного.

Если, if

Простейшая проверка условия происходит при помощи зарезервированного слова if

```
pwd = 'text'
res = input('Input password: ')
if res == pwd:
    print('Доступ разрешён')
```

В строке 3 мы используем if, далее пишем выражение, которое должно вернуть истину или ложь и завершаем строку двоеточием.

В Python доступны все 6 операций сравнения:

«==» — равно	«!=» — не равно
«>» — больше	«<=» — меньше или равно
«<» — меньше	«>=» — больше или равно

Отступы вместо фигурных скобок

Обратите внимание на отступ в четыре пробела в следующей после if строке кода. В отличие от Си-подобных языков программирования? таких как Java, C# и других, Python не используют фигурные скобки для создания логических блоков кода.

Вместо этого мы создаём отступы из четырёх пробелов. Такой стиль повышает читаемость кода и сокращает количество ошибок.

```
pwd = 'text'
res = input('Input password: ')
if res == pwd:
    print('Доступ разрешён')
    print('Но будьте осторожны')
print('Работа завершена')
```

Строки 4-5 написаны с отступом и будут выполнены в случае истинности в строке 3. Строка 6 будет выполнена в любом случае, т.к. написана без отступов.



PEP-8! рекомендует использовать в качестве отступа 4 пробела. Программа будет верно работать и с 2 и с 8 отступами, и даже с символом табуляции (при условии что во всём коде использован единый стиль отступания). Но программы проверки кода — линтеры будут выдавать предупреждения, а коллеги по проекту подзатыльники.

Также обратите внимание, что IDE при верной настройке заменяют нажатие клавиши TAB на клавиатуре на ввод 4 пробелов.

Обратите внимание на отсутствие скобок в строке 3. Python не требует заключать логическое выражение в скобки. А IDE с проверкой синтаксиса будет ругаться на скобки, как на рудимент. Единственный случай, когда вам могут понадобиться скобки, построение сложных логических выражений, где надо поменять порядок логических проверок с традиционного слева направо, на другой.

Иначе, else

Для выполнения кода в случае ложности логического выражения используется зарезервированное слово `else` с обязательным двоеточием после него.

```
pwd = 'text'
res = input('Input password: ')
if res == pwd:
    print('Доступ разрешён')
    print('Но будьте осторожны')
else:
    print('Доступ запрещён')
print('Работа завершена')
```

Слово `else` относится к тому `if`, с которым находится на одном уровне. В примере ниже верхний `if` связан с нижним `else`, а средний `if` со средним `else`.

```
pwd = 'text'
res = input('Input password: ')
if res == pwd:
    print('Доступ разрешён')
    my_math = int(input('2 + 2 = '))
    if 2 + 2 == my_math:
        print('Вы в нормальном мире')
    else:
        print('Но будьте осторожны')
else:
    print('Доступ запрещён')
print('Работа завершена')
```

Еще если, `elif`

Для проверки нескольких выражений используется `elif` - сокращение от `else if`.

```
color = input('Твой любимый цвет: ')
if color == 'красный':
    print('Любитель яркого')
elif color == 'зелёный':
    print('Ты не охотник?')
elif color == 'синий':
    print('Ха, классика!')
else:
    print('Тебя не понять')
```

Проверка работает до первого совпадения. После него дальнейший код пропускается. А если совпадений нет, срабатывает код после `else`.

Выбор из вариантов, `match` и `case`

В Python версии 3.10, т.е. совсем недавно появилась новая возможность множественного сравнения. Это конструкция `match` и `case`. После `match` указываем

переменную для сравнения. Далее идёт блок из множества case с вариантами сравнения. Рассмотрим работу кода на примере.



Важно! Если у вас стоит Python версии 3.9 и ниже, код не будет работать.

```
color = input('Твой любимый цвет: ')
match color:
    case 'красный' | 'оранжевый':
        print('Любитель яркого')
    case 'зелёный':
        print('Ты не охотник?')
    case 'синий' | 'голубой':
        print('Ха, классика!')
    case _:
        print('Тебя не понять')
```

Данный код аналогичен прошлому варианту с elif. Добавлена возможность проверить несколько цветов. Например для красного и оранжевого будет один вывод. Вертикальная черта играет роль оператора "или". Уточню что пользователь вводит один единственный цвет.

Вместо слова else в данной конструкции используется сочетание case _ На этом курсе мы ещё несколько раз будет встречаться с подчеркиванием. И каждый раз он имеет разные эффект в зависимости от применения.

Логические конструкции, or, and, not

В Python доступны три логических оператора:

- and — логическое умножение «И»;
- or — логическое сложение «ИЛИ»;
- not — логическое отрицание «НЕ».

Логика их работы представлена в таблице

first	second	first and second	first or second	not first
True	True	True	True	False
False	True	False	True	True

True	False	False	True	-
False	False	False	False	-

А теперь пример кода на Python чтобы разобраться в правильном синтаксисе построения логических выражений. Вычислим високосный год в Григорианском календаре поэтапно:

```
year = int(input('Введите год в формате yyyy: '))
if year % 4 != 0:
    print("Обычный")
elif year % 100 == 0:
    if year % 400 == 0:
        print("Високосный")
    else:
        print("Обычный")
else:
    print("Високосный")
```

А теперь выберем все случаи, когда год обычный и запишем их в одну строку:

```
if year % 4 != 0 or year % 100 == 0 and year % 400 != 0:
    print("Обычный")
else:
    print("Високосный")
```

Python последовательно слева направо проверяет логическое выражение, формируя финальный ответ — True или False.

Ленивый if

Ещё раз посмотрим на прошлый пример кода.

```
if year % 4 != 0 or year % 100 == 0 and year % 400 != 0:
```

В Python как и в некоторых других языках программирования if "ленивый". Если в логическом выражении есть оператор `or` и первое значение то есть левое вернуло истину, дальнейшая проверка не происходит, возвращается True. Если в логическом выражении есть оператор `and` и левая половина вернула ложь, то возвращается False без проверки правой половины выражения.

Проверка на вхождение, in

На одном из следующих уроках поговорим о коллекциях. Пока для простоты договоримся, что список целых чисел в квадратных скобках — массив данных. Оператор `in` проверяет вхождение элемента в последовательность.

```
data = [0, 1, 1, 2, 3, 5, 8, 13, 21]
num = int(input('Введи число: '))
if num in data:
    print('Леонардо передаёт привет!')
```

А теперь тот же самый код, но с конструкцией `not` — отрицание:

```
data = [0, 1, 1, 2, 3, 5, 8, 13, 21]
num = int(input('Введи число: '))
if num not in data:
    print('Леонардо грустит :-(')
```

Обратите внимание на порядок слов в строке 3. Python использует максимально приближенные к человеческому, а точнее к английскому языку стиль программирования: "Если число не входит в данные, то..."

Тернарный оператор

Завершим разговор о ветвлениях тернарным оператором. Он позволяет записать 4 логические строки в одну.

Было:

```
my_math = int(input('2 + 2 = '))
if 2 + 2 == my_math:
    print('Верно!')
else:
    print('Вы уверены?')
```

Стало:

```
my_math = int(input('2 + 2 = '))
print('Верно!' if 2 + 2 == my_math else 'Вы уверены?')
```

Слева от if записываем выражение для истины, справа от else результат для лжи.



Важно! При использовании тернарного оператора важно помнить, что код должен сохранить свою читаемость. Слишком длинные и сложные конструкции стоит переписать в более привычном виде в 4 строки.



PEP-8! требует, чтобы длина строки кода не превышала 80 символов. Современные мониторы с широкими экранами и высоким разрешением с легкостью вмещают больше символов. Поэтому ограничение было пересмотрено и увеличено до 120 символов. Более длинные строки кода в Python недопустимы, т.к. снижают его читаемость.

Задание

1. Перед вами пример кода. Значения переменных `num` и `name` указаны в первых строчках. Попробуйте прочесть код и выбрать верный вариант без запуска кода. У вас две минуты на каждую пару значений.

```
num = 42
name = 'Bob'
if num > 30:
    if num < 50:
        print('Вариант 1')
    elif name > 'Markus':
        print('Вариант 2')
    else:
        print('Вариант 3')
elif name < 'Markus':
    print('Вариант 4')
elif num != 42:
    print('Вариант 5')
else:
    print('Вариант 6')
```

2. А что получится теперь:

```
num = 64
name = 'Bob'
```

3. И финальные переменные. Какой вариант будет теперь:

```
num = 7
name = 'Neo'
```

4. Циклы

Рассмотрим способы создания циклов в Python.



Важно! Мы снова будем использовать отступы в 4 пробела. Это неотъемлемая часть синтаксиса языка.

Логический цикл while

Цикл while является циклом с предусловием. Проверяем логическое условие, аналогично "если" и в случае истинности выполняем вложенный блок кода. Далее возвращаемся к проверке условия.

Попробуем перебрать всё чётные числа от нуля до введенного пользователем исключительно..

```
num = float(input('Введите число: '))
count = 0
while count < num:
    print(count)
    count += 2
```

Проверка условия после while и выполнение тела цикла продолжается до тех пор, пока условие истинно.

Синтаксический сахар

Обратите внимание на нижнюю строку кода. "+=" - синтаксический сахар Python.

Синтаксический сахар (англ. syntactic sugar) в языке программирования — это синтаксические возможности, применение которых не влияет на поведение программы, но делает использование языка более удобным для человека.

Для неизменяемых типов данных, а числа в Python неизменяемы, две следующие строки кода эквивалентны.

```
num = num + 1
num += 1
num++    # не работает в Python
```

Аналогично можно записать коротко вычитание, умножение, целочисленное деление и другие операции.

Возврат в начало цикла, continue

При необходимости работу цикла можно прервать и досрочно вернуться к проверке условия. Для этого используем зарезервированное слово `continue`.

Выведем все чётные числа (как в прошлом примере), кроме тех, которые кратны 12.

```
num = float(input('Введите число: '))
STEP = 2
limit = num - STEP
count = -STEP
while count < limit:
    count += STEP
    if count % 12 == 0:
        continue
    print(count)
```

`if` внутри цикла проверяет кратность двенадцати. В случае истинности команда `continue` возвращает нас к началу цикла, к `while`.

И пара слов о двух оптимизациях в коде:

1. `STEP = 2` — добавили константу для движения по чётным и ушли от "магических чисел". Теперь изменение условия на "вывод чисел кратных 5" потребует замены числа в одном месте кода.
2. Ввели переменную `limit`. Чтобы цикл не выводил лишние числа, больше введенного `num`, на каждой проверке цикла `while` надо вычитать значение шага. Но шаг — константа. Для экономии ресурсов ПК и ускорения работы

кода логично сделать вычитание один раз перед циклом и сравнивать значения быстрее, без вычитания в строке while

Досрочное завершение цикла, break

Ещё один способ управления циклом — команда break для его досрочного завершения. Она отлично подходит для создания циклов с постусловием, бесконечных циклов с возможностью выхода.

Рассмотрим на примере программы, которая просит ввести число внутри заданного диапазона.

```
min_limit = 0
max_limit = 10
while True:
    print('Введи число между', min_limit, 'и', max_limit, '? ')
    num = float(input())
    if num < min_limit or num > max_limit:
        print('Неверно')
    else:
        break
print('Было введено число ' + str(num))
```

Конструкция while True: создаёт бесконечный цикл. Вместо True можно было бы подставить любое выражение, которое всегда возвращает истину. Но именно такая реализация обеспечивает лучшую читаемость и быстродействие.

Действие после цикла, else

Зарезервированное слово else может применяться не только к ветвлениям, но и к циклам. Для этого else должно быть расположено на том же уровне, т.е. иметь столько же пробельных отступов, что и начало цикла — while.

Доработаем прошлую программу и дадим 3 попытки на попадание в диапазон.

```
min_limit = 0
max_limit = 10
count = 3

while count > 0:
```

```

print('Попытка ' + str(count))
count -= 1

    num = float(input('Введи число между ' + str(min_limit) + ' и ' + str(max_limit) + ': '))
    if num < min_limit or num > max_limit:
        print('Неверно')
    else:
        break

else:
    print('Исчерпаны все попытки. Сожалею.')
    quit()

print('Было введено число ' + str(num))

```

Что изменилось в коде:

- Добавили счётчик count. Цикл проверяет не уменьшился ли счётчик до нуля. Если нет, переходим в тело цикла. Выводим значение счётчика на каждом витке цикла и уменьшает его на единицу.
- Добавили else для while. Логика следующая. Если пользователь ввёл верное число и сработала команда break, блок else для цикла игнорируется. А если числа вводились ошибочно, счётчик досчитал до нуля и произошел выход из цикла по "лжи" в while. В этом случае сработает блок кода после else для цикла.
Если коротко, вызов break в цикле игнорирует else для цикла.
- quit() — ещё одна функция для завершения кода раньше, чем мы дойдём до конца файла. Это вторая функция. Первой, exit() мы пользовались для завершения работы сеанса интерпретатора. Работают они аналогично.
- Мы добавили пустые строки в код. Python игнорирует такие строки. Но разделение кода на блоки по смыслу упрощает чтение. Кроме того, на следующих лекциях мы поговорим о пустых строках, которые прописаны в PEP-8. Но не ставьте пустые строки после каждой строки кода. Если вам нравится большой межстрочный интервал, настройте его в своей IDE.
- **PEP-8!** Кстати, последняя строка в файле должны быть пустой. Именно одна, а не ноль, и не две.

Цикл итератор for in

Цикл for in используется Python разработчиками намного чаще. Прежде чем приступить к его рассмотрению уточню, что команды continue, break и else могут применяться к циклу for in точно так же как и до этого в цикле while.

Рассмотрим работу цикла в качестве итератора последовательности. Итерироваться будем по массиву с числами из части про ветвления и проверку на вхождение.

```
data = [0, 1, 1, 2, 3, 5, 8, 13, 21]
for item in data:
    print(item)
```

Цикл последовательно перебирает все элементы массива data и поочерёдно помещает их в переменную item. После for указываем переменную или переменные для приема значений, они изменяются на каждом витке цикла. После in указываем объект, из которого последовательно берём данные.



Важно! Нельзя изменять содержимое контейнера (в нашем примере data) во время итерации по нему, т.е. внутри цикла. Это приведет к неожиданным ошибкам.

Цикл по целым числам, он же арифметический цикл, функция range()

Для перебора целых чисел цикл for in используется в связке с функцией range(). Она выступает в качестве объекта итератора. Пример печати чётных чисел от нуля до введённого числа может выглядеть так с циклом for:

```
num = int(input('Введите число: '))
for i in range(0, num, 2):
    print(i)
```



Важно! Аргументами функции range() должны быть целые числа, int()

Рассмотрим подробнее варианты работы функции `range()`. Обратите внимание на количество переданных функции аргументов

`range(stop)` — перебираем значения от нуля до `stop` исключительно с шагом один

`range(start, stop)` — перебираем значения от `start` включительно до `stop` исключительно с шагом один

`range(start, stop, step)` — перебираем значения от `start` включительно до `stop` исключительно с шагом `step`.

Пара если.

- Если значение `step` отрицательное, перебор будет в сторону уменьшения.
- Если `start` больше `stop` при положительном `step` или наоборот `start` меньше `stop` при отрицательном `step`, цикл не сработает ни разу. `range(10, 5, 2)` - ничего

Имена переменных в цикле

Исторически сложилось, что для переменных, которые "считают арифметику" в цикле используются переменные `i`, `j`, `k`. Именно в таком порядке с учётом вложенности. Этой традиции более полувека. Так что смело продолжайте её и ваш код поймут. Например так выглядят 3 вложенных цикла. Обратите внимание на отступы, которые показывают уровень вложенности

```
count = 10
for i in range(count):
    for j in range(count):
        for k in range(count):
            print(i, j, k)
```

А так выглядят два последовательных цикла

```
count = 10
for i in range(count):
    print(i)

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

Однако, когда мы перебираем какие-то данные, вместо однобуквенных переменных можно использовать подходящие имена. Например так может выглядеть перебор животных, которые хранятся в массиве данных.

```
animals = ['cat', 'dog', 'wolf', 'rat', 'dragon']
for animal in animals:
    print(animal)
```

Как вы помните читаемость имеет значение.

```
имеем массив животных
для каждого животного в списке животных
    печатаем животное
```

Цикл с нумерацией элементов, функция `enumerate()`

В финале рассмотрим ещё одну функцию, `enumerate()`. Она позволяет добавить порядковый номер к элементам итерируемой последовательности. Доработаем пример с животными. Будем выводить порядковый номер перед указанием животного.

```
animals = ['cat', 'dog', 'wolf', 'rat', 'dragon']
for i, animal in enumerate(animals, start=1):
    print(i, animal)
```

Что изменилось?

- После `for` указано две переменные через запятую. В `i` будет помещаться порядковый номер. В `animal` очередное животное из списка.
- Функция `enumerate()` получила в качестве первого аргумента список животных. Второй аргумент — стартовое значение счётчика, т.е. первое значение, которое попадёт в `i`.

Если второй аргумент не передать, нумерация начнётся с нуля.



Важно! Функция `enumerate` позволяет перебирать только целые числа в порядке возрастания с шагом один.

Задание

Финальное задание с циклами. Перед вами пример программы. Попробуйте мысленно пройти по коду и напишите в чат финальное значение переменной `data`. У вас 2 минуты на каждый вариант кода. Запускать программу не нужно.

1. Вариант кода 1.

```
data = 0
while data < 100:
    data += 2
    if data % 40 == 0:
        break
print(data)
```

2. Вариант кода 2.

```
data = 0
while data < 100:
    data += 3
    if data % 40 == 0:
        break
else:
    data += 5
print(data)
```

3. Вариант кода 3.

```
data = 0
while data < 100:
    data += 3
    if data % 19 == 0:
        continue
    data += 1
    if data % 40 == 0:
        break
else:
    data += 5
print(data)
```

5. Выводы

1. На этой лекции мы разобрали установку и настройку Python. Были изучены правила создания виртуального окружения и работу с pip.
2. Мы повторили основы синтаксиса языка Python. Познакомились с рекомендациями по оформлению кода.

3. Изучили способы создания ветвящихся алгоритмов на Python. Разобрались с разными вариантами реализации циклов.

Краткий анонс следующей лекции

1. Познакомимся со строгой динамической типизацией языка Python.
2. Изучим понятие объекта в Python. Разберёмся с атрибутами и методами объектов.
3. Рассмотрим способы аннотации типов.
4. Изучим "простые" типы данных, такие как числа и строки.
5. Узнаем про математические модули Python.

Домашние задания

1. Установите Python на ваш ПК. Убедитесь, что можете работать как в режиме интерпретатора, так и запускать код в файлах `.py`.
2. Настройте IDE, в которой планируете выполнять задания на семинарах и практические работы. Проверьте возможность запуска программ средствами IDE.