Scripting & Computer Environments

Core Python: an Introduction

IIIT-H

Oct 5, 2013

...So Far & Today...

- Shell Scripting ✓

- Web Programming → Client-Side Scripting
  - `HTML` ✓
  - `XHTML` ✓
  - `CSS` ✓
  - `JavaScript` ✓

- We are headed to → Server-Side Scripting

Today:

Intro to `Python`

Brainstorm

1. Server-Side Scripting (a.k.a. Server Scripting)?
2. The pros and cons (if any)?
3. Some technologies from experience? Python?

Python

- Writen by Guido Van Rossum.
- A *modern*, *interpreted*, *object-oriented* and *versatile* language.
- Multi-paradigm language:
  mainly procedural, object-oriented and functional.
- Can easily be extended (e.g. with C/C++ or libraries/modules).

- Free
- Easy to learn but powerful
- Portable
- Extensible/ Support Libraries (modules)
- Development Speed
- Component Integration
- Object-Oriented
- Dynamic Memory Mgmt

1. Internet Programming ✓

   Standard Internet modules (e.g. network programming) + web-development frameworks (e.g. `web2py`, `Django`)

2. Database Programming

   Interfaces to relational DBMSs such as MySQL, Oracle, Sybase...

3. Scientific Computing (e.g. `Numpy`, `Scipy`)

4. GUIs (e.g. `Tkinter`)

5. Systems Programming

   Standard libraries for OS interfaces (files, processes, sockets, etc)


YouTube


search
Google


NASA

IIIT-H

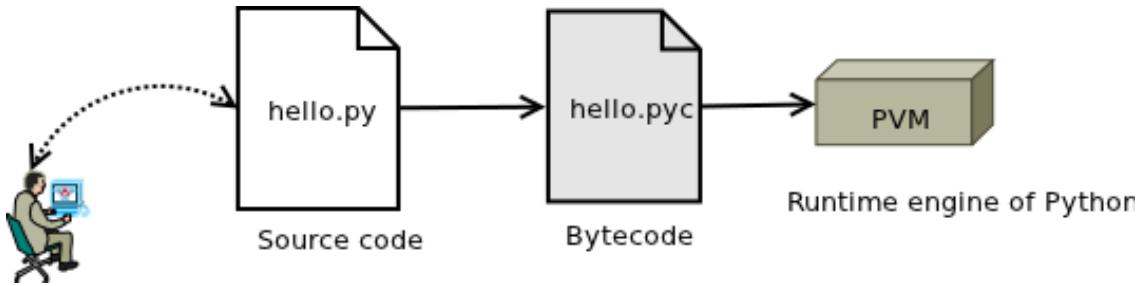

P X R
Animation Studios


NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA


AstraZeneca

The Execution Model

Python is a dynamic, *interpreted* language.



- **Bytecode** is a lower-level, *platform-independent* representation.
- If only source changes is the code 'recompiled'.
- Otherwise, `.pyc` file loaded and run ⇒ an optimization mechanism!
- The **Python Virtual Machine (PVM)** executes the bytecode one-by-one, using the CPU's architecture-specific instructions.
- Everything happens at runtime.

- Running Modes
  1. Interactive mode

```
$ Python
 >>> print "Hello, World!"        # this is comment
```

  2. Script mode

- Error Handling
- Editors and/or IDEs → `Vi, Vim, IDLE ...`
- Interactive Python interpretter → `IPython`

- Everything in Python is an object.

Core Python Objects

- Numbers
- Boolean
- Strings
- Lists
- Tuples
- Dictionaries
- Files
- Functions
- Modules
- Classes

- Mutable vs Immutable objects

Basic Numeric Types

- `Integer` (normal & Long)

  `e.g.  25, 65535, 99999999999999999`

- `Float`

  `e.g.  3.14, 2.17e-30, 6.02E+23`

- `Complex` (RealPart + ImaginaryPart)

  `e.g.  2+5j, complex(0,9)`

Conversion functions → `int()`, `float()`, `long()`, `bin()`, `hex()`

Common math functions are used by *importing* the `math` module.

e.g. `sqrt()`, `pow()`, `min()`, `sin()`, `floor()`, etc

Basic Numeric Types

- `Integer` (normal & Long)

  `e.g. 25, 65535, 99999999999999999`

- `Float`

  `e.g. 3.14, 2.17e-30, 6.02E+23`

- `Complex` (RealPart + ImaginaryPart)

  `e.g. 2+5j, complex(0,9)`

Conversion functions → `int()`, `float()`, `long()`, `bin()`, `hex()`

Common math functions are used by *importing* the `math` module.

e.g. `sqrt()`, `pow()`, `min()`, `sin()`, `floor()`, etc

Operators

- Arithmetic

  `+` `-` `*` `/` `//` `%` `**`

- Relational

  `<` `>` `==` `<=` `>=` `!=` `<>`

- Logical

  `and` `or` `not`

- Bitwise

  `<<` `>>` `&` `|` `^` `~`

Variables

- No need to declare but need to initialize.
- Python is a dynamically-typed language.

```
>>>var = "Bob"
>>>var = 1234
>>>print var
```

The usual identifier naming rules hold true:

- First character be a letter or underscore (_); the rest can be any number of alphanumeric or (_).
- Case-sensitive.
- Not be a reserved word.
- No special characters allowed.

Modules and Imports

- Python programs are composed of modules.
- Modules contain statements.

  e.g. `hello.py` (source code) → a.k.a module "hello"

Module

A file containing Python definitions *&* statements.

- Each module be *imported* to be used.

```
import <module name>
```

- Some standard modules
  - `math` - math functions
  - `sys` - access to `exit()`, `stdout`, `stdin`, `argv` ...
  - `re` - regular expressions
  - `os` - file system, operating system interface, etc

Modules and Imports

- Python programs are composed of modules.
- Modules contain statements.

  e.g. `hello.py` (source code) → a.k.a module "hello"

Module

A file containing Python definitions *&* statements.

- Each module be *imported* to be used.

```
import <module name>
```

- Some standard modules
  - `math` - math functions
  - `sys` - access to `exit()`, `stdout`, `stdin`, `argv` ...
  - `re` - regular expressions
  - `os` - file system, operating system interface, etc

Console Input/Output

- **Console Output**

```
>>>print "Hi there"                                    # method 1

>>>import sys
>>>sys.stdout.write("Hi there")                       # method 2
```

- **Console Input**

```
>>>n = input("Enter a number:")                      # numeric input

>>>name = raw_input("Enter your name:")              # string input
```

Sequence Data Types

- Positionally *ordered* set of objects.
- Notion of left-to-right ordering.

Some Built-in Objects

```
Strings
Lists
Tuples
Dictionaries                # mapping type - unordered
Sets                        # unordered collection
```

- Mutable vs Immutable sequences

Sequence Objects: 1. Strings

- Sequence of bytes or characters.

  e.g. gene sequence, database records, text files, binaries, etc

- No `char` type in Python.
- String Literals

Example

```
>>>str = 'CCCAAGGTTTTTAGGCCCT'

>>>str = "To be, or not to be, that is the question"

>>>str = '''this is also a string literal                # multi-line string
           that spans multiple lines '''

>>>str = """ similar to the above triple quote """    # multi-line string

>>>print str
```

- Sequence of bytes or characters.

  e.g. gene sequence, database records, text files, binaries, etc

- No `char` type in Python.
- String Literals

Example

```
>>>str = ‘CCCAAGGTTTTTAGGCCCT’

>>>str = "To be, or not to be, that is the question"

>>>str = ‘‘‘this is also a string literal              # multi-line string
            that spans multiple lines ’’’

>>>str = """ similar to the above triple quote """    # multi-line string

>>>print str
```

- Strings are immutable.

```
>>>name=‘‘‘Llanfairpwllgwyngyllgogerychwyrndrobwllll...
...antysiliogogogoch’’’                    # longest village name

>>>name[0]
>>>name[-1]
>>>name[3]=‘e’                             # error!
>>>name[100]                               # error!
```

- The usual escape sequences apply

```
>>>print """Faith, hope and love remain.  \n But the
greatest of these is \t love"""
```

String Operations

- Concatenation (+)

```
>>>'ATG' + 'CAGAT'
```

- Repetetion (*)

```
>>>"Hello" * 10
```

- Indexing

```
str[index]
```

- Slicing

```
str[start] - str[end-1]
```

```
>>>S='AGGTTTCCCCCG'

>>>S[2:5]
>>>S[:4]
>>>S[6:]
>>>S[:]
>>>S[1:8:2]
```

String Methods

Assuming, `str = "A string input"`

- `str()`
- `len(str)`
- `str.isalpha()`, `str.isdigit()`, `str.isspace()`, etc
- `str.find('sub')`
- `str.replace('old','new')`
- `str.count('sub')`

String Methods (2)

- `str.split()`
- `str.strip()`
- `str.upper()`, `str.lower()`
- `str.join(sequence)` sequence = list, tuple ...
- `str.startswith('sub')`, `str.endswith('sub')`

String Formatting

- Python offers `printf()`-like facility with the '`%`' operator.

Format Specifiers

- `%d` (int)
- `%s` (string)
- `%o` (octal)
- `%x` (hex)
- `%f` (floating point)
- `%g` (floating point)

- Usage: `<format strings> %(<matching values>)`

Examples

```
>>>import math
>>>x = math.pi
>>>r = 10
>>>print "Area= %f" %(x*r*r)
>>>text = "Hi %s, the result is %f." %('Bob', 2**10/3)
```

String Formatting

- Python offers `printf()`-like facility with the '`%`' operator.

Format Specifiers

- `%d` (int)
- `%s` (string)
- `%o` (octal)
- `%x` (hex)
- `%f` (floating point)
- `%g` (floating point)

- Usage: `<format strings> %(<matching values>)`

Examples

```
>>>import math
>>>x = math.pi
>>>r = 10
>>>print "Area= %f" %(x*r*r)
>>>text = "Hi %s, the result is %f." %('Bob', 2**10/3)
```

- Mutable ordered-collection of arbitrary objects inside '[ ]'.
- Accessed by its offsets.

```
>>>data = [2, 4, 6, 'hello']
>>>print data
>>>print data[0]
>>>print data[4]                          # error
>>>L = []                                 # empty list
>>>print L
>>>matrix = [ [1,2], [3,4], [5,6] ]      # can be nested too
>>>print matrix
```

- Assignment (=) behaves differently in Python.

```
>>>x = data                    # Now, x too points to the list
```

List Operations

- Concatenation (+)

```
>>>data + L
```

- Repetetion (*)

```
>>>data * 3
```

- Indexing

- Slicing

```
>>>print data[1:-1]
>>>print data[:]
```

- Membership (`in` and `not in`)

```
>>>4 in data
>>>'hello' not in L
```

List Methods

- `len(list)`
- `list.append(elem)`
- `list.extend(list2)` `# similar to +`
- `list.remove(elem)`
- `list.pop(index)`

List Methods (2)

- `list.index(elem)`
- `list.insert(index, elem)`
- `list.sort()`
- `list.reverse()`

Example

```
>>>list = [‘thymine’, ‘guanine’]

>>>list.append(‘cytocin’)

>>>list.insert(3,‘adenine’)

>>>list.reverse()

>>>print list

>>>list.extend(list)

>>>list.insert(len(list),‘uracil’)

>>>list.sort()

>>>list.pop(0)

>>>list.remove(‘uracil’)

>>>list
```

Strings $\leftrightarrow$ Lists: Conversion

Given either type , how do you convert it to the other?

1. String to List ?
2. List to String ?

Getting Help

- The built-in `help()` function

Example

```
>>>help('modules')      # list of available modules

>>>help('math')         # docs for 'math' module

>>>help(sys.exit)       # doc for 'exit()' of 'sys' module

>>>dir('math')          # defined members in 'math' module
```

- The official Python documentation is an authoritative source of reference (go there now).

Fun with Programming

- An educational software that teaches programming in a 3D environment. Ideal for OOP concepts.

- Get it from here.
- A tutorial on Alice can be found here.