Object oriented programming and exceptions in Python

Ivan Marković   Matko Orsag   Damjan Miklić
(Srećko Jurić-Kavelj)

University of Zagreb, Faculty of Electrical Engineering and Computing,
Departement of Control and Computer Engineering

2015

University of Zagreb
Faculty of Electrical Engineering
and Computing

Objects and Classes

- In simple terms, object = functions + data
- Support the concept of state
- Classes are "blueprints" of objects
- A natural and powerful way of thinking about problems
- Great for code modularity and reuse
- Rule of thumb: Whenever you are tempted to use a global variable, use a class

Example: A moving average filter

Write a python class that implements a Simple moving average filter, defined by the equation $A_k = \frac{x_k + x_{k-1} + \ldots + x_{k-(n-1)}}{n} = A_{k-1} + \frac{x_k}{n} - \frac{k_{k-n}}{n}$.

A moving average filter: Class design

- Object: filter
- Operations
  - update
  - initialize
- Data
  - buffer length
  - data
  - average

A moving average filter: Definition and Constructor

```
class MAfilter:
    """ A moving average filter. """

    def __init__(self,n):
        """ Initialize filter with buffer length n """
        self.n = 3
        self.data = [0.0 for i in range(n)]
        self.avg = sum(self.data)/n
```

A moving average filter: The update function

```
def update(self, x):
    """ Update filter with new reading. """
    self.avg += float(x - self.data.pop(0))/self.n
    self.data.append(x)

    return self.avg
```

A moving average filter: Object instantiation

```python
if __name__ == '__main__':
    filt = MAfilter(3)
    readings = [1, 17, -5, 9, 2]
    for x in readings:
        print('avg = {0}'.format(filt.update(x)))
```

Object and class gotchas

- Objects are mutable (i.e. they are passed around as references)
- …

OOP and general programming tips

- OOP[1] is all about code reuse
- Use pencil and paper before using the keyboard :)
- Write down a description of your program
  - Nouns are potential classes
  - Verbs are methods
- Break your program down into logical units
  - Functions
  - Classes
  - Modules
- Work incrementally:
  - Write a small chunk of code
  - Test it
  - Integrate
  - Repeat :)

[1] Object-Oriented Programming

Exceptions

- Signaling irregular program conditions
- Allow jumping over arbitrary large chunks of code
- Unhandled exceptions propagate up the call stack

```
def fetcher(obj, idx):
  return obj[idx]
x = 'py'
fetcher(x,5)
```

- Catching exceptions reduces the need of checking for status codes

```
try:
  fetcher(x, 5)
  # Do a lot of possibly dangerous stuff
except IndexError:
  print('Out of bounds!')
```

- We can raise exceptions ourselves (don't overuse!)

Tuples

- Tuples are immutable lists.

```
>>> T = (0, 'Robot', 3.14, [1,2,-5])
>>> T[3][1]
>>> T[1] = 'Human'
>>> 0 in T # Works for all collections!
```

- Tuple assignment

```
T = [(1,2), (3,4), (-5,6)]
for (a,b) in T:
  print(a*b)
```

Assignment

Create a tuple containing only numbers. How could we get a sorted version of the tuple?

Dictionaries

- Dictionaries are unordered collections data, accessed by key.

```
>>> D = {'name': 'Walee', 'age': 7}
>>> D['age']
>>> D['occupation'] = 'robot'
>>> D.keys(); D.values(); D.items()
```

- Iterating over a dictionary

```
D = {'LeBron': 6, 'Wade':3, 'Bosh':1}
for key in D:
  print('Number %d: %s' % (D[key], key))
```

Matplotlib

- A 2D plotting library (MATLAB plot-like interface)

```
user@host$ sudo pip install matplotlib
```

```
>>> from matplotlib.pyplot import plot, show
>>> from math import sin, pi
>>> t = linspace(-pi,pi,100)
>>> f = [sin(x) for x in t]
>>> plot(t,f)
>>> show
```

NumPy/SciPy

- Scientific computing tools for Python (like MATLAB)
- Basic data types: array (N-dim), matrix (2-dim)

```
useer@host$ sudo pip install numpy, scipy
```

```
>>> from numpy import *
>>> M = matrix(’7, 1.2, 1.3; 2.1, 2.2, 2.3; 3.1, 3
>>> P = 5*ones[3]
>>> M[2,4]; M*P; M.T; M.I
>>> A = array([[1.0, 2.0],[3.0,7.0]])
>>> A.shape
>>> import scipy.linalg
>>> linalg.eig(M)
```