**Python Class Notes**

**import this**

---Zen of python

**this.\_\_file\_\_**

---rot13 sample code

'/usr/local/Cellar/python/2.7.12\_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/this.py'

**Note:** Python has an intermediate machine language

**Print:**

Print(“Hello, world!”)

**eg1:**

print("%d this is test %d" % (5,11))

5 this is test 11

**eg2:**

print(num,’this is a number’)

4 this is a number

**Variables:**

-no declarations

-int, float, string, Boolean (Boolean named after George bool)

-Everything is an OBJECT!

-strongly typed

x = 3

x = x + 3 Fails

x = x + str(3) Pass

-No type casting between different data types (But int,float etc will work)

Boolean: True

Variable can be named as true, eg. true = 5

**Built-in Python functions:**

str()

int()

type()

**print**

Python2: print is statement

Python3: print is a function

Works in python2

print 1;

print(1);

**Python Arithmetic:**

3/2 1 in python2

3/2 1.5 python3

3//2 1.5 in python3

**from \_\_future\_\_ import division**

to use python3 division in python2

**div/mod/divmod**

9//5

1

9 % 4

5

divmod(9,5)

(1,4)

q , r = divmod(9,5)

q

1

r

4

**Non-decimal places (Binary,Oct,Hex)**

0b10

2

0o10

8

0x10

16

Sixe of an int 32bits

long 64bits (Eg, 123456789101112L)

**Long:**

>>> 2 \*\* 64

18446744073709551616L

*10 \*\* 100 called googol*

**factorial:**

import math

math.factorial(52)

**strings:**

use single or double quotes, escape using \, \n

+ concatenation

\* duplication

s = hi

print(s \*4)

hihihihi

**‘’’ used for multi line string**

s = ‘’’ asdasd

asdadas

asdasd

‘’’

**len:**

>>> a = 'hello'

>>> len(a)

5

**[]**

* ¥ ¥ ¥ - access a single character via its offset
* ¥ ¥ ¥ - easier to think of as offset instead of index
* ¥ ¥ ¥ - negative offsets count from end of string

>>> a = 'abcdef'

>>> a[0]

'a'

>>> a[1]

'b'

>>> a[-1]

'f'

>>> a[-5]

'b'

**indentation:**

-must be consistent

-either tab or space (don’t mix both)

**comparison:**

> < == !=

**in …**

**loops:**

**NOTE:** loops has an else condition !

**-while loop**

**-for loop**

for letter in ‘string’:

print(letter)

eg2: Range does not include the last range. (ie, range(a,b) then, iterates i=a to i<b)

for num in range(1,5):

print(num)

output: 1,2,3,4

for num in range(1,10,3)

print(num)

output: 1,4,7

**Range:**

Range does not include the last range. (ie, range(a,b) then, iterates i=a to i<b)

-range in python2 pre-generates all numbers as list

-to prevent this we use xrange(0,100)

python3

list(range(0,5))

[0, 1, 2, 3, 4]

**Advanced string functions:**

s=’abcdefghijks’

-slices[] e.g, s[10:20]

-len(s)

- s.startswith()

- s.endswith()

- s.find()

- s.rfind()

- s.count()

- s.strip() s.strip(‘.’)

-s.capitalize()

-s.title()

-s.upper()

-s.lower()

-s.swapcase()

-s.replace(‘abc’,’xyz’)

**Split/Join:**

“this is a split”.split() [‘this’, ‘is’, ‘a’, ‘split’]

“a,b,c,d”.split(‘,’) [‘a’, ‘b’, ‘c’, ‘d’]

‘’.join([‘a’,’b’,’c’]) abc

‘,’.join([‘a’,’b’,’c’]) a,b,c

**Lists:**

(Same as arrays in java)

-duplicates allowed, different types allowed,

**list() – creates a list from string.**

list(‘hello’) [‘h’,’e’,’l’,’l’,’o’]

>>>l = [1,3,5,7]

>>>l

[1,3,5,7]

list1 = [‘1’, ‘3’,’5’]

list2 = [list1, ‘7’] [[1 3 5], 7]

**list functions:**

list.append(‘adas’)

list.insert() – add an item by offset e.g, list.insert(3,’joe’)

extend() , += += add a list to a list

del delete by position

remove(item) remove by value

pop() removes the last item

list.index(‘item) returns the index

in() test if its in list

count(‘item’) count the no. of occurences of the item

**list join()**

**sort/sorted/len**

sort(): sort a list in place

sorted(): return a sorted copy of a list

len(): return length of a list

**lists: assign vs. copy/list/slice**

s = [‘a’,’z’,’b’]

s = s.sort() does not copy

s = list(s.sort()) correct way to copy list, use list() function or copy()

s = s.copy()

**list enumerate()**

for idx, val in enumerate(myList):

print(“index”, idx, “=”,val)

**zip()**

for itema,itemb in zip(list1, list2)

print(itema,itemb)

**ord()**

Ord function gives you the numerical ascii equivalent of the character

eg., ord(char)

ord(‘A’)

**chr()**

Char gives the character of the ascii, i.e reverse of ord.

chr(65) A

**List Comprehensions (“listcomps”)**

string = ‘abcd’

list = [char for char in string]

**Create Cartesian product:**

colors = [‘black’,’blue’]

sizes = [‘small’, ‘medium’]

list = [[color,size] for color in colors

for size in sizes]

**Output:**

**list = [[‘black’,’small’],[‘black,’medium’],[‘blue’,’small’],[‘blue’,’medium’]]**

**Tuples:**

-immutable

-there must always be a comma at the end if there is only one item in a tuple

t = ()

t = (a,) 🡪 Singleton tuple

t = (name, age, salary) 🡪 tuple

Note: \*() is optional for tuple

t = name, age, salary 🡪 also a tuple

Example:

t = ‘jones’, ‘mary’, 1023, true

last, first, empno, status = t 🡪 tuple unpacking

>>last

jones

>>t[1]

jones

>>t[2]

mary

**Dictionaries:**

-Datastructure in python (eg, map (key value pair))

**-unordered and mutable**

-can change the values of keys

-printing by iterating d will return keys by default

d = {}

d = {‘x’:10, ‘v’:5, ‘I’:1}

d[‘k’] = 50 🡪 Add an item

for numeral in d:

print(numeral

x v l k

-d.values() 🡪 gives values

-d.keys() 🡪 gives keys

-d.items() 🡪 gives list of key,value pair tuples

**Iterating through dictionary:**

for key,value in d.items():

print(key,”->”,value)

**get/setdefault:**

>>>d.get(‘x’)

d.get(x,x) second parameter for get is default, if there is no x in then returns x

‘10’

setdefault adds a value to the dictionary if there is no key

**sorted dictionary:**

for k in sorted(mydict , key=mydict.get)

print(k, mydict[k])

**removing items:**

del = remove an item from the dict

dict.pop(key) = remove item and return value

dict.clear() = empty out the dict

**sets:**

unordered collection, no duplicates

s = {‘a’, ‘b’,’c’}

s.add(‘c’)

difference between dict and set

d = {key:value, key:value} 🡪 key value pair

s = {value, value2,value3} 🡪 unique set, no duplicates

Examples:

even = {2,4,6,8}

odd = {3,5,7}

prime = {2,5,7}

>>even & prime 🡪 **&** gives the intersection

{2}

>>odd & prime

{5,7}

>>even | primt 🡪 **|** gives the union

{2,5,7}

>>prime – even 🡪 **-** gives the difference

{5,7}

>>prime ^ odd 🡪 **^** gives the symmetric difference (XOR) (excludes the intersection)

{2,3}

**File I/O:**

fileobj = open(filename, mode) mode is one or two letters

* r = read
* r+ = open for reading and writing
* w = write (create/overwrite)
* x = write, but only if file does not already exist
* a = append, if file exists (unless a+, then create)
* second letter =
* t = text file (default) b = binary

fileobj.close()

**open()** returns file object

**read()** reads bytes

**readline()** reads a line at a time

**readlines()** reads all lines–shouldn't be used can also iterate through a file object a line at a time

**with** statement sets up a temporary context (block) for file I/O and automatically closes file when block is exited

**Functions:**

-**def** introduces a function, followed by a function name.

*def noop():*

*pass*

eg,

*def simfunc(x):*

*if x==1:*

*print(“One”)*

*else:*

*print(“Not one”)*

*return something*

**-text that immediately follows the function definition is the docstring**

def func():

‘’’ this is a doc string, which will be shown in help

‘’’

**Seeing help for a function:**

-help(func) 🡪 prints out formatted doc string

-func.\_\_doc\_\_ 🡪 prints our unformatted doc string

Notes:

1. if function does not return anything, then it returns ‘None’
2. not same as false

**Arguments:**

-arguments does not need to be ordered (better to order though)

-you may specify arguments by name, in any order once you specify a keyword argument, all arguments following it must be keyword arguments

>>menu(‘chili’,dessert=’tattu’,entrée=’polenta’)

-positional arguments

-default arguments

**Variable Arguments:**

example print. It will print out all the arguments

>>>def func(\*args):

print(‘the args are’,args)

**kwargs:**

* if a function needs a bunch of configuration options, having default values which typically aren't overridden?
* one way to do this would be to have the function accept a dict in which these value(s) can be specified better way is to use variable keywords arguments

**Eg:**

def vka(\*\*kwargs):

for key in kwargs:

print(key, kwargs[key])

>>vka(debug=True, x=5, color=’red’)

x 5

debug True

color red

**Functions Recap:**

* Python encourages functions which support lots of arguments with default values
* "Explicit is better than implicit"
* arguments can be passed out of order ONLY if they're passed by keyword
* keywords are more explicit than positions because the function call documents the purpose of its arguments
* variable positional args (\*args) variable keyword args (\*\*kwargs)

**Scopes:**

locals(), globals() 🡪 Gives you a dict{} of all local or global variables in that block

**Python follows “Pass by Assignment”**

**Exceptions:**

* exceptions are "thrown" and either "caught" by an exception handler, or propagated upward
* try block wraps code which may throw an exception, and except block catches exception

Example:  
try:

mylist[5]

except:

print(“Oops, no element”) 🡪 catches all exceptions

except IndexError:

print(‘Bad index!’) 🡪 Catch specific exception

except Exception as e:

print(‘Exception’,e) 🡪 do something with exception e

**try/else:**

try also has an else block.

as long as try succeeds, else will execute.

**try/finally:**

code in the finally block will be executed whether or not an exception is thrown

**Command Line Arguments:**

when we run Python programs directly, we often want to pass arguments to the program

*>>> import sys*

*print(‘Program arguments’,sys.argv)*

**Modules:**

modules are just files of Python code

two ways to import: from module import stuff and

import module

don't use from module import \* except for testing private data is not really private!

packages are directories containing one or more Python modules

**private variables: (works only if u use ‘import module’ or ‘from module import \*’)**

private data is prefaced with an underscore (\_)

eg,

def dummy()

print(‘data’)

public\_data = “public stuff”

**\_private\_data** = “private data”

**Regular Expressions:**

**Modules:**

**OS Module:**

>>os.system(‘ls’)

>>os.getcwd()

>>os.mkdir(‘newdir’)

>>os.path.exists(‘newFile’)

>>os.path.isfile(‘newfile’)

>>os.path.isdir(‘newdir’)

**Sys Module:**

sys.path

sys.argv

sys.exit(12)

**shutil Module(Shell Util):**

shutil.copy(‘newfile’,’newfileCopy’)

shutil.move(‘newCopy’,’newFile’)

**glob Module:**

glob() function matches file or directory names using Linux shell rules rather than regular expression syntax

**subprocess Module:**

supplants os.system()/os.spawn(), both of which used to be standard way to run programs outside of Python

import subprocess

ret = subprocess.getoutput(‘ls’)

**optparse / argparse Module:**

**OO Programming/Classes:**

class = programmer-defined type

>>class Person():

pass

somebody = Person()

**METHODS ARE FUNCTIONS OF A CLASS. IF A FUNCTION BELONG TO A CLASS THEN IT IS CALLED A METHOD.**

* \_\_init\_\_ is a special initialization method that is invoked when the object is instantiated
* \_\_str\_\_ returns a string representation of the object (i.e., for humans), maps to str() function
* \_\_repr\_\_ return unambiguous representation of the object which could be fed to Python interpreter to recreate the object, maps to repr() function