Python Tutorial for CSE 446

Kaiyu Zheng, David Wadden

Department of Computer Science & Engineering
University of Washington

January 2017

Goal

- Know some basics about how to use Python.
- See how you may use Python for CSE 446.

Intro: hello world

Python is a general-purpose interpreted language. It is popular for machine learning because it is easy to code, has diverse libraries, and can use `C` for heavy computation tasks.

Simple hello world:

```
def hello_world():
    print("hello_world")

hello_world()
```

Intro: running the Python Shell

- You can run the Python Shell by typing `python` command on Linux or Mac, and open the Python Shell application if on Windows.

```
% python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for
    more information.
>>>
```

- We do not care about which Python version you use. The code in this tutorial is guaranteed to work on Python 2.7+.

Math Operators

- $+, -, *, /$ work the way you expect them to. For $/$, if either the divisor or dividend is a float, the result is a float; otherwise, the result is an integer.
- $//$ is the truncating integer division operator. `5.0 // 1.5` will yield `3.0`. The decimal part is dropped.
- % is the modulo operator.
- $**$ exponential. $**$ has precedence over $*$, $/$, and $//$.
- `AeB` means $A \times 10^B$, where `A` is an integer or float, and `B` is an integer. `A` and `B` cannot be variables.

Math Operators: Example

```
a = 3
b = 11

b % a # outputs 2
b / a # outputs 3
b / float(a) # outputs 3.666...5
b // a # outputs 3

a**2  # outputs 9
1.5e10 # outputs 15000000000.0
```

Language Basics: Types

In Python, you can convert from one type to another by invoking that type as a function (e.g. `int()`, `str()`). You can check the type of a variable with `type` function. See example below:

```
>>> a = 5
>>> type(a)
<type 'int'>
>>> str(a) + ","
'5,'

>>> b = 0x2424
>>> type(b)
<type 'int'>
>>> str(hex(b))
'0x2424'
```

Language Basics: Conditionals

- Python keywords related to Boolean expressions are: `True`, `False`, `and`, `or`, `not`. For example:

```
>>> False or not ((2 == 3) and (7 <= 5))
True
```

- Comparison operators are `==`, `!=`, `>`, `<`, `>=`, `<=`.
- Operators `is`, `in` are used with data structures (soon).
- Example program with `if-else` syntax:

```
def compare(a, b):
    if a > b:
        print("a is larger!")
    elif a < b:
        print("b is larger!")
    else:
        print("a and b are equal!")
```

Language Basics: Loops

Python supports for-loop and while-loop. Keywords `continue` and `break` are the same as in Java. More examples when discuss DS.

```
import sys
# output: 0, 1, 2, 3, 4,
for i in range(5):
    sys.stdout.write(str(i)+",")

# There will be nothing written!
for i in range(6, 2):
    sys.stdout.write(str(i)+",")

# output: 6, 5, 4, 3
for i in xrange(6, 2, -1):
    sys.stdout.write(str(i)+",")

i = 5
while i >= 0:
    i -= 1
```

Data structures

- The Python data structures that you will use the most are `list`, `dict`, `tuple`, `set`, `string`. We will take a look at them.
- Other data structures, such as queue, stack, priority queue, etc. can either be mimicked using the above ones (e.g. use `list` for stack), or there is some library that implements it (e.g. `heap` and `deque`).
- We won't cover everything here. Refer to Python documentation: `https://docs.python.org/2/library/functions.html`

Data structures: list

Think about `ArrayList` in Java. A `list` is a dynamic-sized integer-indexed array. Here is an example program:

```
def reverse_list(l):
    for i in range(len(l)/2):
        tmp = l[i]
        l[i] = l[-(i+1)]
        l[-(i+1)] = tmp
    l.append("hey!")

l = [2, [0, 1], 'hi', -9]
reverse_list(l) # l becomes: [-9, 'hi', [0, 1], 2, 'hey!']
```

You can take slices off a list as follows.

```
l = [0,1,2,3,4]
l[:4] # returns [0,1,2,3]
l[3:] # returns [3,4]
l[2:4] # returns [2,3]
```

Data structures: dict

Think about `Map` in Java. A `dict` is a hash table. Here is a demonstration of the operations that you can do with it.

```
staff446 = {'Prof': 'Emily Fox', 'TA': ['Dae Hyun Lee',
    'Sachin Mehta', 'David Wadden', 'Kaiyu Zheng']}
staff446['Prof'] # returns 'Emily Fox'
staff446['Coordinator'] = 'Pim Lustig'
staff446[99] = 100
# Won't work. key 100 does not exist yet!
staff446[100] += 1
# Removes the key 99. If 99 isn't a key, returns None.
staff446.pop(99, None)

# Check if key exists
if 'XYZ' in staff446:
    ...
# key pair iteration
for role in staff446:
    ...
```

Data structures: tuple I

A tuple is a finite, ordered list of elements. For example:

```
p0 = (0, 0, 1)
p1 = (1, 'a', [2,3])
```

You can access an element in a tuple just like accessing a list:

```
date = (1, 5, 2017)
month = date[0] # month is 1
day = date[1] # day is 5
year = date[-1] # year is 2017
```

Or, more conveniently, you can unpack a tuple:

```
day, month, year = date
```

Data structures: tuple II

A tuple is hashable if all elements are hashable (i.e. has hash value). So you can have:

```
uw = {}
uw[(1,3,2017)] = 'Quarter starts'
```

Since lists are not hashable, you cannot do

```
uw[p1] = 'VALUE'
```

You can iterate through a tuple with for-loop just like with lists:

```
for e in (1,3,2017):
    ...
```

You can slice a tuple just like a list.

```
a = (0,1,2,3,4)
a[:3] # returns (0, 1, 2). etc.
```

Data structures: set

Think about Set in Java. No duplicated elements, and no indexing of elements. Example code:

```
empty_set = set({})
myset = {1, 2, 3}
myset.add(4)
myset.update([5,6,7]) # add multiple elements
for item in myset:
    ....
```

You can do basic set operations:

```
a, b = {1, 4, 5}, {0, 2, 4, 7}
a | b # Union: set([0, 1, 2, 4, 5, 7])
a & b # Intersection: set([4])
a - b # Difference: set([1, 5])
a ^ b # Symmetric difference: set([0, 1, 2, 5, 7])
```

Data structures: string

A string is created by either putting characters inside single quotes or double quotes, or by casting an object of another type to string using `str`. You can expect python strings to have the same power as Java strings. See `https://docs.python.org/2/library/string.html`. You can iterate over a string just like a list.

```
mystring = "hello, world!"
mystring[0] # character 'h'
mystring + "somestring" # Concatnation
mystring[3:5] # Substring: returns 'lo' (same syntax as
    list slicing)
mystring.find('world') # Substring search: returns 7
for ch in mystring:
  ...
for i in range(len(mystring)):
  ...
```

Example: csv file processing

- You will deal with data in machine learning. One common format to store plain-text data is csv.
- We will go through an example of how a csv data file can be processed with Python.

Example: csv file processing — Laser scan readings

Many mobile robots have an on-board laser scanner, which shoots dozens of laser beams and can sense the distance the beam travelled before it hits an obstacle. Suppose we have a dataset of laser readings. It is in CSV format.

```
1 1482109355.586939 921 7.44846200943 7.43441057205 7.42055177689 7.40688657761 7.393411
2 1482109355.686939 921 7.44846773148 7.43441534042 7.42055797577 7.40689182281 7.393416
3 1482109355.786939 921 7.44849252701 7.43444013596 7.42058229446 7.40691614151 7.393440
4 1482109355.886939 921 7.44846963882 7.43441724777 7.42055988312 7.40689373016 7.393418
5 1482109355.986939 921 7.44846725464 7.43441486359 7.42055749893 7.40689134598 7.393416
6 1482109356.086939 921 7.44849777222 7.434445858 7.42058849335 7.40692138672 7.39344692
7 1482109356.186939 921 7.44845724106 7.43440532684 7.42054843903 7.40688180923 7.393407
8 1482109356.286939 921 7.44846391678 7.43441295624 7.42055416107 7.40688848495 7.393413
9 1482109356.386939 921 7.44846487045 7.43441200256 7.42055511475 7.40688848495 7.393413
10 1482109356.486939 921 7.4484667778 7.43441534042 7.42055654526 7.40689086914 7.393416
11 1482109356.586939 921 7.44845247269 7.43440055847 7.42054176331 7.40687608719 7.39340
12 1482109356.686939 921 7.44847631454 7.43442440033 7.42056512833 7.40690040588 7.39342
13 1482109356.786939 921 7.44847249985 7.43441963196 7.42056179047 7.40689611435 7.39342
14 1482109356.886939 921 7.44848012924 7.43442821503 7.42057037354 7.40690469742 7.39342
15 1482109356.986939 921 7.44843196869 7.43438053131 7.42052221298 7.40685749054 7.39338
16 1482109357.086939 921 7.44838762283 7.43433713913 7.42047977448 7.4068145752 7.393341
17 1482109357.186939 921 7.44840574265 7.43435430527 7.42049646378 7.40683221817 7.39335
18 1482109357.286939 921 7.44838237762 7.43433094025 7.42047405243 7.40681028366 7.39333
19 1482109357.386939 921 7.44838047028 7.43432950974 7.42047262192 7.40680789948 7.39333
20 1482109357.486939 921 7.44837903976 7.43432855606 7.42047119141 7.4068069458 7.393332
21 1482109357.586939 921 7.44843959808 7.43438911438 7.42053222656 7.40686798096 7.39339
22 1482109357.686939 921 7.44846105576 7.43440961838 7.42055273056 7.40688800812 7.39341
```

Example: csv file processing — Laser scan readings

Here is the actual format of this dataset for one row. You don't need to understand what they mean exactly.

```
<id> <timestamp> <n_beams> <readings...> <angle_min>
    <angle_increment> <range_max> <range_min>
```

Note: The number of readings in `<readings ...>` equals to the value in `<n_beams>`.

Example: csv file processing — Code I

Below is the actual code to preprocess a dataset like this. We hope to obtain a list of data rows, and each row is a dictionary. Here is the abbreviated code to show you how you can do this task with Python.

```
def parse_laser(lsfname):
  """Parse given laser scans file and return a list of
      ROS messages in dictionary form"""
  laser_data = []
  with open(lsfname) as f:
      print("Reading laser scans in %s " % lsfname)
      ...
      lines = f.readlines()
      for i, row in enumerate(lines):
          msg = {}
          cols = row.split(' ')

          # assign values
          msg['id'] = cols[0]
```

Example: csv file processing — Code II

```
            msg['time_stamp'] = float(cols[1])
            msg['n_beams'] = int(cols[2])
            msg['ranges'] = [None] * msg['n_beams']
            for k in range(msg['n_beams']):
                msg['ranges'][k] = float(cols[3 + k])
            ...
            laser_data.append(msg)
            # print progress
            sys.stdout.write('Processing file [%.1f%%]\r'
                % (float(i+1)/len(lines)*100))
            sys.stdout.flush()
        sys.stdout.write('\n')
        ...
    print("Finished processing %d laser scans." %
        len(laser_data))

    return laser_data
```

Beyond

For your final project, it is likely that you will use various tools and libraries. Here are some for you to explore:

- PDB: The interactive Python debugger. Really useful. You use it by putting the follow line of code at the breakpoint:

```
import pdb; pdb.set_trace()
```

- NumPy: Useful for dealing with large-scale arrays and matrices, with many math operations.
- matplotlib: Python plotting library, if you want visualization.
- Pandas: Data analysis, IO, etc.
- iPython & Jupyter: more interactive shell (e.g. code completion), and visualize your code as a write-up.

NumPy

Part of SciPy stack, for scientific computing with Python. Visit `www.numpy.org` for documentation.

- Core data structure: *homogeneous nd-array*. Much faster than Python's `list`. Both are written in C though (assuming you use CPython).
- High-level math operations for linear algebra, etc.
- Broadcasting: treating arrays with different shapes.
- Provides C-API, for accessing the array object in C code.

Let's see some quick examples.

NumPy: Examples

```
import numpy as np

a = np.array([[1,2,3],[4,5,6]]) # Creates 2 by 3 matrix
print a.shape # Output: (2,3)
b = a[:1, 1:3] # Slice 1st row, and 2nd + 3rd columns; b = [2,3]
c = a[:1,] # Slice 1st row; c = [1,2,5]
b[0,1] = 5 # Change number at index [0,1] of b to 5.
           # Since a is only a view of a, a is changed as well.
           # So we have b = [2,5], and a = [[1,2,5],[4,5,6]]
np.dot(b, a) # Matrix multiplication; Result: [22, 29, 40]
d = a + c  # Broadcasting; c is added to each row of a
e = np.random.rand(100,4,5,2) # Creates 4-dimensional array
    (100x4x5x2) with random values, each value is in [0,1]
f = a.transpose() # Transpose. Still, f is only a view of a.
```

NumPy: Vectorization

In Python (unlike, for instance, C), writing `for` or `while` loops that iterate over the elements of a vector will result in really slow code. Instead, vectorize. For instance, consider two arrays x and y with a million elements each that you want to add together.

```
# BAD
# x and y stored as built-in Python lists
z = []
n = int(1e6)
for i in range(n):
    z.append(x[i] + y[i])
```

```
# GOOD
# x and y stored as numpy arrays
z = x + y
```

The second version runs 200x faster (try it!)

NumPy: Caveats

- If you need to multiply two vectors or matrices, don't write you own code to do it. Instead, use `numpy.dot`.
- If you need to invert a matrix, don't write your own code or use `numpy.inv`. Instead, use `numpy.solve`.
- In general, any time you're doing heavy numerical work, do it with NumPy functions on NumPy data structures.
- It's worth going through "the basics" in this tutorial: https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

Pandas: Python Data Analysis Library

- The core Pandas data type is a DataFrame, which is like a NumPy array except the row and column indices can be anything you want. It is 2-dimensional.
- If you have some tabular data that you want to get into Python, use `pandas.read_table`.
- To convert DataFrame to a NumPy array, use the `frame.values` attribute.
- Pandas also supports many plotting functions through the `frame.plot` method.
- Much more in the Pandas docs: http://pandas.pydata.org/pandas-docs/stable/

Pandas: process csv file

For the same task of processing csv laser data, you can definitely make use of Pandas.

```
import pandas as pd
import numpy as np
def parse_laser(lsfname):
    df = pd.read_csv(lsfname, delim_whitespace=True,
        header=None) # read csv as DataFrame
    df.columns = ['id', 'timestamp', 'n_beams'] +
        (df.columns[3:3+df.iloc[0][2]].values -
        3).tolist() + ['...the remaining headers...']
    for index, row in df.iterrows():
        ...
    # The DataFrame looks like this (when you print it)
    #   id      timestamp n_beams 0  1 ... 920 ...
    # 0  1  1482109355...     921  ............ ...
    # 1  2  ...
    # 2  3  ...
    # ...
```

Other extensions

- Scipy extends NumPy with more scientific computing capabilities. Probably not necessary for this course.
- Matplotlib allows for convenient plotting. To make a line plot with x-coordinates given by vector x and y-coordinates by vector y you could write:

```
from matplotlib import pyplot as plt
plt.plot(x, y)
```

  More info here: http://matplotlib.org/api/pyplot_api.html
- IPython is an interactive Python console with auto-completion, plotting and debugging support: https://ipython.org/
- To set a breakpoint that will drop you into IPython, use `import ipdb; ipdb.set_trace()`