Python3. Памятка для начинающих

Кирилл Версетти

—

Обновлено 14.01.2021

Оглавление

Установка и настройка рабочего окружения

Какой софт минимально необходим?

1. Интерпретатор Python >= 3.7 с модулями **`pip`** и **`venv`** [Linux | MacOS | Windows]
2. Распределенная система контроля версий Git [Linux | MacOS | Windows]
3. Терминал (Командная строка) [Linux | MacOS | Windows]
4. Редактор кода [Linux | MacOS | Windows]

Как установить интерпретатор Python в Linux?

В операционных системах семейства Linux, интерпретатор Python, как правило, уже установлен. Однако, в репозитории может быть доступна более свежая версия, либо необходимые пакеты не установлены, поэтому повторим установку еще раз:

1. Debian GNU/Linux и Debian based (например, Ubuntu, Linux Mint, Elementary OS):

   ```
   # apt update && apt install python3 python3-pip python3-venv
   ```

2. CentOS 8 (репозиторий appstream):

   ```
   # dnf install python38
   # alternatives --set python3 /usr/bin/python3.8
   ```

3. CentOS 7:

   ```
   # yum install python3
   ```

4. Manjaro 20.1:

   Все необходимые пакеты уже установлены.

5. Fedora 33:

   Все необходимые пакеты уже установлены.

Проверяем, вводим команду и если на экране отобразилась версия интерпретатора, то все установлено верно:

```
$ python3 -V
```

Как установить интерпретатор Python в MacOS?

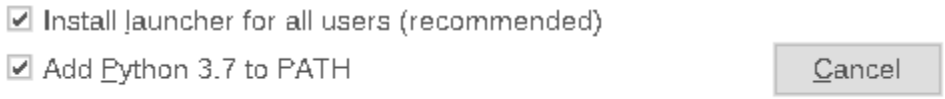
С официального сайта скачать установщик для последнего стабильного релиза.

Проверяем, открываем терминал, вводим команду и если на экране отобразилась версия интерпретатора, то все установлено верно:

```
$ python3 -V
```

Как установить интерпретатор Python в Windows?

1. С официального сайта скачать установщик для последнего стабильного релиза.
2. Запустить установщик и на первой странице **не забыть поставить галочку** “Add Python <VERSION> to PATH”:



3. **Проверить**, чтобы в имени компьютера и имени пользователя не было кириллицы, допустима только латиница.
4. Проверяем, открываем командную строку, вводим команду и если на экране отобразилась версия интерпретатора, то все установлено верно:

```
C:\> python -V
```

Если при попытке запустить интерпретатор из командной строки вы получили ошибку “*python не является внутренней или внешней командой, исполняемой программой или пакетным файлом.*”, то требуется вручную изменить переменную окружения **PATH**:

1. В командной строке найти директорию установки интерпретатора:

```
C:\Users\User> cd \
C:\> dir pip.exe /s
 Volume in drive C has no label.
 Volume Serial Number is 36C1-18C9

 Directory of C:\python\Scripts
```

```
01/14/2021  03:18 AM           106,349 pip.exe
               1 File(s)        106,349 bytes

     Total Files Listed:
               1 File(s)        106,349 bytes
               0 Dir(s)  85,585,444,864 bytes free
```

2. В переменную **PATH** добавить два пути **C:\python** и **C:\python\Scripts**

Как установить Git в Linux?

1. Debian GNU/Linux и Debian based (например, Ubuntu, Linux Mint, Elementary OS):

```
# apt update && apt install git
```

2. CentOS 8, Fedora >=22:

```
# dnf install git
```

3. CentOS 7, Fedora <22:

```
# yum install git
```

4. Arch Linux, Manjaro:

```
# pacman -Sy git
```

Проверяем, вводим команду и если на экране отобразилась версия программы, то все установлено верно:

```
$ git --version
```

Как установить Git в MacOS?

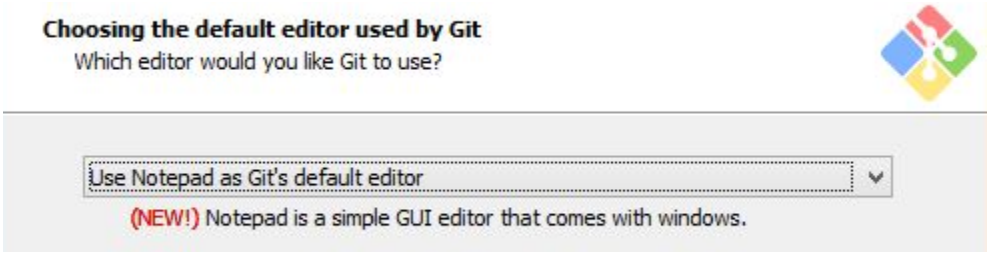
На официальном сайте выбрать один из подходящих способов установки: homebrew, xcode или установщик; самый простой - скачать установщик.

Проверяем, открываем терминал, вводим команду и если на экране отобразилась версия программы, то все установлено верно:

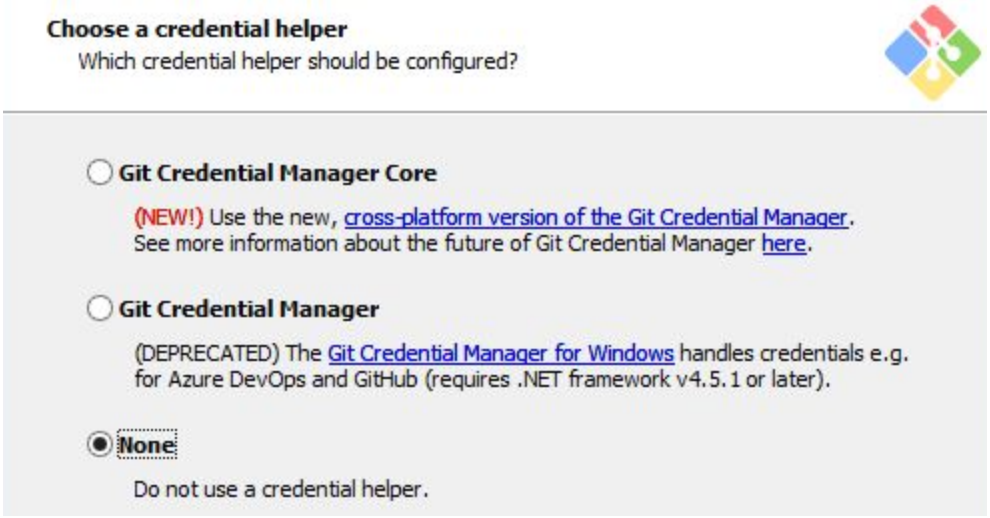
```
$ git --version
```

Как установить Git в Windows?

1. С официального сайта скачать установщик для последнего стабильного релиза.
2. Запустить установщик и изменить значения на указанных шагах:
   2.1. “**Choosing the default editor used by Git**” позволяет задать редактор по-умолчанию, который будет использован, например, когда вы забыли указать сообщение для коммита. Для новичков рекомендуется выбрать стандартный блокнот - “**Use Notepad as Git’s default editor**”. Для продвинутых пользователей, умеющих работать в Vim, можно оставить значение по-умолчанию.



   2.2. “**Choose a credential helper**” отвечает за настройку хранилища учетных данных. Если вы используйте протокол HTTP для доступа к удаленным репозиториям, то при каждом подключении необходимо указывать логин и пароль. Git использует такое поведение по-умолчанию - выбираем **None**.



   Позже вы сможете самостоятельно настроить это поведение удобным и безопасным для вас способом.

Проверяем, открываем командную строку, вводим команду и если на экране отобразилась версия программы, то все установлено верно:

```
C:\> git --version
```

Как открыть терминал в вашей ОС?

1. Linux – вы точно знаете, где он находится =)
2. MacOS – в строке поиска **Spotlight** введите слово Терминал и нажмите **Enter**
3. MS Windows – сочетанием клавиш **Win+R** открыть окно “**Выполнить**”, ввести команду **cmd** и нажать **Enter**

Как выбрать редактор кода (IDE)?

Если вы новичок и хотите выучить новый язык программирования, то отдать предпочтение блокноту с подсветкой синтаксиса, но только не IDLE.

Почему именно такой выбор? Чем проще редактор, тем проще им пользоваться, у вас не будет соблазна нажать “магическую” кнопку, вы все будете делать самостоятельно вручную шаг за шагом. Хороший разработчик знает “как оно работает внутри” и в случае возникновения ошибок или отсутствия привычного инструмента, не окажется в тупике.

Автодополнение зло - когда вы в коде прописываете каждое слово по буквам, у вас работает, как минимум, два вида памяти: зрительная и моторная. А если вы будете голосом проговаривать, что пишите, то сработает и слуховая память. Автодополнение допускает писать код “на глазок” - кажется оно так выглядело, выберу похожее из списка.

Нужно учиться искать ошибки в коде без чьей либо помощи - вы обязаны уметь читать трейсбек, понимать, в каком файле и в какой строке произошла ошибка, что это за ошибка: синтаксическая, арифметическая, логическая и т.д. Зачастую среда может подсветить место в коде, которое не является ошибочным, но новичок этого понять не может.

Нужно уметь работать с утилитами разработчика в терминале - так как терминал это один из основных инструментов питониста и тот, кто пытается доказать обратное, скорее претендует на звание “программиста-домохозяйки”, чем специалиста. Здесь ситуация такая же, как и с “магической” кнопкой, удобно нажать на зеленую стрелочку для запуска программы, но в боевых условиях зеленых стрелочек нет и что-то может пойти не так. Классический вопрос от автора: “Что вы будете делать, если перед вами голая консоль?”.

Вот некоторые популярные кроссплатформенные редакторы, которые из коробки достаточно простые:

- ❏ Sublime Text - субъективно, самый лучший выбор, не требователен к ресурсам, только не забыть настроить трансляцию табуляции в четыре пробела.
- ❏ Atom - редактор от разработчиков GitHub.
- ❏ Visual Studio Code - редактор от Microsoft.

Когда вы запомните синтаксис языка, сможете без проблем прочитать любую ошибку, а терминал не будет вызывать у вас паники, можно перейти на профессиональную IDE, например, PyCharm. Профессиональный инструмент имеет ряд преимуществ перед блокнотом:

- ❏ скорость работы - за счет автодополнения, кодогенераторов, сниппетов и прочих встроенных функций, вы сможете автоматизировать рутинную работу, однако использовать IDE для запуска программы я бы не рекомендовал
- ❏ защита от ошибок - автодополнение вместе с подсветкой синтаксических ошибок может уберечь вас от элементарных ошибок, особенно, если имена идентификаторов в выбранной библиотеке далеки от идеала =)
- ❏ поддержка утилит, необходимых при разработке - в PyCharm есть интеграция с Git, сервером баз данных, Docker, работа с виртуальным окружением, в платной версии поддержка популярных фреймворков, например, Django или Flask

Введение

Переменные и типы данных

Переменная – поименованная область оперативной памяти, в которую в процессе работы программы записываются данные или из которой они извлекаются.

На имена переменных накладываются следующие ограничения:

- ❏ первым символом имени должна быть буква или знак подчеркивания, за которым следует любая комбинация буквенно-цифровых латинских символов или знаков подчеркивания;
- ❏ не может содержать зарезервированные слова или специальные символы;
- ❏ имена чувствительны к регистру;
- ❏ по PEP-8 запрещено использовать любой регистр, кроме нижнего, каждое слово должно быть отделено подчеркиванием - **змеиная нотация**.

Тип данных переменной – является ее характеристикой и отражает:

- ❏ количество выделяемой памяти;
- ❏ формат представления данных;
- ❏ диапазон допустимых значений;
- ❏ операции, допустимые для применения к переменной.

Скалярные типы данных – содержат значения атомарного типа, в один момент времени переменная скалярного типа может хранить только одно значение:

- ❏ bool – логический тип
- ❏ int – целочисленный тип
- ❏ float – вещественный тип
- ❏ complex – комплексное число
- ❏ str – строковой тип
- ❏ bytes – байтовая строка

Структурные (ссылочные) типы данных – в один момент времени переменная структурного типа может хранить более одного значения скалярного или структурного типа:

- ❏ tuple – кортеж
- ❏ list – список
- ❏ set – множество
- ❏ dict – словарь
- ❏ object – объекты

None – пустота или отсутствие значения.

Декларация и инициализация переменной – определение и присвоение значения переменной, в Python не раздельны.

При инициализации переменной, на уровне интерпретатора создается объект, который имеет уникальный идентификатор, хранит тип данных и значение. Посредством оператора присваивания создается ссылка между переменной и этим объектом.

Неизменяемые (immutable) типы данных – bool, int, float, complex, str, bytes, tuple; неизменяемость означает, что в созданный объект больше нельзя вносить изменения – нельзя изменить тип данных или значение, для этого требуется создать новый объект.

Изменяемые (mutable) типы данных – list, set, dict, object; изменяемость означает, что в созданный объект можно вносить изменения без необходимости пересоздавать объект.

Приоритет операторов

В порядке уменьшения приоритета:

Оператор	Описание	Ассоциативность
()	скобки	нет
**	возведение в степень	правая
+ - ~	унарный плюс и минус, побитовый NOT	правая
* / % //	умножение, деление, деление по модулю, целочисленное деление	левая
+ -	сложение, вычитание	левая
<< >>	побитовый сдвиг влево и вправо	левая
&	побитовый AND	левая
^	побитовый XOR	левая
|	побитовый OR	левая
== != > < >= <=	равно, неравно, больше, меньше, больше или равно, меньше или равно	нет
in, not in	принадлежность (вхождение)	левая
is, is not	идентичность объектов	левая
not	логический NOT (инверсия)	правая
and	логический AND (И)	левая
or	логический OR (ИЛИ)	левая

Таблицы истинности логических операторов

X	Y	AND	OR	XOR
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

X	NOT
0	1
1	0

Методы строк

Далее перечислены часто используемые строковые методы. С полным списком методов, их детальным описанием и примерами, можно ознакомиться в документации:

Метод	Описание
"sep".join(iterable)	объединяет все элементы **iterable** в строку, используя разделитель **sep**. **TypeError**, если в **iterable** есть не строки
.split(sep=None, maxsplit=-1)	разбивает строку в список по разделителю **sep** **sep** по умолчанию все пробельные символы **maxsplit** если задан, то разбить не более **maxsplit**
.rsplit(sep=None, maxsplit=-1)	тоже самое, что **split**, но с конца строки
.splitlines([keepends])	разбивает текст в список строк **keepends** если **True**, то удаляет переносы строк
.strip([chars])	удаляет символы **chars** в начале и в конце строки **chars** если не задан, удаляет пробельные символы
.lstrip([chars])	тоже самое, что **strip**, но удаляет в начале строки
.rstrip([chars])	тоже самое, что **strip**, но удаляет в конце строки
.replace(old, new [, count])	поиск и замена подстроки **old** на **new** **count** раз
.find(s [, start [, end]])	поиск подстроки в строке возвращает индекс первого вхождения или -1 **start** индекс, с которого начать поиск **end** индекс, до которого выполнять поиск
.rfind(s [, start [, end]])	тоже самое, что **find**, но с конца строки
.index(s [, start [, end]])	тоже самое, что **find**, но **ValueError** если не найдено
.rindex(s [, start [, end]])	тоже самое, что **index**, но с конца строки
.count(s [, start [, end]])	возвращает количество вхождений подстроки **s** **start** индекс, с которого начать поиск **end** индекс, до которого выполнять поиск
.startswith(s [, start [, end]])	**True** если строка начинается с подстроки **s** **start** индекс, с которого начать поиск **end** индекс, до которого выполнять поиск

.endswith(**s [, start [, end]]** **)**	**True** если строка заканчивается подстрокой **s** **start** индекс, с которого начать поиск **end** индекс, до которого выполнять поиск
.lower()	переводит строку в нижнем регистре
.upper()	переводит строку в верхнем регистре
.swapcase()	переключает регистр каждого символа с нижнего на верхний и наоборот
.capitalize()	переводит первый символ в верхний регистр, а все остальные в нижний
.title()	первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
.isalnum()	**True** если строка содержит только буквы и цифры
.isalpha()	**True** если строка содержит только буквы
.isascii()	**True** если строка содержит только ASCII символы
.isdecimal()	**True** если строка содержит только десятичные символы (можно перевести в 10 СС)
.isdigit()	**True** если строка содержит только цифровые символы, к которым кроме десятичных относятся надстрочные и подстрочные индексы, экзотический пример Кхароштхи
.isnumeric()	**True** если строка содержит только цифровые символы и все символы, имеющие свойство числового значения Unicode, например, римские цифры.
.isspace()	**True** если строка содержит только пробельные символы
.islower()	**True** если строка в нижнем регистре
.isupper()	**True** если строка в верхнем регистре
.istitle()	**True** если все слова в строке начинаются с заглавной буквы

Методы списков[1]

Метод	Описание
.append(x)	добавляет **x** в конец списка
.extend(iterable)	добавляет все элементы **iterable** в конец списка
.insert(i, x)	вставляет **x** в позицию **i**
.remove(x)	удаляет первый элемент со значением **x** **ValueError**, если элемент не существует
.pop([i])	удаляет и возвращает элемент с индексом **i** если индекс не указан, удаляет последний **IndexError**, если индекс не существует
.clear()	очищает список
.index(x [, start [, end]])	возвращает индекс первого элемента со значением **x**, поиск идет от **start** до **end** **ValueError**, если элемент не существует
.count(x)	возвращает количество элементов со значением **x**
.sort(key=None, reverse=False)	выполняет сортировку списка без аргументов сортирует по возрастанию **key**: функция принимает один аргумент и возвращает ключ для сортировки **reverse**: сортировка в обратном порядке
.reverse()	переставляет элементы списка в обратном порядке
.copy()	возвращает плоскую копию списка

[1] Списки в документации - https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

Методы словарей[2]

Метод	Описание
.update([other])	обновляет словарь данными из **other** существующие ключи перезаписываются
.setdefault(key [, default])	возвращает значение ключа если ключ не существует, вместо исключения, создает ключ с значением **default** по умолчанию **default** равен **None**
.get(key [, default])	возвращает значение ключа если ключ не существует, вместо исключения, возвращает значение **default** по умолчанию **default** равен **None**
.keys()	возвращает ключи в словаре
.values()	возвращает значения в словаре
.items()	возвращает последовательность, где каждый элемент кортеж пар (ключ, значение)
.pop(key [, default])	удаляет и возвращает ключ **KeyError**, если ключ не существует **default**, если задан и ключ не существует
.popitem()	удаляет и возвращает пару (ключ, значение) **KeyError**, если словарь пустой
.clear()	очищает словарь
.copy()	возвращает плоскую копию словаря

[2] Словари в документации - https://docs.python.org/3/library/stdtypes.html#typesmapping

Функции при итерациях

Функция	Описание
range(end) **range(start, end [, step])**	возвращает диапазон целых чисел от **start** до **end** с шагом **step**, **end** - не входит в диапазон
enumerate(iterable, start=0)	возвращает итератор, где каждый элемент кортеж пар счетчик-элемент **start** - начальное значение счетчика
zip(*iterables)	возвращает итератор, где каждый элемент кортеж, состоящий из i-тых элементов **iterables** Пример: **zip([1, 2, 3], ["a", "b", "c"])** при итерации возвращает **(1, "a")**, **(2, "b")** и **(3, "c")**
reversed(iterable)	возвращает итератор, где элементы **iterable** идут в обратном порядке
filter(function, iterable)	возвращает итератор из элементов **iterable**, для которых **function** вернула **True**
map(function, iterable)	возвращает итератор, где к каждому элементу **iterable** была применена **function**
all(iterable)	возвращает **True**, если все элементы **iterable** имеют истинное значение, а также если **iterable** пуст
any(iterable)	возвращает **True**, если хотя бы один элемент из **iterable** имеет истинное значение
sum(iterable)	возвращает сумму элементов **iterable**
min(**iterable,** ***[, key, default]** **)** **min(a1, a2, *args [, key])**	Возвращает наименьший элемент из **iterable** или среди двух и более аргументов. Возвращает **default**, если **iterable** пуст. **ValueError**, если **iterable** пуст и **default** не задан. **key** - функция, которая принимает один аргумент и используется для упорядочивания.
max(**iterable,** ***[, key, default]** **)** **max(a1, a2, *args [, key])**	Возвращает наибольший элемент из **iterable** или среди двух и более аргументов. Аргументы такие же, как у функции **min**.

Функция обратного вызова

Callback function (функция обратного вызова) - это функция, переданная в другую функцию (при вызове) в качестве значения аргумента, которая будет вызвана в теле другой функции с заранее известными аргументами.

В синхронном программировании callback-функции используют тогда, когда нужно дать возможность изменять определенные шаги алгоритма. Например, функция фильтрации элементов списка, не может сама принять решение оставить элемент в списке или удалить, эту задачу она делегирует callback-функции; если callback-функция вернет истину, элемент будет оставлен, если ложь - удален.

В асинхронном программировании callback-функции используют как обработчики, которые выполняются после завершения асинхронной операции. Например, функция отправляет асинхронный HTTP-запрос на удаленный сервер и завершает свою работу. Когда запрос завершится, то будет вызвана callback-функция, которой будет передан результат запроса.

Callback-функцию можно задать с помощью **`def`**, если тело функции содержит более одной инструкции, либо с помощью **`lambda`**, если тело функции содержит только одну простую инструкцию. Самое частое применение lambda-функций - это функции обратного вызова.

Пример, как бы могла быть реализована встроенная функция filter, если бы ее не было в языке:

```
def filter_(callback, iterable):
    result = []

    for i in iterable:
        if callback(i):
            result.append(i)

    return result
```

```
def txt_files(value):
    return value.endswith('.txt')


lst = ['1.txt', '2.html', '', '3.mp3', '8.txt']
txt = filter_(txt_files, lst)
print(txt) # => ['1.txt', '8.txt']
```

Более производительное решение с использованием генератора, более близкое к оригинальной функции:

```
def filter_(callback, iterable):
    for i in iterable:
        if callback(i):
            yield i
```

Callback-функция может принимать любое количество аргументов и возвращать любое значение. В документации, автор функции обязан описать какие аргументы передаются в callback-функцию при вызове и какой результат он ожидает. Следуйте документации при написании своих callback-функций.

Работа с базой данных

PEP-249 описывает программный интерфейс взаимодействия с базами данных. Таким образом любой модуль, который работает с БД, должен реализовывать (поддерживать) этот интерфейс. Благодаря этому, на примере встроенного модуля sqlite3, вы сможете изучить и опробовать все методы. Если в будущем вам понадобится работать с СУБД MySQL или иной, то достаточно установить сторонний модуль, все методы вам уже знакомы.

СУБД и модули

- MySQL (MariaDB) - PyMySQL (мой выбор), MySQLdb
- PostgreSQL - psycopg2
- Oracle - cx_Oracle
- MS SQL Server - pyodbc
- SQLite3 - встроенный модуль sqlite3

Виды баз данных

- **реляционные** - совокупность таблиц и связей между ними. Примеры: MySQL, PostgreSQL, Oracle, MS SQL Server, SQLite.
- **документно-ориентированные** - хранит иерархические структуры данных (документы), как правило реализована с помощью подхода NoSQL. Примеры: MongoDB, CouchDB
- **объектно-ориентированные** - хранят информацию в виде объектов, как в объектно-ориентированных языках программирования.
- **графовые** - предназначены для хранения взаимосвязей и навигации в них, данные хранятся в виде структуры данных граф, где узлы хранят сущности, а ребра связи. Примеры: OrientDB, Amazon Neptune, Neo4j

Ключевые определения

Реляционная база данных – это совокупность отношений, содержащих всю информацию, которая должна храниться в БД. Однако пользователи могут воспринимать такую базу данных как совокупность таблиц.

Таблицы состоят из **строк** и **колонок**. Колонки часто называют **полями**.

Каждая строка должна иметь уникальный идентификатор, в терминах БД первичный ключ и фиксированное число полей.

Первичный ключ (Primary key) - это поле или набор полей со значениями, которые являются уникальными для всей таблицы.

Каждой колонке присваивается уникальное, в рамках текущей таблицы, имя, тип данных, значение по умолчанию и другие поддерживаемые характеристики.

Внешние ключи (Foreign key) - используются для организации связей между таблицами базы данных (родительскими и дочерними) и для поддержания ограничений ссылочной целостности данных.

SQL (Structured Query Language) - язык структурированных запросов, обычно используется в в реляционных базах данных для выполнения запросов.

Условно, можно сказать, что язык кроссплатформенный (одинаковый для любой СУБД), однако кроссплатформенность заканчивается в тот момент, когда в запросе используются специфичные для выбранной СУБД типы данных, функции или иной синтаксис.

Файл schema.sql - скрипт на языке SQL, содержащий определения таблиц; полей каждой таблицы с указанием названий, типов данных и опций полей; связей между таблицами (внешних ключей); создание индексов. Используется для инициализации пустой БД. Если приложение может использовать разные СУБД, то обычно для каждой создается свой schema.sql.

Алгоритм взаимодействия с БД

1. Установка соединения с сервером БД
   1.1. Опционально (для СУБД), выбрать базу данных
2. Выполнение запроса:
   2.1. Получить объект курсора (опционально)
   2.2. Выполнить запрос с помощью execute
   2.3. Если запрос на изменение данных или структуры БД:
      2.3.1. Нужно зафиксировать изменения (опционально)
   2.4. Если запрос на получение данных (SELECT):
      2.4.1. Фактические данные нужно раз-fetch-ить:
         fetchall() - получить все строки таблицы в список
         fetchone() - получить одну строку из таблицы
         fetchmany(N) - получить нужное кол-во строк (N) из таблицы
3. Закрыть соединение с сервером БД

Виртуальное окружение

По-умолчанию, все сторонние пакеты, установленные с помощью пакетного менеджера **`pip`** или вручную, используя команду **`python setup.py install`**, копируются в специальные директории в ОС. Такой способ установки не позволяет иметь две разные версии одной зависимости. Эту проблему позволяет решить виртуальное окружение.

Виртуальное окружение - это способ изолировать зависимости пакета, позволяет использовать в разных пакетах одинаковые зависимости, но разных версий. Зависимости ставятся локально в проект (в виртуальное окружение). Таким образом мы не засоряем систему и не столкнемся с проблемами отсутствия прав суперпользователя.

Важно понять и запомнить, что виртуальное окружение это НЕ:

- ❏ не виртуальная машина
- ❏ не контейнер
- ❏ не jail ("тюрьма"/chroot)
- ❏ не настоящий интерпретатор, а лишь ссылка на установленный в системе
- ❏ не хранится в git
- ❏ не переносится с ПК на ПК (привязан к текущему ПК)

Какой модуль использовать?

- ❏ В Python3 стандартный модуль venv (в Linux ставится отдельно)
- ❏ В Python2 сторонний модуль virtualenv (ставится pip-ом)

Как создать виртуальное окружение?

Переходим в корневую директорию с проектом и выполняем команду:

```
python3 -m venv env
```

- ❏ аргумент **`-m`** загружает модуль **`venv`**
- ❏ **`env`** имя директории, в которой будет располагаться виртуальное окружение, эту директорию нужно добавить в файл **`.gitignore`**

Как активировать виртуальное окружение?

Переходим в корневую директорию с проектом и выполняем команду:

Платформа	Оболочка	Команда
Posix	bash/zsh	`. ./env/bin/activate`
Windows	cmd.exe	`env\Scripts\activate.bat`
Windows	PowerShell	`env\Scripts\Activate.ps1`

В большинстве дистрибутивов Linux команда “точка” - псевдоним для команды **`source`**.

Как деактивировать виртуальное окружение?

Для деактивации или для выхода - введите команду **`deactivate`**.

Пара слов о Pipenv

В классическом подходе к разработке пакетов на Python с использованием виртуального окружения, **`pip`** и файлов **`requirements*.txt`** есть существенные недостатки. О них можно прочитать в этой статье., где также речь идет об альтернативном подходе с использованием **`pipenv`**.

Если вы новичок, то не торопитесь бежать и ставить себе **`pipenv`**, т.к. несмотря на 1,5 года с момента первого релиза, этот инструмент еще очень сырой и обуздать его в силах только профессионал. Но за **`pipenv`** будущее, им можно и нужно начинать пользоваться, когда наберетесь опыта.

Плюсы хорошо описаны в статье выше, из минусов лично я заметил два больших:

1. **не очевидный трейсбек** - из ошибок моих студентов, в 100% случаев виной был **`setup.py`**. Видимо потому, что **`pipenv`** замораживает зависимости, при установке любого пакета он просматривает все **`setup.py`** и если в каком-то есть ошибка, то падает с исключением;

2. **нет проверки успешности установки** - если ошиблись в имени пакета, то придется вручную удалять пакет из **`Pipfile`**, иначе будет постоянные ошибки при установке других пакетов;
3. **Windows**. Во-первых, в имени ПК должна быть только латиница, это относится не только к **`pipenv`**, это общая рекомендация. Во-вторых, в имени пользователя должна быть только латиница. В-третьих, если вы установили интерпретатор в какую-то нестандартную директорию, то все символы в пути должны содержать только латиницу. Самый простой выход из ситуации, это переименовать ПК и установить интерпретатор в корень диска **`C`**.

Пользовательские директории

Программе может потребоваться сохранить свои данные или конфигурационные файлы. Самый очевидный и простой способ - это корневая директория приложения. Однако, если программа одновременно используется несколькими пользователями ПК, то необходимо позаботиться о том, как хранить данные каждого пользователя раздельно, а возможно, и ограничить доступ к ним. Вторая проблема - это потеря данных после переустановки системы. Третья проблема - пользователи Windows не смогут синхронизировать данные своих приложений.

В ОС существуют специальные пользовательские директории для хранения данных и конфигурационных файлов приложений. Расположение этих директорий зависит от используемой вами ОС:

Linux	
Данные	**/home/<USER>/.local/share/<APP_NAME>**
Конфиг	**/home/<USER>/.config/<APP_NAME>**
Кеш	**/home/<USER>/.cache/<APP_NAME>**
Логи	**/home/<USER>/.cache/<APP_NAME>/log**
MacOS	
Данные	**/Users/<USER>/Library/Application Support/<APP_NAME>**
Конфиг	**/Users/<USER>/Library/Application Support/<APP_NAME>**
Кеш	**/Users/<USER>/Library/Caches/<APP_NAME>**
Логи	**/Users/<USER>/Library/Logs/<APP_NAME>**
Windows	
Данные	**C:\\Users\\<USER>\\AppData\\Local\\<APP_AUTHOR>\\<APP_NAME>**
Конфиг	**C:\\Users\\<USER>\\AppData\\Roaming\\<APP_AUTHOR>\\<APP_NAME>**
Кеш	**C:\\Users\\<USER>\\AppData\\Local\\<APP_AUTHOR>\\<APP_NAME>\\Cache**
Логи	**C:\\Users\\<USER>\\AppData\\Local\\<APP_AUTHOR>\\<APP_NAME>\\Logs**

В Python нет встроенного модуля, решающего этот вопрос, однако есть сторонний - appdirs. Если вы уже используете какой-то фреймворк, то обязательно уточните, нет ли в нем возможности работать с пользовательскими каталогами. Например, Click, имеет функцию, которая возвращает путь до каталога с конфигурацией приложения:

```
import click

APP_NAME = 'Demo App'
config_dir = click.get_app_dir(APP_NAME)
```

Приложение 1. PEP-8 коротко и по-русски[3]

- ❏ используйте 4 пробела для отступа и не используйте tab, не смешивайте их;
- ❏ максимальная длина строки 79 символов;
- ❏ функции верхнего уровня и определения классов отделяйте двумя пустыми строками;
- ❏ определения методов внутри класса отделяйте одной пустой строкой;
- ❏ дополнительные пустые строки используйте для логической группировки методов;
- ❏ кодировка файлов для Python3 должна быть utf-8;
- ❏ каждый импортируемый модуль на новой строке;
- ❏ импорты всегда помещаются в начало файла, порядок импортов модулей:
    - ❏ встроенные модули (стандартная библиотека)
    - ❏ сторонние модули (установленные **`pip`** или иначе)
    - ❏ локальные модули (из текущего проекта);

  пустая строка между каждой группой импортов;

- ❏ **`import *`** стоит избегать
- ❏ избегайте лишних пробелов внутри скобок, перед запятыми, точкой с запятой и двоеточиями;
- ❏ избегайте лишних пробелов перед скобками с аргументами функций и скобками с индексами;
- ❏ избегайте больше чем одного пробела между любыми операторами;
- ❏ избегайте пробелов вокруг `=`, если он используется для обозначения именованного аргумента или значения параметров по-умолчанию;
- ❏ избегайте составных инструкций - несколько команд в одной строке;
- ❏ обновляйте комментарии вместе с кодом;
- ❏ пишите комментарии по-английски;
- ❏ пишите комментарии для всех публичных модулей, функций, классов и методов;

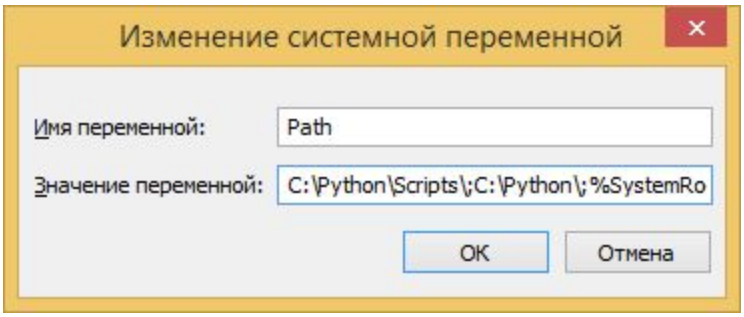
[3] читайте полную версию в официальной документации

- ❏ избегайте однобуквенных идентификаторов **`l`** (маленькая латинская “эль”), **`O`** (заглавная латинская O), **`I`** (заглавная латинская “ай”);
- ❏ для модулей и пакетов используйте короткие имена в нижнем регистре, можно использовать символы подчеркивания для отделения слов, но это не рекомендуется в именах пакетов;
- ❏ для классов используйте **`CamelCase`** имена;
- ❏ для функций и методов используйте **`snake_case`** имена;
- ❏ имена защищенных методов и переменных начинайте с **`_`**;
- ❏ имена закрытых методов и переменных начинайте с **`__`**;
- ❏ всегда используйте **`self`** в качестве первого аргумента метода экземляра объекта;
- ❏ всегда используйте **`cls`** в качестве первого аргумента метода класса;
- ❏ для проверки на **`None`** используйте **`is`** или **`is not`** , не используйте операторы сравнения;
- ❏ используйте **`isinstance()`** для проверки типа;
- ❏ не сравнивайте булевы переменные с **`True`** и **`False`**.

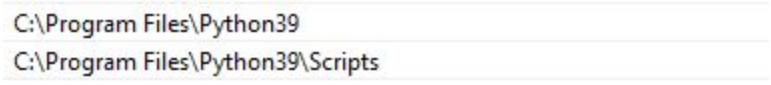
Приложение 2. Заметки для пользователей Windows

Как задать или настроить переменную окружения PATH в Windows?

1. Сочетанием клавиш **`Win+R`** открыть окно “Выполнить” и ввести команду **`sysdm.cpl`**
2. В открывшемся окне “Свойства системы” перейти на вкладку “Дополнительно” и найти кнопку **Переменные среды...**
3. В открывшемся окне “Переменные среды” нас интересует раздел “Системные переменные”. Находим в списке переменную **`PATH`** (регистр значения не имеет) и кликаем два раза. Если переменная не существует, то создаем.
4. Если у вас Windows ниже версии 10, в открывшемся окне в поле “Значение переменной” нужно добавить путь до директории с исполняемым файлом, например с **`python.exe`**. Разделителем путей служит символ точка с запятой:



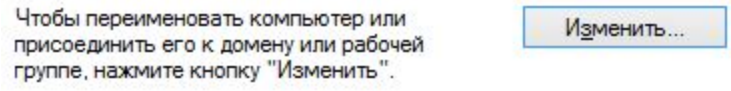
5. Если у вас Windows 10, то в открывшемся окне находим кнопку **Создать**, появится новое поле для ввода, в которое мы записываем путь до директории с исполняемым файлом, например с **`python.exe`**:



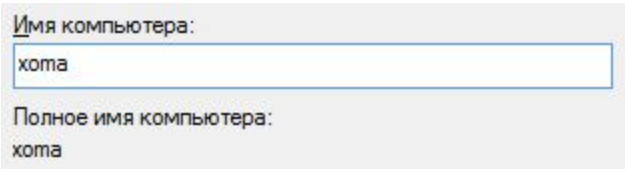
6. Нажать **`OK`** поочередно во всех открытых окнах и перезапустить командную строку, если ранее она была открыта.

Как изменить имя компьютера в Windows?

1. Сочетанием клавиш **Win+R** открыть окно “Выполнить” и ввести команду **sysdm.cpl**
2. В открывшемся окне “Свойства системы” найти кнопку “Изменить”, которая сопровождается подсказкой о смене имени ПК:



3. В открывшемся окне “Изменение имени компьютера или домена” указать новое имя компьютера на латинице и нажать **OK**:



4. Перезагрузить компьютер.

Как переименовать пользователя в Windows?