BIG DADDY'S PYTHON TOOLBOX For 3.6+

www.wikipython.com *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!**

Coding Operators

\ Multiline (explicit join) Statements: Not needed within [], {}, or ()
; Multiple Statements on a Line: not used/needed with **for, if, while**
**line comment**
''' **block comment** **'''**

Operators

Math: =(execute/assign, = can value swap; a, b = b, a); **+**; **-**; ***; **/**; ****** (exp); **+=**; **-=**; ***=**; ****=**; **/=**; **//=** ("floor" div, truncated no remainder)
% (modulo): ↳ 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*

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

String Tools

Functions

ascii(str) ↳ like repr, esc non-ascii
chr(i) ↳ character of Unicode 97= 'a'
input(prompt) ↳ user input as str
len() ↳ length of str; count of iterable items (list/dictionary/tuple/set)
ord(str)↳ value of Unicode char.
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 .isallnum(), .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(xdict) the string is a key value in dictionary xdict
.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() see **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**(12345.678, **" =+12,.2f"**) ↳ + 12,345.68
f-string: print(**f**"{'Pay $'}{9876.543:,.2f}") ↳ Pay $ 9,876.54 *NEW in 3.6*
.ljust(width [, fillchar]) or **.rjust(***same args***)**
.lower()/.upper ↳ text case converted
.strip([chars]), lstrip(), rstrip() ↳ a string with leading and trailing [chars] removed. If [chars] omitted or None, the argument removes whitespace
.swapcase() ↳ upper -> lower & vise versa
.title() ↳ titlecased version - words cap'ed
.zfill(width) - left fill with '0' to len width

Decision Making

if elif else:
if somenum == 1**:** do something
elif somenum == 2**:** do something else
else:
otherwise do this
The ternary if: an inline **if** that can be use in formulas
print(x **if** x **in** myword **else** ", end=")

Looping

while (*expression evaluates as True*)**:**
process data statements; **else:**
break ends while loop, skips 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

Error Management

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

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 valid in an outer function

Creating a Function

(required in red, optional in green)
***1** (*examples: return & generator functions*)*
command key word *arguments*
def *name* (input or defined params)**:**
*new **function name*** *colon*
***2** a docstring
2,3-x** **code block
***last** **return**(expression to pass back)*
or a ***generator*** passed using **yield**:

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

↳ **aei**

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

Miscellaneous

definition: 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) ↳**T**rue if it is
help(object) invokes built-in help system, (for interactive use)
id*(object)* ↳ unique identifier
:= (New in 3.8) - assignment expression operator assigns values to variables inside a larger expression

BIG DADDY'S PYTHON 2 TOOLBOX For 3.6+

www.wikipython.com *in this document the symbol ↳ means yields, results in, or produces

 vPro2020A V082420

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 **in Common** by 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			

Unique Data Type 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: x =* **D[k]**; *like D[k] but* ↳ x if no k x = **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()** *support* **len**(D.view), **iter**(D.view), **x in** D.view, **reversed**(D.view) ← an iterator

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) **S ^=** other; keep unique elements only **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**): **len(SF); x in SF; x not in SF; 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*

The old format: (1) the old string % syntax will eventually be **deprecated**: ex: print("$%.2f buys %d %ss"%(1.2, 2, 'hot dog')) (2) for **'f string'** options (available in version 3.6) **and others, 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

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=False, 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 a 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)
new_list= [item*2 for item in atuple if item>0]
print(atuple, new_list)
```
↳ (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 provides 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 and 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", * for tuples, ** for dictionaries; examples:

```
a,*b,c = [1,2,3,4,5];
y={1:'a', 2:'b'}; z={2:'c', 3:'d'}
c={**y, **z}
```
↳ **b**=[2,3,4]
↳ 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')

Example: function, *, *args

```
def myfunc(*args):
    print(*args)
my_list = ['a1','b2','x','c3']
myfunc(*my_list)
del my_list[2]
myfunc(*my_list)
```

File Access

```
wholefilepath="C:\\file\\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

About a dozen functions not shown here

BIG DADDY'S

vPro2020A © 2020 John A. Oakey V082420

www.wikipython.com *in this document the symbol ↳ means yields, results in, or produces

PYTHON 3 TOOLBOX For 3.5

Python Documentation: Tables & Lists

Functions * boldface not in basic toolbox

abs()	callable()	enumerate()	hasattr()	list()	pow()	**staticmethod**
all()	chr()	eval()	hash()	**locals()**	print()	str()
any()	**classmethod**	exec()	help()	map()	**property()**	sum()
ascii()	compile()	filter()	hex()	max()	range()	**super()**
bin()	complex()	float()	id()	**memoryview**	repr()	tuple()
bool()	delattr()	format()	input()	min()	reversed()	type()
breakpoint()	dict()	frozenset()	int()	next()	round()	**vars()**
bytearray()	**dir()**	getattr()	**isinstance()**	**object()**	set()	zip()
bytes()	divmod()	**globals()**	**issubclass()**	oct()	setattr()	
			iter()	open()	slice()	
			len()	ord()	sorted()	

Keywords (reserved)

and as assert async await break class continue def del elif else except False finally for from global if import in is lambda nonlocal None not or pass raise return True try while with yield

Built-in Constants

False, True, None, NotImplemented, Ellipsis (same as literal '…'), __debug__, quit(), exit(), copyright, credits, license

Built-in Types

numerics, sequences, mappings, classes, instances, exceptions

Numeric Types

int **float** **complex**
constructors: ***(im defaults to 0)***
int() **float()** **complex(re, im)**

Numeric Operations

Operation	Result
x + y	sum of *x* and *y*
x - y	difference of *x* and *y*
x * y	product of *x* and *y*
x / y	quotient of *x* and *y*
x // y	floored quotient of *x* and *y*
x % y	remainder of x / y
-x	*x* negated
+x	*x* unchanged
abs(x)	absolute value or magnitude of *x*
int(x)	*x* converted to integer
float(x)	*x* converted to floating point
complex (re, im)	a complex number, real part *re*, imaginary part *im*. defaults to 0
c.conjugate()	conjugate of complex number *c*
divmod(x, y)	the pair (x // y, x % y)
pow(x, y)	*x* to the power *y*
x ** y	*x* to the power *y*

int and float also include these operations
math.trunc(x) math.floor(x) math. Ceil(x) round(x[,n])

Sequence Operations (4.6.1)

x in s True if an item of s is equal to x
x not in s False if an item of s is equal to x
s + t the concatenation of s and t
s * n or n * s concatenate s n times
s[i] ith item of s, origin 0
s[i:j] slice of s from i to j
s[i:j:k] slice of s from i to j with step k
len(s) length of s
min(s) smallest item of s
max(s) largest item of s
s.index(x[, i[, j]]) index of the first occurrence of x in s (at or after index i and before index j)
s.count(x) number of occurrences of x in s

Mutable Sequence Operations

s[i] = x item i of s is replaced by x
s[i:j] = t slice of s from i to j is replaced by the contents of the iterable t
del s[i:j] same as ***s[i:j] = []***
s[i:j:k] = t the elements of s[i:j:k] are replaced by those of t
del s[i:j:k] removes the elements of s[i:j:k] from the list
s.append(x) appends x to the end of the sequence
s.clear() removes all items from s (same as ***del[:]***)
s.copy() creates a shallow copy of s (same as ***s[:]***)
s.extend(t) or ***s +=*** extends s with the contents of t (for the most part the same as ***[len(s):len(s)] = t)***
s *= n updates s with its contentsrepeated n times
s.insert(i, x) inserts x into s at the index given by i(same as *s[i:i] = [x]*)
s.pop([i]) retrieves the item at i and also removes it from s
s.remove(x) remove the first item from s where s[i]== x
s.reverse() reverses the items of s in place

For important notes see:
https://docs.python.org/3.6/library/stdtypes.html

Operators and Precedence

lambda
if – else
or · **and** · **not x** Boolean OR, AND, NOT
in · **not in** · **is** · **is not**
< · **<=** · **>** · **>=** · **!=** · **==**
| · **^** · **&** bitwise OR, XOR, AND
<< · **>>**
+ · **-**
*** · @ · **/** · **//** · **%** (Multiplication, matrix multiply, division, floor div, remainder)
+x · **-x** · **~x** (pos, neg, bitwise NOT)
** (exponentiation)
await x (Await expression)
x[index] · **x[index:index]** · **x(argu-ments...)** · **x.attribute** (subscription, slicing, call, attribute reference)

Comparisons

Op	Meaning		
<	strictly less than	**<=**	less than or equal
>	strictly greater than		
>=	greater than or equal		
==	equal	**!=**	not equal
is	object identity	**is not**	negated object identity

Integer Bitwise Operations

Operation	Result
x | y	bitwise *or* of *x* and *y*
x ^ y	bitwise *exclusive or* of *x* and *y*
x & y	bitwise *and* of *x* and *y*
x << n	*x* shifted left by *n* bits
x >> n	*x* shifted right by *n* bits
~x	the bits of *x* inverted

Bytes and Bytearray Operations

x. = means this methon can be used with "bytes." or "bytearray." i.e.,
x.count(*sub*[, *start*[, *end*]]) is same as
bytes.count(sub[, start[, end]])
bytearray.count(sub[, start[, end]])
x.decode(encoding="utf-8", errors="strict")
x.endswith(suffix[, start[, end]])
x.find(sub[, start[, end]])
x.index(sub[, start[, end]])
x.join(iterable)
static bytes.maketrans(from, to)
static bytearray.maketrans(from, to)
x.partition(sep)
x.replace(old, new[, count])
x.rfind(sub[, start[, end]])
x.rindex(sub[, start[, end]])
x.rpartition(sep)
x.startswith(prefix[, start[, end]])
x.translate(table, /, delete=b'')
x.center(width[, fillbyte])
x.ljust(width[, fillbyte])
x.lstrip([chars])

continued next column

www.wikipython.com

x.rjust(width[, fillbyte])
x.rsplit(sep=None, maxsplit=-1)
x.rstrip([chars])
x.split(sep=None, maxsplit=-1)
x.strip([chars])
x.capitalize()
x.expandtabs(tabsize=8)
x.isalnum()
x.isalpha()
x.isascii()
x.isdigit()
x.islower()
x.isspace()
x.istitle()
x.isupper()
x.lower()
x.splitlines (keepends=False)
x.swapcase()
x.title()
x.upper()
x.zfill(width)

Constants

False
True
NotImplemented
ellipsis (same as '…')
__debug__
added by the site module
quit()
exit()
copyright
credits
license

Boolean Operations

Operation	Result (ascending priority)
x or y	if *x* is false, then *y*, else *x*
x and y	if *x* is false, then *x*, else *y*
not x	if *x* is false, True, else False

Open File Modes: *open for*

'r'	reading (default)
'w'	writing, truncating the file first
'x'	exclusive creation, fails if it already exists
'a'	writing, appending to the end of the file **if** it exists
'b'	binary mode
't'	text mode (default)
'+'	for updating (reading and writing)

f-string Formatting : conversion types

'd'	Signed integer decimal.
'i'	Signed integer decimal.
'o'	Signed octal value.
'u'	Obsolete type – it is identical to 'd'.
'x'	Signed hexadecimal (lowercase).
'X'	Signed hexadecimal (uppercase),
'e'	Floating point exponential format (lowercase).
'E'	Floating point exponential format (uppercase).
'f'	Floating point decimal format.
'F'	Floating point decimal format.
'g'	Floating point format. Uses lowercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
'G'	Floating point format. Uses uppercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
'c'	Single character - accepts integer or single character str
'r'	String (converts any Python object using repr()).
's'	String (converts any Python object using str()).
'a'	String (converts any Python object using ascii()).
'%'	No argument converted, puts '%' character before result

f-string : conversion flags

'#'	conversion will use the "alternate form"
'0'	conversion zero padded for numerics
'-'	value is left adjusted (overrides the '0')
' '	(space) A blank should be left before a + #
'+'	A sign character ('+' or '-') will precede the conversion (overrides a "space" flag).

Warnings Module

- Warning, FutureWarning, ImportWarning, ResourceWarning PendingDeprecationWarning, RuntimeWarning SyntaxWarning, UnicodeWarning, UserWarning

BIG DADDY'S PYTHON 4 TOOLBOX For 3.5

www.wikipython.com *in this document the symbol ↳ means yields, results in, or produces

Errors

ArithmeticError	DeprecationWarning	LookupError	SystemError
AssertionError	EOFError	MemoryError	TabError
AttributeError	EnvironmentError	ModuleNotFoundError	TimeoutError
BaseException	FileExistsError	NameError	TypeError
BlockingIOError	FileNotFoundError	NotADirectoryError	UnboundLocalError
BrokenPipeError	FloatingPointError	NotImplementedError	UnicodeDecodeError
BufferError	IOError	OSError	UnicodeEncodeError
BytesWarning	ImportError	OverflowError	UnicodeError
ChildProcessError	IndentationError	PermissionError	UnicodeTranslateError
ConnectionAbortedError	IndexError	ProcessLookupError	ValueError
ConnectionError	InterruptedError	RecursionError	WindowsError
ConnectionRefusedError	IsADirectoryError	ReferenceError	ZeroDivisionError
ConnectionResetError	KeyError	RuntimeError	
	KeyboardInterrupt	SyntaxError	

Escape Sequences

\ newline
\\ Backslash (\)
\' Single quote (')
\" Double quote (")
\a ASCII Bell (BEL)
\b ASCII Backspace (BS)
\f ASCII Formfeed (FF)
\n ASCII Linefeed (LF)
\r ASCII Carriage Return (CR)
\t ASCII Horizontal Tab (TAB)
\v ASCII Vertical Tab (VT)
\ooo Character with octal value ooo
up to 3 octal digits accepted
\xhh Character with hex value hh
exactly 2 hex digits required

The power of Python is its ability to add functions to fit just about any conceived programming need through the importation of specialized **MODULES** that integrate with, and extend, Python. About 230 of these modules, the "Standard Library", are downloaded automatically when Python is installed. There are over another 1,000,000 packages contributed by users in the PyPi online storage. A few highlights of the modules are noted below. Find PyPi at: **https://pypi.org/**

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

The Python Standard Library highlighted here

Text Processing Services - 7 modules including:
string — Common string operations
re — Regular expression operations
textwrap — Text wrapping and filling
Binary Data Services - 2 modules
Data Types – 13 modules including:
datetime — Basic date and time types
calendar — General calendar-related functions
copy — Shallow and deep copy operations
enum — Support for enumerations
pprint — Data pretty printer
Numeric and Mathematical Modules – 7 modules inc:
numbers — Numeric abstract base classes
math — Mathematical functions
cmath - complex #; decimal - accurate fp arithmetic
random — Generate pseudo-random numbers
statistics — Mathematical statistics functions
fractions — rational numbers
Functional Programming Modules – 3 modules:
File and Directory Access – 11 modules including:
pathlib — Object-oriented filesystem paths
os.path — Common pathname manipulations
fileinput — iterate lines from multiple inputs
filecmp — File and Directory Comparison
shutil — High-level file operations
Data Persistence – 6 modules including:
pickle — Python object serialization
marshal — Internal Python object serialization
sqlite3 — DB-API 2.0 interface for SQLite databases
Data Compression and Archiving – 6 modules inc:
zipfile — Work with ZIP archives
tarfile — Read and write tar archive files
File Formats – 5 modules including:
csv — CSV File Reading and Writing
Cryptographic Services – 3 modules:
Generic Operating System Services – 16 modules inc:
os — Miscellaneous operating system interfaces
time — Time access and conversions
io — Core tools for working with streams
platform — Accesss to platform identifying data
Concurrent Execution – 10 modules including:
threading — Thread-based parallelism
multiprocessing — Process-based parallelism
Interprocess Communication and Networking - 9 mods
Internet Data Handling – 10 modules:
Structured Markup Processing Tools – 13 modules:
Internet Protocols and Support – 21 modules:
Multimedia Services – 9 modules including:
wave — Read and write WAV files
Internationalization – 2 modules:
Program Frameworks – 3 modules including:
turtle — Turtle graphics
Graphical User Interfaces with Tk – 6 modules including:
tkinter — Python interface to Tcl/Tk
IDLE
Development Tools – 9 modules:
Debugging and Profiling – 7 modules:
Software Packaging and Distribution – 4 modules including: distutils — Building and installing modules
ensurepip — bootstrapping the pip installer
Python Runtime Services – 14 modules including:
sys — System-specific parameters and functions
sysconfig — Access to Python's config information
__main__ — Top-level script environment
inspect — Inspect live objects
Custom Python Interpreters – 2 modules
Importing Modules – 5 modules including
zipimport — Import modules from Zip archives
runpy — Locating and executing Python modules
Python Language Services – 13 modules:
keyword — Testing for Python keywords
py_compile — Compile Python source files
Miscellaneous Services – 1 module:
MS Windows Specific Services – 4 modules
Unix Specific Services - 13 modules:
Superseded Modules – 2; ***Undocumented Modules*** – 1
pypi.org another 257M+ modules including:
RPI.GPIO, Pillow, pandas, fuzzywuzzy, Anaconda, miniconda, conda, playsound, Poetry, Numpy, etc.

Selected Standard Library Module Constants and Methods for New Users

↳ = returns/yields

NOTES: (1) Just because you installed a module under Python version 3.x does NOT mean it will be available to you in 3.x+1
(2) How to find installed modules: from python:
>>> help('modules')

calendar import calendar
a couple of fun examples:
c=calendar.**TextCalendar**(calendar.SUNDAY)
c.pryear(2021,w=2,l=1,c=6,m=3) **or try**
c=calendar.**TextCalendar**(calendar.MONDAY)
c.setfirstweekday(calendar.SUNDAY)
print(c.formatmonth(2021,1,w=0,l=0))
many functions - **see: www.wikipython.com** -> OTHER MODULES -> calendar

cmath - A suite of functions for complex #

copy - import copy relevant for compound objects, (objects containing other objects)
.**copy**(x) <-relies on references to objects
.**deepcopy**(x[, memo]) <-copies objects (so you can change the copy and not the original)

csv import csv comma separated values
.**reader**(csvfile, dialect='excel',**fmtparams) file and list objects ; file obj opens with newline=""
.reader objects: **csv.reader(file path). +** __next__() , dialect , line_num , or fieldnames
.**writer**(csvfile, dialect='excel',**fmtparams)
writer objects: csv.writer(file path). + writerow(row) , writerows(rows) , dialect , or writeheader()
.**list_dialects**() - *return registered dialect names*
includes classes to read/write dictionary objects
Constants: .QUOTE_ALL, .QUOTE_MINIMAL, .QUOTE_NONNUMERIC, .QUOTE_NONE

datetime import datetime
Constants: MINYEAR, MAXYEAR
From datetime import timedelta: tools for duration and difference between dates and times. Supports functions for the following **types: .date, .time, .datetime, .timedelta., .tzinfo, .timezone**
Minimum constructor:datetime.datetime(year,month,day);
To create today's date object: datetime.date.**today**();
object attributes: .year, .month, .day (iyr=idte.year)
Get a time tuple: **datetime.datetime.timetuple**(arg)
ex: ↳ time.struct_time(tm_year=2020, tm_mon=7, tm_mday=26, tm_hour=11, tm_min=10, tm_sec=0, tm_wday=6, tm_yday=208, tm_isdst=-1)
ex: Using timedelta to find a future date:
*start = **datetime.date(2019, 1, 1)***
*duration = **datetime.timedelta**(days=180)*
enddate = start + duration
print(enddate) ↳ ***2019-06-30***
***with** idte=datetime.datetime.**today**(), instance attributes are: .year, .month, .day, .hour, .minute, .second, .microsecond, .tzinfo, .fold ex: idte.minute much more - *also in **PyPi** see new **python-dateutil** module*

decimal fast, correctly rounded fp math with a gazillion functions and pages of instruction

ensurepip - boostraps pip into an existing Python environment. pip is the installer for modules not in the Standard Library.
Windows **command line invocation**:
python –m ensurepip -- upgrade

enum - from enum import enum
mimicks enum in C, fast integer access and iter.

filecmp import filecmp
.**cmp**(f1, f2, shallow=True) Compare f1 and f2, returning True if they seem equal

fileinput import fileinput
for line in fileinput.input():
process(line)
.**input** (files=None, inplace= False, backup='', *, mode='r', openhook=None)
.filename() ↳ file being red
.fileno() ↳ file descriptor (-1 is none open)
.**lineno**() ↳ cumlatiave # of last line read
.**filelineno**() ↳ line # in current
.**isfirstline**() ↳ True if first line of its file
.**isstdin**() ↳ True if last line was read from sys.stdin
.**nextfile**() close file, read next line from next file
.**close**() close

BIG DADDY'S PYTHON 3 TOOLBOX For 3.5

www.wikipython.com *in this document the symbol ↳ means yields, results in, or produces

fractions.py import fractions

.**Fraction** (numerator=0, denominator=1)
.**Fraction**(other_fraction)
.**Fraction**(float)
.**Fraction**(decimal)
.**Fraction**(string)
a= '22' ; print(fractions. Fraction(a)) ↳ 11/50
print(fractions.Fraction(math.pi))
↳ 884279719003555/281474976710656

idlelib IDLE is Python's native IDE see:

https://docs.python.org/3.6/library/idle.html

io import io *three types: text, binary, raw*

for example: *continued next page*
f = open("myfile.txt", "r", encoding="utf-8")
f = open("myfile.jpg", "rb")
f = open("myfile.jpg", "rb", buffering=0)

keyword import keyword

.**iskeyword**(str) .**kwlist**

math - import math functions include

.ceil(x) *smallest int >= x*
.**comb**(n,k) *ways to choose k items from n*
.**copysign**(x,y) *absolute value of x, sign of y*
.**fabs**(x) ↳ *absolute value of x*
.**factorial**(x) ↳ *x factorial as integer*
.**floor**(x) ↳ *largest int <= x*
.**fmod**(x,y) *mathematically precise ver of x%y*
.**frexp**(x) ↳ *mantissa and exponent of x (m,e)*
.**fsum**(iterable) *returns fp sum of values*
.**gcd**(a,b) ↳ *greatest common divisor of a & b*
.**isclose**(a, b, *, rel_tol=1e-09, abs_tol=0.0) *True if a & b are close, otherwise False, relative or abs tolerance*
.**isfinite**(x) ↳*True if x not infinity or a NaN*
.**isinf**(x) *True if x is a positive or negative infinity*
math.isnan(x) ↳ *True if x is a NaN (not a number), False otherwise.*
.**isqrt**(n) (new 3.8) ↳*the integer square root of the nonnegative integer n. This is the floor of the exact square root of n, or equivalently the greatest integer such that a² ≤ n. To compute the ceiling of the exact square root of n, a positive number, use a = 1+ isqrt(n - 1).*
.**ldexp**(x, i) ↳ *x * (2**i); inverse of **frexp()***
.**modf**(x) ↳ fractional and integer parts of x
.**trunc**(x) ↳Real value of x truncated to integral
.**exp**(x) ↳ e**x.
.**expm1**(x) ↳ e**x - 1
.**log**(x[, base]) 1 argument, ↳ natural logarithm of x (to base e). 2 arguments, ↳ the logarithm of x to the given base, calculated as log(x)/log(base).
.**log1p**(x) ↳ the natural logarithm of 1+x (base e). *accurate for x near zero*
.**log2**(x) ↳ the base-2 logarithm of x
.**log10**(x) ↳ base 10 log of x
.**pow**(x,y) ↳ x raised to y
.**sqrt**(x) ↳ square root of x

Trigonometric Functions: ↳**radians** .**atan2**(y,x)
.**hypot**(x,y) ↳ sqrt(x*x + y*y) .**acos**(x) .**asin**(x)
.**atan**(x) .**cos**(x) .**sin**(x) .**tan**(x)
.**degrees**(x) anglex from radians to degrees
.**radians**(x) anglex from degrees to radians

Hyperbolic Functions: ↳**radians**
.**acosh**(x) ↳ the inverse hyperbolic cosine of x
.**asinh**(x) ↳ the inverse hyperbolic sine of x
.**atanh**(x) ↳ the inverse hyperbolic tangent of x
.**cosh**(x) ↳ the hyperbolic cosine of x
.**sinh**(x) ↳ the hyperbolic sine of x
.**tanh**(x) ↳ the hyperbolic tangent of x

Constants:
math.pi π = 3.141592…
math.e e = 2.718281…
math.tau τ = 6.283185…, New in version 3.6.
math.inf A floating-point positive infinity. (For negative infinity, use -math.inf.) New in 3.5.
math.nan A floating-point "not a number" (NaN)

numbers - operations from abstract base classes - four classes defined: Complex(components: real, imag), Real, Rational (adds numerator and denominator properties), Integral

os import os **hundreds of functions, many os specific; a few universal

.**environ**['HOME'] *home directory*,
.**chdir**(path) change working dir
.**getcwd**() current working dir
.**listdir**(path) .**mkdir**(path) .**mkdirs**(path) *make all intermediate directories* .**remove**(path)
.**strerror**() translate error code to message
.**curdir**() .**rename**(src, dst) .**rmdir**(path)
.**walk**(start directory, topdown=True) produces a generator of filenames in a directory tree
.**system**(command) Unix and Windows, execute the command in a subshell

os.path *Lib/posisxpath or Lib/ntpath (windows)* import os.path [as osp]

.**abspath**(*path*) normalized absolutized version of the pathname *path*.
.**basename**(*path*) base name of pathname *path*.
.**commonpath**(*paths*) longest common sub-path.**commonprefix**(*list*) ↳ the longest prefix
.**dirname**(*path*) ↳ directory name of *path*
.**expandvars**(*path*) ↳ *environment variables expanded*
.**exists**(*path*) ↳ True if *path* exists
.**getsize**(*path*) ↳ n the size, in bytes, of *path*.
.**isabs**(*path*) ↳True if *path* is absolute pathname
.**isfile**(*path*) ↳ True if *path* is existing file
.**isdir**(*path*) ↳ True if *path* is existing directory
.**islink**(*path*) ↳True if ref is an existing directory
.**join**(*path*, **paths*) Join one or more path components intelligently.
.**normcase**(*path*) Normalize case of a pathname
.**normpath**(*path*) On Windows, converts forward slashes to backward.
.**relpath**(*path*, *start=os.curdir*) ↳ relative filepath from the current directory or an optional start
.**samefile**(*path1*, *path2*) ↳ True if both pathname arguments refer to the same file or directory.
.**sameopenfile**(*fp1\fp2*) ↳ True if the same
.**samestat**(*stat1*, *stat2*) Return True if the stat tuples *stat1* and *stat2* refer to the same file.
.**split**(*path*) Split *path* into a pair, (head, tail)

pathlib (3.5) from pathlib import Path [as pt]

(For PurePath objects see online documentation.)
For MOST of the following concrete **methods** use:
.cwd() ex: my_current_dir = pt.cwd(); **.home**()
For a user defined file the following functions require that the file variable first be instantiated with the Path class; i.e., for "myfile" holding "C:\temp.txt": test_file = **pt**(myfile), then, **pt.exists** (test_file) is noted just as: **.exists (test_file)**
is_dir(test_dir) **.is_file**(test_file)
*** **.glob**—apparently .glob does NOT work outside the **sorted()** wrapper:
print(sorted(p.**glob**(test_file, '*.py')))
.iterdir() - creates an iterator; for directory x:
for dir in x.iterdir(): print(dir)
.mkdir (mode=0o777, parents=False, exist_ok=False) create new directory F
.open(mode='r', buffering=-1, encoding= None, errors=None, newline=None)
.read_text(); **.rename(target)**; **.rmdir()** - remove empty directory **(file_path).resolve (strict=False)** - make absolute path
.write_text(data, encoding=None, errors=None) - *open, write, close - all in one fell swoop*

pickle import pickle - non-human-readable object serialization (json is text only)

.**dump**(obj.file, protocol=None, *,fix_imports=True, buffer_callback=None)
.**dumps**(obj, protocol=None, *, fix_imports=True, buffer_callback=None)
.**load**(file, *, fix_imports=True, encoding="ASCII", errors="strict", buffers=None)
.**loads**(data, *, fix_imports=True, encoding="ASCII", errors="strict", buffers=None)

platform import platform

.**machine**() ↳ *machine type*
.**node**() ↳ *network name*
.**processor**() ↳ *real processor name*
.**python_version** ↳ *version as string*
.**system**() ↳ *'Linux', 'Darwin', 'Java', 'Windows'*

pprint import pprint

allows output of objects, including objects holding other objects in a reasonably readable format. Begin by creating an instance: (assume a list "mylist")
pp = **pprint.PrettyPrinter**(indent=3) *set indent*
then use your instance ("pp" above) to output:
pp.**pprint**(mylist)
some PrettyPrinter objects new/changed in 3.8
.pformat(obj), .pprint(obj), pp.isreadable(obj), more
ex: print(pp.**isreadable**(mylist)

py_compile.py import py_compile

.**compile**(file) - the compiled file is placed on file path in added directory "/__pycache__/ "

random import random

only for non-cryptographic applications
.**seed** initialize the random number generator
.**getstate**() ret object with internal generator state
.**setstate**() restores internal state to getstate value
.**getrandbits**(k) ret integer with k random bits
For integers:
.**randrange**(stop) .**randrange**(start, stop[, step])
.**randint**(a, b) fileinput.filename() a random integer N such that a <= N <= b. Alias for randrange(a, b+1).
For sequences:
.**choice**(sequence) ↳ random element
.**shuffle**(x [,random]) shuffle sequence in place
.**random**() ↳ the next random floating point number in the range (0.0, 1.0).
.**uniform**(a, b) ↳ fp between a and b

re import re complex search and match

re.**search**(pattern, string, flags=0)
re.**match**(pattern, string, flags=0)
re.**ignorecase**

shutil import shutil

.**copyfileobj**(fsrc, fdst[, length])
.**copyfile**(src, dst, *, follow_symlinks=True)
.**copymode**(src, dst, *, follow_symlinks=True)
Copy the permission bits from src to dst.
.**copystat**(src, dst, *, follow_symlinks=True)
Copy the permission bits, last access time, last modification time, and flags from src to dst
.**copy**(src, dst, *, follow_symlinks=True)
Copies the file src to the file or directory dst. src and dst should be strings.
.**copy2**(src, dst, *, follow_symlinks=True)
copy2() also attempts to preserve file metadata
.**copytree**(src, dst, symlinks=False, ignore=None, copy_function=copy2, ignore_dangling_symlinks= False, dirs_exist_ok=False)
.**disk_usage**(path) ↳ disk usage stats as tuple (total, used and free) in bytes—a file or a directory

Sound if your objective is to play a sound using a Python Standard Library module save your time - **none** of the modules listed under Multimedia Services do that. SEE: PyPi — playsound

sqlite3 import sqlite3

initialize using the Connection cursor() object:
conn = sqlite3.**connect**("'database_name'.db")
use special name **:memory** *to create in RAM*
cur = **conn.cursor** *#create cursor object,*
connection objects: cursor(), **commit**()., **rollback**(), **close**(), **execute**(), **executemany**(), **backup**()**executescript**(), **create_function**(), **iterdump**(), **create_aggregate**()
cursor objects: execute(), executemany(), executescript(), fetchone(), fetchmany(), fetchall() or close()
row objects: keys()

BIG DADDY'S PYTHON 6 TOOLBOX For 3.5

www.wikipython.com *in this document the symbol ↳ means yields, results in, or produces

statistics import statistics
.**mean**(data) average
.**harmonic_mean**(data) harmonic mean
.**median**(data) middle value
.**median_low**(data) low middle value
.**median_high**(data) high middle value
.**median_grouped**(data) 50th percentile
.**mode**(data) most common
.**pstdev**(data,mu=None) population std dev
.**pvariance**(data,mu=None) pop variance
.**stdev**(data, xbar=None) sample std dev
.**variance**(data, xbar=None) sample variance
more...extensive normal distribution functions

string
Constants
string.ascii_letters,
string.ascii_lowercase string.ascii_uppercase
string. digits string.hexdigits
string.octdigits string.punctuation
string.printable string.whitespace
string.**capwords**(str, sep=None)

sys import sys mostly advanced functions
.**exit**([arg]) - exit python
.**getwindowsversion**()
.**path** - search paths list
.**version** - Python version #

tarfile import tarfile extensive archive including gzip, bz2 and lzma compression
ex: (assumes import tarfile – to extract to cwd)
tar = tarfile.**open**("sample.tar.gz")
tar.**extractall**()
tar.**close**()

textwrap import textwrap
textwrap.**wrap**(text,width=x,**kwargs)**Lib/Lib/**

time import time or from time import
a new user must understand terminology found at:
https://docs.python.org/3.8/library/time.html
print(time.**time**()) #seconds since the epoch
↳ 1596486146.111275
mytime = time.**time**() #capture it
print(time.**localtime**(mytime)) #demo the tuple
↳ time.struct_time(tm_year=2020, tm_mon=8, tm_mday=3, tm_hour=16, tm_min=22, tm_sec=26, tm_wday=0, tm_yday=216, tm_isdst=1)
time_tuple=time.**localtime**(mytime) #capture it
print("The hour is: " + str(time_tuple[3])) #demo
↳The hour is: 16
print(time.**strftime**("%a, %d %b %Y %H:%M:%S +0000", time.**gmtime**()))
↳Mon, 03 Aug 2020 20:22:26 +0000
seconds=5 ; print("Wait 5 seconds!")
time.**sleep**(seconds) # delay of five seconds
print(time.**asctime**(time.localtime()))
↳ Mon Aug 3 16:22:31 2020
print(time.**ctime**(mytime))
↳ Mon Aug 3 16:22:26 2020

tkinter from tkinter import *
****there is a 10 page tkinter toolbox available to review at www.wikipython.com with a link to a free download on GitHub**

wave import wave
.**open**(file, mode=None) If file is a string, open the file by that name, otherwise treat it as a file-like object. mode can be: 'rb' (read), 'wb' (write) ex:
with wave.**open**("D:\\aloop.wav", "rb") as tstfile:
print(tstfile.**getnframes**()) # length in frames
once an object is returned by open(),
wave_read objects have these methods:
.**close**() i.e., object.close .**getnchannels**()
.**getsampwidth**() .**getframerate**() .**getnframes**(n)
.**readframes**(n) .**rewind**()
wave_write objects have these methods:
.**close**() Make sure nframes is correct
.**setnchannels**(n) .**setsampwidth**(n) -
.**setframerate**(n) .**setnframes**(n) Set frames to n.
.**setparams**(tuple) tuple should be (nchannels, sampwidth, framerate, nframes, comptype, compname), with values valid for the set*() methods. Sets all parameters.
.**tell**() Return current position in the file
.**writeframesraw**(data) Write audio frames, without correcting nframes.¶

A Few Interesting PyPi Modules

Anaconda, Conda, MiniConda - 3 related programs offering environment management at different levels. **Anaconda** manages all variations and compatibility issues unavoidable with many scientific, analytic, machine learning, statistical, web, visualization, distributed computing and data applications. Over 300 applications come "installed" in the base (root) environment, and over 8000 more are available to easily be added. **Its installation(s) can be huge.** It also qualifies as a language within itself. Plan on 90 days to get "sort of" oriented; a lifetime to explore a fraction of its options. Numerous IDEs are available in any Anaconda environment including Spyder, Visual Studio Code, IDLE, Jupyter Notebooks ... more. A few of the popular modules managed include Numpy, pip, Sqlite, wheel, zlib, cryptography, Astropy, cubes, matplotlib, pandas, SciPy, scikit-learn, scikit-image, ScientificPython + thousands more.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/aio/index.html

Miniconda is a lightweight version. **Conda** is similar to pip but is also an environment manager.

NumPy - powerful **N-dimension array** objects
NumPy says installation works best with a prebuilt package, see: https://scipy.org/install.html where they suggest a "scientific distribution" but do give **pip** directions: python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose obviously adding a whole bunch of other modules but violating the best advice of Jonathan Helmus at Anaconda: *"avoid all 'users' installs." so...* with conda: *from the Anaconda prompt type: conda install numpy* (did not test this)
with pip: *python –m pip install numpy* (worked ok)
import numpy as np
.**array**([elements list][element list][...])
.**zeros**(# of 0 elements)
.**ones**(# of 1 elements)
.**empty**(# of elements, values are random)
.**arange**(# of elements) np.arrange(5)↳([0,1,2,3,4])
.**linspace**(start, stop, # of elements) <-linearly spaced: .linspace(2,10,5) ↳([2,4,6,8,10])
dtype - default datatype is fp, but can specify with dtype: xarray = np.ones(3, **dtype**=np.int)
np.**sort**(array variable)
.**ndim** the # of axes/dimensions, of the array
.**size** the total number of elements of the array
.**shape** a tuple of integers indicating # of elements in each dimension
one zillion more functions

*For Raspberry Pi Aficionados

Rpi.GPIO – module to control Raspberry Pi GPIO channels; see GPIO toolbox and download link at: www.wikipython.com

Pillow - by Alex Clark, updated Aug 2020, a friendly version of Fredrik Lundh's **Python Imaging Library**
Pillow version 7.2 works in Python 3.5 to 3.8
install: **python3 -m pip install --upgrade Pillow**
from PIL import Image
im = Image.**open**(testfilepath)
print(im.**format**, im.**size**, im.**mode**)
im.**show**()

playsound is a cross platform program pulled from **Pypi** that is very easy to use. From windows: python -m pip install playsound for example:
from playsound import playsound
testwave = "C:\\Windows\\Media\\Alarm09.wav"
playsound(testwave)

Poetry is a smaller more efficient way to manage dependencies for 3.4+ but it's a little complicated. Start with: https://pypi.org/project/poetry/

pandas for tabular data — "aims to be the fundamental" module for "real world data analysis" - it is part of the Anaconda distribution (also installs with Miniconda) but can be installed with pip:
pip install pandas then **import pandas as pd**
tables are **DataFrame**(s) and columns are **Series**
see the docs @: https://pandas.pydata.org/

fuzzywuzzy imprecise string comparison - requires **python-Levenshtein** dependency
pip install fuzzywuzzy
pip install python-Levenshtein
functions return a matching ratio %
.**ratio**(string1, string2)
.**partial_ratio**(string1, string2)
.**token_sort_ratio**(string1, string2)
.**token_set_ratio**(string1, string2)
use process module for compare to list of choices
process.extract(query, choices) ↳ score sorted list
process.extractOne(query, choices) ↳ top choice

yahoo-finance 1.4.0, yfinance, lxml
we are talking about incredible stock data—prices, historical prices, splits, etc. **yahoo-finance** was deprecated when yahoo and google got hand slapped for making this available but the data has a back door module, **yfinance** has conflicting claims for dependencies but requires **lxml** which is a library for processing XML and HTML using C libraries from Python. It has literally millions of downloads. Once you install **lxml** you just:
pip install yfinance (no need to install yahoo-finance) import yfinance as yf
*NOTE—a few functions no longer work
create stock instance with .**Ticker**(stock symbol)
ex: jnj = yf.Ticker("JNJ") *i.e.[Johnson & Johnson]*
.**history**(period ="short cut symbol") valid periods 1d,5d,1mo,3mo,6mo,1y,2y,5y,10y,ytd,max **or**
.**history**(start="yyyy-mm-dd", end="yyyy-mm-dd")
.**actions** - dividends, splits
.**dividends** or .**splits** - show dividends or splits
.**financials** or .**quarterly_financials**
.**major_holders** .**institutional_holders**
.**calendar** .**recommendations**

plotly.express and Kaleido - **plotly.express** is built-in to the **plotly** library and is considered a "starting point" but may be all you ever need. *Plotly is an MIT Licensed module.* plotly.express requires a determined effort to learn because it creates more than 35 types of graph images—which is why there is not group of highlighted commands here. It does **not** export your graph as a static image—which is why you need **Kaleido** also. plotly has lots of dependencies, kaleido has none. Both import easily: pip install plotly==4.9.0 (as of Aug 2020) or just pip install plotly_express==0.4.0 (Bid Daddy did not test this) and pip install kaleido.

What is not mentioned in this General Toolbox?
We estimate 99.83% of Python capability available has no mention in this toolbox, so forge ahead, and happy coding!

Can important key methods of your favorite module be briefly summarized? We would **really** like to hear your suggestion(s)! email:

oakey.john@yahoo.com

www.wikipython.com

No registration, cookies, fees, ads, hidden agendas, no financial contributions accepted, nobody looking for a job - secure site & downloads.