Python Lab: Fundamentals

Proteomics Informatics, Spring 2014
Week 3
11th Feb, 2014

himan.grover@gmail.com
Himanshu.Grover@nyumc.org

Programs

- perform computations
  - on some **input data** to generate some **output**
- according to a well-defined series of steps (**algorithm**)
- implemented using some programming language
- which follows a **syntax**

Data – Types & Values

Object Type	Example Creation
Numbers (integers or floating-point/real-numbers)	10, 1234.5, 22.7
Strings	“Are you kidding?”, ‘This Python class rocks’
Boolean	True, False
…	…
…	…

- Fundamental “things” that programs work with (Data)
- Every data object has a type
  - Try in ipython shell:
    - type(1), type(‘PEPTIDE’)
- Creation
  - from keyboard, files, remote server, through data processing

Variables

- Store for later use

- Assign data to variables
  - x = 2
  - y=‘abc’ or “abc”
  - name = ‘Roger Federer’
  - aa_sequence = “GAMRPODSTK”

- Use meaningful names

Operations

- Numbers:
  - Operators (Ex. *, /, +, -, ** etc.)
  - Mathematical funcs (Ex. math.pow, math.sqrt)
    - import math
    - math.pow?, math.pow(10, 2)
- Strings (aa_seq = “GAMRPODSTK”)
  - Access:
    - Indexing (forward and backward): x[2], x[-1]; index starts at 0
    - Slicing: aa_seq[0:3], aa_seq[1:], aa_seq[1:5:2], aa_seq[::2], x[::-1]
  - Concatenation: aa_seq + aa_seq, aa_seq + “MDP”
  - aa_sesq.<tab> (in ipython shell)
  - Repetition: aa_seq*8
  - Membership: ‘G’ in aa_seq
  - aa_seq.find(‘E’), aa_seq.replace(“MR”, “NT”), aa_seq.lower(); len(x)
- Built-in vs. user-defined

Data Structures

- Containers for build composite/complex data objects

- Stored in a specific format, depending on data access and processing needs

Lists

- Most general sequence object
- Positionally ordered collection of arbitrarily typed objects: x = [1, ‘a’, [1,2,3]]
  - Heterogeneous
  - Arbitrarily nest-able
- Various operations:
  - Ex. x = [1,2,3,4,5]
  - Indexing, slicing etc. same as strings
  - Slice assignment: x[1:3] = [“replacement”, “string”]
  - **Generic functions:** len(x), type(x)
  - **Type-specific functions:** x.append(‘b’), x.pop(2), x.reverse(), x+y, x*3, x.extend([5,6,7]), x.sort()… x.<tab> to check the entire list

Dictionaries

- Unordered collection of mappings (key-value pairs)
  - Ex. y = {‘a’: 1, ‘b’: 2}
  - Ex. codon → AminoAcid
  - Ex. geneID → nucleotide sequence; protID → AminoAcid seq
- Keys must be unique
- Objects are accessed by ‘keys’ and not by relative position
  - y[‘a’] = 1
- Heterogeneous, can be arbitrarily nested:

```
rec = {'name': {'first': 'Bob', 'last': 'Smith'},
       'job': ['dev', 'mgr'],
       'age': 30}
```

- Various operations:
  - **Generic:** len(y), type(y)
  - **Type-specific:** y.has_key(‘a’) or ‘a’ in y, y.update(<another dict>), y.keys(), y.values(), y.items()

Statements

- **Assignment:**

  thisIs = “Python Lecture 2”
- **Function calls:**

  x.reverse() where x is a list obj
- **Print:** print x, “\t”, y
- **Select action:**

```
If <condition 1>:
    action 1
elif <condition 2>:
    action 2
else:
    action 3
```

- **Sequence Iteration (loops):**

```
for x in myList:
    action
while X > Y:
    action/pass
```

- **Building Functions:**

```
def add(x, y):
    return x+y
```

- **Module access:**

```
import numpy
from numpy import ndarray
from matplotlib import pyplot as plt
```

- **Exception Handling, OOP, global, del, exec, assert, documentation strings/comments etc.**

- **Indentation for compound statements**

Getting help: Documentation

- IPython shell
  - x = [1,2,3,4]
  - x.<tab>
  - help(x), help(x.<function name>), x.<function name>?
    - Ex. x.append?
    - L.append(object) -- append object to end
      - Similar to f(x) notation in math. But more general…

Interactive vs. Script Mode

Practice some!!

- Create some data objects (list, dictionary string) in IPython shell
- Check their “types”, and explore documentation for some of the their operations/functions
- Run some functions to see what happens

Next Class

- Compound statements:
  - for/while loops

- Real examples