Python Notes

Steven Wang

March 27, 2016

```
#Author: Steven Wang    Date:20151221
#codecademy python

{} #curly braces

no datatype
use white spave to structure(no ";", use indent)
"True" "False"
comments:
#comments

"""
comments
"""


string:
print text
print "a string"
print "a" + "big" + "hat" #concatenate

raw_input("Please input:") #input function with hint message
variable.isalpha() #False if there is non-letter characters

"cat"[0] == c
"cat"[2] == t
"cat"[1:2] == at
"cat"[0:len("cat")] == cat

len(variable)
variable.lower()
variable.upper()
str(variable)    #explicit string conversion
#methods that use dot notation only work with strings, or other
    words; len() and str() can work on other datatypes

```

```
% formatting
print ("Hello, %s") %(variable)     # %s, placeholder
print ("Hello, %s and %s") %(variable1,variable2)

'Alpha' "Bravo" "Thailand's people" #no need for backslash

data & time:
from datetime import datetime #inport "datetime" module
now = datetime.now()
now.year now.month now.day now.hour now.minute now.second

"**" #to the power of

boolean operator:
not, and, or
-----------> priority

if __:
elif __: #else if, elif is only checked if the original "if"
   statement is False
else __:

function:
1.header: def function_name(arguments):
2.optional comment
3.body    #indented

import module
1.generic import:  import math
    call: math.sqrt()
2.function import: from math import sqrt  #pulling in just a
   single function from a module
    call: sqrt()
3.universal import: from math import *    #CAUTION: fill the
   program with many variable and function names
    call: sqrt()
#best to stick with module name

    import math
    everything = dir(math)
    print everything
#show everything available in math

max()
min()
abs()
type()
```

```
80
81 list:
82 list_name = [item_1, item_2]
83 empty_list = []
84
85 list_name[0]    #call by index
86 list_name.append()
87 len(list_name)
88 list_name[1:3]   #return the 2nd to the 4th
89 #think of string as a list of characters
90 list_name[:3]
91 list_name[6:]
92
93 list_name.index() # search the first corresponding element
94 list_name.insert('index', value)
95
96 for variable in list_name:   #loop eg. for i in list_1:
97
98 list_name.sort()   #modify the list so it is in order
99
100 dictionary # map in c++
101 d = {'key1':1, 'key2':2, 'key3':3}
102 dict_name[new_key] = new value
103 empty_dict = {}
104
105 del dict_name[key_name]
106 list_name.remove()
107
108 LOOP:
109
110 for key in dict_name:
111     print dict_name[key]
112 #CAUTION: dictionary is unordered
113
114 sum(variable) # summation
115
116 5/2 = 2
117 float(5)/ 2 = 2.5
118 "/" #continuation charater, next line is following up
119
120 list_name.pop(index) # return value and remove
121 list_name.remove(item) # find the item and remove
122 del(list_name[index]) # like 'pop' but no return
123
124 range # a shortcut to generate a list
125 range(6) #=> [0,1,2,3,4,5]  range(stop)
126 range(1,6) #=> [1,2,3,4,5]  range(start,stop)
```

```
range(1,6,3) #=> [1,4]  range(start,stop,step)

for item in my_list:
    print item

for i in range(len(my_list)):
    print my_list[i]

["0"] * 5 #=> ['0','0','0','0','0']

my_list = ['a','b','c']
#turn list into string
' '.join[my_list] #=> a b c
"---".join[my_list] #=> a---b---c
# '' and "" are interchangeable

form random import randint
int(raw_imput("hint")) #raw_input always returns string type

if x not in range(8) or y not in range(5)
break # to get out of the loop

print a,b

while loop_condition:

while/else: #else executes as long as while loop condition is
   evaluated False; if the loop exits as the result of break, [
   else] will not be executed

for/else: # [else]excutes after [for] only if [for] ends
   normally(not because of [break])

print char, #[,] character after [print char] keeps next [print
   ] keep its output in the same line

for key in my_dict: #you get key

enumerate: # return index
    for index, item in enumerate(list):
    print index, # output the index
    print item # output the list element

zip: #create pairs of elements, stops at the end of shorter
   list
    for a,b in zip(list_a, list_b):
        print a, b
```

```
169
170 "*" * 4 # return "****"
171
172 my_string.split() #split the string into a list of strings
173
174 if a not in b:
175
176 sorted(list) # return the ordered list
177
178 sum[my_list] #returns sum of all list elements
179
180 !!! # always devide by 2.0
181 float(len(my_list))
182
183 #dictionary is unordered key/value pairs
184 print my_dict.item() #=> print all keys and its values in
      random order
185 my_dict.keys() # returns an array of dictionary's keys
186 my_dict.item() # returns an array of dictionary's values
187
188 # tuple: a sequence of immutable object
189 # tuples are surrounded by "()"
190
191 print item, # the trailing comma keeps printing on the same
      line
192
193 #list comprehension: for/in if
194 my_list = [ i for i in range(51) if i % 2 == 0]
195 my_list = [x*2 for x in range(1,6)]
196 my_list = [x*2 for x in range(1,6) if (x*3) % 3 == 0]
197
198 #list slicing:
199 [start:end:step] #start is inclusive; end is exclusive
200 #omitting indices:
201 Python will pick up default:
202 #default start index is 0
203 #default ending index is the end
204 #default step is 1
205 my_list = [1,2,3,4,5]
206 my_list[3:] #=>[4,5]
207 my_list[:2] #=>[1,2]
208 my_list[::2] #=>[1,3,5]
209 my_list[::-1] #=>[5,4,3,2,1] {reversing a list)
210
211 #functional programming(anonymous function):
212 lambda; filter
213     my_list = range(16)
```

```
    filter(lambda x: x%3 == 0, my_list)

#bitwise operator:
#In python: to write numbers in bianry format by starting the number with [0b].
    0b100
    0b1101
bin() # takes an integer as input and returns its binary
oct() # like bin()
hex() # like bin()

int("42") #=> 42
int("110") #=> 6
int("0b110") #=> 6

# floor division in python is integer division;
# in python 3: 5/2 = 2.5; 5//2 = 2

0b0001 << 2 #=> 0b0100
0b0100 >> 2 #=> 0b0001

#bitwise compare
a = 0b0101010
b = 0b0001111

& #AND
0&0=0
0&1=0
1&0=0
1&1=1
a&b = 0b1010

| #OR
0|0=0
0|1=1
1|0=1
1|1=1
a|b = 0b0101111

^ #XOR: exclusive or operator; for a bit, either of the two is 1 but not both
0^0=0
0^1=1
1^0=1
1^1=0

a^b = 0b0100101
```

```
259
260 ~ #NOT opeartor: add one the number and then make it negative
261
262 # bit mask
263 # a bit mask can help you turn specific bits on, turn others
    off, or just collect data from an integer
264
265 data = 0b010011010
266 mask = 0b000010000
267 print data&mask
268
269 & # using[&] to turn on bits
270 | # using[|] to turn a corresponding bit on if it is off and
    leave it on if it is already on
271 ^ # using[^] to flip bits
272 << >> # using [<<] and [>>] to slide mask into place
273
274 # class:
275 # when a class has its own functions, those functions are
    called methods
276 # ["Eric"] and [my_dict] are instances of the [str] and [dict]
    class.
277 class NewClass(object): # in the parenthesis is the class from
    which the new class inherits
278                         # [object] is the simplest, most basic
    class
279                         # by convention, user-defined Python
    class names start with a capital letter
280     def __init__(self, age, name):  #[__intit__()] exist by
    default
281         self.age = age              #Python will use the 1st
    parameter that [__init__()] reveives to refer to the object
    being created.
282         self.name = name            #Python convention: it's
    overwhelming common to use [self] as the 1st parameter in [
    __init__()]
283
284     def method_1(self): # for any method in a class, you need
    to provide [self] as the 1st argument
285         pass                        # [pass] statement is a
    null operation; nothing happens when it executes; works as a
     placeholder
286
287     def method_d(self,a,b):
288         pass
289
290     member_variable_1 = True
```

```
291     member_variable_2 = "test"
292
293 my_object = NewClass(18, "python")  # parameter list starts
    from the parameter after [self] (you don't need to give [
    self])
294 print my_object.name                 # dot notation: to access
    attributes of out objects
295
296 # class scope:
297 # needs further attention
298 global variable #like [public] in c++
299 member variable #like [protected] in c++
300 instance veriable   #like [private] in c++
301
302 # Inheritance: "is-a" relationship
303 superclass  subclass
304 baseclass   derivedclass
305 parentclass childclass
306
307 class BaseClass(object):
308     def __init__(self,a,b):
309         pass
310
311 class DerivedClass(BaseClass):
312
313 #override(overload): # in a new Derived class, override a
    function is to define a function with the same function name
314
315 super # [super] directly access the attributes or methods of a
    superclass
316     def method(self,a,b):
317         return super(DerivedClass, self).method(a,b)
318
319 # 2 ways of initialization
320 #1.
321     def __init__(self,a,b,c,d):
322         BaseClass.__init__(self,a,b)
323         self.c = c
324         self.d = d
325
326 #2.
327     def __init__(self,a,b,c,d):
328         super(DerivedClass, self).__init__(a,b) # no [seld] in
    the 2nd parenthesis
329         self.c = c
330         self.d = d
331
```

```python
__repr__() # a built-in methods: representation, tell Pyhon how to represent an object of our class
    def __repr__(self):
        return 'DerivedClass(a=%s, b=%s)' % (self.a, self.b)

DerivedClass #=> "DerivedClass(a=1, b=2)" (assume a="1" and b="2")

#__repr__ goal is to be unambiguous
#__str__ goal is to be readable
#Container s __str__ uses contained objects  __repr__

#file I/O:
my_file = open("output.txt", "w")   # open "output.txt" in "w" mode("w" stands for "write"), store the result of this operation in a file object [my_file]
# "w": write only mode
# "r": read only mode
# "r+": read and write mode
# "a": append mode

my_file.write("abcd\n")
my_file.read()
my_file.close() # During I/O process, data is buffered. Python doesn't flush the buffer, that is, write data to the file until it's sure you've done writing. One way to do this is to close the file.

#automatically close the files: file objects contain a pair of built-in methods:
__enter__()
__exit__()  #when [__exit__()] is invoked, the file will be closed

with/as # use [with][as] to invoke [__exit__()]
with open("text.txt","w") as my_file:
    my_file.write("abcd")

#check if the file is closed:
print my_file.closed #Python file objects have a [closed] attribute, [True] for closed, [False] for not closed

#End of file
```

python