BIG DADDY'S PYTHON TOOLBOX For 3.5

Reserve Words

Comparsion / Conjunction

true **==** (equal) **false** **none** (i.e., null) **and** **not** **or**
in list, tuple, string, dictionary
is true if **same** object

Definition

class create a class
def create a function
del items in lists (del mylist[2]), whole strings, whole tuples, whole dictionaries

Module Management

import connects mod, i.e, import math
from gets a function from math import cos
as creates an alias for a function

Miscellaneous

pass (placeholder – no action)
with wrapper ensures _exit_ method

Functions

def, return(obj), yield, next
inside functions **yield** is like **return** except it returns a generator whose sequential results are triggered by **next**
global declares global inside a function
non local a variable inside a nested function is good in the outer function
lambda anonymous inline function with no return statement

```
a = lambda x: x*2
for i in range (1,6):
   print (a(i))
```

Error Management

raise forces a ZeroDivisionError
try except finally else return used in error handling blocks

try:	code with error potential
except:	do this if you get the error
else:	otherwise do this code
finally:	do this either way

assert condition=False raises **AssertionError**

Looping

while (some statement is true)
for example:

```
alist=['Be','my','love']
for wordnum in range(0,len(alist)):
   print(wordnum, alist[wordnum])
```

range range (1,10) iterates 123456789
break **continue**
break ends the smallest loop it is in;
continue ends current loop iteration

Decision Making

if elif else

```
def if_example(a):
if a == 1:
   print('One')
elif a == 2:
   print('Two')
else:
   print('Some other')
```

The Ternary if Statement
An inline **if** that works in formulas:
myval = (high if (high > low) else low) * 3

Multi-line Statements \
Not needed within the [], {}, or ()
Multiple Statements on a Line ; not with statements starting blocks

Reading Keystrokes

```
text = ""
while 1:
  c = sys.stdin.read(1)
  text = text + c
  if c == '\n':
     break
print("Input: %s" % text)
```

You must import sys before you can use the standard input (sys.stdin.read) function.

Major Built-In Functions

String Handling (↳=converts / returns)

str(object) ↳string value of object
repr(object) ↳ printable string
ascii(str) ↳ printable string
eval(expresion) ↳ value after evaluation
chr(i) ↳ character of Unicode [chr(97) = 'a']
input(prompt) ↳ user input
len(—) ↳ length of str, items in list/dict/tuple
ord(str) ↳ value of Unicode character
slice(stop) or **slice(start, stop [,step])**
↳a slice object specified by slice (start, stop, and step) word = "Python"; word[2:5]='thon'
format(value [,format_spec]) ↳ value in a formatted string—**extensive and complex** - 2 examples (comma separator & % to 3 places)
print('{:,}'.format(1234567890)) yields '1,234,567,890'
print('{:.3%}'.format(11.23456789)) yields '1123.457%'

Number Handling

abs(x) ↳ absolute value of x
bin(x) ↳ integer to binary bin(5)='0b101' (one 4, no 2's, one 1)]
divmod(x,y) takes two (non complex) numbers as arguments, ↳a pair of numbers - quotient and remainder using integer division.
float(x) ↳ a floating point number from a number or string
hex(x) ↳ an integer to a hexadecimal string hex(65536) = ox10000
int(x) ↳ an integer from a number or string
pow(x,y [,z]) ↳ x to y, if z is present returns x to y, modulo z
round(number [,digits]) ↳ floating point number rounded to digits; Without digits it re-turns the nearest integer.

Miscellaneous Functions

bool(x) ↳ true/false, ↳ false if x is omitted
callable(object) ↳true if object callable
help(object) invokes built-in help system, (for interactive use)
id(object) ↳unique object integer identifier
print(*objects, sep=' , end='\n', file= sys.stdout, flush=False) prints objects separ-ated by sep, followed by end; **%** *see other side*

Data Container Functions type=list/tuple/dict

all(iterable) ↳ TRUE if all elements are true or it is empty
any(iterable) ↳ TRUE if any element is are true, FALSE if empty
type(enumerate(iterable, start = 0))
plist = ['to','of','and']
print(**list**(enumerate(plist)))
↳ [(0,'to'), (1,'of'), (2,'and')]

Use enumerate to make a dictionary: ex 2:
mydict = {**tuple**(enumerate(mytup))}
For dictionaries it enumerates keys unless you specifiy values, ex 3:
print(**dict**(enumerate(mydict.values())))

type([iterable]) ↳a mutable sequence; if a=[7,8,9] then list([a]) returns [[7, 8, 9]]
max(type) min(type) - **not** for ~~tuples~~
sum(iterable [, start]) must be all numeric, if a=[8,7,9] then sum(a) returns 24
sorted(iterable [,key=][,reversed]) reversed is Boolean with default False; strings without key sorted alphabetically, numbers high to low; key examples: print (sorted(strs, key=len)) sorts by length of each str value; 2 examp: key= strs.lower, or key = lambda tupsort: tupitem[1]
reversed(seq) - **reversed is tricky**, does **not** return a reversed list; if a=[4,5,6,7] then for i in reversed(a) yeilds 7/6/5/4; to get a reversed list for list mylist use:newlist = **list**(reversed(mylist))

range (stop) or
range (start, stop [, step])
tuple(iterable) not a function, an immutable sequence, mytuple=('dog',42,'x')
next(iterator [,default]) next item from iterator by calling next(iter). Default is returned if the iterator is exhausted, otherwise StopIteration raised.

```
>>> Mylist =[2,4,6,8];   MyItNum = iter(Mylist)
>>> next(MyItNum)  —> 2
>>> next(MyItNum)  —> 4          ......etcetera
```

File open (and methods)

fileobject=**open**(file [,mode],buffering])
The basic modes: **r, r+, w, w+, a** ..more
file object methods: **.read(size)**, **.readline**, **.readlines**, **list(fo)**, **.write(string)**, **.close**

```
with open("C:\Python351\Jack.txt",'r+') as sprattfile:
  sprattlist=sprattfile.read().splitlines() *<- removes '/n'
print(sprattlist)
-->['Jack Spratt', 'could eat ', 'no fat.', 'His Wife', 'could eat', 'no lean.']    *The WITH structure auto closes the file.
```

Other Functions filter(), vars(), dir(), super(), globals(), map(), dict(), setattr(), bytearray(), oct(), set(), classmethod(), zip(), locals(), __import__(), object(), memoryview(), hasattr(), issubclass(), exec(), compile(), hash(), isinstance(), complex(), bytes(), iter(), delattr(), property(), type(), getattr(), frozenset(), staticmethod()

String Methods

.find(sub[, start[, end]])
↳1st char BEFORE sub is found or -1 if not found
.capitalize() ↳first character cap
.lower() ↳ a copy of the string with all t converted to lowercase.
.center(width[, fillchar]) string is centered in an area given by width using fill character 'fillchar'
.ljust(width [, fillchar]) or **.rjust()**
.count(sub[, start[, end]]) number of substrings in a string
.isalnum() .isnumeric() .isalpha
.isdigit() .isspace() .islower()
.isupper .isprintable() may be null
↳ true if all char meet condition & at least one char in length
.replace(old, new[, count])
↳ a copy of the string with substring old replaced by new. If opt argument count is given, only first count are replaced.
.rfind(sub[, start[, end]])
↳ the **highest index** in the string where substring sub is found, contained within slice [start:end]. Return -1 on failure.
.strip([chars]) ↳ a copy of the string with the leading and trailing characters removed. The chars argument is a string specifying the set of characters to be removed. If omitted or None, the chars argument removes whitespace.
.zfill(width) ↳ a copy of the string left filled with ASCII '0' digits to make a string of length width. A leading sign prefix ('+'/'-') is handled by inserting the padding after the sign character rather than before. The original string is returned if width is less than or equal to len(str).
str.split() - separates words by space