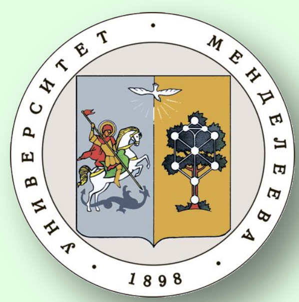
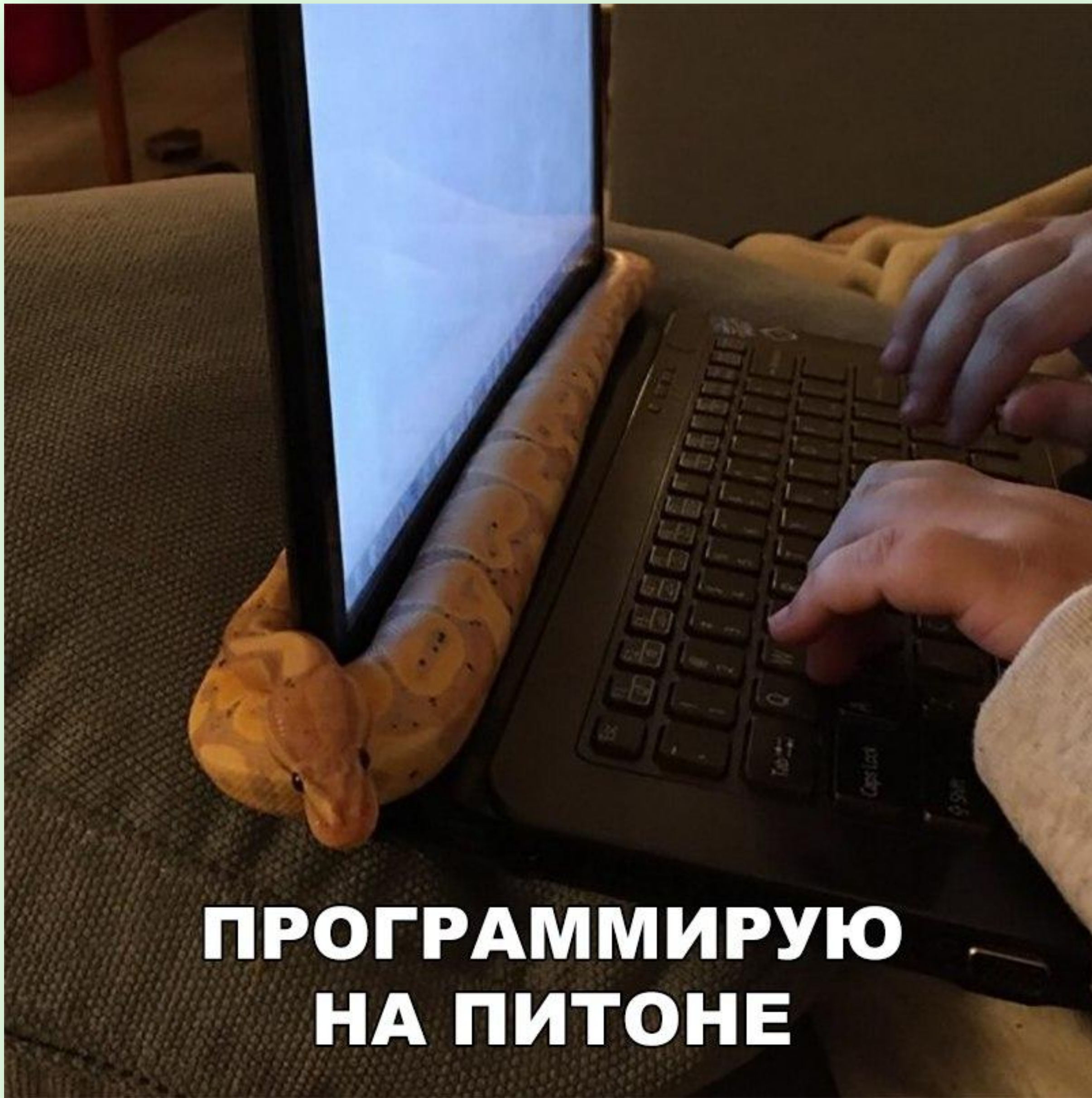


Информационные технологии

Основы программирования на Python 3

Каф. ИКТ РХТУ им. Д.И. Менделеева
Ст. преп. Васецкий А.М.

Москва, 2018


ПРОГРАММИРУЮ
НА ПИТОНЕ

Лекция 1. Основы работы

1. Введение
2. Интерпретатор Python и среды разработки
3. Основные понятия
4. Анализаторы кода
5. Структура кода
6. Модули
7. Доступ к документации

Список источников. Книги

1. Изучаем Python. Программирование игр, визуализация данных, веб-приложения. — СПб.: Питер, 2017. — 496 с.: ил. — (Серия «Библиотека программиста»).
2. Рейтц К., Шлюссер Т. Автостопом по Python. — СПб.: Питер, 2017. — 336 с.: ил. — (Серия «Бестселлеры O'Reilly»).
3. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с., ил.
4. Прохоренок Н. А., Дронов В. А., «Python 3. Самое необходимое» — СПб.: БХВ-Петербург. – 2016, 464 с
5. Прохоренок Н.А. Дронов В. А., «Python 3 и PyQt. Разработка приложений» — СПб.: БХВ-Петербург. – 2016, 832 с.
6. Любанович Б: Простой Python. Современный стиль программирования. – СПб.: Питер, – 2016, 480 с.
7. Васецкий А. М., Красильников И.В., Информационные технологии. Введение в язык программирования Python : учеб. пособие. – М. : РХТУ им. Д. И. Менделеева, 2019. – 140 с.

Сайты

1. https://docs.python.org/3/ – Оригинальная документация
2. http://pythonicway.com
3. http://pythonz.net
4. https://tproger.ru/tag/python/
5. https://pythonworld.ru
6. http://pythontutor.ru
7. https://pythoner.name – перевод документации
8. https://wiki.python.org/moin/PythonEditors - список редакторов Python

Индекс языков программирования

Oct 2019	Oct 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.884%	-0.92%
2	2		C	16.180%	+0.80%
3	4	↑	Python	9.089%	+1.93%
4	3	↓	C++	6.229%	-1.36%
5	6	↑	C#	3.860%	+0.37%
6	5	↓	Visual Basic .NET	3.745%	-2.14%
7	8	↑	JavaScript	2.076%	-0.20%
8	9	↑	SQL	1.935%	-0.10%
9	7	↓	PHP	1.909%	-0.89%
10	15	↑↑	Objective-C	1.501%	+0.30%
11	28	↑↑	Groovy	1.394%	+0.96%
12	10	↓	Swift	1.362%	-0.14%
13	18	↑↑	Ruby	1.318%	+0.21%
14	13	↓	Assembly language	1.307%	+0.06%
15	14	↓	R	1.261%	+0.05%
16	20	↑↑	Visual Basic	1.234%	+0.58%
17	12	↓↓	Go	1.100%	-0.15%
18	17	↓	Delphi/Object Pascal	1.046%	-0.11%
19	16	↓	Perl	1.023%	-0.14%
20	11	↓↓	MATLAB	0.924%	-0.39%

https://tiobe.com/tiobe-index/

1. Введение

- **Python** – язык высокого уровня
- Синтаксис языка Python минималистический и гибкий.
- На нём можно составлять простые и эффективные программы.
- Стандартная библиотека для этого языка содержит множество полезных функций, что значительно облегчает процесс создания программ.
- Язык Python поддерживает несколько парадигм программирования, включая **структурное**, **объектно-ориентированное** и **функциональное** программирование.

Применение Python

- разработка сценариев для работы с Web и Intenet-приложений;
- сетевое программирование;
- средства поддержка технологий HTML и XML;
- приложения для работы с электронной почтой и поддержки Intemet,
- Протоколов приложения для обслуживания всевозможных баз данных;
- программы для научных расчетов;
- приложения с графическим интерфейсом;
- создание игр и компьютерной графики,

Особенности Python

- Python относится к интерпретируемым языкам программирования, т.е. код выполняется с помощью специальной программы-интерпретатора.
- Интерпретатор выполняет программный код построчно (с предварительным анализом).
- Иногда исходный программный код компилируется в промежуточный код, и этот промежуточный код выполняется непосредственно интерпретатором
- Ошибки выявляются на этапе выполнения и скорость работы меньше, чем компилируемых языках.
- Преимуществом является б***о***льшая скорость разработки, т.к. меньше времени тратится на компиляцию

Реализации Python

- **Python** – это спецификация для языка, которая может быть реализована множеством разных способов
- **CPython** (http://www.python.org) – базовая реализация Python, написанная на языке С.
- **Stackless Python** (https://bitbucket.org/stackless-dev/stackless/wiki/Home) – это обычный вариант CPython Эта версия языка имеет патч, отвязывающий интерпретатор Python от стека вызовов, что позволяет изменять порядок выполнения кода

Реализации Python (продолжение)

- **PyPy** (http://pypy.org/) – это интерпретатор Python, реализованный в ограниченном подмножестве статически типизированных языков Python (которое называется RPython), что позволяет выполнить оптимизацию.
- В данный момент PyPy быстрее CPython более чем в пять раз.

Jython

- **Jython** (http://www.jython.org/) – реализация интерпретатора Python, компилирующая код Python в байт-код Java, который затем выполняется JVM *(Java Virtual Machine)*. Дополнительно он может импортировать и использовать любой класс Java в качестве модуля Python

Jython в данный момент поддерживает версии Python вплоть до Python 2.7 (http://bit.ly/jython-supports-27)

Python под .NET

- **IronPython** (http://ironpython.net/) – это реализация Python для фреймворка .NET. Она может использовать библиотеки, написанные как на Python, так и с помощью .NET, а также предоставлять доступ к коду Python другим языкам фреймворка .NET.

IronPython поддерживает версию Python 2.7

- **PythonNet** – (http://pythonnet.github.io/) – пакет, который предоставляет почти бесшовную интеграцию оригинально установленного Python и общеязыковой среды выполнения .NET.

Такой подход противоположен подходу, которым пользуется IronPython. Т.е. **PythonNet** и **IronPython** дополняют друг друга.

Прочие реализации

- **Skulpt** (http://www.skulpt.org/) – реализация Python на JavaScript.

Skulpt поддерживает большую часть функциональности версий Python 2.7 и Python 3.3.

- **MicroPython** (https://micropython.org/) – реализация Python 3, оптимизированная для микроконтроллеров. Поддерживает 32-битные процессоры ARM, имеющие набор инструкций Thumb v2.

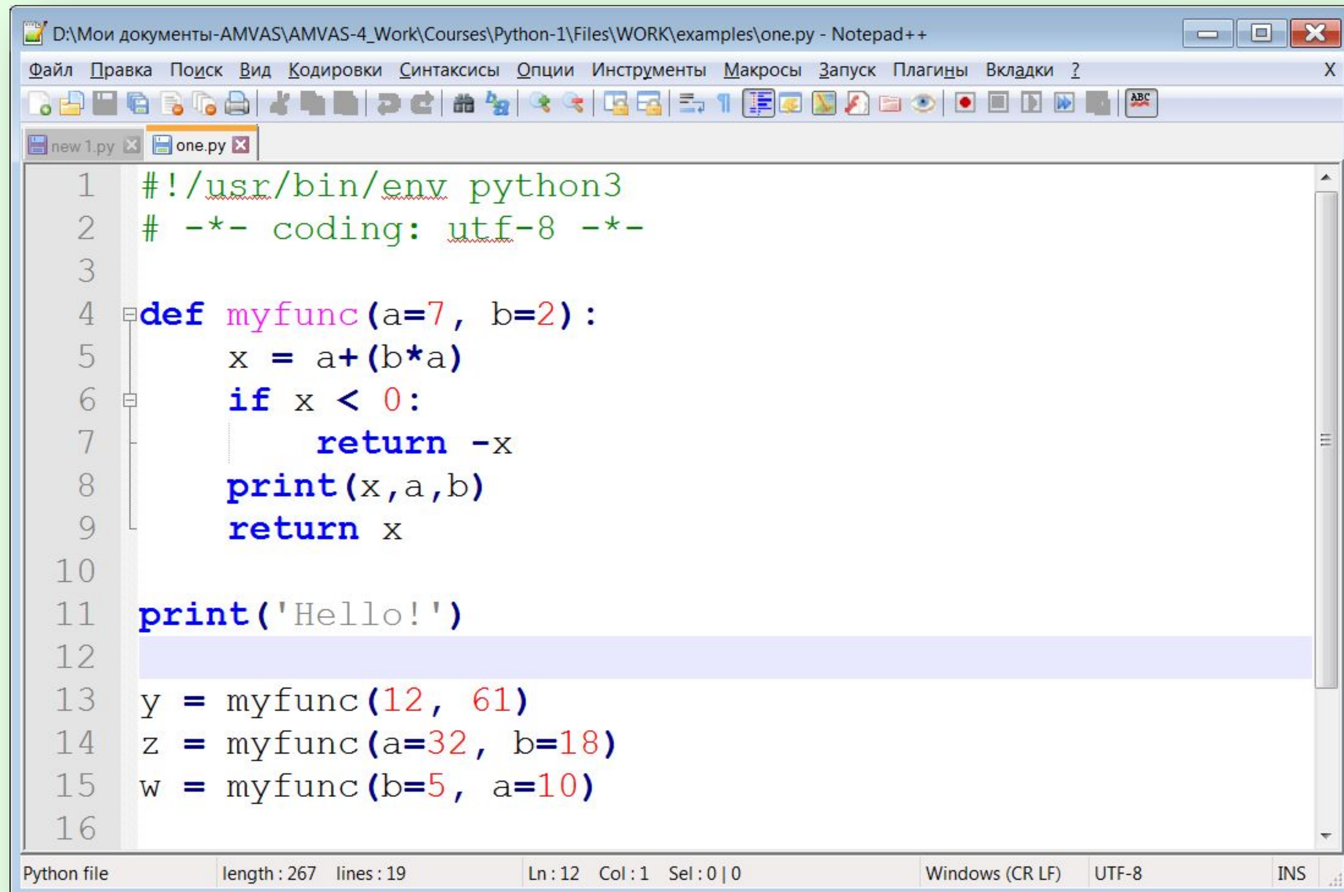
Python в разных операционных системах

- Поддержка Python уже установлена на большинстве компьютеров Linux
- Python в системах Windows и MacOS требует установки.
  - ✔ Требуется скачать соответствующий пакет с http://python.org/download/ и установить его.
  - ✔ Для научных расчётов наиболее удобен дистрибутив **«Anaconda»** https://www.anaconda.com/download/

Примечание: ***Anaconda*** не любит, когда имя пользователя написано русскими буквами, поэтому придётся создать нового пользователя, в имени которого все буквы английские, и дальнейшую работу вести от имени этого пользователя.

Набор кода

- Сам код набирается в любом текстовом редакторе, либо в среде программирования.
- Notepad++ (пример)



Проект "Jupyter"

- **Jupyter Notebook** – командная оболочка для интерактивных вычислений. Может использоваться не только с ***Python***, но и с другими языками программирования: ***Julia, R, Haskell и Ruby***.
- Часто используется для работы с данными, статистическим моделированием и машинным обучением.
- Кроме того, это удобный инструмент для создания красивых аналитических отчетов, поскольку он позволяет хранить вместе код, изображения, комментарии, формулы и графики

Подробнее см.
http://devpractice.ru/python-lesson-6-work-in-jupyter-notebook/
https://tproger.ru/translations/jupyter-notebook-python-3/
https://habrahabr.ru/company/wunderfund/blog/316826/
http://jupyter.org/

Пример документа Jupyter

IP[y]: Notebook spectrogram Last Checkpoint: a few seconds ago (autosaved)

IPython (Python 3)

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

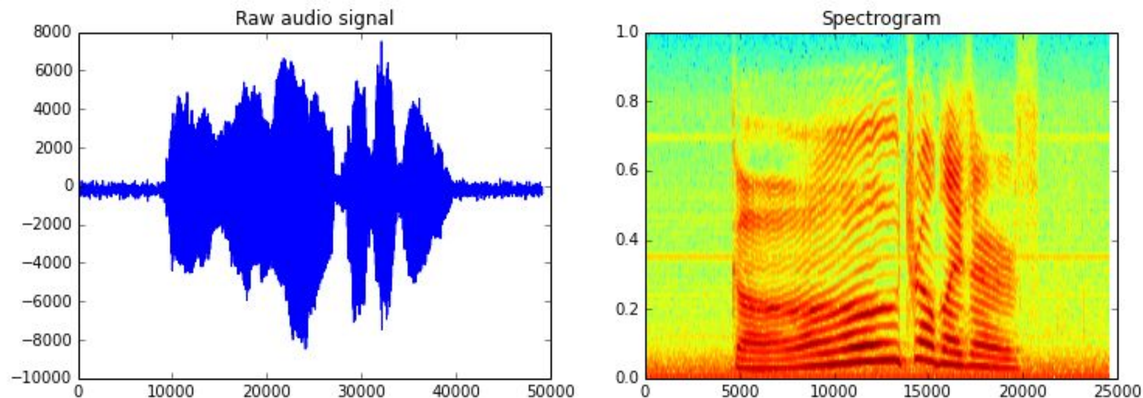
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

In [1]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

In [2]:
```
%matplotlib inline
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```



2. Интегрированные среды разработки

IDE: Интегрированная среда разработки (англ. Integrated Development Environment)

- ✔ PyCharm
- ✔ PyDev
- ✔ WingWare
- ✔ Komodo IDE
- ✔ Eric
- ✔ Spyder
- ✔ IDLE

P.S. …есть и другие https://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Среда программирования PyCharm

https://www.jetbrains.com/pycharm/

- Создатель: **JetBrains**
- Исходный код: **Закрытый**
- ОС: **Windows, Linux, MacOS**
  - ✔ Возможности интегрированного модульного тестирования, проверки кода, интегрированного контроля версий, инструменты переработки (рефакторинга) кода, набор инструментов для навигации проекта, выделения и автоматического завершения. Поддержка ряда сторонних фреймворков для веб-разработки, таких как Django, Pyramid, web2py, Google App Engine и Flask

Среда программирования PyDev

http://www.pydev.org/

- Плагин Python для Eclipse
- Поддерживается всеми операционными системами
- Источники находятся в свободном доступе по публичной лицензии Eclipse
  - ✔ Обработка кода, интеграция отладки Python, инструменты переработки кода и пр.
  - ✔ Возможность создания новых проектов Django, выполнение команд Django при помощи горячих клавиш и использование отдельной конфигурации запуска только для Django.

Среда программирования WingWare

https://wingware.com/

- Создатель: **WingWare**
- Исходный код: **Закрытый**
- ОС: **Windows, Linux, MacOS**
  - ✔ Есть мощный инструмент отладки, который позволяет устанавливать контрольные точки, возможность пошагового выполнения кода, проверка данных, удалённая отладка и отладка шаблонов Django.
  - ✔ Поддержка **matplotlib**, с автоматическим обновлением графиков. Также предоставляется доработка кода, подсветка синтаксиса, исходный браузер, графический отладчик и поддержка систем управления версиями.

Среда программирования Komodo IDE

- Создатель: **ActiveState**
- Исходный код: **Закрытый**
- ОС: **Windows, Linux, MacOS**
  - ✔ Поддержка Django: подсветка синтаксиса и завершение кода для шаблонов.
  - ✔ Содержит базовые функции, такие как переработку (рефакторинг) кода, автозаполнение, calltips (подсказки), сопоставление скобок, браузер кода, переход к определению, графическая отладка, многопроцессная отладка, многопоточная отладка, конфигурация точки остановки, профилирование кода, интеграция со сторонними библиотеками, такими как pyWin32.
  - ✔ Интеграция менеджера пакетов, отслеживание изменений, инструмент просмотра заметок, быстрые закладки, переход ко всему (Commando) и многое другое.

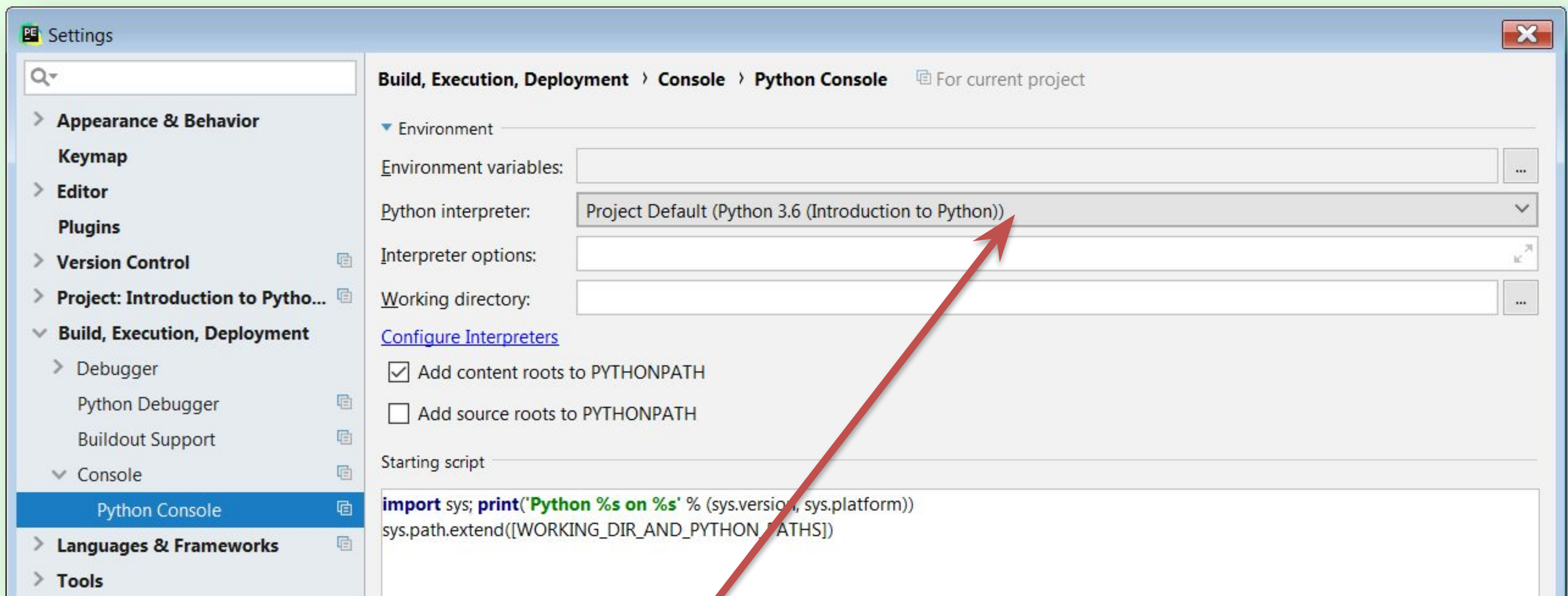
Eric IDE

https://eric-ide.python-projects.org/

- Создатель: **Detlev Offenbach**
- Исходный код: **Открытый**
- ОС: **Windows**
  - ✔ Содержит отладчик Python и Ruby, покрытие кода, автоматическая проверка кода, оболочку Python и Ruby, браузер класса и пр.
  - ✔ Имеются функции для совместного редактирования. Диалоги Regex и Qt, опции для создания сторонних приложений в редакторе, диаграммы приложения, возможности управления проектами, и интерактивная оболочка Python.
  - ✔ Многоязычный пользовательский интерфейс, который включает в себя и русский
  - ✔ Контроль версии для Subversion, Mercurial и Git, использование объявлений в плагинах, и многое другое.

Конфигурирование IDE

- Поскольку обновления Python выходят достаточно часто, при настройке среды разработки важно указать путь к вашей текущей версии Python



Настройка пути интерпретатора Python в pyCharm

3. Основные понятия
Python Enhancement Proposals (PEP)

- Одной из причин, почему код Python прост для понимания, является наличие информативного руководстве по стилю написания кода (представлено в двух Предложениях по развитию Python PEP 20 и PEP 8) и питонских идиомах.

PEP 8

- Фактически представляет собой руководство по стилю написания кода Python. Здесь рассматриваются
  - соглашения по именованию,
  - структура кода,
  - пустые области (табуляция против пробелов)
  - и другие аналогичные темы.
- Все сообщество Python старается следовать принципам, изложенным в этом документе.
- С помощью программы **pycodestyle (pep8)**

(https://github.com/PyCQA/pycodestyle), которая запускается из командной строки, можно проверить код на соответствие принципам PEP 8.

PEP 20 – The Zen of Python

- (https://www.python.org/dev/peps/pep-0020/) (набор принципов для принятия решений в Python) всегда доступен по команде import this в оболочке Python

1. Красивое лучше, чем уродливое.
2. Явное лучше, чем неявное.
3. Простое лучше, чем сложное.
4. Сложное лучше, чем запутанное.
5. Одноуровневое лучше, чем вложенное.
6. Разреженное лучше, чем плотное.
7. Читаемость имеет значение.
8. Особые случаи не настолько особые, чтобы нарушать правила.
9. При этом практичность важнее безупречности.
10. Ошибки никогда не должны замалчиваться.
11. Если не замалчиваются явно.
12. Встретив двусмысленность, отбросьте искушение угадать.
13. Должен существовать один – и желательно только один – очевидный способ сделать это.
14. Хотя он поначалу может быть и не очевиден, если вы не голландец.
15. Сейчас лучше, чем никогда.
16. Хотя никогда зачастую лучше, чем прямо сейчас.
17. Если реализацию сложно объяснить – идея плоха.
18. Если реализацию легко объяснить – идея, возможно, хороша.
19. Пространства имен – отличная штука! Будем делать их побольше!

Явное лучше чем неявное

- Неявно передаются переменные **(плохо!)**

```
def make_dict(*args):
    x, y = args
    return dict(**locals())
```

- Явно указываются принимаемые и возвращаемые значения **(хорошо!)**

```
def make_dict(x, y):
    return {'x': x, 'y': y}
```

Разреженное лучше, чем плотное

- В каждой строке рекомендуется размещать только одно выражение

```
print(' Так ' ); print('неверно')
```

```
print('Так ')
print('верно')
```

Неверно:

```
if (<условие1> and <условие2>): <инструкции>
```

Верно:

```
cond1 = <условие1>
cond2 = <условие2>
if cond1 and cond2: <инструкции>
```

Философия Python сосредоточена во фразе «Мы все – ответственные пользователи»

- Любой клиентский код может переопределить свойства и методы объекта. Однако сообщество Python предпочитает полагаться на набор соглашений, которые указывают, к каким элементам нельзя получить доступ напрямую.
- Основным соглашением для закрытых свойств и деталей реализации является обязательное добавление к именам подобных элементов **нижнего подчеркивания** (например, ***sys._getframe***).
- Если клиентский код в нарушение этого правила получает доступ к отмеченным элементам, то считается, что любое неверное поведение или проблемы вызваны именно клиентским кодом.

Структура программы

1. Программы делятся на модули.
2. Модули содержат инструкции.
3. Инструкции состоят из выражений.
4. Выражения создают и обрабатывают объекты.

Синтаксис языка Python по сути построен на инструкциях и выражениях. Выражения обрабатывают объекты и встраиваются в инструкции. Инструкции представляют собой более крупные логические блоки

Структура кода на Python

- Python отличается от других языков тем, что пробелы в нем используются для того, чтобы задать структуру программы.
- Любая программа более читаема, если её строки достаточно короткие. Рекомендуемая максимальная длина строки равна 80 символам
- Формат инструкций:

  <Основная инструкция>:

  <Вложенный блок инструкций>
- Конец строки является концом инструкции
- Конец отступа – это конец блока
- Перенос строки – символ **"\"**
- См. Руководство для оформления программ на Python
- Обычно на каждой строке располагается одна инструкция, но для большей компактности **иногда** можно записать несколько инструкций в одной строке, разделив их **";"**

См. также

https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html

https://www.calculate-linux.org/main/ru/python_style_guide

http://www.python.org/doc/essays/styleguide.html

Комментарии

- Комментарии предназначены для вставки пояснений в текст программы, интерпретатор полностью их игнорирует. Внутри комментария может располагаться любой текст, включая инструкции, которые выполнять не следует.

 # Пример комментария
S = 1 # и это комментарий

Начало файла

#!/usr/bin/env python3
 # -*- coding: utf-8 -*-

Это связано с переносимостью скриптов. Путь к python на разных системах может отличаться, а вот путь к ***env*** на всех системах неизменный. Поэтому, вызывая ***env*** и передавая ей в качестве аргумента нужный интерпретатор, можно быть уверенным, что скрипт будет запущен вне зависимости от того, где на самом деле находится интерпретатор (главное, чтобы он был в PATH).

Вторая строка показывает систему кодировки кода

Продолжение длинной строки

- Если строка длиннее принятого в PEP8 значения (80) то рекомендуется заключить элементы в круглые скобки. Если интерпретатор Python встретит незакрытую круглую скобку в одной строке, он будет присоединять к ней следующие строки до нахождения закрытой скобки. То же поведение верно для фигурных и квадратных скобок

```
long_string = (
' Это длинная строка'
' с продолжением'
)
```

Примечание: по возможности следует избегать таких строк, чтобы не вредить читабельности

Временная переменная

- Если необходимо присвоить какое-то значение, но сама переменная не нужна, воспользуйтесь двойным подчеркиванием **(__)**.
- Иногда рекомендуют для этих целей использовать одинарное подчёркивание **(_)**, однако проблема в том, что оно зачастую применяется как псевдоним для функции ***gettext.gettext()*** и как интерактивное приглашение сохранить значение последней операции, что может вызвать перезапись этой переменной.

4. Анализаторы кода

- Поскольку в Python используется интерпретатор, а не компилятор, то многие ошибки обнаруживаются уже в ходе выполнения.
- Поэтому имеет смысл использовать анализаторы кода.
- Например
  - ✔ https://www.pylint.org/
  - ✔ https://pypi.python.org/pypi/pyflakes
- **PyLint** можно интегрировать в **IDE**, такие как **PyCharm** и текстовые редакторы
- Подробнее об анализаторах кода см.

https://python-scripts.com/code-analysis

5. Модули

Один из основных уровней абстракции. Уровни абстракции позволяют программисту разбивать код на части, которые содержат связанные данные и функциональность.

Импортирование модуля производится командой ***import***

Модули могут быть **встроенными**, **сторонними** и **внутренними** (пользовательскими).

Примеры встроенных модулей – **os, sys**.

Примеры сторонних пакетов в среде – Requests, NumPy

Пример:

```
import sys     #Встроенный модуль
import matplotlib.pyplot as plt # Сторонний
```

Примеры пользовательских модулей:

- Модуль взаимодействия с пользователем
- Модуль записи в файл
- Модуль считывания из файла
- Модуль построения графиков
- Управляющий модуль
- Расчётный модуль
- Модуль функций

Именование модулей

- Модулям присваиваются короткие имена, которые начинаются со строчной буквы.
- Не нужно использовать символы типа (.) или (?)
- Также не следует злоупотреблять (_), поскольку можно спутать с именем переменной

Рекомендации

- Не используйте расширения имен файлов в инструкциях ***import*** и ***reload***.
- Не указывайте полные пути к файлам и расширения в инструкциях ***import.*** Например, следует писать ***import mod***, а не ~~***import mod.py***~~).

Упаковка

Python предоставляет достаточно понятную систему упаковки, которая расширяет механизм модулей так, что он начинает работать с каталогами.

Любой каталог, содержащий файл ***__init__.py***, считается пакетом Python.

Каталог высшего уровня, в котором находится файл ***__init__.py***, является ***корневым пакетом***.

Разные модули пакетов импортируются аналогично простым модулям, но файл ***__init__.py*** при этом будет использован для сбора всех описаний на уровне пакета.

Признаком хорошего тона является поддержание файла ***__init__.py*** пустым, когда модули и подпакеты пакета не имеют общего кода.

Пакетирование модулей

- Модули можно группировать по назначению и располагать в соответствующих папках. Например:

sound/	Верхний уровень
__init__.py	Инициализатор пакета
formats/	подпакет конверсии форматов
__init__.py	
wavread.py	
wavwrite.py	
aiffread.py	
effects/	Подпакет эффектов
__init__.py	
echo.py	
surround.py	
reverse.py	
filters/	Подпакет фильтров
__init__.py	
equalizer.py	
vocoder.py	
karaoke.py	

См. https://docs.python.org/3/tutorial/modules.html#packages

Импорт модулей из пакетов

Файл **__init__.py** нужен, чтобы показать, что папка содержит модули. Сам файл может быть пустым, либо содержать инициализацию переменной **__all__** или код инициализации модулей

Импорт подмодуля: **import sound.effects.echo**
Обращение к нему: **sound.effects.echo.echofilter (input, output, delay=0.5, atten=5)**

Альтернативно: **from sound.effects import echo**
Обращение к нему: **echo.echofilter (input, output, delay=0.5, atten=5)**

Альтернативно: **from sound.effects.echo import echofilter**
Обращение к нему: **echofilter(input, output, delay=0.5, atten=5)**

Импорт модулей (продолжение)

Пусть в файле \effects\ __init__.py определена:

__all__ = ["echo", "surround", "reverse"]

Тогда

from sound.effects import *

импортирует все три названных подмодуля

Если же не определить **__all__** , тогда НЕ импортируются подмодули из **sound.effects**

Читаемость импортированных модулей

```
from modu import *      #Непонятный код
x = sqrt(4)             #Откуда что?

from modu import sqrt   #Импортируем
                        # конкретную функцию
x = sqrt(4)

import modu             # Импортируем модуль
x = modu.sqrt(4)        # И ссылаемся на
# функцию sqrt внутри него
```

Импорт отдельных подмодулей

Из вышеприведённого примера можно импортировать отдельные подмодули в ***sounds/effects/surround.py*** с использованием относительных путей:

#импорт sound/effects/echo.py

from . import echo

#импорт sound/formats

from .. import formats

#импорт sound/filters/equalizer.py

from ..filters import equalizer

Структура, это главное

- Следует избегать большого количества циклических зависимостей

Например, если файл ***one.py*** зависит от файла ***two.py***, который зависит от файла ***one.py***, то это может вызвать ошибку **ImportError**.

- Избыточное использование глобального состояния или контекста

Например, вместо явной передачи данных используются глобальные переменные, которые модифицируются другими структурными единицами. Отследить, какие именно внесли критические изменения в такие переменные очень сложно.

Читабельность важнее скорости

- **Спагетти-код.** Вложенные условия *if*, расположенные на нескольких страницах подряд, и циклы ***for***, содержащие большое количество скопированного текста.

Подробнее см. https://habrahabr.ru/post/187154/

- **Равиоли-код** состоит из множества небольших логических фрагментов, зачастую классов или объектов, которые не имеют хорошей структуры. Т.е. это чересчур мелкоструктурированный код.

ООП

- В **Python** все элементы являются **объектами** и работают как объекты.
- **Функции** являются объектами первого класса. Функции, классы, строки и даже типы считаются в Python объектами: все они имеют тип, их можно передать как аргументы функций, они могут иметь методы и свойства.
- Однако, в отличие от Java, в Python парадигма объектно-ориентированного программирования (ООП) не основная. Проект, написанный на Python, вполне может и **не быть** объектно-ориентированным

Функциональное программирование

- Парадигма, которая в своей чистейшей форме не имеет операторов присваивания и побочных эффектов и вызывает функции одну за другой, чтобы выполнить задачу.
- В Python имеются инструменты, которые позволяют заниматься функциональным программированием, хотя он не является чисто функциональным языком.
- Подробнее с этим аспектом **Python** можно ознакомиться http://bit.ly/functional-programming-python

6. Доступ к документации

- Документ

Пуск>Программы (Все программы)>Python 3.x> Python 3.x Manuals.

- Сервер документов (откроется в браузере)

Пуск>Программы (Все программы)>Python 3.x> Python 3.x Module Docs.

- Там находятся страницы с описанием всех классов, функций и констант, объявленных в стандартных модулях
- Чтобы завершить работу, следует переключиться в его окно, ввести команду ***q*** и нажать ***<Enter>.*** А введенная там команда ***b*** повторно выведет страницу со списком модулей.
- Оригинальную документацию онлайн см. https://docs.python.org/3/

Функция help

- С помощью функции ***help()*** можно получить документацию по конкретной функции и по всему модулю сразу.
- Для вывода помощи по классу ***str*** в модуле ***builtins*** набрать в консоли:

>>>help(str)

- Сначала импортируем модуль

>>> import builtins

- А затем выведем справку по нему

>>> help(builtins)

Строки документирования

- для комментирования большого фрагмента кода используются утроенные кавычки или утроенные апострофы. Такие строки не являются комментариями в полном смысле этого слова. Вместо комментирования фрагмента создается объект строкового типа, который сохраняется в атрибуте ***__doc__.***
- Функция ***help()*** при составлении документации получает информацию из этого атрибута.

Пример документирования

```
# -*- coding: utf-8 -*-
"""описание модуля testd.py """
def func():
    """ Описание функции"""
    pass
```

Подключаем модуль testd и выводим документацию:

```
# -*- coding: utf-8 -*-
import testd
help(testd)
```

Результат в консоли

Help on module testd:

NAME

testd - описание модуля testd.py

FUNCTIONS

func()

Описание функции

FILE

c:\users\пользователь\pycharmprojects\modulestest01\testd.py

Атрибут __doc__:

```
import testd
print(testd.__doc__)
print(testd.func.__doc__)
```

Вывод:

описание модуля testd.py
Описание функции

Список всех атрибутов

- Список всех атрибутов, имеющихся у любого объекта, можно получить с помощью функции ***dir(X)***

Например: ***print(dir(print))*** **выдаст**

['__call__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__self__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__text_signature__']

Спасибо за внимание