**Введение в Python и его установка**

Надо будет поставить в VS code python.

Низкоуровневые языки работают непосредственно с аппаратной частью компьютера, поэтому скорость их работы, в отличии от высокоуровневых, выше.

**Процессор** – различает только два сигнала. Принято говорить о них как 1 и 0. Но по факту сигнал либо есть либо нет. Процессор работает только с командами состоящими из нулей и единиц – **машинным кодом.**

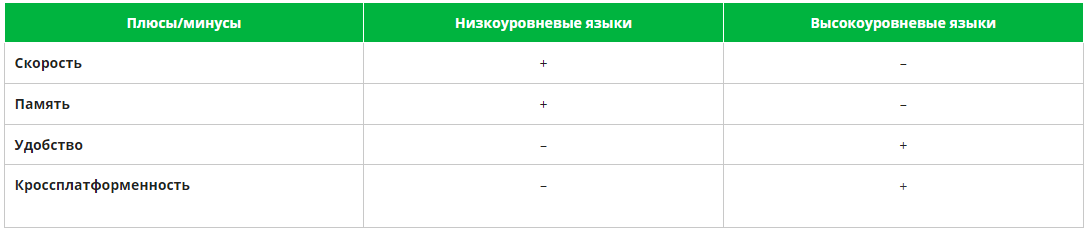
Низкоуровневые языки уже состоят из набора таких команд, поэтому эти команды и понятны компьютеру.

**Но эти языки не являются кроссплатформенными.** Ну то есть машинный код на **windows,** не получится просто запустить на **Linux.** Связано с архитектурой этих платформ.

**Высокоуровневые языки** наоборот могут запускать код не зависимо от операционной системы. И синтаксис этих языков более понятный.

**Python** кстати к синтаксису довольно строг. Особенно к табуляции.

Также программы написанные на высокоуровневых языках, требуют для своей работы и хранения значительно больше памяти, чем низкоуровневые.



**Ещё языки программирования делятся на компилируемые и интерпретируемые.**

В **компилируемых** языках код программы преобразуется в машинный код с помощью компилятора. На выходе получается новая скомпилированная программа, которую выполнит компьютер.

**Интерпретатор** принимает удобный для восприятия человеком код и начинает построчно исполнять его, преобразуя каждую строку в машинный код. Новой программы по итогу не случится.

**JS** интерпретируемый, как и **python** и **PHP.**

**C++, C#, Java, Golang –** компилируемые.

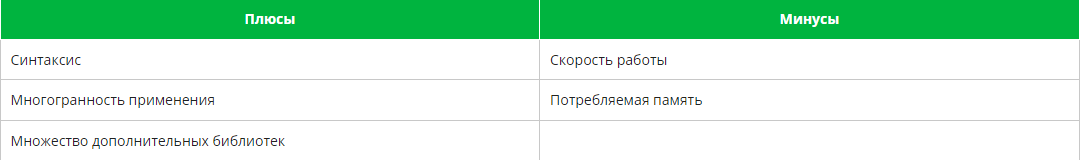
**Версии Python**

В январе 2020 года прекратилась поддержка **Python 2.7.18**

Сейчас самая актуальная версия это **Python 3.** Но и среди них есть устаревшие версии.

**Python 3.4 включительно и старше постепенно перестают поддерживаться разработчиками**

**Актуальные:** любая из версий 3.5, 3.6, 3.7, 3.8. Постепенно ранние версии также прекращают свою поддержку, поэтому старайтесь пользоваться наиболее поздней из доступных версий. В данном курсе будем пользоваться **Python 3.8.**



Ну зато честно.

**А вот и нет. Хотя ладно. Python 3.10.6 и 3.8 Скачаю оба. А на ноутбуке у меня вообще 3.9 еще стоит.**

Если что-то будет не работать, просто включу другой интерпретатор.

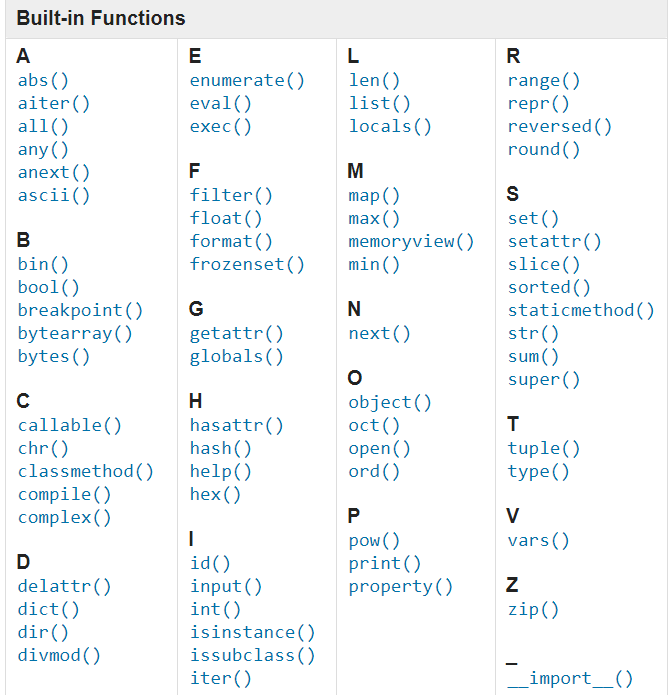
**Как работать с документацией по Python**

Тупо ссылки

* [Модули для работы с текстовым форматом данных](https://docs.python.org/3/library/text.html). Например, строки, регулярные выражения.
* [Расширенные типы данных](https://docs.python.org/3/library/datatypes.html). Позволяют работать с датами, очередями, массивами.
* [Модули для работы с числами](https://docs.python.org/3/library/numeric.html). Модули с расширенными математическими операциями.
* [Взаимодействие с файловой системой](https://docs.python.org/3/library/filesys.html).
* [Работа с операционной системой](https://docs.python.org/3/library/allos.html).
* [Простейший графический интерфейс](https://docs.python.org/3/library/tk.html).

Тут чуть полезнее. Список встроенных функций с их описанием.

<https://docs.python.org/3/library/functions.html>



Встроенных типов данных меньше чем этих функций. Основные: целые числа, числа с плавающей запятой, строки, списки и словари. Словари, насколько помню, что-то вроде объектов. Могу ошибаться.

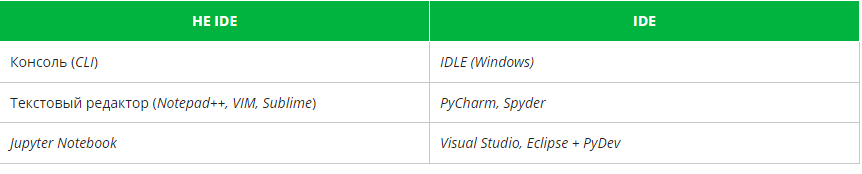
**Установка IDE и необходимых пакетов**

**Интегрированная среда разработки**

Вместе с установкой интерпретатора вместе с ним установилась программа под названием **IDLE.**

**IDLE (Integrated Development and Learning Environment) –** интегрированная среда разработки и обучения. Но конкретно эта, на **Python** больше подходит для обучения, чем для работы. Общие программы именно для разработки, поиска ошибок и тд называются **IDE.**

**IDE –** интегрированная среда разработки.



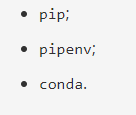
**IDE** должен иметь: текстовый редактор, транслятор (у **python** интерпретатор), который исполняет код, отладчик, для пошагового выполнения кода. Блин в рамках этого курса планируется использовать **Pycharm.** Не то, чтобы это была проблема. Но для начала буду все же в **VS code.**

**Пакетные менеджеры и виртуальное окружение**

Работа с зависимостями. Имеются виду модули и библиотеки, нужные проекту. Существует каталог пакетов **Python(PyPI – Python Package Index),** где содержатся пакеты, написанные другими разработчиками. Хранятся они все здесь:

<https://pypi.org/>

Устанавливаются они через пакетные менеджеры. Вот основные:



**PIP** модуль написанный на **Python.** Он позволяет работать только с установкой **Python-пакетов.**

**Pipenv** использует связку **pip** и **virtalenv,** это что-то упрощает, не понял.

**Conda** обладает немного большим фонкционалом и может устанавливать пакеты не только для **Python.**

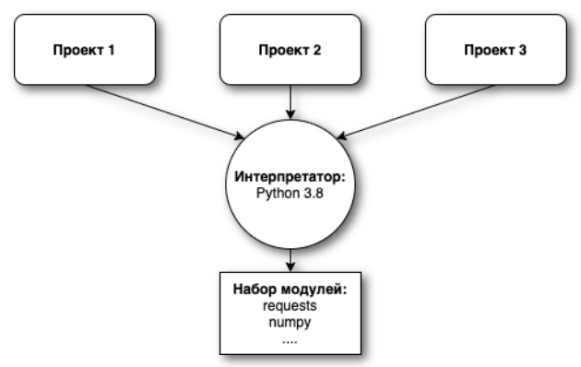
**Виртуальные окружения**

Я помню. В свое время заметно помучался с ними. В программировании есть 2 способа хранения зависимостей в проекте.

Все проекты используют единый набор зависимостей, которые еще называют общесистемным..

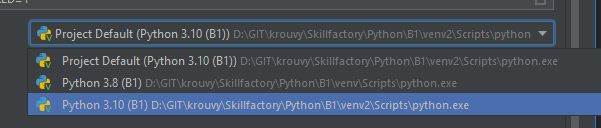
Каждый проект обладает собственным набором зависимостей, которые доступны только ему.

**Большая проблема Python — использование подхода с общесистемным набором зависимостей по умолчанию.** Модули будут устанавливаться, по умолчанию все в один интерпретатор.



Первая проблема это версии этих модулей. Называются они одинаково, а значит буду перезаписываться. Для интереса попробую создать еще виртуальную среду.

Да, вот.



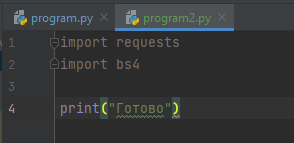
Да, и к этим средам легко теперь можно обращаться и в VS code. Странно только, что venv2 в **git** не добавился.

В этих папках был **git ignore.** Вот и ответ. Ну больше сред устанавливать не буду. Одна среда 11мб весит.

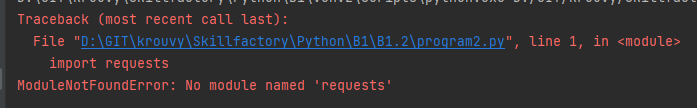
По заданию в виртуальную среду я установил 2 эти библиотеки.



Я установил их в среду **3.8**



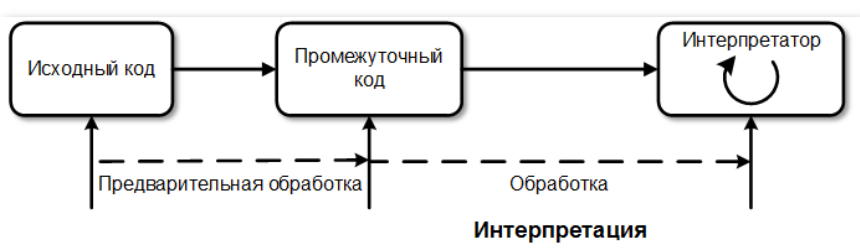
Все запустилось без конфликтов. А если выбрать не ту среду…



**Интерпретатор Python**

Интерпретатор будет прогонять код, пока не закончатся все строки или не дойдет до строки, обозначающей **конец интерпретации.**

После компиляции в байт-код рядом с файлом с исходным кодом создастся файл с таким же названием, но расширение у него будет .pyc. Он создается при каждом изменении исходного кода, если же изменений не было, будет сразу запускаться этот файл. Именно код из этого файла и выполняется построчно.

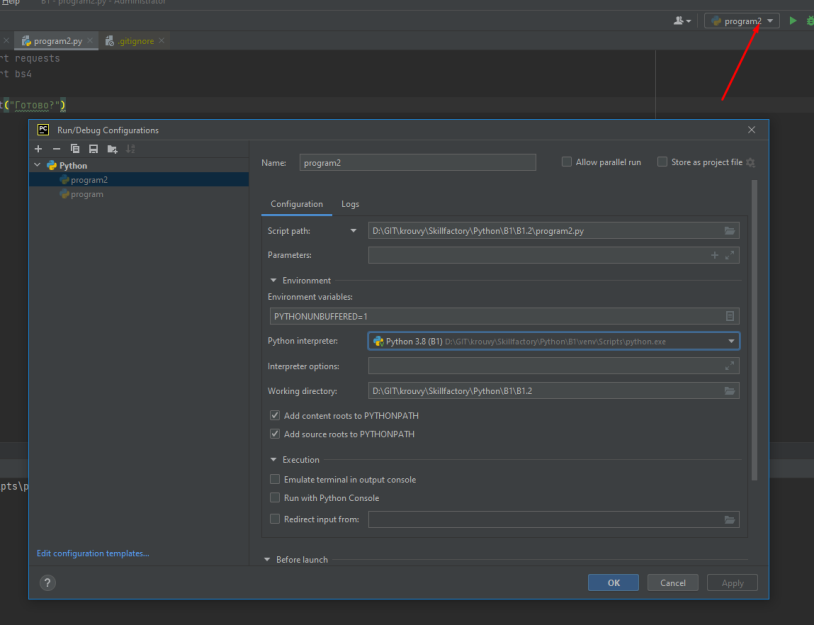
****

В общем интерпретатор работает так:

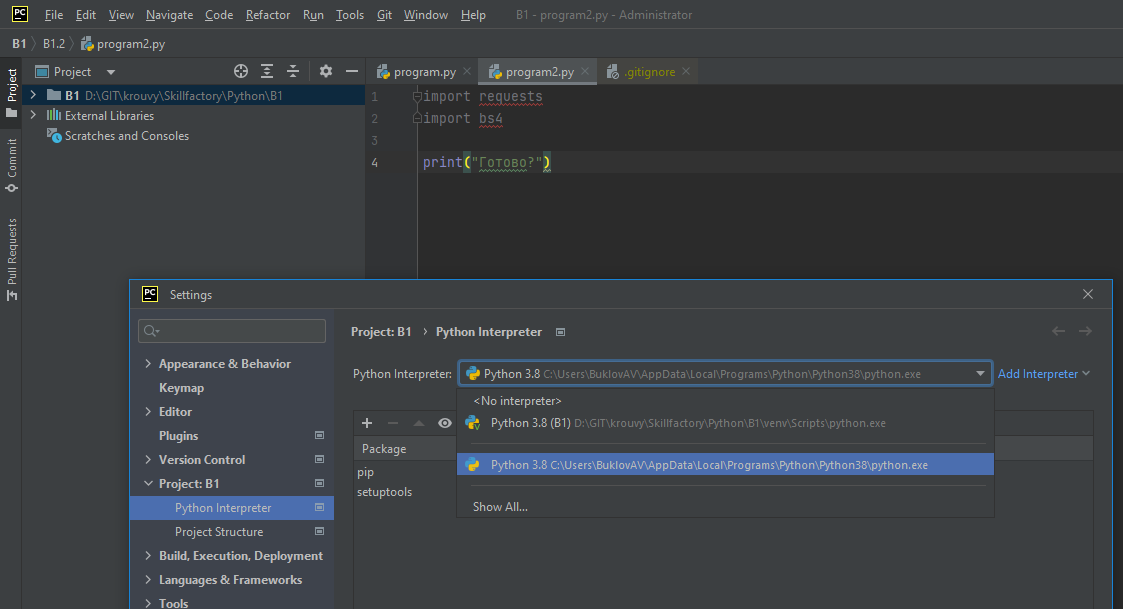
****

**Короче, важное открытие. Виртуальные среды создавать можно без проблем. Но. Их лучше на гит не заливать. Так как созданные они пытаются ссылаться на тех же пользователей, где и были созданы. А значит на других системах могут не запускаться.**

Еще кое-что. Здесь выбирается интерпретатор, который запустит проект.



Но можно еще выбрать интерпретатор, с помощью которого проект будет видеть код.



Если тут выбрать не тот интерпретатор, то проект будет думать, что библиотеки не подгружены. Хотя проект и будет работать, благодаря правильно выбранному интерпретатору у самой программы.

Сейчас пробую установить библиотеки в виртуальную среду. Для этого ее конечно нужно активировать, но на рабочем компе не хватает прав, так что только на ноутбуке. Сначала ее нужно активировать:



Чтобы выключить среду нужно сделать тоже самое, только **deactivate**

Что ж. После появится круглая кнопка с названием среды.



Теперь через команды **pip install** можно устанавливать библиотеки и они установятся именно в эту среду.

А через **pip freeze** можно посмотреть список установленных пакетов.

