BIG DADDY'S PYTHON ❶ TOOLBOX For 3.6+

*in this document the symbol ↳ means yields, results in, or produces

print() is a function
print(*objects*, separator="", end='\n')
print("Hello World!") ↳ **Hello World!**

Multiline (explicit join) **Statements: **
Not needed within [], {}, or ()
Multiple Statements on a Line: ; can not be used with statements like **if**

Number Tools

abs(x) ↳ absolute value of x
bin(x)↳ int to binary bin(5)= '0b101' (a 4, no 2's, a 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; if x="1.1" print(float(x)*2) ↳ 2.2
hex(x) ↳ int to hex string hex(65536) ↳ 0x10000 or hex(65536)[2:] ↳ '10000'
oct(x) ↳ int to octal
int(x) ↳ int from float, string, hex
pow(x,y [,z]) ↳ x to y, if z is present returns x to y, modulo z
pow(5,2)=25, pow(5,2,7)=4
round(number [,digits]) floating point number rounded to digits; without digits returns the nearest integer
Round(3.14159, 4) ↳ 3.1416
max, **min**, **sort** - see data containers
None -> **constant** for null; x=None

Operators

Math: =(execute/assign, = can value swap; a, b = b, a); **+**; **-**; *****; **/**; ****** (exp); **+=**; **-=**; ***=**; ****=**; **/=**; **//=** ("floor" div truncated no remainder;
% (**mod**ulo): ↳ remainder from division
Boolean: True, False (1 or 0)
Logical: and, or, not modify compare
Comparison: == (**same as**); **!=** (is **not** equal); **<**; **<=**; **>**; **>=**; **is**; **is not**; all ↳ a Boolean value (T/F)
Membership: in; **not in**; - a list, tuple, string, dictionary, or set
Identity: is; **is not** the same object
Bitwise: & (and); **|** (or); **^** (xor 1 not both); **~** inversion, = -(x+1); **<<** (shift left); **>>**(shift right) bin(0b0101 <<1) ↳ '0b1010'
Sequence Variable Operators (for strings) + ↳ concatenate , ***** ↳ repetition ; s[i] **single** slice; s[i:j:k] **range slice from, to, step** -> start at i, end j-1, increment by count

Decision Making

if elif else:
if somenum == 1:
  do something
elif somenum == 2:
  do something else
else:
  otherwise do this

Comments:
# line comment
""" block comment """

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

much more at **www.wikipython.com**
comments appreciated: oakey.john@yahoo.com

String Tools

Functions

ascii(str) ↳ like repr, escapes non-ascii
chr(i) ↳ character of Unicode [chr(97) = 'a']
input(prompt) ↳ user input as a string
len() ↳ length of str, or count of items in an iterable (list, dictionary, tuple or set)
ord(str)↳ value of Unicode character
repr(object) ↳ printable string
str(object) ↳ string value of object
slice selection **str[:***stop***]; str[***start***:***stop***[:***step***]]** ↳ a string object created by the selection

Methods

Attribute Info: .isnumeric(), .isdigit(), .isalpha(), .islower(), .isupper(), .isidentifier(), .isdecimal(), .isprintable(), .istitle(), .isspace(), .isalnum(), .isascii(), may be null, ↳ ***True*** if all characters in a string meet the attribute condition and the string is at least one character in length
.casefold() ↳ casefold - caseless matching
.count(sub[,start[,end]]) ↳ # of substrings
.encode(*encoding="utf-8", errors="strict"***)**
.endswith ***(suffix[, start[, end]])***
.expandtabs() replace tabs with spaces
.format_map(mapping) similar to format()
.index(sub[,start[,end]])=.find+ "ValueError"
"*sep***".join([string list])** joins strings in iterable with sep char; can be null - "" in quotes
.partition(sep) ↳ 3 tuple: **before, sep, after**
.replace(old, new[, count]) ↳ substring old replaced by new in object; if count is given, only the count number of values are replaced
.rfind(sub[, start[, end]]) ↳ lowest index of substring in slice [start:end]. -1 on fail
.rindex() like rfind but fail ↳ *ValueError*
.rsplit() like split except splits from right
.rstrip([chrs]) trailing chars or " " removed
.split() ↳ word list with intervening spaces
.splitlines(keepends=False) ↳ list of lines broken at line boundaries
.startswith(prefix[,start[,end]]**)** ↳ True/False
.find(sub[, start[, end]]**)** ↳ the index of **substring** start, or -1 if it is not found;
print('Python'.find("th")) ↳ 2
.translate(table) map to translation table

String Format Methods

.center(width[, fillchar]) string centered in width area using fill character 'fillchar'
.capitalize() ↳ First character capitalized
.format() - **see Format Toolbox!**
method: (1) substitution (2) pure format
(1) '***string {sub0}{sub1}***'**.format(0, 1)**
print("Give {0} a {1}".format('me','kiss'))
(2) '**{:*format_spec*}**'**.format(*value*)**
function: format(***value, format_spec***)
format_spec: ("format mini-language")
[[fill] align] [**sign**] [# - alt form]
[0 - **forced pad**] [width] [,] [.precision] [type]
x = **format**(12345.6789, " =+12,.2f") ↳ + 12,345.68
f-string: print(f"{'Charge $'}{9876.543: ,.2f}")
↳ Charge $ 9,876.54 *NEW in version 3.6, -> format language*
.ljust(width [, fillchar]) or **.rjust(***same args***)**
.lower() ↳ text converted to lowercase
.strip([chars]), lstrip(), rstrip() ↳ a string with leading and trailing characters removed. [chars] is the set of characters to be removed. If omitted or None, the [chars] argument removes whitespace
.swapcase() ↳ upper -> lower & vise versa
.title() ↳ titlecased version - words cap'ed
.upper() ↳ text converted to uppercase
.zfill(width) - left fill with '0' to len width

Looping

while (*expression evaluates as True*):
  process data statements; **else:**
for *expression to be satisfied*: ex:
alist=['A','B','C']; x=iter(alist)
for i in range (len(alist)):
  print(i+1, next(x)) *can use **else:***
else: while and **for** support else:
range (start, stop [,step])
continue skips to next loop cycle
break ends while loop, skips else:

Error Management

use in error handling blocks (**with**)
try: code with error potential
except [*error type*]**:** do if error
else: otherwise do this code
finally: do this either way
assert: condition = **False** will raise an *AssertionError*
raise forces a specified exception

Programmed Functions

def create function: def functName(args):
return(variable object) - return the value a function derived - *or* -
yield/next; in a generator function, returns a **generator** with sequential results called by **next**
global x creates global variable - defined inside a function
nonlocal a variable in a nested function is good in outer function

Creating a Function:

(required in red, optional in green)
Line 1 (*note example: a generator function*)
command key word *arguments*
Def ***name*** (input or defined params):
new function name *colon*
Line 2 a docstring (*optional*)
Line 2 or 3 to ? code block
Usual last line **return(**expression to pass back**)** *keyword to pass result*
or a *generator* passed using **yield**:

```
def gen1(wordin):
   for letter in wordin:
     yield(letter)
vowels, myword = 'aeiouy','idea'
for x in gen1(vowels):
  print(x if x in myword else '', end='')
  next
```

↳ **aei**

Lambda Function:

an unnamed inline function
lambda [parameters]**:** expression
z = lambda x: format(x**3,",.2f");
print(z(52.1)) ↳ **141,420.76**

Module Management

import get module, ex: import math
from get a single module function: from math import cos; print (cos(9))
as creates an alias for a function

File Management

wholefilepath="C:\\file\\test\\mytest.txt"
open(file[,mode],buffering])
basic modes: **r, r+, w, w+, a** ..*more*
helpful methods: **.readline(), .read(size), .readlines(), .write(string), .close(), list(openfile), .splitlines([keepends]),**
with open(wholefilepath) as textfile:
  textfile=mytest.read().splitlines()
The WITH structure closes a file automatically
Note: about a dozen functions not shown here

BIG DADDY'S PYTHON ❷ TOOLBOX For 3.6+

Data Containers Methods / Operations

In notes below: i,j,k: **indexes**; x: a value or **object**

L / T / D / S / F / SF ↳ **instances** of: **list**, **tuple**, **dictionary**, **set**, **frozen set**, **both**

Methods used by **multiple** iterable types

Method	Action	L	T	D	S	F
.copy()	duplicate iterable	x		x	x	x
.clear()	remove all members	x		x	x	
.count(x)	# of specific x values	x	x			
.pop(i)	return & remove i^th item	x		x	x	
.index(x)	return slice position of x	x	x			

Data Type *unique* statements/methods

LISTS: *create* **L=[]**; **L=list(L/T/S/F)**; **L=[x,x,...]**; *add* **.append**(x) or **+=**; **insert**(i,x); **.extend** (x,x,...); *replace* **L[i:j]**=[x,x...]; *sort* **L.sort(**key=none, reverse= False); *invert member order* **L.reverse**(); *get index, 1st value of x* = **L.index (x[**,*at/after index* i [,*before index* j**])**

TUPLES: *create* **T=()**; **T=(x,[[x],(x)...])**; **T= tuple(T/L/S/F)**; *create or add single item* **+=(x,)**; *clear values* **T=()** *get slice values* **x,x,...=T[i:j]**; *reverse order* **T[::-1]**; **sorted (T,** *reverse=True/False***)**; ex: T=sorted(T, reverse = True)

DICTIONARIES: *create* **D={k:v, k:v,...}**; **=dict.fromkeys(L/F [,*1 value*]); =dict (**zip(L1, L2)**)**; **=dict(**kwargs)**; *revalue & extend* **D.update(**D2**)**; *get values: v map to k:* **D[k]**; *like D[k] but* ↳ x if no k **D.get (**k[,x]**)**; **D.setdefault(k[,default])** *if k in dictionary, return value, if not, insert and return default*; *change value:* **D[k]=value**; *views:* **D.items(); D.keys(); D.values()** also see mapping from a list in more tools↗

SETS: *(no duplicates!, not immutable)* *create* **S=set(L/T/F)**; **S={**x,x,x**}**; **S**='string' ↳ unique letters; Change Set Data: **S.add**(element); **S1.update**(iterable) or **S |=** S1|S2|... **S.intersection_update(**iterable) or **S &=** iterable & ... **S.difference_update**(iterable) or **S -=** S1 | S2 |... *or any iterable* **S.symmetric_difference_update**(iterable) or S ^= iterable **S.remove**(element) *Key Error if missing;* **S.discard**(element) *no error*

FROZENSETS: *immutable after creation; create* **S=frozenset**([iterable]) ☛ *only*

Boolean Testing (**Sets** & **Frozensets**):
SF.isdisjoint(S2) common items?
SF.issubset(S2) *or* **<=** contained by
SF<S1 set is a proper subset
SF.issuperset(S2) *or* **SF=>**S2 contains
SF>S1 set is a proper superset

Change **Sets** or **Frozensets** Data:
SF.union(S2) *or* **SF=**S1**|**S2[**|**...] merge
SF.intersection(S2) *or* **S &** S1 intersection of S & S1 *ex: S3 = S1.intersection(S2)*
SF.difference(S2) *or* **S-**S2 *unique in S*
SF.symmetric_difference(S2) or **S^**S2 *elements in either but not both*

more on format: (1) the old string % syntax will eventually be **deprecated**: print("$%.2f buys %d %ss"%(1.2, 2, 'hot dog')) *try it* (2) **for 'f string' options** available in version **3.6** **see www.wikipython.com : format toolbox**

More Data Container Tools

all(iterable) ↳ True if all elements are True
any(iterable) ↳ True if any element is True
*all and any are both FALSE if empty
del(iterable instance) - delete
enumerate(iterable, start = 0) ↳ list of tuples
alist = ['x','y','z']; l1 = list(enumerate(alist)); print(l1)
↳ **[(0,'x'), (1,'y'), (2,'z')]**

Use enumerate to make a dictionary. ex: mydict = dict(enumerate(mylist))

filter(function, iterable) iterator for element of iterable for which function is True
in/not in - membership, True/False
iter and **next(iterator [,default])** create iterator with **iter**; fetch items with **next** ; default returned if iterator exhausted, or StopIteration

```
team = ['Amy', 'Bo', 'Cy'];   it1 = iter(team);   myguy = ""
while myguy is not "Cy":
    myguy = next(it1, "end")
    print(myguy)
```

The **collections** module adds **ordered dictionaries** and **named tuples**.

len(iterable) count of instance members
map(function, iterable) can take multiple iterables - function must take just as many
alist=[5,9,13,24]; x = lambda z: (z+2)
list2 = list(map(x, alist)); print(list2) ↳ [7, 11, 15, 26]
max(iterable[,key function, default]**)** *see*
min(iterable[,key function, default]**)** *lambda*
reversed() reverse **iterator**: **list** or **tuple**

```
alist=["A","B","C"]; print(alist)
alist.reverse(); print(alist);
rev_iter = reversed(alist)
for letter in range(0, len(alist)):
    print(next(rev_iter), end=", ")
```

↳ ['A', 'B', 'C'] ['C', 'B', 'A'] A, B, C,

sum(iterable [, start]) must be all numeric, if a=[8,7,9] then sum(a) returns 24
sorted(iterable [,key=][,reverse])
reverse is Boolean, default=False; strings without keys are sorted alphabetically, numbers high to low; key ex: print (sorted(list, key= len)) sorts by length of each str value; ex2: key= alist.lower, ex3: key = lambda tupsort: tupitem[1]
type([iterable]) ↳ a datatype of any object
zip() creates aggregating iterator from multiple **iterables**, ↳ iterator of tuples of i^th iterable elements from each sequence or iterable

Other Commands & Functions

Working with object attributes - most useful for created class objects, but can be educational:

```
listatr = getattr(list, '__dict__')
for item in listatr:
    print(item, listatr[item], sep="  |  ")
```

getattr(object, 'name' [, default])
setattr(object, 'name', value)
hasattr(object, 'name')
delattr(object, 'name')
range ([start,] stop [,step])

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

↳ 0 Amy 1 Bo 2 Cy

exec(string or code obj[, globals[, locals]]) dynamic execution of Python code
compile(source, filename, mode, flags=0, don't_inherit=Fales, optimize=-1) create a code object that exec() or eval() can execute
hash(object) - ↳ integer hash value if available
dir() - ↳ names in current local scope
dir(object) - ↳ list of valid object attributes

List Comprehensions

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

new_list = [(*modified*)item for item in old_list if some -item-attribute of (item)] *Example:*

```
atuple=(1,-2,3,-4,5)
newLst=  [item*2 for item in atuple if item>0]
print(atuple, newLst)   ↳ (1, -2, 3, -4, 5)  [2, 6, 10]
```

if modifying items only: **up1list =[x+1 for x in L]**

CLASS - an object **blueprint** or **template**

(required in red, optional in green)
Common components of a class include:
(1) *inheritance creates a "derived class"*
command key word colon
class class-name (inheritance):
your class name-class ***definition header***
Class creates a namespace and supports **two operations**: instantiation and attribute reference
(2) a **docstring**, '''Docstring example'''
(3) **instantiation** with **special method:**
def __init__(self, arguments):
which is autoinvoked when a class is created; Arguments are passed when a class instantiation is called. Includes variable name assignments, etc.
(4) **function definitions, local variable assignments**

ex:

```
❶ class mammalia(object):
❷   "A class for mammal classification"
❸   def __init__(self, order, example):
       self.ord = order
       self.ex = example
       self.cls="mammal"
❹   def printInfo(self):
       info="class/order: " + self.cls + "/"+\
       self.ord +", Example:" + self.ex
     print(info)
```

mam_instance = mammalia("cetacea","whales")
mam_instance.printInfo()
↳ **class/order: mammal/cetacea, Example: whales**

*/** for iterable unpack

or "argument unpack", 2 examples:
a,*b,c = [1,2,3,4,5]; ↳ b=[2,3,4]
y={1:'a', 2:'b'}; z={2:'c', 3:'d'}
c={**y, **z} ↳ c={1:'a',2:'c',3:'d'}

*args and *kwargs:

used to pass an unknown number of arguments to a function.

***args** is a **list**

```
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
```

***kwargs** is a **keyword -> value pair** where keyword is **not** an expression

```
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')
```

Miscellaneous

ITERABLE: a data container with changeable items
pass (placeholder – no action)
del deletes variables, data containers, items in iterables: del mylist[x]
breakpoint enters debugger
with wrapper ensures **_exit_** method
eval(Python expression) ↳ value
bool(expression) ↳ **T/F**(F default)
callable(object) ↳ True if it is
help(object) invokes built-in help system, (for interactive use)
id(object) ↳ unique identifier

Selected Escape Characters

Nonprinting characters represented with backslash notation, **'r'** *(raw) ignores esc chars before a literal*
\n newline, **\b** backspace, **\f** formfeed, **\t** tab, **\v** vertical tab...