Python

Урок 1

Введение в программирование на языке Python

Что такое Python?

Python (чаще всего произносится «питон», хотя некоторые предпочитают произносить как «пайтон») — мощный и простой для изучения язык программирования. В нём предоставлены проработанные высокоуровневые структуры данных и простой, но эффективный подход к объектно-ориентированному программированию. Сочетание изящного синтаксиса и динамической типизации, совмещённых с интерпретируемой сущностью, делает Python идеальным языком для написания сценариев и ускоренной разработки приложений в различных сферах и на большинстве платформ.

Вот как описывает Python создатель языка Гвидо ван Россум:

Python -- интерпретируемый, объектно-ориентированный высокоуровневый язык программирования с динамической семантикой. Встроенные высокоуровневые структуры данных в сочетании с динамическими типизацией и связыванием делают язык привлекательным для быстрой разработки приложений (RAD, Rapid Application Development). Кроме того, его можно использовать в качестве сценарного языка для связи программных компонентов. Синтаксис Python прост в изучении, в нем придается особое значение читаемости кода, а это сокращает затраты на сопровождение программных продуктов. Python поддерживает модули и пакеты, поощряя модульность и повторное использование кода. Интерпретатор Python и большая стандартная библиотека доступны бесплатно в виде исходных и исполняемых кодов для всех основных платформ и могут свободно распространяться.

История языка Python

Создание Python было начато **Гвидо ван Россумом** (Guido van Rossum) в 1991 году, когда он работал над распределенной ОС Амеба. Ему требовался расширяемый язык, который бы обеспечил поддержку системных вызовов. За

основу были взяты ABC и Модула-3. В качестве названия он выбрал Python в честь комедийных серий BBC "Летающий цирк Монти-Питона", а вовсе не по названию змеи. С тех пор Python развивался при поддержке тех организаций, в которых Гвидо работал. Особенно активно язык совершенствуется в настоящее время, когда над ним работает не только команда создателей, но и целое сообщество программистов со всего мира. И все-таки последнее слово о направлении развития языка остается за Гвидо ван Россумом.

Почему программисты используют Python?

Качество программного кода
Python — удобочитаемый язык. Программный код на языке Python легко читается, а значит, многократное его использование и обслуживание выполняется гораздо проще, чем использование программного кода на других языках сценариев. Единообразие оформления программного кода на языке Python облегчает его понимание даже для тех, кто не участвовал в его создании. Все это само по себе позволяет проще создавать качественное ПО.

Высокая скорость разработки
По сравнению с компилирующими или строго типизированными языками, такими как C, C++ и Java, Python во много раз повышает производительность труда разработчика. Объем программного кода на языке Python обычно составляет треть или даже пятую часть эквивалентного программного кода на языке C++ или Java.

Переносимость программ
Большая часть программ на языке Python выполняется без изменений на всех основных платформах. Перенос программного кода из операционной системы Linux в Windows обычно заключается в простом копировании файлов программ с одной машины на другую.

Библиотеки поддержки
В составе Python поставляется большое число собранных и переносимых функциональных возможностей, известных как стандартная библиотека. Эта библиотека предоставляет массу возможностей, востребованных в прикладных программах, начиная от поиска текста по шаблону и заканчивая сетевыми функциями. Кроме того, Python допускает расширение как за счет ваших собственных библиотек, так и за счет библиотек, созданных сторонними разработчиками.

Интеграция компонентов
Сценарии Python легко могут взаимодействовать с другими частями приложения благодаря различным механизмам интеграции. На сегодняшний день программный код на языке Python имеет возможность вызывать функции из библиотек на языке C/C++, сам вызываться из программ, написанных на языке

C/C++, интегрироваться с программными компонентами на языке Java, взаимодействовать с такими платформами, как COM и .NET, и производить обмен данными через последовательный порт или по сети с помощью таких протоколов, как SOAP, XML-RPC и CORBA.

Удовольствие
Благодаря непринужденности языка Python и наличию встроенных инструментальных средств процесс программирования может приносить удовольствие. На первый взгляд это трудно назвать преимуществом, тем не менее, удовольствие, получаемое от работы, напрямую влияет на производительность труда.

Кто в настоящее время использует Python?

- Google
- Youtube
- BitTorrent
- Instagram
- Disqus
- Dropbox
- Яндекс
- Mail.ru
- и многие, многие другие, в том числе Maya, NASA, Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, IBM...

Что можно делать с помощью Python?

- Системные скрипты
- Программы с графическим интерфейсом
- Веб-сайты и веб-приложения
- Интегрировать различные компоненты в единую систему
- Приложения баз данных
- Быстро создавать прототипы приложений
- Математические и научные вычисления
- Игры, изображения, искусственный интеллект, XML роботы и многое другое

Сильные стороны Python

- Объектно-ориентированность
- Python может использоваться и распространяться совершенно бесплатно
- Переносимость - программы python работают практически на всех основных платформах

- Мощность языка:
  - Динамическая типизация
  - Автоматическое управление памятью (сборщик бусора)
  - Модульное программирование
  - Встроенные типы объектов (списки, словари, кортежи, строки)
  - Мощные стандартные инструменты работы с объектами
  - Большая коллекция инструментов стандартной библиотеки
  - Утилиты сторонних разработчиков
- Возможность интеграции с программами C/C++
- Удобство использования - не нужно компилировать
- Простота изучения

Философия Python

```
>>> 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!
```

Что в переводе звучит примерно так:

1. Красивое лучше, чем уродливое
2. Явное лучше, чем неявное

3. Простое лучше, чем сложное
4. Сложное лучше, чем запутанное
5. Плоское 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, отбрось 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 имён — отличная штука! Будем делать их побольше!

Эти принципы на практике:
http://artifex.org/~hblanks/talks/2011/pep20_by_example.html

Версии Python 2.x и 3.x

В настоящее время большинство программного обеспечения Python написано на версии языка 2.x. Однако версии ветки 3.x уже давно выпущены и доступны для установки. Почему программисты с трудом переходят на третью версию? Потому что версии Python 2.x и 3.x не совместимы между собой и нужно вносить значительные изменения в код и прибегать к некоторым приемам, чтобы было возможно запускать приложения во второй и в третьей версиях языка.

В данном курсе мы рассмотрим обе версии и вы сможете применять ту или иную версию языка в зависимости от ситуации.

Введение в Python

Python является динамическим, интерпретируемым языком. В исходном коде не объявляются типы переменных, параметров или методов. Это делает код коротким и гибким. Python отслеживает типы всех значений во время выполнения.

В Python не надо объявлять тип переменной — просто присвойте ей значение и двигайтесь дальше. Python вызовет ошибку времени выполнения, если вы попытаетесь прочитать переменную, которой не было присвоено значение. Как C++ и Java, Python чувствителен к регистру — так "a" и "A" это разные переменные. Конец строки означает конец оператора, и в отличие от C++ и Java, Python не требует точки с запятой в конце каждой инструкции. Вы можете включать точку с запятой в конце Python заявления (возможно, просто по привычке), но это не лучший стиль. Комментарии начинаются с '#' и продолжаются до конца строки.

Первая программа Python

Исходные файлы Python имеют расширение ".py". Если у вас есть программа hello.py, самый простой способ запустить ее — набрать в консоли команду "python hello.py»

Системные программы UNIX (#!)

Для того, чтобы программа Python в UNIX-подобных системах исполнялась также просто, как обычные системные программы, необходимо сделать следующее

1. Первая строка файла должна иметь специальный формат: #!<путь к интерпретатору Python>. Для интерпретатора Python это простой комментарий, а для системы — указание на путь к интерпретатору для выполнения остального кода в файле.
2. Файл должен иметь разрешение на выполнение: chmod +x file.py

```
#!/usr/local/bin/python
... ваш код ...
```

В некоторых версиях системы UNIX можно избежать явного указания пути к интерпретатору Python, если специальный комментарий в первой строке оформить следующим образом:

```
#!/usr/bin/env python
... ваш код ...
```

При таком подходе программа env отыщет интерпретатор Python в соответствии с настройками пути поиска (то есть в большинстве командных оболочек UNIX поиск будет произведен во всех каталогах, перечисленных в переменной окружения PATH). Такой способ может оказаться более универсальным, так как он не требует жестко указывать во всех сценариях путь к каталогу, куда был установлен Python.

Типовая программа

Вот простейшая типовая программа Python:

```
#!/usr/bin/python

# В начале импортируются модули - sys один из стандартных
import sys

# Основной код скрипта в функции main()
def main():
    print('Hello there', sys.argv[1])
    # Аргументы командной строки: sys.argv[1], sys.argv[2] ...
    # sys.argv[0] содержит само название скрипта

# Стандартный шаблон вызова функции main(),
# чтобы начать программу
if __name__ == '__main__':
    main()
```

Запуск программы из командной строки выглядит таким образом:

```
$ python hello.py Guido
Hello there Guido
$ ./hello.py Alice    # (Unix)
Hello there Alice
```

Выше демонстрируется специальный прием, позволяющий импортировать файлы как модули и запускать их как самостоятельные программы. Каждый модуль обладает встроенным атрибутом __name__, который устанавливается интерпретатором следующим образом:

- Если файл запускается как главный файл программы, атрибуту __name__ на запуске присваивается значение “__main__”.
- Если файл импортируется, атрибуту __name__ присваивается имя модуля, под которым он будет известен клиенту.

Благодаря этому модуль может проверить собственный атрибут __name__ и определить, был ли он запущен как самостоятельная программа или импортирован другим модулем.

Функции

Функции в Python определяются таким образом:

```
# Определим функцию «repeat», принимающую 2 аргумента
def repeat(s, exclaim):
    """Возвращает строку s, повторенную 3 раза.
    Если exclaim == True — поставим восклицательные знаки.
    """

    result = s + s + s # также можно использовать "s * 3"
    if exclaim:
        result = result + '!!!'
    return result
```

Заметьте также, что строки, которые составляют функции или условный оператор if, все имеют один и тот же уровень отступа.

"def" определяет функцию. В скобках задаются ее параметры. Все строки функции имеют определенный начальный отступ. В первой строке функции может быть размещена строка документации («docstring»), которая описывает то, что делает эта функция. Строка документации может быть в однострочной, или многострочной, как в примере выше. Переменные, определенные в функции, являются локальными для этой функции, так что переменная "result" в приведенной выше функции является независимой от переменной "result" в какой-либо другой функции.

Строки

Форматирование строк

```
>>> template = '{0}, {1} and {2}' # Порядковые номера позиционных
 аргументов
>>> template.format('spam', 'ham', 'eggs')
 'spam, ham and eggs'
>>> template = '{motto}, {pork} and {food}' # Имена именованных
 аргументов
>>> template.format(motto='spam', pork='ham', food='eggs')
 'spam, ham and eggs'
>>> template = '{motto}, {0} and {food}' # Оба варианта
>>> template.format('ham', motto='spam', food='eggs')
 'spam, ham and eggs'
```

Строковые методы

Вот некоторые из наиболее распространенных методов строк. Метод очень похож на функцию, но он выполняется "на" объекте. Если переменная s является строкой, то s.lower() выполнит метод lower() для этой строки и возвращает результат (это одна из основных идей объектно-ориентированного программирования, ООП). Вот некоторые из наиболее распространенных методов строк:

s.lower(), s.upper() — возвращает строчный или прописной вариант строки
s.strip() — возвращает строку с удаленными пробелами в начале и в конце
s.startswith("str"), s.endswith("str") — возвращают True, если строка начинается или заканчивается на «str»
s.find("str") — ищет в данной строке строку «str» (не регулярное выражение), и возвращает индекс первого вхождения, или -1, если «str» найдена
s.replace("old", "new") — возвращает строку, в которой все вхождения «old» были заменены на «new»
s.split("разделитель") — возвращает список подстрок, разделенных данным разделителем. Разделитель не является регулярным выражением, это простой текст. "aaa,bbb,ccc".split(',') -.> ['aaa', 'bbb', ccc']. Если вызвать .split() без параметров, то вернется список слов строки (в качестве разделителя будет использован пробел).
s.join(list) — противоположность split(), соединяет элементы данного списка вместе, используя строку в качестве разделителя. Например '---'.join(['aaa', 'bbb', ccc']) -.> ‘aaa---bbb---ccc’.

Тесты

s.isalnum() — возвращает True, если строка содержит только буквенно-цифровые символы и строка содержит хотя бы один символ. False в противном случае.
isalpha() — возвращает True, если строка содержит только буквы и строка содержит хотя бы один символ. False в противном случае.
s.isdigit() — возвращает True, если строка содержит только цифровые символы и строка содержит хотя бы один символ. False в противном случае.

Полный список строковых функций можно найти в официальной документации по методам строк.

Кстати, в Python нет отдельного типа Символ (Char), как в некоторых других языках.

Оператор If

Python не использует {} для определения блоков для условий, циклов, функций и тд. Вместо этого, Python использует двоеточие (:) и отступы-пробелы, чтобы группировать операторы. Логическая проверка для условия if не обязательно должна заключаться в скобки (большое отличие от C + + / Java).

Любое значение может быть использована в логической проверке. «Нулевые» значения все считаются Ложью: None, 0, пустая строка, пустой список, пустой словарь. Существует также логический тип с двумя значениями: истина и ложь (1 и 0). Python имеет обычные операции сравнения: ==, =, <, <=,>,> =!. Логические операторы пишутся словами *and*, *or*, *not* (Python не использует &&, ll, ! в C-стиле).

Цикл for ... in

Цикл в Python реализуется очень просто:

```
squares = [1, 4, 9, 16]
sum = 0
for num in squares:
    sum += num
print(sum)  ## 30
```

Не удаляйте и не добавляйте ничего в список во время цикла.

Конструкция in позволяет с легкостью проверить, есть ли элемент в списке (или в любой другой коллекции) — value in collection — проверяет, есть ли значение в коллекции и возвращает True или False:

```
list = ['larry', 'curly', 'moe']
if 'curly' in list:
    print('yay')
```

Также for/in можно использовать для строк, перебирая в строке символ за символом.

range()

Функция range(n) возвращает числа 0, 1, ..., n-1, и range(a, b) возвращает a, a +1, ..., b-1 — до последнего числа, но не включая его. Комбинация цикла for и

функции range() дает вам возможность задавать традиционный цикл с заданным количеством повторений:

```
## print the numbers from 0 through 99
for i in range(100):
    print(i)
```

Цикл while

В Python также существует стандартный while-цикл, и операторы «break» и «continue», влияющие на ход выполнения внутреннего цикла.

```
# Выведем каждый 3й элемент списка
i = 0
while i < len(a):
    print(a[i])
    i = i + 3
```

Списки

Методы списков

list.append(elem) — добавляет один элемент в конец списка. Частая ошибка: не возвращает новый список, просто модифицирует исходный.
list.insert(index, elem) — вставляет элемент в заданную позицию, сдвигая элементы вправо.
list.extend(list2) — добавляет элементы из list2 в конец списка. Тоже самое, что использование + или += для списков.
list.index(elem) — ищет заданный элемент с начала списка и возвращает его индекс. Выбрасывает исключение ValueError если элемент не находится (используйте in для проверки без ValueError).
list.remove(elem) — ищет первый экземпляр данного elem и удаляет его (выбрасывает исключение ValueError если элемента нет в списке)
list.sort() — сортирует данный список (не возвращает его). (Желательно использовать для сортировки функцию sorted()).
list.reverse() — изменяет порядок данного списка на обратный (не возвращает новый список)
list.pop(index) — удаляет и возвращает элемент по заданному индексу. Возвращает последний элемент, если индекс опущен (в этом смысле работает как противоположность методу append()).
list.count(elem) — возвращает количество элементов elem в списке.

Сортировка списков

Для сортировки предпочтительно использовать функцию sorted(list).

```
>>> a = [5, 1, 4, 3]
>>> sorted(a)
[1, 3, 4, 5]
>>> a
[5, 1, 4, 3]
```

Можно сортировать в обратном порядке:

```
>>> strs = ['aa', 'BB', 'zz', 'CC']
>>> sorted(strs)
['BB', 'CC', 'aa', 'zz'] # (case sensitive)
>>> sorted(strs, reverse=True)
['zz', 'aa', 'CC', 'BB']
```

Для более сложных случаев сортировки можно использовать ключ key, в который можно передать функцию, трансформирующую каждый элемент перед сравнением. Эта функция принимает одно значение и возвращает 1 значение, и возвращаемое значение используется для сравнения при сортировке. Например:

```
>>> strs = ['ccc', 'aaaa', 'd', 'bb']
>>> sorted(strs, key=len)
['d', 'bb', 'ccc', 'aaaa']
```

Где получить помощь?

- Погуглить нужную строчку — «python string functions», «python list» и тд. Обычно самый эффективный способ найти справку по интересующему вопросу;
- Обратиться к официальной документации Python — docs.python.org. Обычно именно на нее попадаешь, когда ищешь что-то в Гугл);
- Много полезной информации можно найти на StackOverflow;
- Использовать встроенные функции help() и dir().

help() выводит краткую документацию для модулей, функций и методов.
dir() дает возможность быстро посмотреть список доступных методов объекта.

```
help(len)
help(list.append)
dir(sys)
```