EML4930/EML6934: Lecture 01 - About Python

Basics: data types, math, logic, if statement

Charles Jekel

August 31, 2017

Results from the first HW

Attempts: 37 out of 37

Which Python installation did you go with? Either answer Anaconda or Enthought Canopy

Anaconda	29 respondents	78 %	✓
Enthought Canopy	8 respondents	22 %	✓

Which version of Python did you install? Your choices were 2.7, 3.5, or 3.6.

2.70	18 respondents	49 %	✓
3.50	4 respondents	11 %	✓
3.60	15 respondents	41 %	✓

The operating systems used by the class

Windows 7		0 %	✓
Windows 8	1 respondents	3 %	✓
Windows 10	23 respondents	62 %	✓
OS X	8 respondents	22 %	✓
Linux	3 respondents	8 %	✓
other		0 %	✓
Something Else	2 respondents	5 %	

Textbook for this lecture

A whirlwind Tour of Python by Dr. VanderPlas is a short book to prepare users with the bare essentials for working with Python.

It is also available for free at `http://www.oreilly.com/programming/free/files/a-whirlwind-tour-of-python.pdf`

or `https://github.com/jakevdp/WhirlwindTourOfPython`

Comments

```
# This is a comment in Python

'''
This is a bulk comment in python

Any line between the start and the end

is part of the comment
'''
```

Assigning variables

```
# Variables can be easily assigned
# this code assigns integer 10 to x
x = 10
```

This officially defines a pointer named x that points to the integer 10.
We can change what x points to at any time.

```
x = 10        # x is the integer 10
x = 10.       # x is the floating point EEE-754 double precision
x = 10.0      # same as x = 10.
x = 'ten'     # x is now a string
x = (0,1,2) # x is now a tuple
x = [0,1,2] # x is now a list
x = 1; y = 2; z = 3 # assigns x = 1, y = 2, and z = 3
```

Demo - Consequences of pointers

Be careful.

```
x = [1,2,3]
y = x
x[0] = 4
print(y)
```

Numpy warning!

```
import numpy as np
x = np.array([10])
y = x
x += 5
print(y)
# this doesn't  happen if I used x = x + 5
# this doesn't happen if x = 10 (instead of np.array([10]))
```

Math operators

Operater	Name	Description
a + b	Addition	Sum of a and b
a - b	Subtraction	Difference of a and b
a * b	Multiplication	Product of a and b
a / b	True division	Quotient of a and b
a // b	Floor division	Quotient of a and b , removing fractional parts
a % b	Modulus	Remainder after division of a by b
a ** b	Exponentiation	a raised to the power of b
-a	Negation	The negative of a
+a	Unary	plus a unchanged (rarely used)

Page 18 of A Whirlwind Tour of Python by VanderPlas.

Division in Python2

Let's take a look at division in Python2

```
a = 3
b = 2
c = a/b
```

So we have integer a divided by integer b and I'm familiar with programming so I expect c to be an integer.

Python3 broke division!

or *Fixed* it, because in Python3 an integer divided by an integer magically becomes a float. Remember that import future command? We can use it to get Python3 division in Python2.

```
from __future__ import division
a = 3
b = 2
c = a/b
```

Built in data types

Here are the built in data types for Python. Use

```
type(x)
```

to display the data type of x.

Type	Example	Description
int	x = 1	Integers (i.e., whole numbers)
float	x = 1.0	Floating-point numbers (i.e., real numbers)
complex	x = 1 + 2j	Complex numbers
bool	x = True	Boolean: True/False values
str	x = 'abc'	String: characters or text
NoneType	x = None	Special object indicating nulls

str(x) converts x to a string int(x) converts x to an integer float(x) converts x to a floating point

Page 24 of A Whirlwind Tour of Python by VanderPlas.

Comparison operators

Comparison operators will return a boolean

```
True
False
```

Operation	Description
a == b	a equal to b
a != b	a not equal to b
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a >= b	a greater than or equal to b

Page 21 of A Whirlwind Tour of Python by VanderPlas.

Floating point precision

```
0.1+0.2 == 0.3
```

this returns False! Why?

You can use the numpy isclose function to set a tolerance for floating point comparison.

Data Structures

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

Page 30-31 of A Whirlwind Tour of Python by VanderPlas.

Mutable

Can be changed and modified

Immutable

Can not be changed or modified

Lists are amazing

Lists

- basic **ordered** and **mutable** data collections
- Lists can be any shape and contain any data type
- you can have lists, floats, integers, tuple, dictionaries, sets in lists
- a float of 10.0 can be added to a list x by x.append(10.0)
- list x can be sorted by x.sort()
- the number of items in list x can be found with len(x)

Pimp my list


YO DAWG I HEARD YOU LIKE LISTS
SO I PUT A LIST WITHIN A LIST, WITH A LIST OF LISTS
memegenerator.net

Listception

An example list

```
x = [] # initialize an empty list
x.append(0.0) # append float 0.0 to x
x.append(1) # append integer 1 to x
x.append([3,4,'hi',(10,8)]) # append a list with a tuple to x
x.sort() # sort x from low to high
n = len(x) # count the number of items in list x
print('There are ',n,' items in list x')
print(x)
```

Note: x.sort() doesn't work in this case with Python 3 since you would need to compare different data types!!!

List forward indexing - lists start at 0

```
x = [7, 77, 777, 7777]
```

The first item in the list can be called using

```
in : x[0]
out: 7
```

The second item of the list can be called using

```
in : x[1]
out: 77
```

The third item of the list can be called using

```
in : x[2]
out: 777
```

The last item of the list can be called using

```
in : x[3]
out: 7777
```

List backward indexing - last index item in Python is -1

```
x = [7, 77, 777, 7777]
```

The last item in the list can be called using

```
in : x[-1]
out: 7777
```

The second to last item of the list can be called using

```
in : x[-2]
out: 777
```

The third to last item of the list can be called using

```
in : x[-3]
out: 77
```

The fourth to last item of the list can be called using

```
in : x[-4]
out: 7
```

List slicing

```
x = [7, 77, 777, 7777]
```

List slicing is [startPoint : endPoint] where startPoint is inclusive and endPoint is exclusive. In Mathematics would define define the interval as [startPoint, endPoint).

If we wanted the first and second item in a list

```
in : x[0:2]
out: [7, 77]
```

So if we want the second through fourth item in list x

```
in : x[1:4]
out: [77, 777, 7777]
```

Slicing with step size

```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
     10, 11, 12, 13, 14, 15,
     16, 17, 18, 19] # creates a list of integers
```

We can slice with [startPoint:endPoint:stepSize], just like before, but now stepSize is the step size of our slice.

```
in : x[0:20:2]
out: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

in : x[0:20:5]
out: [0, 5, 10, 15]

in : x[1:20:3]
out: [1, 4, 7, 10, 13, 16, 19]
```

Reverse the order of a list

Sometimes it is useful to reverse the order of a list. We can do this by consider a backwards slice.

```
x = [7, 77, 777, 7777]
```

So to see the reverse order of x we would run

```
in : x[::-1]
out: [7777, 777, 77, 7]
```

Creating lists with range (Python2 only)

Note: Python 3.5 changed how range works, range no longer creates lists, it is an iterable. We won't use range for numerical work, we'll use a numpy function instead.

Let's use the range function to create lists of integers. How do we view the docstring of range?

```
print(range.__doc__)
```

range(stop) $->$ list of integers range(start, stop[, step]) $->$ list of integers

Return a list containing an arithmetic progression of integers. range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to 0. When step is given, it specifies the increment (or decrement). For example, range(4) returns [0, 1, 2, 3]. The end point is omitted! These are exactly the valid indices for a list of 4 elements.

Tuples

Tuples are just like lists, except you define a tuple with parentheses instead of square brackets. List indexing a slicing of Tuples works exactly like lists, and you'll use square brackets to call items of a tuple.

Tuples are immutable which means that once they are created they can't be modified in any way. I hardly ever use Tuples.

Let's create a tuple of 2, 4, 6.

```
x = (2, 4, 6)
```

```
in : x[0]
out: 2
```

```
in : x[-1]
out: 6
```

Dictionaries

Dictionaries map keys to values. There is no index with dictionaries, instead there is a key. Dictionaries are sometimes used to pass parameters into a function.

Let's take a look at this param dictionary for the XGDBoost library as an example dictionary. `http://xgboost.readthedocs.io/en/latest/python/python_intro.html`

```
param = {'max_depth':2, 'eta':1, 'silent':1,
 'objective':'binary:logistic' }
param['nthread'] = 4
param['eval_metric'] = 'auc'
```

We don't can add keys to the dictionary at any time! Let's take a look at param.

```
in : print(param)
out: {'silent': 1, 'eval_metric': 'auc', 'nthread': 4,
 'eta': 1, 'objective': 'binary:logistic', 'max_depth': 2}
```

Dictionaries - param continued

Find values of keys with square brackets.

```
in : param['objective']
out: 'binary:logistic'
```

We can easily assign a new value for a key

```
in : param['objective'] = 'reg:linear'
in : print(param['objective'])
out: 'reg:linear'
```

Sets

Sets are like lists and tuples, but are defined with curly brackets. Sets obey mathmatical set definitions. VanderPlas provides a good example on page 36-37. Which I'll show here.

```
in : primes = {2, 3, 5, 7}; odds = {1, 3, 5, 7, 9}
in : primes.union(odds) # union of primes and odds
out: {1, 2, 3, 5, 7, 9}
in : primes.intersect(odds) # intersection of primes and odds
out: {3, 5, 7}
in : primes.difference(odds) # items in primes but not odds
out: {2}
in : primes.symmetric_difference(odds) # items in just one set
out: {1, 2, 9}
```

Membership operator

Operator	Description
a is b	True if a and b are identical objects
a is not b	True if a and b are not identical objects
a in b	True if a is a member of b
a not in b	True if a is not a member of b

Page 23 of A Whirlwind Tour of Python by VanderPlas.

If-then statements

Let's take a look at a if, elif, and else statement in Python.

```
x = 9
if x < 0:
    print(x,' is a negative number')
elif x > 0:
    print(x,' is a positive number')
elif x == 0:
    print('Single')
else:
    print(x,' makes me confused!')
```

This obviously returns (9, ' is a positive number')

If-then statements - notes

```
x = 9
if x < 0:
    print(x,' is a negative number')
elif x > 0:
    print(x,' is a positive number')
elif x == 0:
    print('Single')
else:
    print(x,' makes me confused!')
```

- in Python code blocks are denoted by indentation
- remember the emphasis on Python is to create highly readable code
- by forcing you to indent your code block
- the recommendation is that you use four spaces to denote an indent
- but you can also use tab
- indentations are preceded by a :

Subsequent code blocks are also indented

```
x = 9
if x > 0:
    print(x,' is a greater than zero')
    if x > 5:
        print(x,' is a greater than five')
        if x > 10:
            print(x,' is greater than 10')
    # this is the continued x > 0 code block
    print('the date type of x is ', type(x))
```

Note that this will print the type of x only when x $>$ 0!

tabs vs spaces

In a survey it was found that those who use spaces make more money
`https://stackoverflow.blog/2017/06/15/developers-use-spaces-make-money-use-tabs/`

tabs vs spaces

HW 01 - turn in one week from today in Canvas

Turn in the 5 questions as a single .py file onto canvas. Use comments to clearly indicate which question you are working on. Your filename should end as _py2.py if you use Python2 and _py3.py if you use Python3.

1. Name one difference between Python2 and Python3. Print your answer as a string in Python.
2. You are given a list x = [0,1,2,3,4,5,6]. Print the third item in list x.
3. Assign y as the reversed order of list x. Print y.
4. Use list slicing to assign z [1,3,5] from x. Print z.
5. Your friend is new to Python. He is confused why his if statement isn't working, he has a number x and wants to print 'x is positive' by checking with an if statement. His code is following.

   ```
   x = 99
   if (x > 0) is True
   print('x is positive')
   ```

   This returns an 'invalid syntax error'. Copy this code into your .py file, and correct the code.