BIG DADDY'S PYTHON TOOLBOX For 3.5+

Reserve Words

Comparsion / Conjunction

True, == (is same as), **False, none** (i.e., null), **and, not, or, in** list/tuple/string/dictionary/set; **is** or **is not** == comparison ↳ 'True' or 'False'

Definition

class create class: class ClassName: see below
def create function: def FName(args):
del deletes variables, data containers, items in iterables: del mylist[x]
ITERABLE: a data container with changeable items

Module Management

import connects module, ex: import math
from get a single module function: from math import cos; print (cos(9)) *no module preface
as creates an alias for a function

Miscellaneous

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

Functions

def, return(obj), yield, next
def creates; inside functions **yield** is like **return** but returns a generator whose sequential results are triggered by **next**;
global x declares global var in function
non local a variable inside a nested function is good in the outer function
lambda unnamed inline function, no return needed

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

Error Management

raise forces a ZeroDivisionError
try except else finally assert
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 expression:

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

range (start, stop, [step])
See data container functions
break ends the smallest loop it is in;
continue ends current loop iteration

Decision Making

if elif else

```
def if_example(MyInt):
  if MyInt == 1:
     print('One')
  elif MyInt  == 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 [], {}, or ()
Multiple Statements on a Line ; not with statements starting blocks like **if**

Functions not covered here:

vars(), dir(), super(), globals(), memoryview(), setattr(), bytearray(), classmethod(), locals(), __import__(), object(), hasattr(), issubclass(), isinstance(), compile(), hash(), complex(), bytes(), exec(), delattr(), property(), getattr (), staticmethod()
for some of those not covered here see:
www.wikipython.com

Major Built-In Functions

String Handling (↳=converts/returns)

str(object) ↳string value of object
repr(object) ↳printable representation string
ascii(str) ↳like repr but escape non-ascii
eval(expresion) ↳ value after evaluation
chr(i) ↳ character of Unicode [chr(97) = 'a']
ord(str) ↳ value of Unicode character
input(prompt) ↳ user input as a string
len(—) ↳ length of str, items in list/dict/tuple
slice selection **[[start[:]] [[:]stop] [:step]]**
↳a new string object created by the selection
str.join('string seperator',[string list])
format(value [,format_spec]) ↳ value in a formatted string—**extensive and complex** - 2 syntactical structures (1) simple format only: **format(number/string,'format string')**
(2) format and/or substitution: **'{:order or format string}'.format(objects);**
format string attributes/required order:

[[fill] align] [sign] [#-alt form] [0 forc-ed pad] [width] [,] [.precision] [type]

Key **types**: '**f**'/'F' fixed point, default=6; '**g**'/'G' general; '**e**'/'E' exponential; **%** percent; '**c**' Unicode char; ex: format(number,'**0=+20,.3f**')
↳ +000,000,012,345.679
Substitution using format():
"{variable to output} | {numeric format}…".format ('string' or numeric values…)
'**{0[x]}**' selects the xth value in a tuple which format specifies: ex: print ('**{0[x]}**'.format(mytup))
Also: format dates with help of datetime module. SEE WWW.WIKIPYTHON.COM ➔ TB4: Formatting Options

Number Handling

abs(x) ↳ absolute value of x
bin(x) ↳ integer to binary **bin(5)=** '0b101' (one 4, no 2's, one 1) **bin(7)[2:] = '111'**
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 an integer or string **A='1.1'; print(float(A)*2)** ↳**2.2**
hex(x)↳integer to hex string **hex(65536)** ↳**0x10000** or **hex(x)[2:]='10000'** also **oct(x)** ↳int to octal
int(x) ↳ integer from a decimal, string, hex
pow(x,y [,z]) ↳ x to y, if z is present returns x to y, modulo z **pow(2,7)=128, pow(2,7,3)=2**
round(number [,digits]) ↳ floating point number rounded to digits; Without digits it re-turns the nearest integer. **Round(3.14159,4)=3.1416**

Miscellaneous Functions

bool(x) ↳ true/false, ↳ false if x is omitted
callable(object) ↳true if object is 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 separated by sep, followed by end;

File open (and methods)

wholeFilePath = "C:\\\\file\\\\test\\\\mytest.txt"
fObj=**open**(file[,mode],buffering]) basic modes: **r, r+, w, w+, a** ..more helpful object methods: **.read(size), .readline(), .readlines(), .write(string), .close(), .splitlines ([keepends]), list(openfile)**

```
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.*

Operators

Math: = (= can also value swap; a, b = b, a), **+**, **-**, *****, **/**, **//** (floor or truncated division - no remainder), ****** (exponent), **%** (mod or modulo returns the remainder) x = 8%3; print(x) ↳2
Boolean/Logical: and, or, not *not(a [and/or] b)*
Comparison: == (same as), <, <=, >, >=, is, is not, !=(is not equal); operators can be chained
Membership: in , not in
Identity: is/is not checks for same object
Bitwise: & (and), | (or), ^ (xor 1 not both), ~ flips last bit << (shift left), >>(shift right) >>> bin(0b0101 <<1) ↳'0b1010'
Assignment: (execute & assign) =, //=, -=, +=, *=, /=, **=, %=
Sequence Variable Opers (for strings) + is concatenation (strx + stry) , * is repetition (strx*3)=strx+strx+strx; s[i] single slice, s[i:j:k] range slice from, to, step -> *starts at 0, end - count from 1; ie 1 more than qty needed* ☹
r'*str*' **raw** string/byte obj suppresses ESC chrs

Escape Characters

Nonprintable characters represented with backslash notation: **r** ignores esc chars;
\n Newline, **\b** Backspace, **\s** Space, **\cx** or **\C-x** Control-x, **\e** Escape, **\f** Formfeed, **\t** Tab, **\v** Vertical tab, **\x** Character x, **\r** Carriage return, **\xnn** Hexadecimal notation, n is in the range 0-9, a-f, or A-F; **many more**

Helpful String Methods

.find(sub[, start[, end]])
↳First char BEFORE sub is found or -1 if not found ex: print('Python'.find("th")) ↳ 2
.rfind(sub[, start[, end]])
↳ the **highest index** in the string where substring sub is found, contained within slice [start:end]. Return -1 on failure.
.capitalize() ↳first character cap'ed
.lower() ↳ a copy of the string with all text converted to lowercase; **.upper()**
.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
Attributes: **isalnum**, **isalpha**, **isdecimal**, **isdigit**, **isidentifier**, **islower**, **isnumeric**, **isprintable**, **isspace**, **istitle**, **isupper** - may be null, ↳ true if all char meet condition and variable is 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.
.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. Also **lstrip / rstrip**
.split() - returns list of words extracted by an intervening space.
str.join(iterable) - concatenates strings in iterable; str is the separator
Others include: casefold, join, encode, endswith, expandtabs, format, format_map, index, partition, maketrans, rindex, rpartition, rsplit, , splitlines (keepends), **title, startswith, swapcase, translate, upper, zfill**

BIG DADDY'S PYTHON TOOLBOX For 3.5+

Data Containers Methods / Operations

In notes below: (i/j/k-> an **index**; x->value or **object**; L/T/D/S-> an **instance** of a **list**, **tuple**, **dictionary**, or **set**.

LISTS: create: L=**[x,x,...]**; L=**[]**; **.insert**(i,x); **.append**(x); **.extend**(x,x,...); **.remove**(x); **del L**; **.pop**(); **.pop**(i); **L[i]=x** *replace*; **L[i:j]** =[x,x...] *replace multi-items;* **index#=L.index (x**[, *at or after index* i [, *before index* j]]**)** *retrieve index number of first value of x*; **V**=**iter**(L) *creates iteration generator*; **next** (V,default) *to step thru iteration*; **len**(L); **.count**(x); **.max**(L), **.min**(L); if v **in** L *determine membership*; **.copy**(); **.sort** (key=none, reverse=False); **.reverse; .clear**; L=**list(tuple)**

List Comprehensions

Make a new list with item exclusions and modifications from an existing list: brackets around the expression, followed by 0 to many **for** or **if** clauses; clauses can be nested

NewLst = [[modified]item for item in OldLst if some -conditional-item-attribute of (item)] or if modifying x only, ex: up1lst =[x+1 for x in ylist]

TUPLES: create =**(x,[x],(x),...)** *objects can include lists and other tuples*; *parens not required; **+=** *add items*; **+=(x,)** *add singe item*; **tuple[i:j]** *start is 0, end j-1*; **x,x,...=T[i:j]** *retrieve values*; **i=T.index(x[,***at or after index* **i [,***before index* **j]])**; **for int in T**; v=**iter(T)** *creates iteration generator*; **next** (v) *next iteration*; **len(T)**; **.count**(x); **.max** (T); **.min**(T); x **in** T; T**[::-1]**; **sorted (**T, **reverse=False)**; **T=()** *clears values*; **del** T; **T=tuple(somelist)** *creates tuple from a list*

DICTIONARIES: create: D=**{k:v, k:v,...}** ; D=**dict.fromkeys**(keys/list[,values]); D.**update**(D2) *adds D2 to D*; **D[k]** *returns v mapped to k*; **del D[k]** deletes key and item; D.**pop**(k[,default]); D.**popitem()**; D.**items()** *key and value*; D.**keys()**; D.**values()**; D.**get** **(**k[,x]**)** *like D[k] but D**.get(k,x)*** ↳ x if no k; v=**iter**(D) *creates iteration variable*; **next**(v) *step thru iterations*; **len**(D); v **in** D; v **not in** D; D.**has_key**(v); D.**copy()**; D.**clear()**; **del** D; D.**setdefault(k[,default])** *if k is in the dictionary return the key value, if not, insert it with default value and return default*

SETS: ***no duplicates*** create: S=**set**()←*empty;* S=**{x,x,x}**; S=**set**(**L**) *use list as set items*;; **S="string** ↳ *unique letters*; **.union**(S2); **.update**(S2); .**intersection**(S2); **.add**(x); **.difference**(S2); **.remove**(x) *gives KeyError if not present*; **.discard**(x); **.pop()**; **.copy()**; **.isdisjoint**(S2) *true if no common items*; **.issubset**(S2) *or* **S<=S2** *contained by*; **S<S2** *true if both S<=S2 and S!=S2 (is not equal)*; **.issuperset**(S2) *or* **S>=S2**; **S>S2**; v=**iter**(S) *create iteration variable*; next(v); **len**(S); S **in**; S **not in**; **.clear()** *all elements*; **del** S

FROZEN SET: *a set immutable after creation*
S=**frozenset**([iterable])

comments and suggestions appreciated:
john@johnoakey.com

Data Container Functions

all(iterable) ↳ TRUE if all elements are true
any(iterable) ↳ TRUE if any element is true
both all and any are FALSE if empty
enumerate(iterable, start = 0) ↳list

```
alst = ['x','y','z']
print(list(enumerate(alst)))
↳ [(0,'x'), (1,'y'), (2,'z')]
```

Use enumerate to make a dictionary: ex: mydict = dict(enumerate(mylist)) **Dictionaries** enumerate keys & yield values unless values specified; print (**dict** (enumerate(mydict.values()))) yields keys

type([iterable])
↳a datatype of any object
max(type) min(type)
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; ex: key= strs.lower, or key = lambda tupsort: tupitem[1]
reverse() reverses in place; mylist.**reverse()**
reversed() reverses access order—list or tuple

```
alist=["Amy","Bo","Cy"]
alist.reverse()
for i in alist:
   print(i)
for i in reversed(alist):
   print(i)
```

↳
```
Cy
Bo
Amy
Amy
Bo
Cy
```

Reverse a word
```
word = "Python"
iterword = iter(word)
newword =""
for i in reversed(word):
    newword +=i
print (word, newword)
```

range (stop) or **(start, stop [,step])**

```
alist=["Amy","Bo","Cy"]
for i in range (0,len(alist)):
  print(i, alist[i])  #note slice
```

↳
```
0 Amy
1 Bo
2 Cy
```

iter and **next(iterator [,default])** Create iterator then fetch next item from iterator. Default returned if iterator exhausted, otherwise StopIteration raised.

```
alist=["Amy","Bo","Cy"] ; IterNum = iter(alist)
print(next(IterNum, "listend"))
print(next(IterNum, "listend"))
print(next(IterNum, "listend"))
print(next(IterNum, "listend"))
```

↳
```
Amy
Bo
Cy
listend
```

map(function,iterable) can take multiple iterables but function must take just as many

```
alist=[5,9,13,24]
x = lambda z: (z**2 if z**2 < 150 else 0)
itermap = map(x,alist)
for i in alist:
    print(next (itermap))
```

zip an iterator that merges iterables left to right
filter(function, iterable) iterator for element of iterable for which function is True.
getattr(object, 'name' [, default])
setattr(object, 'name', value)

*args and *kwargs:

***args and *kwargs:** used to pass an unknown number of arguments to a function. *args is a list; *kwargs is a keyword->value pair where keyword is not an expression

```
def testargs (a1, *argv):
  print('arg#1: ',a1)
  for ax in range(0,len(argv)):
    print ("arg#"+str(ax+2)+" is "+argv[ax])
testargs('B', 'C', 'T', 'A')
```

↳
```
arg#1:  B
arg#2 is C
arg#3 is T
arg#4 is A
```

```
def testkwargs(arg1, **kwargs):
    print ("formal arg:", arg1)
    for key in kwargs:
        print ((key, kwargs[key]))
testkwargs(arg1=1, arg2="two", dog='cat')
```

↳
```
formal arg: 1
('dog', 'cat')
('arg2', 'two')
```

CLASS:

CLASS: (Your very own complex data object **blueprint**.)

DESIGN:
```
class YourClassName (inheritance, most commonly: object):
  def __init__(self, mandatory variables,...):
    self.accessname = mandatory variable1  ...repeat as necessary
add other functions:  def getAttribute1(self):
                          return self.accessname
```

MyInstanceName is the variable for **self**

CREATE AN INSTANCE:
```
MyInstanceName = ClassName(mandatory variables values)
```

ACCESS YOUR INSTANCE DATA: ex: calling a **get value** function
```
print(MyInstanceName.getAttribute1())
```

re-Regular Expresions module

A language in itself. It supports pattern matching on (1) a module level - for **1 time use** and (2) compiled expressions. To compile an expression set a variable like **patrn = re.compile (pattern)** then use it to **search** or **match. patrn** can be split over several lines. **Match** searches from first character only. Also you can use: **findall()** and **finditer().**

```
import re #if not found 'None'; attribs error
teststring = "Taking Math 101 is fun!"
mypat = re.compile(r'\d+', flags=0) see below
myso = mypat.search(teststring)
print (myso)
print ('group()', myso.group())   ↳  101
print ('start()', myso.start())   ↳  12
print ('end()', myso.end())       ↳  15
print ('span()', myso.span())     ↳  (12,15)
```

...or don't compile it...
```
print(re.search(r'\d', teststring).start() )
```

Special characters . ^ $ * + ? { } [ ] \ | ( )
Use Python r (raw) to process \ commands
r'(pattern)' matches literally: . any except newline \d decimal digit \D non-decimal \w any alphanumeric \W non-alphanum \s any white space chr \S non-whtspace * 0 or more + 1 or more ? 0 or 1 X{n} exactly n ,'X' chars X{m,n} between m & n X's **$** end of str **|** OR: a|b matches a OR b (...) whatever re is in the parens (?abcdef) one or more letters in parens (?=...) a look ahead assertion, "only if" (?!=...) negated look-ahead assertion, "not if" \A match only at start of string \Z match only end of string \b empty string at the start/end of a word [ ] contains a set of chars to match: '-' a range – [a-c] matches a,b,or c special chars lose meaning inside [ ], ^ as 1st char starts complimentary match

Flags: DOTALL any char, A escapes match ASCII, IGNORECASE, MULTILINE affecting ^$, VERBOSE) ***About Backslashes****: use the '\' character to indicate special forms or allow a character's use without invoking its special meaning—be aware this conflicts with Python's usage of the same character in string literals.*

To test for true result, use bool operator:
```
if bool(re.search(r'\d', teststr))==True:
```

Useful Module/Functions

Python Standard Library Module
https://docs.python.org/3.5/library
math like Excel math functions ceil (x) fsum(iterable), sqrt(x), log (x[,base]), pi, e, factorial(x) **random** seed([x]), choice(seq), randint(a, b), randrange(start, stop [, step]), random(x) - floating point [0.0 to 1.0] **sys** stdin standard input, stdout std output, exit(error msg) **datetime** date.today(), datetime.now(), **time** asctime (t), clock(), sleep(secs) **calendar**—a world of date options

```
>>> c = calendar.TextCalendar
(calendar.SUNDAY)
>>> c.prmonth(2016, 9)
   September 2016
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

This only works with a mono-spaced font like **Consolas** .

tkinter Python's defacto GUI; also see **ttk; tix**; see TB4 on **wikipython;** older version was Tkinter (capital T); **os** deep operating system access **array** arrays; **tarfile/zip-file** - file compression; **wave** - interface to wav format; **RPi.GPIO** - control Raspberry Pi pins via Python; **csv** access data: comma separated vals

A note on format: (1) **new f string options** available in version **3.6** (2) the old string % syntax will eventually be deprecated: print("$%.2f buys %d %ss"%(1.2,2,'hot dog')) *try it*