ELEG 5040
Tutorial 1
Introduction to Python

Kai KANG (kkang@ee.cuhk.edu.hk)

Outline

- Tutorial Plan
- Piazza System
- Python Basics
- Popular Python Packages

Tutorial Plan

Week	Content	Note
1		
2	Python Introduction	
3	Theano	
4-5	CUDA/GPU Programming (Invited Talks)	Jan 24 & 25
6-7	Deep Learning Toolbox	
8-10	Caffe	
11-12	Research Experience on Deep Learning	
13-14	Review	

Piazza System

- Course website: www.piazza.com/cuhk.edu.hk/spring2015/eleg5040/home
- Resources: piazza.com/cuhk.edu.hk/spring2015/eleg5040/resources
    - Announcements (Email preferences)
    - Lecture and tutorial notes
    - Homework and solutions
    - Reading materials
- Q & A Section
    - Public and private questions, anonymous posts (if you are shy)
    - Collaborative editing: anyone can contribute
    - Image, attached files, LaTex math forms, code highlighting, etc

Why Python?

Why Python?

[If programming languages were vehicles]

Python is great for everyday tasks: easy to drive, versatile, comes with all the conveniences built in. It isn't fast or sexy, but neither are your errands.

Why Python?

- Convenience
  - Interpreted and interactive
  - Clear and concise syntax
  - Dynamic typing
  - Portable
  - Modules
- High-level
  - Object-oriented
  - Interfaces to many libraries
- Large community

Online Materials

- Official Python tutorial (https://docs.python.org/3/tutorial/index.html#tutorial-index)

- CodeAcademy interactive tutorial (http://www.codecademy.com/en/tracks/python)

- Dive Into Python free digital book (http://www.diveintopython.net)

- Python Challenge (http://www.pythonchallenge.com)

Python Basics

Numbers

Arithmetics

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
```

Floor division and remainder

```
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2  # result * divisor + remainder
17
```

Numbers

Power operations

```
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

Variables and assignments

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Numbers

Beyond int and float

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction()
Fraction(0, 1)
>>> Fraction('3/7')
Fraction(3, 7)

>>> 3+5j
(3+5j)
>>> 3+5J
(3+5j)
>>> 6+7j + 8-2J
(14+5j)
>>> complex('6-5j')
(6-5j)
```

Strings

Use either single quotes or double quotes

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Use \ to escape sequences

```
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> 'Hello, World!\n'
'Hello, World!\n'
>>> print('Hello, World!\n')
Hello, World!

>>>
```

Strings

Concatenation and repetition

```
>>> 'He' + 'llo'
'Hello'
>>> ('He' + 'llo') * 3
'HelloHelloHello'
>>> 'He''llo'
'Hello'
>>> ('He' "llo") * 3
'HelloHelloHello'
```

Indexing (zero-based)

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

Lists

Lists may not contain items with same types

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

Accessing, updating and deleting items

```
>>> list1 = ['physics', 'chemistry', 1997, 2000];
>>> list2 = [1, 2, 3, 4, 5, 6, 7 ];
>>> print "list1[0]: ", list1[0]
list1[0]:  physics
>>> print "list2[1:5]: ", list2[1:5]
list2[1:5]:  [2, 3, 4, 5]
>>> list1[2]=2001
>>> list1
['physics', 'chemistry', 2001, 2000]
>>> del list1[2]
>>> list1
['physics', 'chemistry', 2000]
```

Concatenating and repeating

```
>>> [1, 2, 3] + [4, 5]
[1, 2, 3, 4, 5]
>>> [6, 7] * 4
[6, 7, 6, 7, 6, 7, 6, 7]
```

Tuples

Tuples are very similar to lists, only difference is that tuples are immutable

```
>>> tup1 = ('physics', 'chemistry', 1997, 2000);
>>> tup2 = (1, 2, 3, 4, 5, 6, 7 );
>>> print "tup1[0]: ", tup1[0]
tup1[0]:  physics
>>> print "tup2[1:5]: ", tup2[1:5]
tup2[1:5]:  (2, 3, 4, 5)
>>> tup1 = (50,)
```

No enclosing delimiters needed

```
>>> a, b = 0, 1
>>> a, b = b, a + b
>>> a, b
(1, 1)
```

Dictionaries

Creating dictionaries with {}

```
>>> dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
>>> dict
{'Beth': '9102', 'Alice': '2341', 'Cecil': '3258'}
>>> dict1 = { 'abc': 456 };
>>> dict2 = { 'abc': 123, 98.6: 37 };
>>> dict2
{98.6: 37, 'abc': 123}
```

Indexing with keys

```
>>> dict['Alice']
'2341'
>>> dict1['abc']
456
>>> dict2[98.6]
37
```

Looping through keys

```
>>> for key in dict2:
...     print key, dict2[key]
...
98.6 37
abc 123
```

`if` Statements

Indentation is important in Python, semicolons are not

```
>>> a = 3
>>> b = 4
>>> if a < b:
...     print(a)
... else:
...     print(b)
...
3
>>> print(a if a < b else b)
3
```

No switch and case statements, use if..elif..elif.. sequence

```
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
```

for Statements

for statements iterate within sequences

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

Looping in a range

```
>>> for i in range(5):
...     print i
0
1
2
3
4
>>> range(1,5)
[1, 2, 3, 4]
>>> range(1,5,2)
[1, 3]
```

break and continue

break jumps out the smallest loop

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:
...         # loop fell through without finding a factor
...         print n, 'is a prime number'
```

continue jumps to next iteration

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print "Found an even number", num
...         continue
...     print "Found a number", num
```

`pass` Statements

pass statements do nothing but make syntax right

```
>>> while True:
...     pass  # Busy-wait for keyboard interrupt (Ctrl+C)
...

>>> class MyEmptyClass:
...     pass
...

>>> def initlog(*args):
...     pass   # Remember to implement this!
...
```

Functions

Define a function

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Functions are objects

```
>>> fib
<function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

Functions

return statements

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to
n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Functions

Default argument values

```
>>> def inc(a, b = 1):
...     return a + b
...
>>> inc(10)
11
>>> inc(10,5)
15
```

Keyword arguments

```
>>> def inc(st, step=1):
...     return st + step
...
>>> inc(10)
11
>>> inc(10, 5)
15
>>> inc(10, step=6)
16
```

Arbitrary argument list

```
def write_multiple_items(file, separator, *args):
    file.write(separator.join(args))
```

Classes

Define a class: instance variables and methods

```python
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world'
```

```python
>>> x = MyClass()
>>> x.i
12345
>>> x.f()
'hello world'
```

Instantiations: __init__()

```python
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Classes

Instance variables

```
>>> class Dog:
...     def __init__(self, name, kind='canine'):
...         self.name = name
...         self.kind = kind
...
>>> d = Dog('Fido')
>>> e = Dog('Buddy', 'husky')
>>> d.kind
'canine'
>>> e.kind
'husky'
>>> d.name
'Fido'
>>> e.name
'Buddy'
```

Classes

Inheritance

```
>>> class Dog:
...     def __init__(self, name, kind='canine'):
...         self.name = name
...         self.kind = kind
...
```

```
>>> class Husky(Dog):
...     def __init__(self, name):
...         Dog.__init__(self, name, 'husky')
...
>>> h = Husky('Rocky')
>>> h.name
'Rocky'
>>> h.kind
'husky'
```

Run Python Scripts

```
#!/usr/bin/python3

class Duck:
    def quack(self):
        print('Quaaack!')

    def walk(self):
        print('Walks like a duck.')

def main():
    donald = Duck()
    donald.quack()
    donald.walk()

if __name__ == "__main__": main()
```

Popular Packages

- cPickle, Protocol Buffers
- NumPy, SciPy
- scikit-learn, scikit-image, PIL, matplotlib
- h5py, leveldb
- ipython
- …

NumPy for Matlab Users

- http://wiki.scipy.org/NumPy_for_Matlab_Users

MATLAB	numpy	Notes
a && b	a and b	short-circuiting logical AND operator (Python native operator); scalar arguments only
a || b	a or b	short-circuiting logical OR operator (Python native operator); scalar arguments only
`1*i,1*j,1i,1j`	`1j`	complex numbers
`eps`	`spacing(1)`	Distance between 1 and the nearest floating point number
`ode45`	`scipy.integrate.ode(f).set_integrator('dopri5')`	integrate an ODE with Runge-Kutta 4,5
`ode15s`	`scipy.integrate.ode(f).\ set_integrator('vode', method='bdf', order=15)`	integrate an ODE with BDF

NumPy for Matlab Users

- Linear Algebra Equivalents

MATLAB	numpy.array	numpy.matrix
`ndims(a)`	`ndim(a)` or `a.ndim`	
`numel(a)`	`size(a)` or `a.size`	
`size(a)`	`shape(a)` or `a.shape`	
`size(a,n)`	`a.shape[n-1]`	
`[ 1 2 3; 4 5 6 ]`	`array([[1.,2.,3.], [4.,5.,6.]])`	`mat([[1.,2.,3.], [4.,5.,6.]])` or
`[ a b; c d ]`	`vstack([hstack([a,b]), hstack([c,d])])`	`bmat('a b; c d')`
`a(end)`	`a[-1]`	`a[:,-1][0,0]`
`a(2,5)`	`a[1,4]`	
`a(2,:)`	`a[1]` or `a[1,:]`	
`a(1:5,:)`	`a[0:5]` or `a[:5]` or `a[0:5,:]`	
`a(end-4:end,:)`	`a[-5:]`	
`a(1:3,5:9)`	`a[0:3][:,4:9]`	
`a([2,4,5],[1,3])`	`a[ix_([1,3,4],[0,2])]`	

NumPy for Matlab Users

- Linear Algebra Equivalents

MATLAB	numpy.array	numpy.matrix
a(3:2:21,:)	a[2:21:2,:]	
a(1:2:end,:)	a[::2,:]	
a(end:-1:1,:) or flipud(a)	a[::-1,:]	
a([1:end 1],:)	a[r_[:len(a),0]]	
a.'	a.transpose() or a.T	
a'	a.conj().transpose() or a.conj().T	a.H
a * b	dot(a,b)	a * b
a .* b	a * b	multiply(a,b)
a./b	a/b	
a.^3	a**3	power(a,3)
(a>0.5)	(a>0.5)	
find(a>0.5)	nonzero(a>0.5)	
a(:,find(v>0.5))	a[:,nonzero(v>0.5)[0]]	a[:,nonzero(v.A>0.5)[0]]
a(:,find(v>0.5))	a[:,v.T>0.5]	a[:,v.T>0.5)]

NumPy for Matlab Users

- Linear Algebra Equivalents

<table>
<tr><th>MATLAB</th><th>numpy.array</th><th>numpy.matrix</th></tr>
<tr><td>a(a<0.5)=0</td><td colspan="2">a[a<0.5]=0</td></tr>
<tr><td>a .* (a>0.5)</td><td>a * (a>0.5)</td><td>mat(a.A * (a>0.5).A)</td></tr>
<tr><td>a(:) = 3</td><td colspan="2">a[:] = 3</td></tr>
<tr><td>y=x</td><td colspan="2">y = x.copy()</td></tr>
<tr><td>y=x(2,:)</td><td colspan="2">y = x[1,:].copy()</td></tr>
<tr><td>y=x(:)</td><td colspan="2">y = x.flatten(1)</td></tr>
<tr><td>1:10</td><td>arange(1.,11.) or
r_[1.:11.] or
r_[1:10:10j]</td><td>mat(arange(1.,11.)) or
r_[1.:11.,'r']</td></tr>
<tr><td>0:9</td><td>arange(10.) or
r_[:10.] or
r_[:9:10j]</td><td>mat(arange(10.)) or
r_[:10.,'r']</td></tr>
<tr><td>[1:10]'</td><td>arange(1.,11.)[:, newaxis]</td><td>r_[1.:11.,'c']</td></tr>
<tr><td>zeros(3,4)</td><td>zeros((3,4))</td><td>mat(...)</td></tr>
<tr><td>zeros(3,4,5)</td><td>zeros((3,4,5))</td><td>mat(...)</td></tr>
<tr><td>ones(3,4)</td><td>ones((3,4))</td><td>mat(...)</td></tr>
<tr><td>eye(3)</td><td>eye(3)</td><td>mat(...)</td></tr>
<tr><td>diag(a)</td><td>diag(a)</td><td>mat(...)</td></tr>
<tr><td>diag(a,0)</td><td>diag(a,0)</td><td>mat(...)</td></tr>
<tr><td>rand(3,4)</td><td>random.rand(3,4)</td><td>mat(...)</td></tr>
</table>

Reading and Writing Mat Files

```
>>> import scipy.io as sio
>>> import numpy as np
>>> x = np.arange(12).reshape((3,4))
>>> x
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> sio.savemat('example.mat', {'x': x})
>>> y = sio.loadmat('example.mat')
>>> y
{'x': array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]]), '__version__': '1.0', '__header__':
'MATLAB 5.0 MAT-file Platform: posix, Created on: Fri Jan 16 13:04:04
2015', '__globals__': []}
>>> new_x = y['x']
>>> new_x
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Pickle

```
>>> import cPickle
>>> import numpy as np
>>> x = np.arange(12).reshape((3,4))
>>> y = np.random.rand(3,4)
```

Serialization

```
>>> pkl_file = open('example.pkl','wb')
>>> cPickle.dump((x,y), pkl_file)
>>> pkl_file.close()
```

De-serialization

```
>>> new_x, new_y = cPickle.load(open('example.pkl','rb'))
>>> new_x
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> new_y
array([[ 0.64956048,  0.79727908,  0.28350831,  0.07649368],
       [ 0.11551531,  0.11434357,  0.89143651,  0.23523877],
       [ 0.21506583,  0.16611042,  0.15059062,  0.30306736]])
```

References

- Official Python tutorial (https://docs.python.org/3/tutorial/index.html#tutorial-index)
- CodeAcademy interactive tutorial (http://www.codecademy.com/en/tracks/python)
- Dive Into Python free digital book (http://www.diveintopython.net)
- Python Challenge (http://www.pythonchallenge.com)
- NumPy for Matlab Users (http://wiki.scipy.org/NumPy_for_Matlab_Users)
- Scikit-learn (http://scikit-learn.org/stable/)
- SciPy, NumPy, matplotlib (http://www.scipy.org)
- h5py (http://docs.h5py.org/en/latest/)
- Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/)
- Google Python Style Guide (https://google-styleguide.googlecode.com/svn/trunk/pyguide.html)