



Введение в Python

основные понятия, требования к trainee/junior, дзен Python



Что такое язык программирования

Язык программирования – формальный язык, который предназначен для записи компьютерных программ



Какие бывают языки программирования

Низкоуровневые

- Языки программирования, которые близки к программированию непосредственно в машинных кодах (assembler)

Высокоуровневые

- Языки программирования, разработанные для скорости и удобства использования программистом. Основная черта – абстракция, то есть введение смысловых конструкций, описание которых на машинном коде очень длинны или сложны для понимания (C, C++, Java, JavaScript, Python)



Какие бывают языки программирования

Компилируемые

- Языки программирования, исходный код которых переводится компилятором (программой, переводящей текст, написанный на ЯП) в машинный код и записывается в файл особым заголовком и/или расширением для последующей идентификации этого файла

Интерпретируемые

- Языки программирования, исходный код которых выполняется методом интерпретации (сперва транслируется в машинный код и сразу выполняются-интерпретируются). Позволяют динамические изменения по ходу выполнения программы
- Примеры: Python, JavaScript, PHP



Какие бывают языки программирования

Со статической типизацией

- Переменная, параметр программы, возвращаемое значение функции должны быть связаны с некоторым типом в момент объявления и не может быть изменен позже
- Примеры: C, C++, Java, Go

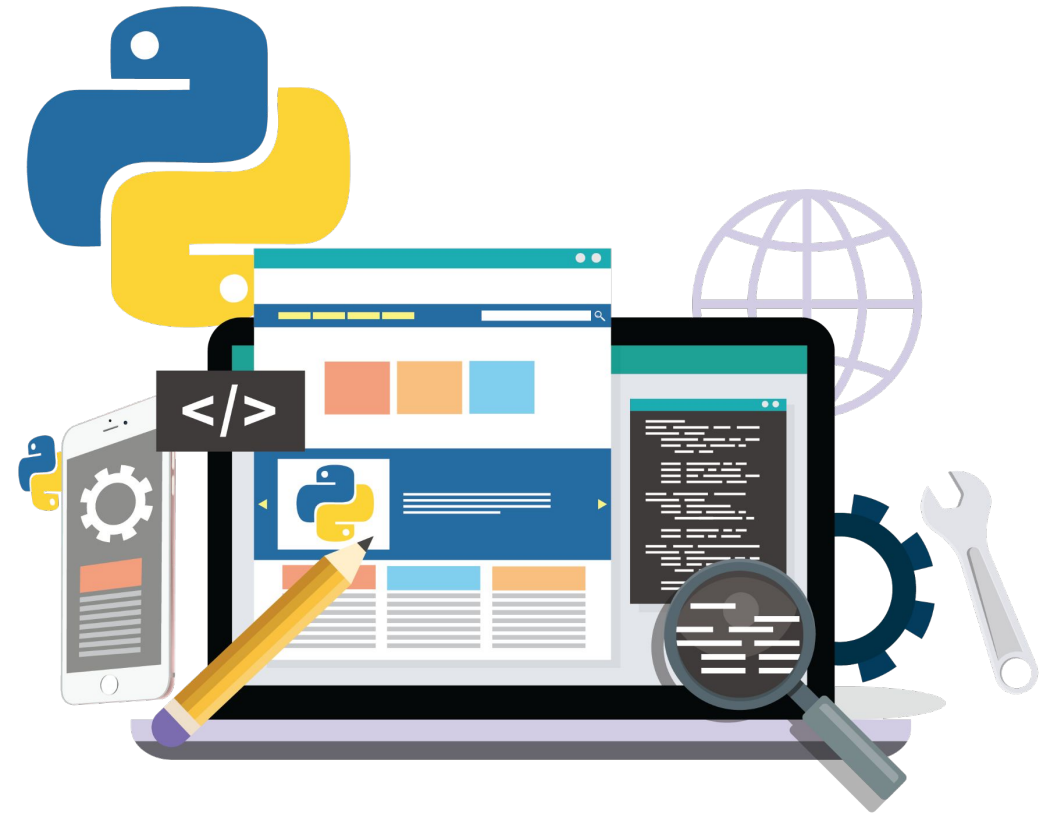
С динамической типизацией

- Переменная связывается с типом в момент присваивания значения, а не в момент объявления
- Примеры: Python, Ruby, PHP, JavaScript



Что такое Python

Python – высокоуровневый интерпретируемый объектно-ориентированный язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода



Факты о Python

- Разрабатывался с конца 80-х (первый релиз в феврале 1991 года)
- Назван в честь британского телешоу «Летающий цирк Монти Пайтона», а не от семейства пресмыкающихся
- В синтаксисе не используются фигурные скобки (как в C/C++/Java/JavaScript и тд)
- Очень популярный язык программирования (3 место TIOBE, 1-2 место stackoverflow)
- Существуют 2 версии (2-я, 3-я)



Гвидо ван Россум



Плюсы и минусы Python

- ✓ Универсальный (общего назначения)
- ✓ Кроссплатформенный
- ✓ Много готовых инструментов
- ✓ Востребованный на рынке
- ✓ Быстроразвивающийся
- ✓ Сильное community
- ✓ Мощная поддержка крупных компаний (Google, Facebook, Netflix, Uber, Instagram, Spotify и др.)
- ✓ Строгие требования к написанию кода
- ✗ Относительно медленный
- ✗ Относительно большее потребление памяти
- ? GIL
- ? Низкий порог вхождения (простой и понятный синтаксис, идеален для старта)
- ? Слишком свободный



Что можно сделать на Python

- Системные утилиты (Python, как правило, установлен по умолчанию в дистрибутивах linux)
- Веб-сайты (Django, Flask, Aiohttp, Tornado)
- Научные расчеты (NumPy, SciPy)
- Машинное обучение, нейронные сети (TensorFlow, Scikit-learn)
- Desktop приложения (tkinter, PyQt, wxPython)
- Игры (Pygame)
- Мобильные приложения (kivy)



Python vs. другой язык программирования

Python

```
1 class SingletonMeta(type):
2     _instances = {}
3
4     def __call__(cls, *args, **kwargs):
5         if cls not in cls._instances:
6             instance = super().__call__(*args, **kwargs)
7             cls._instances[cls] = instance
8         return cls._instances[cls]
9
10
11 class Singleton(metaclass=SingletonMeta):
12     def some_business_logic(self):
13         pass
14
15
16 if __name__ == "__main__":
17     s1 = Singleton()
18     s2 = Singleton()
19
20     if id(s1) == id(s2):
21         print("Singleton works")
22     else:
23         print("Singleton failed")
```

PHP

```
1 <?php
2
3 namespace RefactoringGuru\Singleton\Conceptual;
4
5 class Singleton
6 {
7     private static $instances = [];
8
9     protected function __construct() { }
10
11     protected function __clone() { }
12
13     public function __wakeup()
14     {
15         throw new \Exception("Can't unserialize a singleton.");
16     }
17
18     public static function getInstance(): Singleton
19     {
20         $cls = static::class;
21         if (!isset(self::$instances[$cls])) {
22             self::$instances[$cls] = new static();
23         }
24
25         return self::$instances[$cls];
26     }
27
28     public function someBusinessLogic()
29     {
30         // ...
31     }
32 }
33
34 function clientCode()
35 {
36     $s1 = Singleton::getInstance();
37     $s2 = Singleton::getInstance();
38     if ($s1 === $s2) {
39         echo "Singleton works, both variables contain the same
40 instance.";
41     } else {
42         echo "Singleton failed, variables contain different
43 instances.";
44     }
45 }
46 clientCode();
```



Python vs. другой язык программирования

Python

```
1 class SingletonMeta(type):
2     _instances = {}
3
4     def __call__(cls, *args, **kwargs):
5         if cls not in cls._instances:
6             instance = super().__call__(*args, **kwargs)
7             cls._instances[cls] = instance
8         return cls._instances[cls]
9
10
11 class Singleton(metaclass=SingletonMeta):
12     def some_business_logic(self):
13         pass
14
15
16 if __name__ == "__main__":
17     s1 = Singleton()
18     s2 = Singleton()
19
20     if id(s1) == id(s2):
21         print("Singleton works")
22     else:
23         print("Singleton failed")
```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "sync"
6 )
7
8 var lock = &sync.Mutex{}
9
10 type single struct {
11 }
12
13 var singleInstance *single
14
15 func getInstance() *single {
16     if singleInstance == nil {
17         lock.Lock()
18         defer lock.Unlock()
19         if singleInstance == nil {
20             fmt.Println("Creating single instance now.")
21             singleInstance = &single{}
22         } else {
23             fmt.Println("Single instance already created.")
24         }
25     } else {
26         fmt.Println("Single instance already created.")
27     }
28
29     return singleInstance
30 }
31
32 func main() {
33
34     for i := 0; i < 30; i++ {
35         go getInstance()
36     }
37     fmt.Scanln()
38 }
```



Python vs. другой язык программирования

Python

```
1 class SingletonMeta(type):
2     _instances = {}
3
4     def __call__(cls, *args, **kwargs):
5         if cls not in cls._instances:
6             instance = super().__call__(*args, **kwargs)
7             cls._instances[cls] = instance
8         return cls._instances[cls]
9
10
11 class Singleton(metaclass=SingletonMeta):
12     def some_business_logic(self):
13         pass
14
15
16 if __name__ == "__main__":
17     s1 = Singleton()
18     s2 = Singleton()
19
20     if id(s1) == id(s2):
21         print("Singleton works")
22     else:
23         print("Singleton failed")
```

Java

```
1 package refactoring_guru.singleton.example.non_thread_safe;
2
3 public final class Singleton {
4     private static Singleton instance;
5     public String value;
6
7     private Singleton(String value) {
8         try {
9             Thread.sleep(1000);
10        } catch (InterruptedException ex) {
11            ex.printStackTrace();
12        }
13        this.value = value;
14    }
15
16    public static Singleton getInstance(String value) {
17        if (instance == null) {
18            instance = new Singleton(value);
19        }
20        return instance;
21    }
22 }
23
24
25 package refactoring_guru.singleton.example.non_thread_safe;
26
27 public class DemoSingleThread {
28     public static void main(String[] args) {
29         System.out.println("If you see the same value, then
30 singleton was reused (yay!)" + "\n" +
31 "If you see different values, then 2 singletons
32 were created (boo!!)" + "\n\n" +
33 "RESULT:" + "\n");
34         Singleton singleton = Singleton.getInstance("FOO");
35         Singleton anotherSingleton =
36 Singleton.getInstance("BAR");
37         System.out.println(singleton.value);
38         System.out.println(anotherSingleton.value);
39     }
40 }
```



Python vs. другой язык программирования

Python

```
1 class SingletonMeta(type):
2     _instances = {}
3
4     def __call__(cls, *args, **kwargs):
5         if cls not in cls._instances:
6             instance = super().__call__(*args, **kwargs)
7             cls._instances[cls] = instance
8         return cls._instances[cls]
9
10
11 class Singleton(metaclass=SingletonMeta):
12     def some_business_logic(self):
13         pass
14
15
16 if __name__ == "__main__":
17     s1 = Singleton()
18     s2 = Singleton()
19
20     if id(s1) == id(s2):
21         print("Singleton works")
22     else:
23         print("Singleton failed")
```

C++

```
1 class Singleton
2 {
3
4 protected:
5     Singleton(const std::string value): value_(value)
6     {
7     }
8
9     static Singleton* singleton_;
10    std::string value_;
11
12 public:
13     Singleton(Singleton &other) = delete;
14     void operator=(const Singleton &) = delete;
15     static Singleton *GetInstance(const std::string& value);
16
17     void SomeBusinessLogic()
18     {
19         // ...
20     }
21
22     std::string value() const{
23         return value_;
24     }
25 };
26
27 Singleton* Singleton::singleton_ = nullptr;
28
29 Singleton *Singleton::GetInstance(const std::string& value)
30 {
31     if(singleton_==nullptr){
32         singleton_ = new Singleton(value);
33     }
34     return singleton_;
35 }
36
37 void ThreadFoo(){
38     std::this_thread::sleep_for(std::chrono::milliseconds(1000));
39     Singleton* singleton = Singleton::GetInstance("FOO");
40     std::cout << singleton->value() << "\n";
41 }
42
43 void ThreadBar(){
44     std::this_thread::sleep_for(std::chrono::milliseconds(1000));
45     Singleton* singleton = Singleton::GetInstance("BAR");
46     std::cout << singleton->value() << "\n";
47 }
48
49
50 int main()
51 {
52     std::cout << "If you see the same value, then singleton was reused (yay!)\n" <<
53         "If you see different values, then 2 singletons were created (boooo!!)\n\n" <<
54         "RESULT:\n";
55     std::thread t1(ThreadFoo);
56     std::thread t2(ThreadBar);
57     t1.join();
58     t2.join();
59
60     return 0;
61 }
```



2 версии Python

Python 2.x

- Признан устаревшим, новые проекты на нем не создаются. Последняя версия 2.7.6
- Не совместим с Python 3.x
- `print "Hello"`
- `range` и `xrange`:
 - `range` возвращает список
 - `xrange` возвращает итератор
- `mro` в глубину

Python 3.x

- Текущая версия 3.9.1
- Не совместим с Python 2.x
- `print('Hello')`
- Добавлена поддержка Unicode
- Добавлена поддержка асинхронности «из под коробки» (`asyncio` с Python 3.4)
- New-style классы. Все классы – наследники `object`
- Есть только `range`, возвращает итератор
- Появилась распаковка переменных (`*a`)
- F-строки (с python 3.6)
- `mro` в ширину

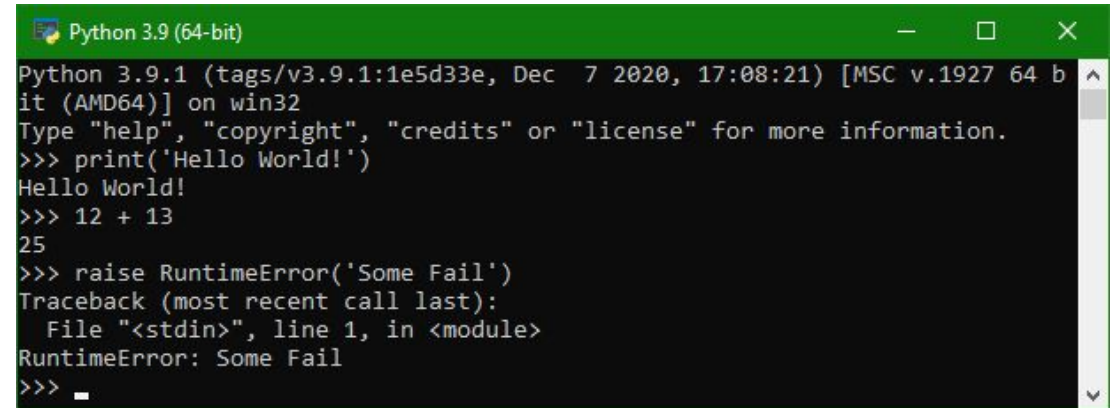


Интерпретатор Python

Интерпретатор – программа выполняющая анализ, обработку и выполнение исходного кода программы или запроса

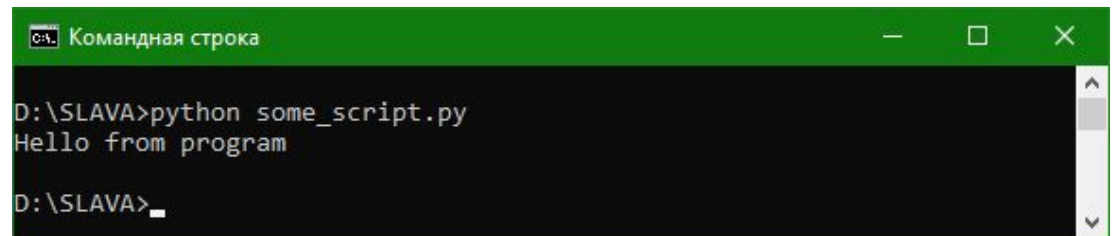
Два режима работы:

- Интерактивный
- Выполнение программ



```
Python 3.9 (64-bit)
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 b
it (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World!')
Hello World!
>>> 12 + 13
25
>>> raise RuntimeError('Some Fail')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: Some Fail
>>> _
```

1. Интерактивный режим



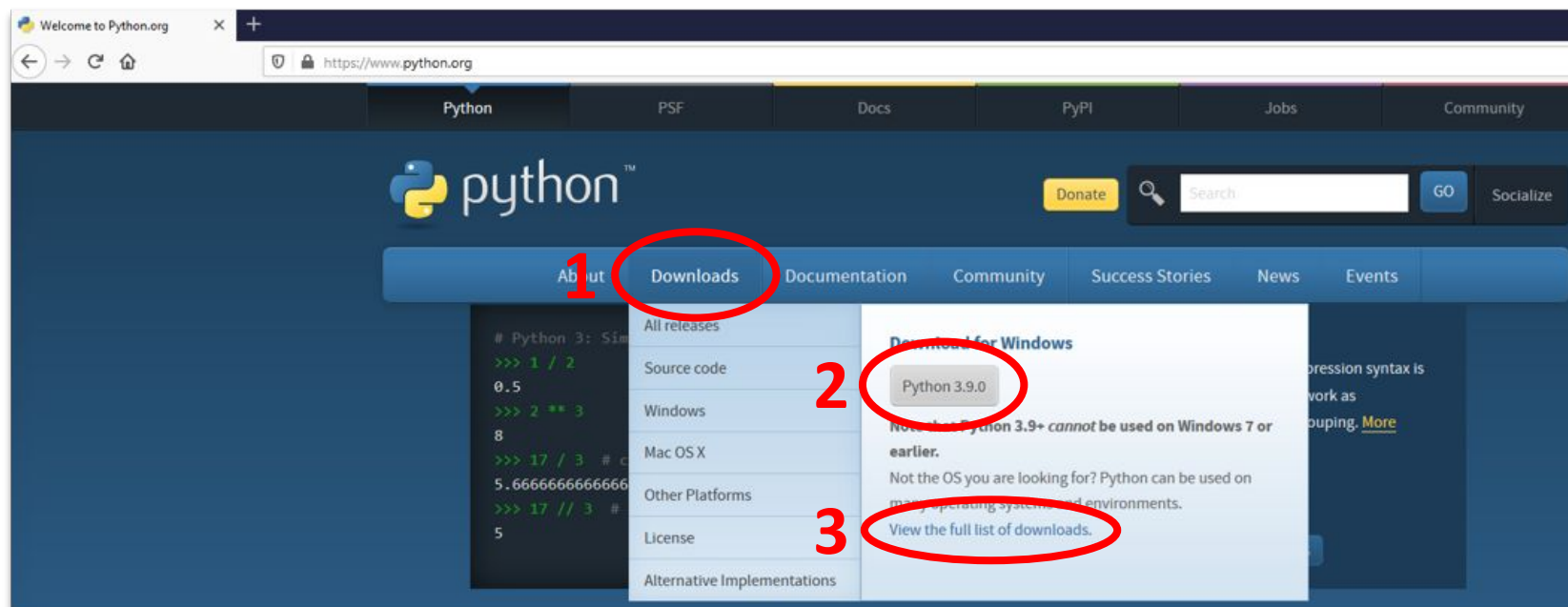
```
Командная строка
D:\SLAVA>python some_script.py
Hello from program
D:\SLAVA>_
```

2. Выполнение программ



Установка Python в Windows

- Зайти на сайт: <https://www.python.org/>
- В разделе Downloads (1) скачать последнюю версию интерпретатора (2)
- Если нужна другая версия Python – перейти по ссылке (3) <https://www.python.org/downloads/> и скачать нужную версию



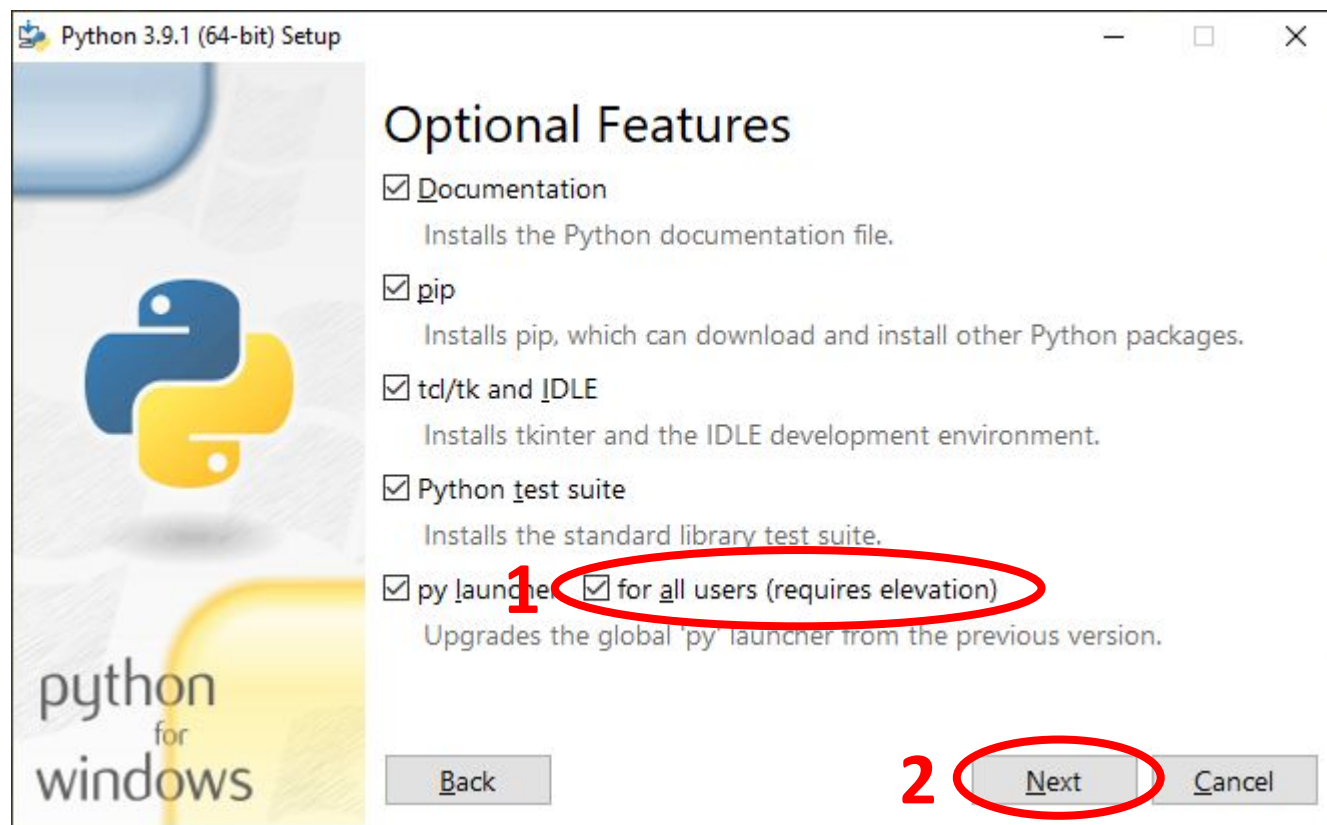
Установка Python в Windows

- Не забываем добавить Python в PATH! (1)
- Если в системе один пользователь, либо мы хотим установить Python только для текущего пользователя, то выбираем пункт «Install Now» (2), как правило, этого достаточно
- Если мы хотим установить Python для всех пользователей, то выбираем вариант «Customize installation» (3)



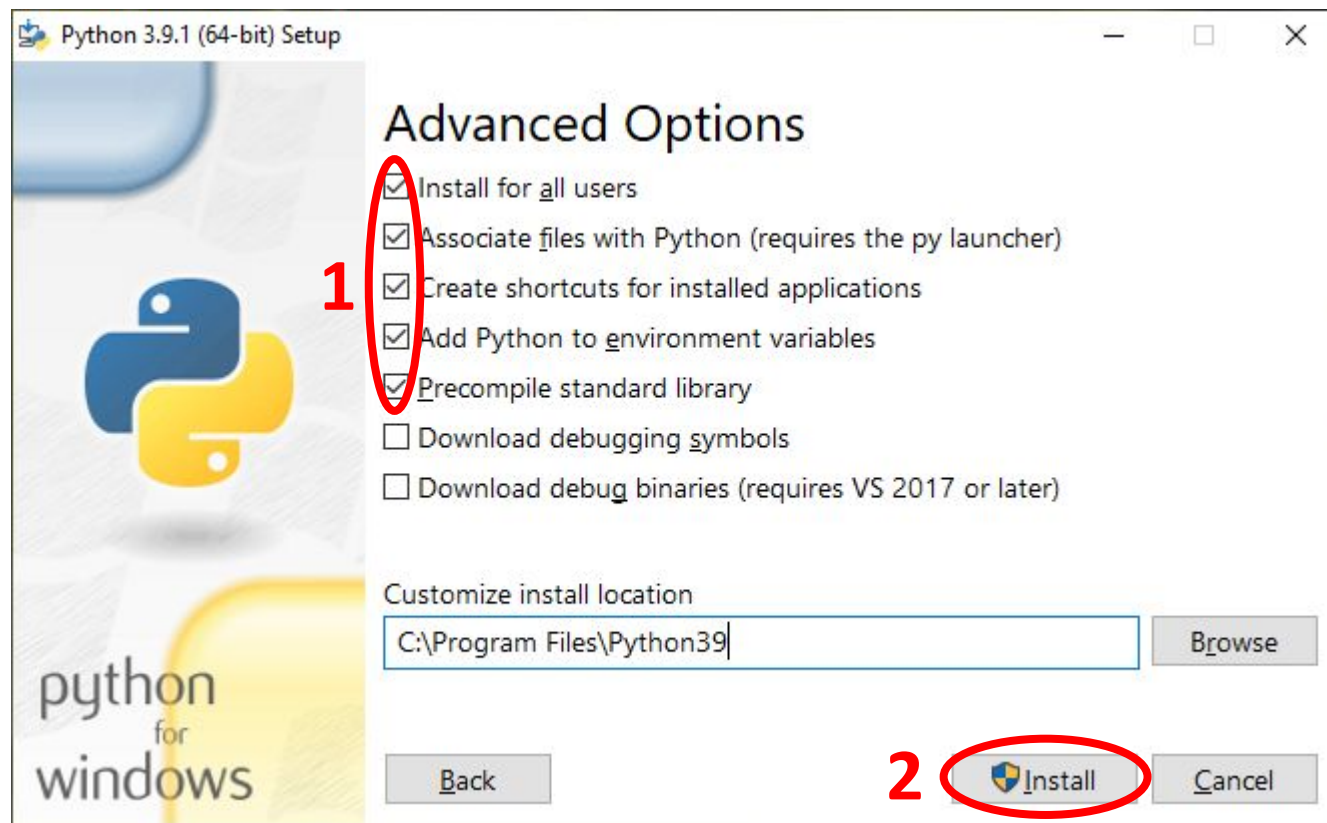
Установка Python в Windows

- Ничего не меняем, проверяем, чтобы был выбран пункт «for all users» (1)
- Нажимаем кнопку «Next» (2)



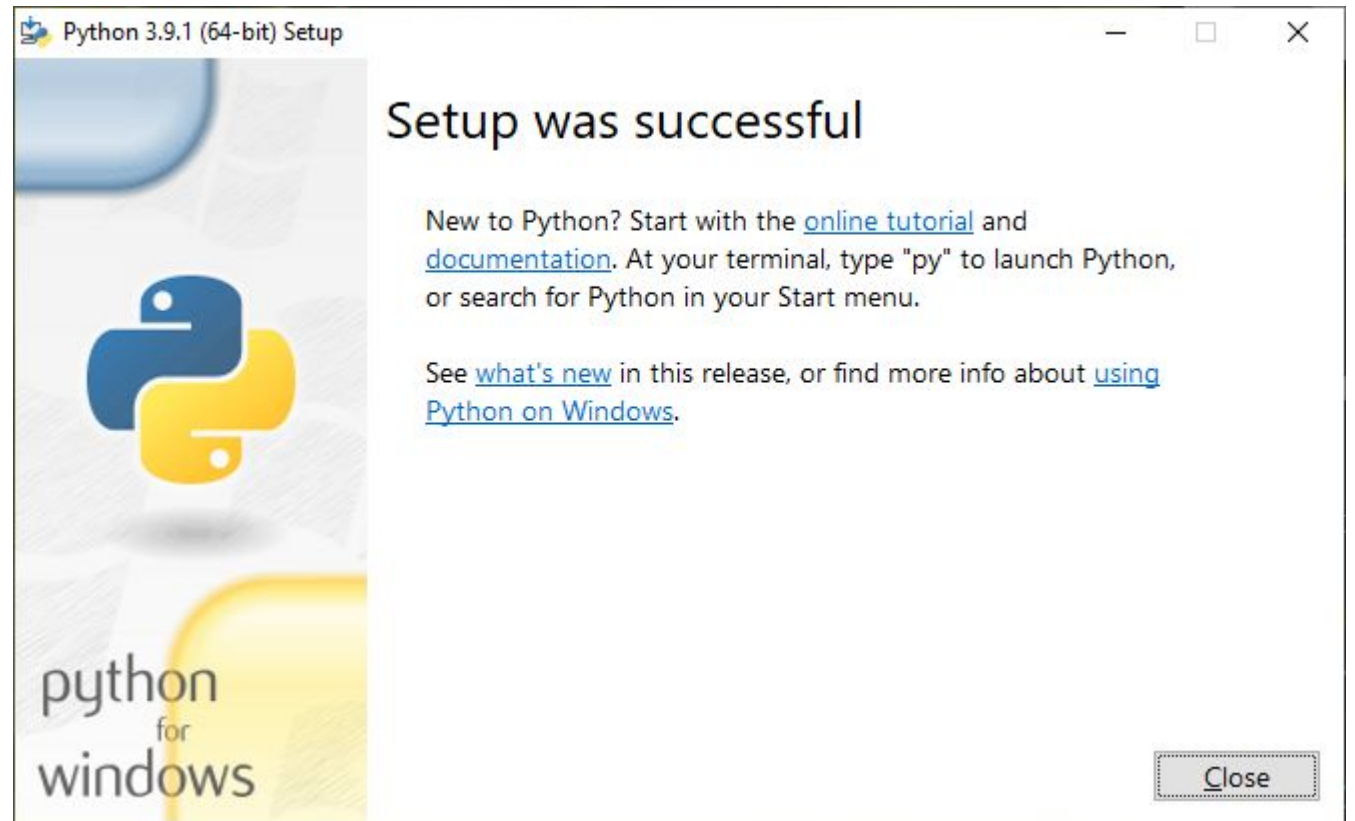
Установка Python в Windows

- Ничего не меняем, проверяем, чтобы был выбран следующие пункты (1)
- Нажимаем кнопку «Install» (2)



Установка Python в Windows

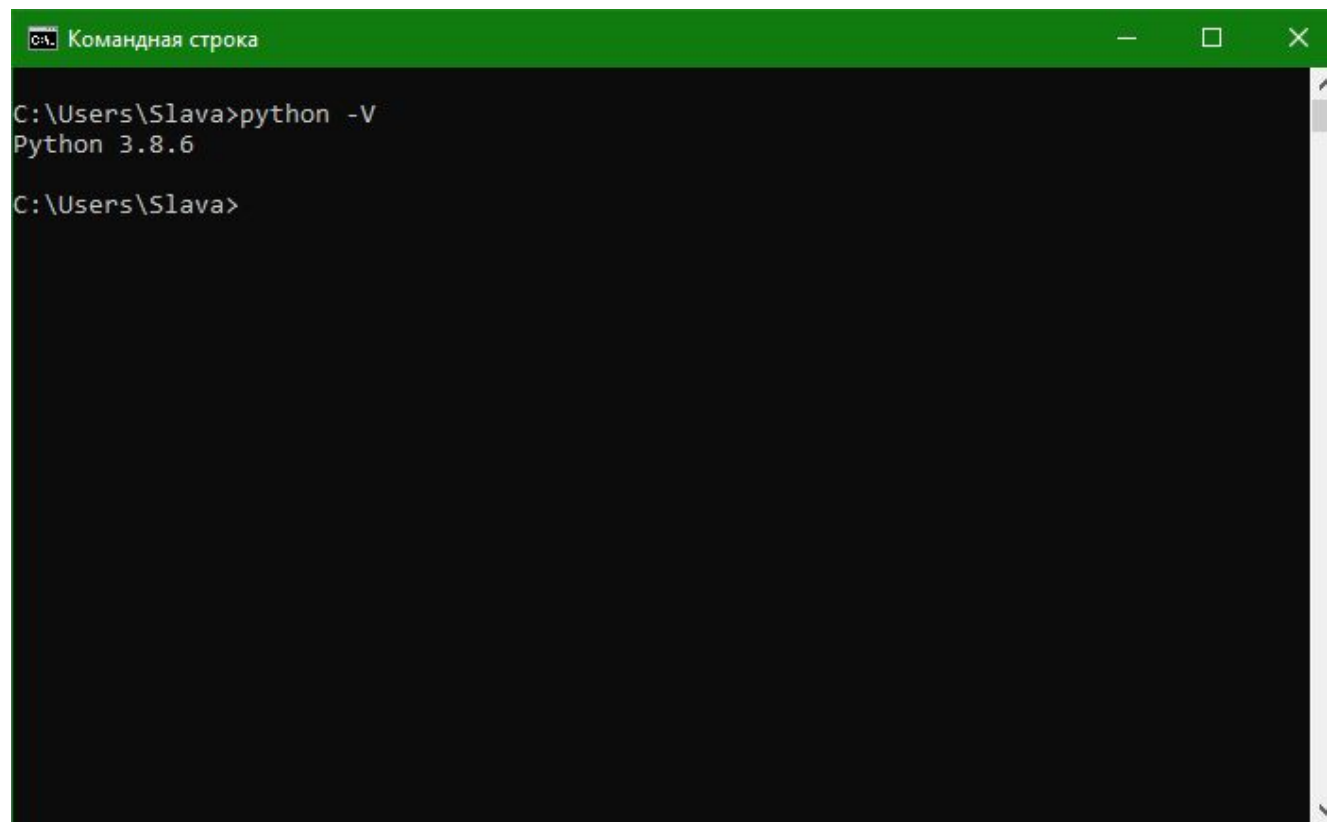
Когда видим следующее окно, значит, Python установлен в систему



Установка Python в Windows

Проверка установки:

- Открываем командную строку (Win+R, вводим cmd)
- Вводим команду:
 - `python -V`
- Если на экран выведена информация об установленной версии Python, то установка прошла успешно



```
Командная строка
C:\Users\Slava>python -V
Python 3.8.6
C:\Users\Slava>
```



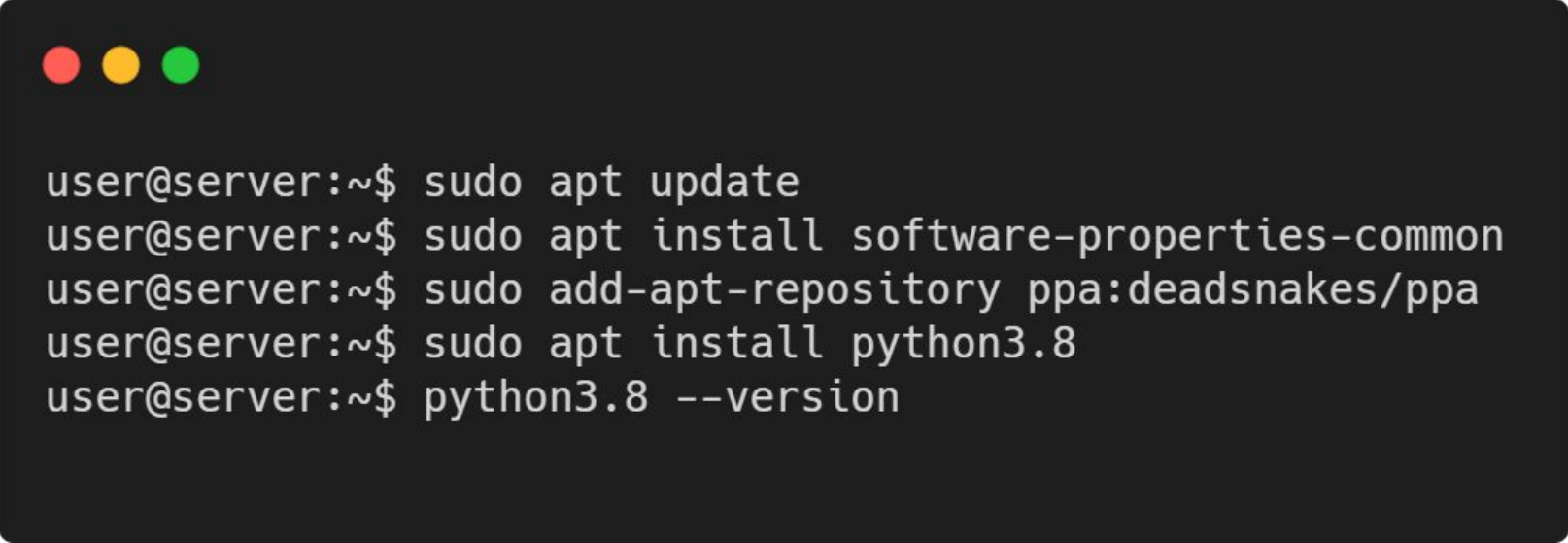
Вопрос-ответ

- ? Что делать, если я забыл(а) добавить Python в PATH при установке?
- ✓ Необходимо добавить путь к интерпретатору и папке scripts в системные переменные, а после перезагрузить компьютер



Установка в Linux (Ubuntu)

Для установки в ОС Ubuntu необходимо выполнить следующие команды в терминале (если нужная версия Python не установлена):



```
user@server:~$ sudo apt update
user@server:~$ sudo apt install software-properties-common
user@server:~$ sudo add-apt-repository ppa:deadsnakes/ppa
user@server:~$ sudo apt install python3.8
user@server:~$ python3.8 --version
```



Установка в MacOS

- Перед началом, вам нужно установить **Homebrew**:
- Перейти на страницу <http://brew.sh/>. Выбрать код начальной загрузки под **Install Homebrew**. Далее нажимаем **Cmd+C**, чтобы копировать его в буфер обмена. Убедитесь в том, что вы полностью выделили текст команды, так как в противном случае установка будет неудачной
- Открыть окно **Terminal.app**, вставить код начальной загрузки **Homebrew**, затем нажать **Enter**. Начнется установка
- Если вы делаете это в свежей версии macOS, может появиться предупреждение, в котором предлагается установка инструментов командной строки разработчика от Apple. Это нужно для того, чтобы закончить установку, так что подтвердите диалоговое окно, нажав на **install**.
- Подтверждаем диалог «Программное обеспечение было установлено» установочного файла инструментов разработчика
- Возвращаемся к терминалу, нажимаем **Enter** для продолжения установки Homebrew
- Homebrew попросит вас ввести свой пароль для окончания установки. Введите свой пользовательский пароль и нажмите Enter, чтобы продолжить
- В зависимости от того, какое у вас подключение к интернету, Homebrew займет несколько минут времени для загрузки необходимых файлов. После окончания установки, вам нужно будет вернуться к окну терминала



Установка в MacOS

После того, как Homebrew установлен, возвращаемся к терминалу и выполняем следующие команды



```
user-macbook:~ user$ brew install python3  
user-macbook:~ user$ python3 --version
```



Что такое PEP

Развитие языка Python происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов PEP.

PEP (Python Enhancement Proposal) — это предложения по развитию питона. Процесс PEP является основным механизмом для предложения новых возможностей и для документирования проектных решений, которые прошли в Python.

- <https://www.python.org/dev/peps/>

Index by Category

Meta-PEPs (PEPs about PEPs or Processes)

	PEP	PEP Title	PEP Author(s)
P	1	PEP Purpose and Guidelines	Warsaw, Hylton, Goodger, Coghlan
P	4	Deprecation of Standard Modules	Cannon, von Löwis
P	5	Guidelines for Language Evolution	Prescod
P	6	Bug Fix Releases	Aahz, Baxter
P	7	Style Guide for C Code	GvR, Warsaw
P	8	Style Guide for Python Code	GvR, Warsaw, Coghlan
P	10	Voting Guidelines	Warsaw
P	11	Removing support for little used platforms	von Löwis, Cannon
P	12	Sample reStructuredText PEP Template	Goodger, Warsaw, Cannon
P	387	Backwards Compatibility Policy	Peterson
P	609	PyPA Governance	Ingram, Gedam, Harihareswara



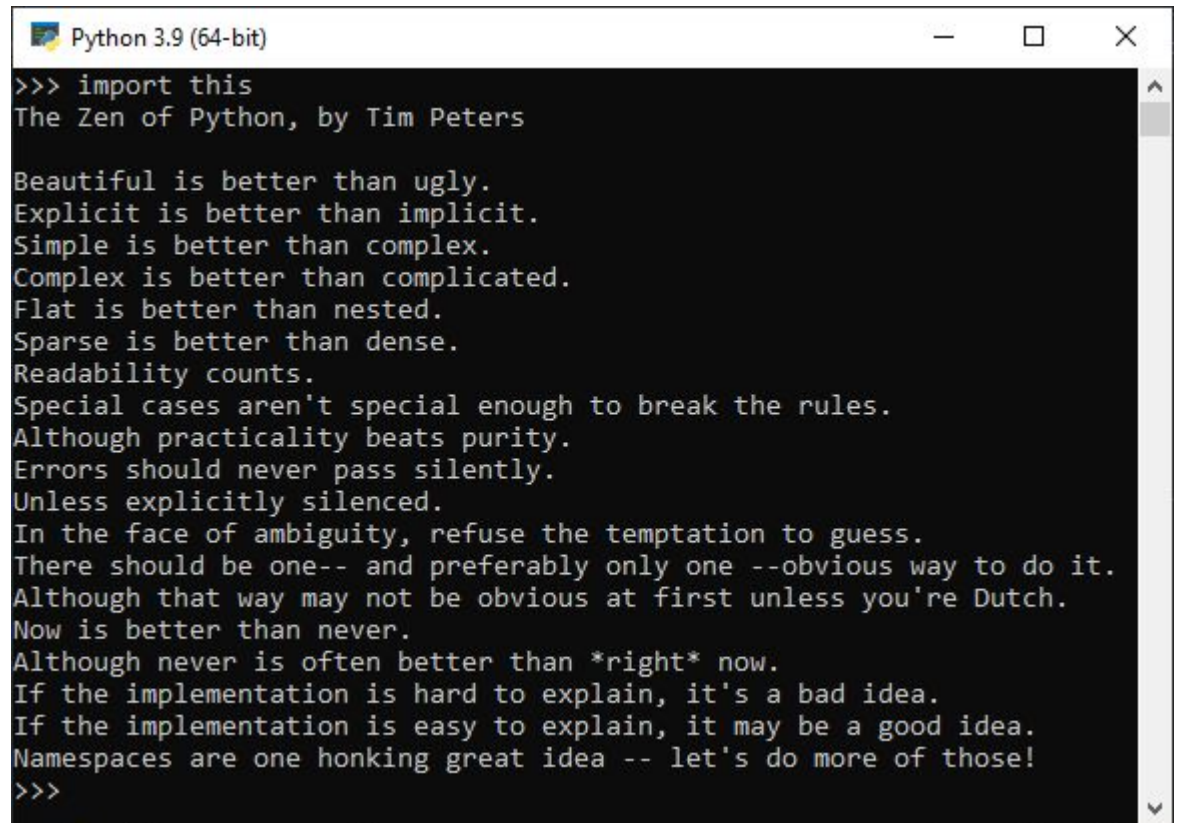
The Zen of Python, by Tim Peters (PEP 20)

Каждый язык программирования имеет свой стиль. В Python встроен небольшой текст, который выражает его философию (насколько я знаю, Python — это единственный язык программирования, содержащий подобное «пасхальное яйцо»)



The Zen of Python, by Tim Peters (PEP 20)

Каждый язык программирования имеет свой стиль. В Python встроен небольшой текст, который выражает его философию (насколько я знаю, Python — это единственный язык программирования, содержащий подобное «пасхальное яйцо»)



```
Python 3.9 (64-bit)
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```



The Zen of Python, by Tim Peters (PEP 20)

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если они не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один и, желательно, только один очевидный способ сделать это.
- Хоть и лучше, чем никогда.



Хорошо

Введение

IDE

Pip, venv

Основы

Что хотят IT компании от trainee/junior

Какие требования чаще встречаются в вакансиях для Python разработчиков уровня trainee/junior:

- ✓ Знание принципов ООП и паттернов проектирования
- ✓ Знание популярных Python фреймворков/библиотек (Django/Flask/aiohttp, asyncio, SQLAlchemy, NumPy/SciPy/scikit-learn/pandas/matplotlib, Celery)
- ✓ Базовые знания SQL и базовое понимание принципов работы с базами данных (PostgreSQL, MySQL)
- ✓ Базовое умение работать с системой контроля версий Git



Nice to have НАВЫКИ

Какие технологии, как правило, отмечают для Python разработчиков как «Будет плюсом»:

- ✓ Знание Frontend технологий (JavaScript, HTML, CSS)
- ✓ Знание JavaScript фреймворков/библиотек (React, Vue, Angular, JQuery)
- ✓ Умение работать с Docker
- ✓ Умение работать с NoSQL базами (MongoDB, Redis, Memcached)
- ✓ Умение работать с брокерами сообщений (RabbitMQ, Apache Kafka)
- ✓ Персональные качества (обучаемость, коммуникабельность и т. д.)





Ваши вопросы

что необходимо прояснить в рамках
данного раздела





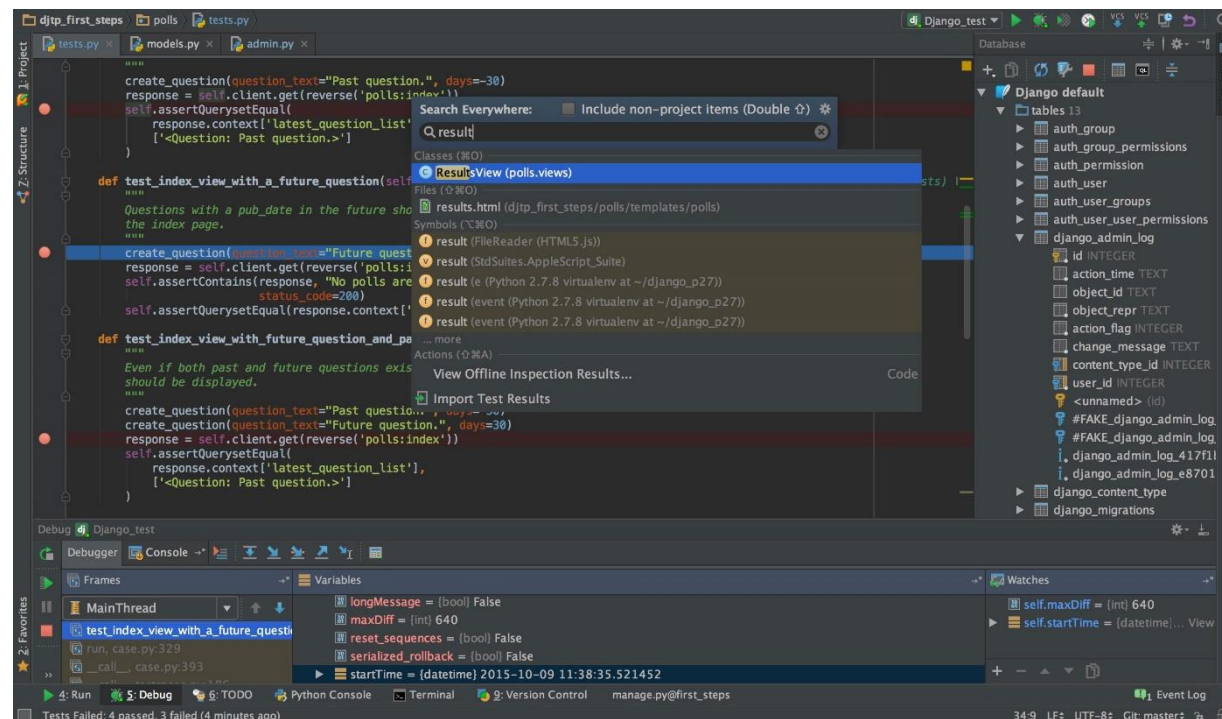
Выбор среды разработки

что такое IDE, что должно быть в IDE, сравнение IDE



Среда разработки (IDE)

IDE (интегрированная среда разработки или integrated development environment) - система программных средств, используемая программистами для разработки программного обеспечения



IDE для работы с Python

PyCharm



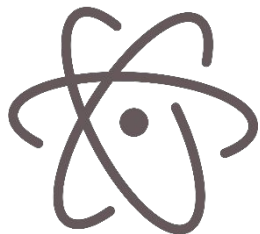
Jupyter



Visual Studio
Code



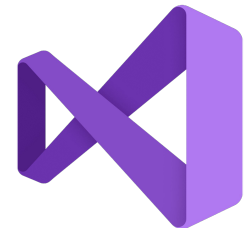
Atom



Spyder



Visual Studio

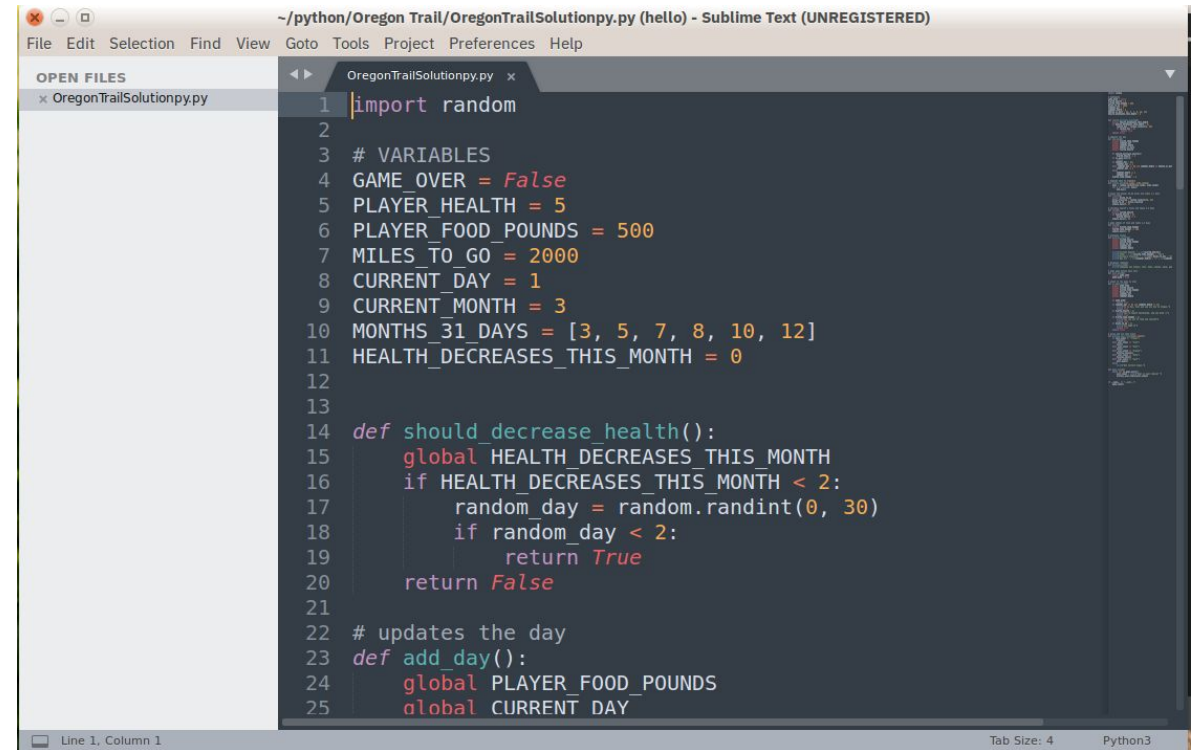


Что должно быть в IDE

- Работа с файлами (открытие, создание, редактирование, сохранение)
- Запуск кода внутри среды
- Поддержка возможности отладки (возможность пошагово выполнить программный код)
- Подсветка синтаксиса (облегчение восприятия программного кода)
- Автоматическое форматирование кода (добавление отступов, символов двоеточия и т.д.)
- Дополнительные необходимые инструменты (для работы с базами данных, сетевыми подключениями, системой контроля

Sublime Text

- Тип: редактор кода
- Сайт:
 - <https://www.sublimetext.com/>
- Цена:
 - \$80
- Плюсы:
 - Кроссплатформенность
 - Много поклонников (особенно среди РНР разработчиков)
 - Быстрый
 - Легковесный
 - Хорошая поддержка
 - Широкие возможности кастомизации
- Минусы:
 - Бывает трудно настроить
 - Много расширений, в которых начинающий может запутаться
- Комментарий: встречается в некоторых компаниях установленным по умолчанию



```
~/python/Oregon Trail/OregonTrailSolution.py (hello) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

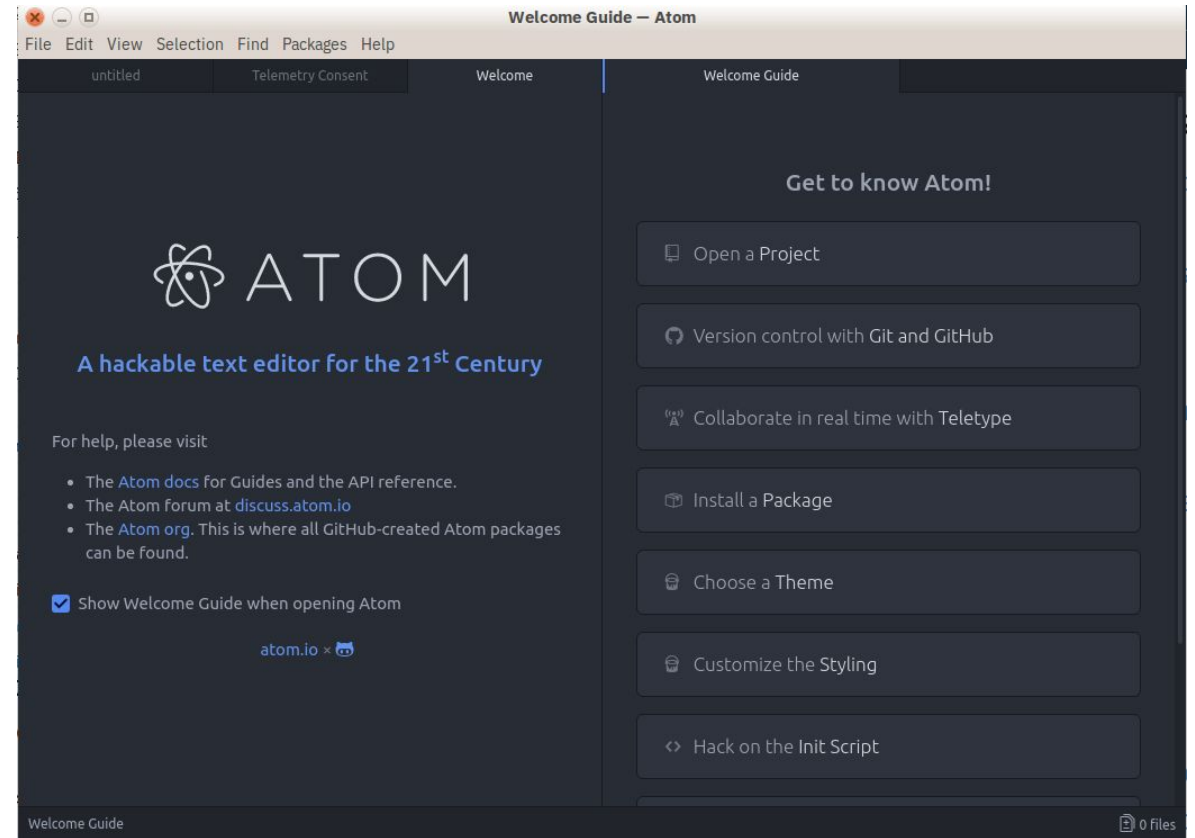
OPEN FILES
x OregonTrailSolution.py

1 import random
2
3 # VARIABLES
4 GAME_OVER = False
5 PLAYER_HEALTH = 5
6 PLAYER_FOOD_POUNDS = 500
7 MILES_TO_GO = 2000
8 CURRENT_DAY = 1
9 CURRENT_MONTH = 3
10 MONTHS_31_DAYS = [3, 5, 7, 8, 10, 12]
11 HEALTH_DECREASES_THIS_MONTH = 0
12
13
14 def should_decrease_health():
15     global HEALTH_DECREASES_THIS_MONTH
16     if HEALTH_DECREASES_THIS_MONTH < 2:
17         random_day = random.randint(0, 30)
18         if random_day < 2:
19             return True
20     return False
21
22 # updates the day
23 def add_day():
24     global PLAYER_FOOD_POUNDS
25     global CURRENT_DAY
```



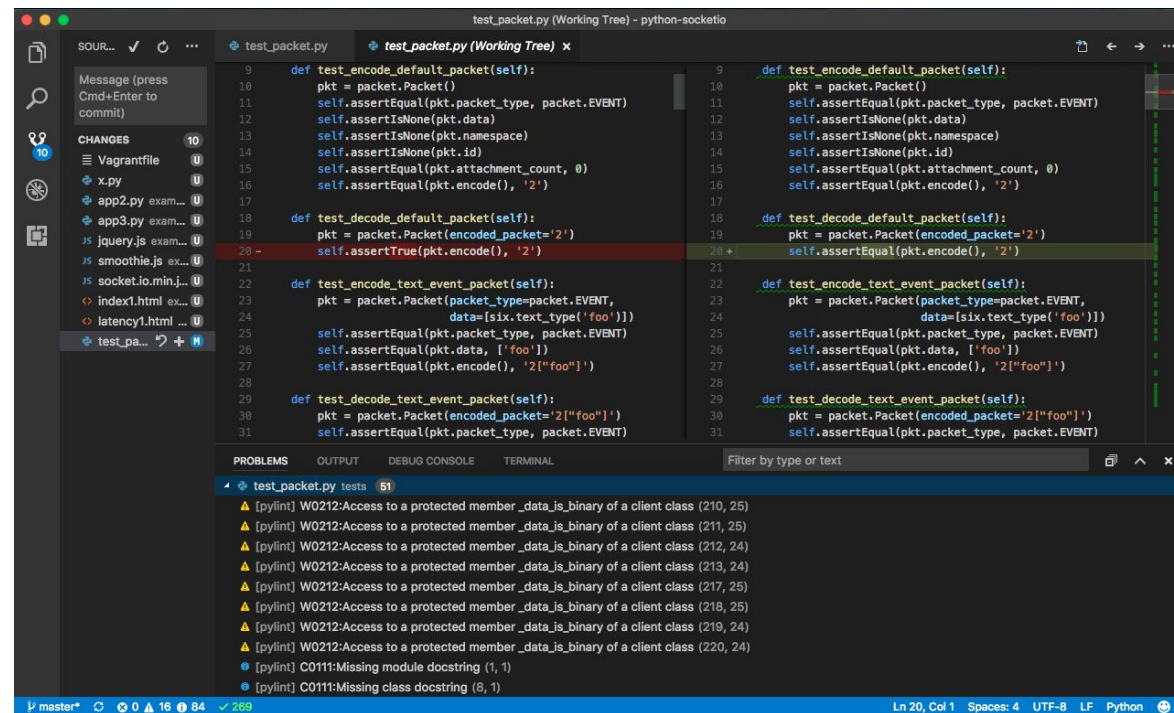
Atom

- Тип: редактор кода
- Сайт:
 - <https://atom.io/>
- Цена:
 - Бесплатно
- Плюсы:
 - Кроссплатформенность
 - Легковесный

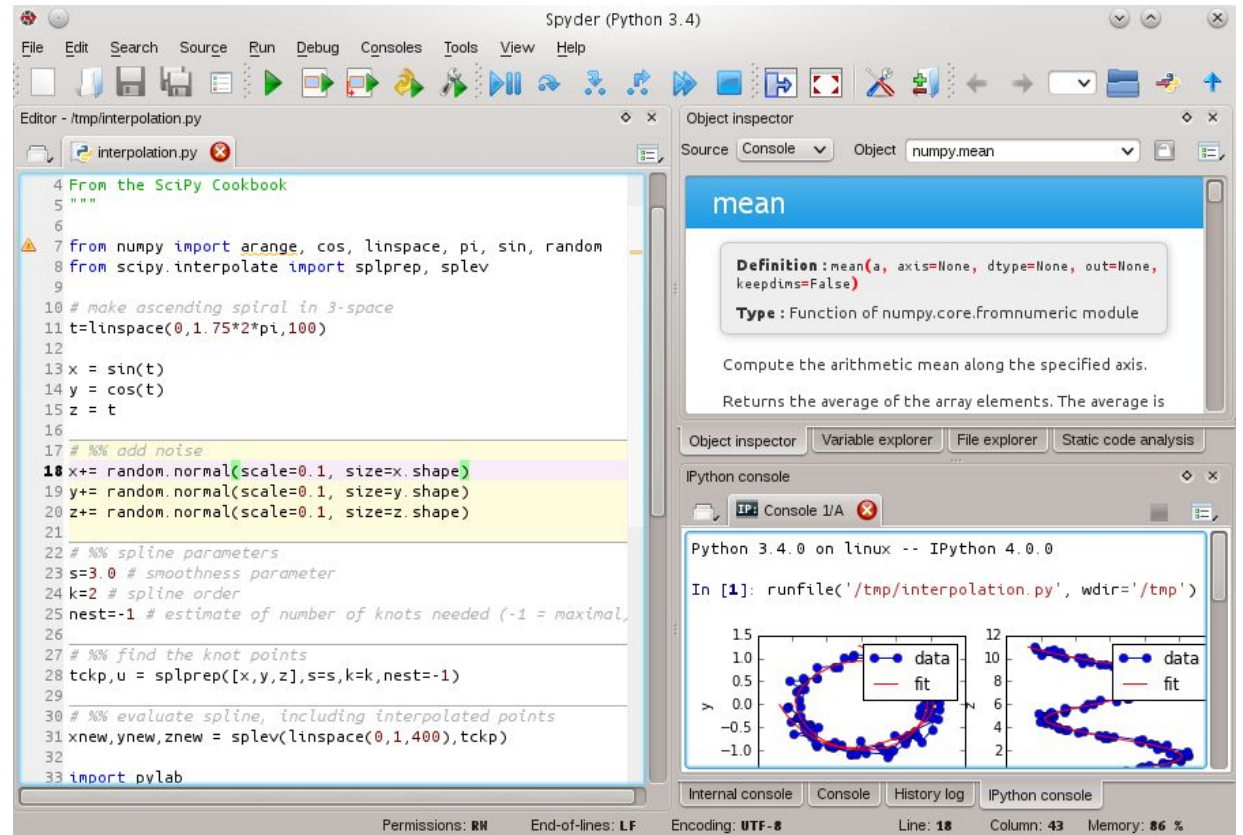


Visual Studio Code

- Тип: редактор кода
- Сайт:
 - <https://code.visualstudio.com/>
- Цена:
 - Бесплатно
- Плюсы:
 - Кроссплатформенность
 - Много поклонников
 - Легко кастомизировать
 - Легковесный
- Комментарий: неплохой инструмент в качестве замены стандартного текстового редактора. Для полноценной работы необходимо кастомизировать

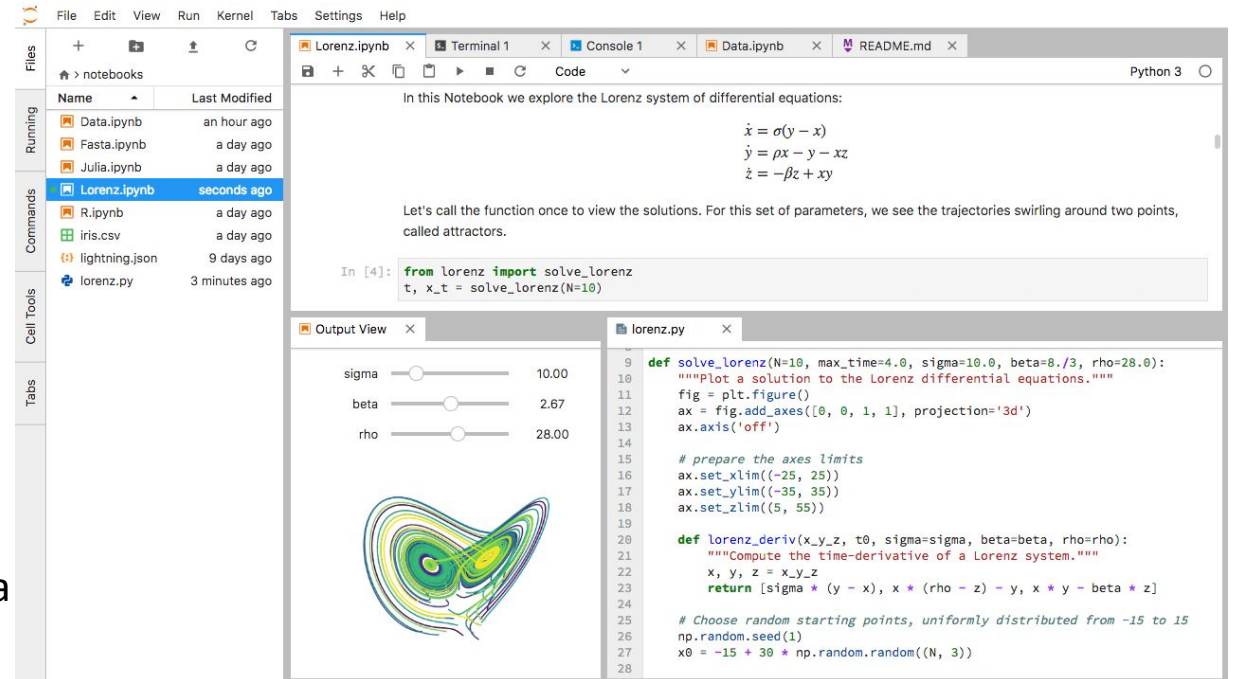


- Тип: IDE
- Сайт:
 - <https://www.spyder-ide.org/>
- Цена:
 - бесплатно
- Плюсы:
 - В комплекте сразу идет Anaconda
 - Много поклонников
 - Легковесная
 - Много функционала «из под коробки»
- Минусы:
 - Меньший функционал по сравнению с другими IDE. Можно рассматривать как инструмент под определенную задачу, а не как основной рабочий инструмент
- Комментарий: целевая аудитория Data scientist'ы. Отличительной особенностью Spyder является наличие проводника переменных. Он позволяет просмотреть значения переменных в форме таблицы прямо внутри IDE.



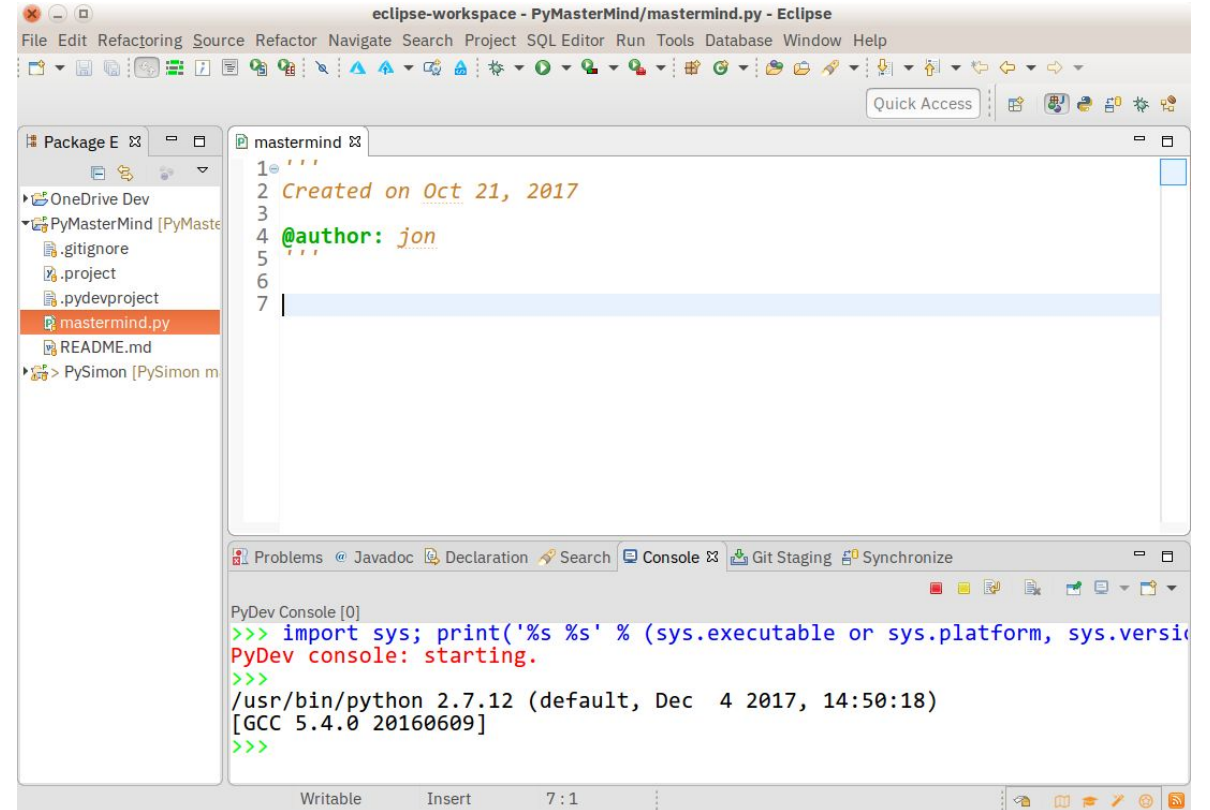
Jupyter

- Тип: IDE
- Сайт:
 - <https://jupyter.org/>
- Цена:
 - бесплатно
- Плюсы:
 - Кроссплатформенность
 - Есть web-интерфейс, а также нативное приложение
 - Популярна
 - Лучшая платформа для работы в области data science
- Минусы:
 - Нет некоторых мощных функций
- Комментарий: встречается в некоторых компаниях установленным по умолчанию



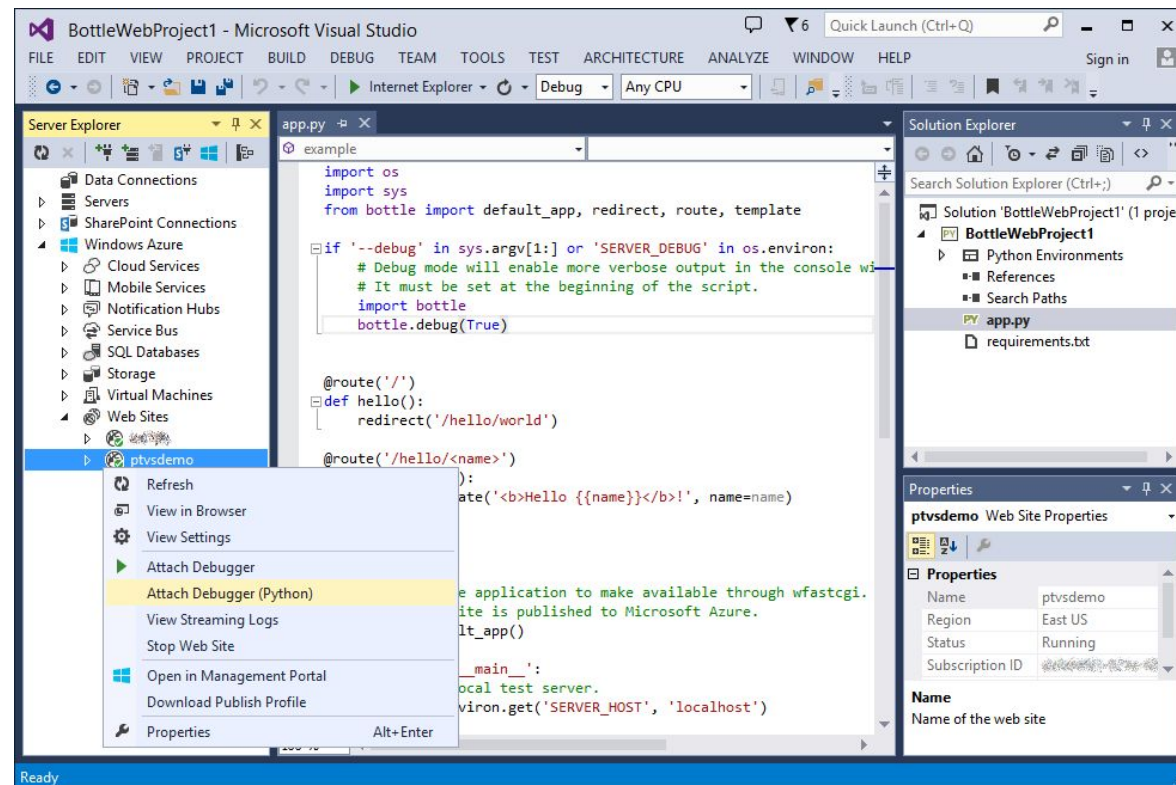
Eclipse + PyDev

- Тип: IDE
- Сайт:
 - <https://www.eclipse.org/>
 - <https://www.pydev.org/>
- Плюсы:
 - Кроссплатформенность
- Цена:
 - Бесплатно
- Минусы:
 - Трудна для начинающего пользователя
- Комментарий: по сути является open-source IDE для разработки на Java. Для работы с Python необходимо установить расширение pydev



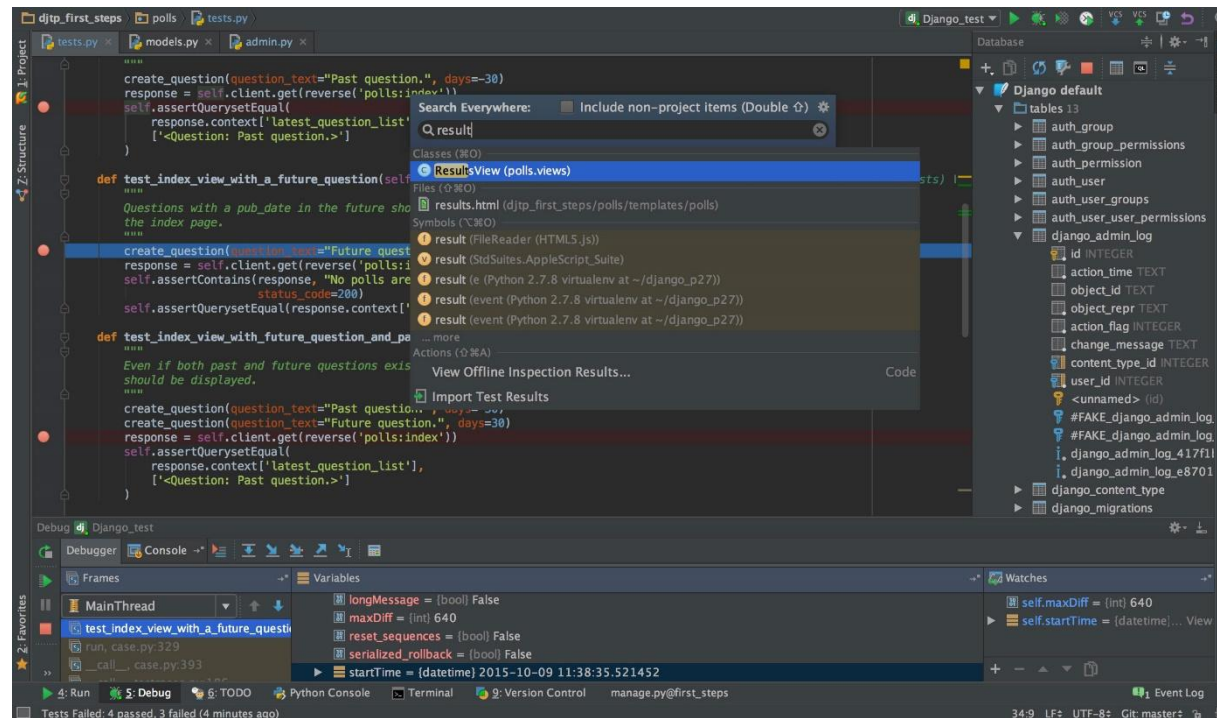
Visual Studio

- Тип: IDE
- Сайт:
 - <https://visualstudio.microsoft.com/ru/vs/>
- Цена:
 - Community – бесплатно
 - Professional - \$45 в месяц
- Плюсы:
 - Кроссплатформенность
 - Много поклонников (особенно среди C/C++/C# разработчиков)
 - Очень много функционала «из под коробки»
- Минусы:
 - Не кроссплатформенна (Windows, MacOS)
 - Как правило, оверхед, если вы не пишете на c/c++/c#
 - Если надо купить, то в Беларуси это не всегда просто (из собственного опыта)
- Комментарий: для разработки на Python будет слишком тяжелым решением



PyCharm

- Тип: IDE
- Сайт:
 - <https://www.jetbrains.com/pycharm/>
- Цена:
 - Community – бесплатно
 - Professional – \$8,90 в месяц
- Плюсы:
 - Кроссплатформенность
 - Много поклонников
 - Легковесная
 - Много функционала «из под коробки»
- Минусы:
 - В community нет поддержки работы с Python фреймворками
 - Немного требовательна к «железу»
- Комментарий: наверное, самый популярный инструмент для разработки на Python. Личная рекомендация выбрать его в качестве своего рабочего инструмента



Вопрос-ответ

? Чем отличается PyCharm Community от PyCharm Professional?

	PyCharm Professional Edition	PyCharm Community Edition
Функциональный редактор Python	✓	✓
Инструмент запуска тестов и графический отладчик	✓	✓
Навигация по коду и рефакторинг	✓	✓
Инспекции кода	✓	✓
Поддержка систем контроля версий	✓	✓
Инструменты для научных вычислений	✓	
Веб-разработка	✓	
Веб-фреймворки Python	✓	
Python-профилировщик	✓	
Возможности удаленной разработки	✓	
Поддержка баз данных и SQL	✓	



Вопрос-ответ

? Обязательно ли работать в IDE?

✓ Не обязательно. Но IDE – инструмент, который совмещает в себе много инструментов, которыми программист пользуется в повседневной работе. IDE – удобная и быстрая работа с комплексом необходимого функционала

? За IDE нужно платить?

✓ Есть бесплатные инструменты (VS code, PyCharm Community Edition), есть платные (PyCharm Professional Edition). Вопрос лишь в ваших предпочтениях, но, как правило, IDE стоят не дорого (например, PyCharm стоит около 8\$ в месяц) и это основной инструмент, который увеличивает продуктивность программиста, что позволяет работать быстрее, заработать больше

