01-python-overview

January 19, 2017

1 Python Features

- easy to learn
- simple syntax
- concise notation for common tasks
- very useful data structures are built into the language
- interactive
  - interpreted, not compiled(usually)
  - read-eval-print-loop(repl)
  - easy to play with things
- no type declarations on variables
- expedites experiments and prototyping
- but there's a downside
  - lose compiler error checking
  - A python program can have all kinds of bugs that are impossible in Java/C++
  - Errors are detected at run type - program can bomb unexpectedly
  - harder for humans to understand code
  - many optimizations that compilers do are not possible
  - slower than Java or C++ at some tasks
- multi-paradigm: supports procedural, object oriented, and functional programming
- exceedingly popular in research environments like Columbia
- out in the 'real world' C, C++, and Java probably more popular
  - programming language survey
- Python is pulling together many great ideas from other languages, including
- Lisp
- Matlab
- R
- Mathematica
- huge number of 3rd party libraries
- free, open source

2 Which Version of the Python Language?

- Python painted itself into a bit of a corner several years ago

- decided to fix a number of problems with 2.7 version
- fixes became version 3.X, but 3.X is incompatible with 2.7
- for several years many people refused to use 3.X, but the tide seems to have turned
- we will use 3.5 - quite a bit cleaner than 2.7
- We will briefly discuss strategies for dealing with legacy 2.7 code

3 Which Implementation of Python?

- We will use the "standard" CPython
  - included in the download from Anaconda
- There are special purpose implementations we will discuss later

4 Ways to Run Python

- interactive interpreters
  - "vanilla" python
  - ipython
    - time saving help features
  - jupyter notebooks
    - to run the notebooks i distribute, cd to the directory the notebooks are in, and enter 'jupyter notebook'
    - a tab will appear in your browser with the notebook files
    - click on a file to open it in a new tab
- inside an IDE
  - spyder(recommended)
    - in your home dir, open anaconda/bin/spyder to start it
    - be patient, it takes a while to come up
    - in preferences/object inspector/automatic connections, turn on "editor", "python console" "ipython". this enables help while you type
  - eclipse
    - has a python mode, but I find it difficult to use
- python programs can be invoked and run without user interaction(scripts)
  - much nicer than bash
  - computeCp
  - pbackup
- via web servers
  - flask
  - django
- Python can be embedded into other programs
  - blender
    - running line below will move a vertex of the default cube
    - bpy.data.objects["Cube"].data.vertices[0].co.x += 1.0

5 Python Memory Model

5.0.1 EVERYTHING is an Object

- even integers are objects
- objects have a type, state(instance variables), and executable procedures(methods)
- methods may reference and modify the object's state
- Python model is simple and elegant, but incurs considerable memory overhead
- Java and C++ have "things" which are not "objects"

5.0.2 All objects are stored in the heap

- automatic memory management via reference counting
- when no references to an object are left, the object's memory is reclaimed

5.0.3 Objects are mutable or immutable

- the state of a mutable object can be modified at any time
  - example: list
- immutable objects can not be modified after creation
  - example: string
  - "functional programming" favors immutables
- the type of an object never changes

5.0.4 Memory is not directly accessible

- Python variables only hold "references" to objects
- there are no pointers to memory locations, and in theory, it not possible to corrupt memory and crash
  - like Java
  - unlike C++

```
In [ ]:
```