OOPython

Лекция 2. Введение в синтаксис языка, часть А

Лирическая часть

- Создатель – Гвидо-ван-Россум, 1991 г. Свое название язык получил не из-за вида змей, а из-за Monty Python - комик-группы из Великобритании (1969 г).
- Питон используется в различных областях:
  как скриптовый язык shell в linux;
  обработка текстов;
  GUI;
  веб-приложения;
  лучший язык для обучения программированию;
  альтернатива Matlab'у
- В научных исследованиях Питон можно использовать для:
  замены Matlab
  создания новых программных интерфейсов для существующих программ на C/Fortran
  "склеивания" существующих программ вместе (например, GUI с вычислительной программной)
- У Питона простой и понятный синтаксис
  Java

```
class HelloWorld
{
        public static void main (String argv[])
        {
        System.out.println("Hello, World!");
        }
}
```

  C/C++

```
#include <stdio.h>
int main(char** argv, int argc)
{
        printf("Hello, World!\n");
        return 0;
}
```

  Python

```
print "Hello, World!"
```

  (Удобство очевидно)

- **Выемки из “Zen of Python”:**
  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!

- **Наличие большого количества библиотек, в частности - для научных вычислений:**
  numpy: «быстрые» массивы
  matplotlib: построение графиков различных видов
  scipy: численные методы
  sympy: символьные вычисления
  Программный код всех этих библиотек открытый.

- Питон - интерпретируемый (скриптовый) язык. Существует 2 вида трансляции программы в бинарный код: компилятор читает всю программу целиком, делает ее перевод и создает законченный вариант программы на машинном языке, который затем и выполняется. Интерпретатор переводит и выполняет программу строка за строкой. Файл исполняется **построчно** в отличие от компилируемых языков. В отличие от компилятора, интерпретатор не порождает на выходе программу на машинном языке. Распознав команду исходного языка, он тут же выполняет ее. Как в компиляторах, так и в интерпретаторах используются одинаковые методы анализа исходного текста

программы. Но интерпретатор позволяет начать обработку данных после написания даже одной команды.
Плюсы интерпретации по сравнению с компиляцией: не нужно время на компиляцию. Минусы: медленное исполнение, но существуют специальные оптимизации.

- Питон - высокоуровневый язык, высокий уровень абстракции. Уже есть встроенные и хорошо оптимизированные структуры данных + библиотеки. На Питоне очень приятно писать программы (после программирования на C++).
- У Питона имеется сборщик мусора.

Практическая часть

1. Написание и запуск программ. Возможно использовать несколько способов:
   - **текстовый редактор**
   - **IDE (Spyder, PyCharm)**
   - **Jupyter Notebook** - интерактивная среда (чем-то напоминает среду Wolfram Mathematica), с отдельными ячейками, в которых можно запускать код. Интерактивное окно открывается в браузере. Можете использовать любой из способов на ваш вкус. Мы будем пользоваться Jupyter Notebook.
2. Использование языка как калькулятора. Открыть терминал, набрать команду python - это запустит интерпретатор в интерактивном режиме.

   >>> **print** 5*0.6 - 0.5*9.81*(0.6**2)

   Стандартные арифметические операции записываются как “+”, “-”, “*”, “/” (как в C++). Операция возведения в степень “**”.

   Код запуска любой программы с использованием интерпретатора:

   python program.py

   Бинарные исполняемые файлы **не создаются**, создаются только текстовые файлы расширением **.py** + файлы **.pyc** для виртуальной машиной Питона (PVM), их интерпретация.
3. Переменные. Питон - динамически типизированный язык. То есть тип любой переменной перед началом исполнения программы неизвестен, в отличие от статически типизированного C/C++. Программист явно не задает типы int, double или char. В Питоне тип переменной во время исполнения может быть изменен. В Питоне всё (переменная, функция, массив и пр.) является **объектом (забегаю вперед).** Объект можно считать участком памяти. Адресной арифметики в Питоне нет. Поэтому разыменовать нулевой указатель просто невозможно. В действительности все переменные в Питоне являются ссылками на объекты; ссылки – особый тип данных, являющийся скрытой формой указателя, который при использовании автоматически разыменовывается.

Ссылка не является **указателем**, а просто является другим именем для объекта. Ссылка “примерно равна” указателю. Например, запись

a = b

означает "**в переменную a скопировать ссылку из переменной b**". Присваивание и разыменование ссылок производится автоматически. Каждый объект в Питоне хранит счетчик ссылок, и при таком копировании ссылки этот счетчик увеличивается. Счетчик же ссылок того объекта, на который переменная a указывала раньше - уменьшается. Когда счетчик достигает 0, объект считается неиспользуемым - память объекта освобождается.

Откройте gedit или vim соответствующей командой в терминале. Введите:

```
v0 = 5
g = 9.81
t = 0.6
y = v0*t - 0.5*g*t**2
print y
```

Именование переменных.
Способы написания составных имен переменных

- **Использование “_”.**
- **CamelCase (“верблюжий регистр”)** - стиль написания составных имен переменных, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово пишется с заглавной буквы.

```
initial_velocity = 5
acceleration_of_gravity = 9.81
TIME = 0.6
VerticalPositionOfBall = initial_velocity*TIME - \
0.5*acceleration_of_gravity*TIME**2
print VerticalPositionOfBall
```

Используйте тот стиль, который вам удобен.

Символ переноса на новую строку: **“\”.**
Желательно писать по одному выражению на строке. Если хотите вывести более одного - ставится **”;”**. Делать подобным образом не рекомендуется, потому что это снижает читабельность программы.

4. **Комментарии** начинаются со знака “#”.

```
# Program for computing the height of a ball in vertical motion.
```

```
v0 = 5 # initial velocity
g = 9.81 # acceleration of gravity
t = 0.6 # time
y = v0*t - 0.5*g*t**2 # vertical position print y
```

Совет по стилю: как вы знаете, сопровождение кода комментариями является правилом хорошего тона в программировании. Подробные комментарии вместе с правильно подобранными именами переменных крайне желательны для программ длиной более нескольких строк. В комментариях никогда не повторяйте то, что уже ясно из имен функций и переменных, вместо этого опишите в них неочевидную логику программы, описав важные детали функционирования.

- **Комплексные числа:** a = 2+1j; a.real; a.imag; a.conjugate()
- **Строки: a = ‘string’; a = “string”; a = ‘’’string’’’; a + b; a[0:3]; a[:3]; a[-1];**
- **Списки (обобщение массивов):** a = []. Могут хранить переменные различных типов.Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться). Чтобы использовать списки, их нужно создать. Создать список можно несколькими способами. Например, можно обработать любой итерируемый объект (например, строку) встроенной функцией list: >>> list('список')
  ['с', 'п', 'и', 'с', 'о', 'к']
- Список можно создать и при помощи литерала:
- >>>
- >>> s = [] # Пустой список
  >>> l = ['s', 'p', ['isok'], 2]
  >>> s
  []
  >>> l
  ['s', 'p', ['isok'], 2]
- Как видно из примера, список может содержать любое количество любых объектов (в том числе и вложенные списки), или не содержать ничего.
  В сложных случаях лучше пользоваться обычным циклом for для генерации списков.
- Функции и методы списков
- Создать создали, теперь нужно со списком что-то делать. Для списков доступны основные встроенные функции, а также методы списков.
- Таблица "методы списков"

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L

list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента от start до end со значением x
list.count(x)	Возвращает количество элементов со значением x
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка (новое в python 3.3)
list.clear()	Очищает список (новое в python 3.3)

Форматирование кода сдвигами (замена “{“ и “}” в C/C++)

- Циклы: for, while.
  Когда мы собрали переменные в список, зачастую требуется провести одну операцию над всеми его элементами. Этой цели служит цикл for:

  ```
  degrees = [0, 10, 20, 40, 100]
  for C in degrees:
      print ’list element:’, C
      print ’The degrees list has’, len(degrees), ’elements’
  ```

  Общий вид записи:
  for **element** in **somelist**:
  <process **element**>

  Здесь **element** - итератор, т.е. специальный объект, который используется для последовательного доступа к элементам списка.

  Стиль C-программистов и использованием инкрементов:

  ```
  for i in range(len(somelist)):
      element = somelist[i]
      <do smth>
  ```

  В Python так делать нежелательно, лучше придерживаться предыдущей формулировкой цикла.

Цикл **while**.

```
while eps >= 1e-3:
        eps /= 2
print eps
```

- Еще один способ создать список - это генераторы списков. Генератор списков - способ построить новый список, применяя выражение к каждому элементу последовательности. Генераторы списков очень похожи на цикл **for**.
  >>> c = [c * 3 for c in 'list']
  >>> print c
  --- ['lll', 'iii', 'sss', 'ttt']