Quick & Dirty Python

Professor Marie Roch

stickpng.com

Quick and dirty Python 3.x

- About the language
  - Interpreted high level language
  - Reasonably simple to learn
  - Rich set of libraries

- For details, see texts in syllabus or www.learnpython.org or www.diveintopython3.net

- Python comment
  # comment from hash character to end of line

Python data types

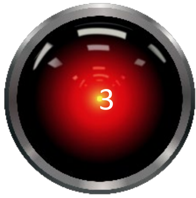
- float, int, complex: 42.8, 9, 2+4j

- Strings: single or double quote delimited

  ‘hi there’ “Four score and seven years ago…”

- Dictionaries: Python’s hash table

```
quotes = dict()  # new dictionary
quotes[“Lincoln”] = “Four score and seven years ago…”
OR
quotes = {“Lincoln” : “Four…”,
              “Roosevelt”: “The only thing we have to fear…”}
```



Python data types

- Sequences
  - Lists [“Four”, “score”, “and”]
  - tuples (“Four”, “score”, “and”)
- Difference between tuple and list
  - List – can grow or shrink
  - Tuple – Fixed number of elements
    - Faster
    - Can be used as hash table indices
    - Non-mutable
    - Need to make a tuple of size 1: (var,)

Python data types

- None – special type for null object
- Booleans: True, False

- Variable names can be bound to values of any type

- User defined types are available with dataclasses as of Python 3.7. We'll go over these after we discuss classes.

Python Expressions

- assignment: count = 0
- list membership: value in [4, 3, 2, 1]
- indexing 0 to N-1: listvar[4], tuplevar[2]
- slices [start:stop:step]
  listvar[0:N] → items 0 to N-1
  listvar[:N] → items 0 to N-1
  listvar[3:] →items 3 to end
  listvar[0:5:2] → even items at 0, 2, 4

  listvar[1::2] → odd items from start of list
  listvar[-4:-1] → 4th to the last to 2nd to the last
- write out logical operators: and, or, not

Python expressions

- comparison operators: < > >= <= !=
- basic math operators: + - / *
- exponentation: x ** 3 # x cubed
- bitwise operators: & | ~ and ^ (xor)

Python control structures

- Use indentation to denote blocks
- Conditional execution

```
if expression:
        statement(s)
elif expression:
        statements(s)
else:
        statement(s)
```

Python control structure

- Iteration

```
done = False
while not done:
    statements(s)
    done = expression

for x in range(10):  # 0 to 9
    print(x)
    print(f“x={x}.”)  # f is a format-string (see docs)
```

Alter iteration behavior with break and continue (usual semantics)

Many types of objects are iterable: lists, tuples, even some classes

Python functions

```
def foobar(formal1, formal2, formal3=None):
        “foobar doesn’t do much” # doc string
        # Use “”” multi-line text “”” for long doc strings
        statement(s)
        return value
```

- formal3 defaults to None if not supplied

- Variable scope rules
  local, enclosing function, global, builtin names

Python objects

```
class Board:
    "Grid board class"
    def __init__(self, rows, cols):  # constructor
        "construct a board with specified rows and cols"
        self.rows = rows
        self.cols = cols
        # list comprehension example
        self.board = [[None for c in range(cols)] for r in range(rows)]
    def place(self, row, col, item):
        "place an item at position row, col"
        self.board[row][col] = item
    def get(self, row, col):
        "get an item from position row, col"
        return self.board[row][col]
```

Python objects

- Create: b = Board(8,8)
- b.place(2, 7, ‘black-king’)
- b.get(2,7)
  “black-king”

Iterators

- Objects that can be looped over
- Raises StopIteration exception on end of sequence
- Rely on implementation of
    - __iter__ to return an object that can be looped over (possibly the object being called)
    - __next__ to return the next item in sequence

```
# Fibonacci sequence
fib = Fib(50)  # Numbers <= 50
# loop calls __iter__ on entry
# and __next__ each time
for f in fib:
    print(f)
```

Iterator example

```
class Fib:
    '''iterator that yields numbers in the Fibonacci sequence, series where next number is
       sum of the previous two'''

    def __init__(self, max):
        self.max = max          # stop when next Fibonacci number exceeds this

    def __iter__(self):
        self.a = 0    # initialize the Fibonacci sequence
        self.b = 1
        return self

    def __next__(self):
        fib = self.a
        if fib > self.max:
            raise StopIteration
        self.a, self.b = self.b, self.a + self.b  # evaluate RHS first, then assign pair
        return fib
```

Example from Pilgrim’s *Dive Into Python 3*

Exceptions

```
try:
        some code…
except RunTimeError as e:
        e is bound to the exception object
        do what you want…

# Other exceptions are not caught
# Read about finally clause
```

Dataclass (Python 3.7+)

- Requires importing dataclass decorator from dataclasses
- Declares a class, usually without any methods and a set of typed variables, e.g.:

```
from dataclasses import dataclass
@dataclass Framing:
   advance_ms: float
   length_ms: float
```

To use, `frame_params = Framing(10, 20)`

```
   frame_params.advance_ms returns 10.0
```

Python versions

- Versions of Python
  - Python.org – stock Python, sometimes called CPython
  - Anaconda – bundles with lots of libraries and Spyder IDE
    A variant called miniconda is less bloated.
  - Many other variants exist, see Python implementations if you are curious: https://wiki.python.org/moin/PythonImplementations

What should I install?

- CS 550 – Use C Python or Anaconda/miniconda
- CS 682 – Use Anaconda/miniconda, it makes installing tensorflow easier

A bit about Anaconda

- Supports 1+ virtual environment
- Allows easy switching between environments
- Can be managed in text or graphical mode
    - GUI: Getting started
    - Text: Getting started

Virtual environments are stored in the envs subdirectory of where you installed Anaconda. If you use a non-bundled development environment, select the Python interpreter residing in the appropriate subdirectory of envs:

e.g. /home/myacct/anaconda/envs/tensorflow if you created an environment named tensorflow

A few useful packages

- numpy – Numerical library (https://numpy.org/) that provides high performance number crunching
- scipy – Scientific and engineering libraries
- scikit learn – Machine learning libraries
- matplotlib – Plotting tools, other packages exist (e.g. seaborn)
- pysoundfile – Library for reading audio data
- pythonsounddevice – Library for audio recording/playback

Most of these can be installed easily with Anaconda or Python’s own package manager pip.

Examples installs

```
conda install scipy
pip install scipy
```

Python

Integrated development environments (IDEs)

I use these

- Eclipse with PyDev
- Pycharm
- Komodo (ActiveState)
- Visual Studio Code
- Spyder (bundled with Anaconda)
- others (see Python.org)

You are welcome to use whatever IDE you like, but I can only help you with problems for the IDEs that I use. Submissions must be pure Python code, Jupyter notebooks are not accepted.

Setting up pycharm

- Download: https://www.jetbrains.com/pycharm/
- Register as student for free professional version

- Educational materials on JetBrains site and elsewhere

Setting up elcipse

- Download from eclipse.org
- Follow the instructions on installing a plugin: https://www.pydev.org/download.html

Specifying the interpreter

Regardless of the IDE you use, you may need to indicate which version of Python to use.

- Pycharm instructions
- Eclipse instructions

Pycharm: setting the interpreter