

print() is a function
print(objects, separator="", end='\n')
 print("Hello World!") \hookrightarrow 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) \hookrightarrow absolute value of x
bin(x) \hookrightarrow 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,
 \hookrightarrow a pair of numbers - quotient and
 remainder using integer division
float(x) \hookrightarrow a floating point number
 from an integer or string; x="1.1"
 print(float(x)*2) \hookrightarrow 2.2
hex(x) \hookrightarrow int to hex string
 hex(65536) \hookrightarrow 0x10000 or
 hex(65536)[2:] \hookrightarrow '10000'
oct(x) \hookrightarrow int to octal
int(x) \hookrightarrow int from float, string, hex
pow(x,y [,z]) \hookrightarrow 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]) \hookrightarrow
 floating point number rounded to digits;
 Without digits it returns the nearest
 integer Round(3.14159, 4) = 3.1416
max, min, sort - see data containers
None \rightarrow constant for null; x=None

Operators

Math: =(execute/assign, = can value
 swap; a, b = b, a); +; -; *; /; **
 (exp); +=; -=; *=; **=; /=; //=
 ("floor" div truncated no remainder;
 % (modulo): \hookrightarrow 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 \hookrightarrow 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) \hookrightarrow '0b1010'
Sequence Variable Operators
 (for strings) + \hookrightarrow concatenate, * \hookrightarrow
 repetition; s[i] single slice; s[i:j:k]
range slice from, to, step \rightarrow 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

The ternary if Statement

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

More Python toolboxes available on
www.wikipython.com

String Tools

Functions
ascii(str) \hookrightarrow like repr, escapes non-ascii
chr(i) \hookrightarrow character of Unicode [chr(97) = 'a']
input(prompt) \hookrightarrow user input as a string
len() \hookrightarrow length of str, or count of items in
 an iterable (list, dictionary, tuple or set)
ord(str) \hookrightarrow value of Unicode character
repr(object) \hookrightarrow printable string
str(object) \hookrightarrow string value of object
slice selection str[:stop]; str[start:stop[:step]]
 \hookrightarrow a string object created by the selection
Methods **Attribute Information:**
 .isprintable(), .isidentifier(), .isnumeric(),
 .isalpha(), .isdigit(), .islower(), .isdecimal(),
 .istitle(), .isspace(), .isalnum(), .isupper()
 may be null, \hookrightarrow True if all characters in a
 string meet the attribute condition and the
 string is at least one character in length
.casefold() \hookrightarrow casefold - caseless matching
.count(sub[,start[,end]]) \hookrightarrow # 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 w/ ValueError
"sep".join([string list]) joins strings in
 iterable with sep char; can be null - "" in quotes
.replace(old, new[, count]) \hookrightarrow copy of the
 string with substring old replaced by new; if
 count is given, only first count # are replaced
.rfind(sub[, start[, end]]) \hookrightarrow the lowest
 index in the string where substring sub is
 found, contained within slice [start:end].
 \hookrightarrow -1 on failure
.rindex() like rfind but fail \hookrightarrow ValueError
.partition(sep) \hookrightarrow 3 tuple: before, sep, after
.split() \hookrightarrow word list with intervening spaces
.splitlines(keepends=False) \hookrightarrow list of
 lines broken at line boundaries
.startswith(prefix[,start[,end]]) \hookrightarrow True/False
.find(sub[, start[, end]]) \hookrightarrow the index of
 substring start, or -1 if it is not found;
 print('Python'.find("th")) \hookrightarrow 2
.translate(table) map to translation table
String Format Methods
.center(width[, fillchar]) string centered in
 width area using fill character 'fillchar'
.capitalize() \hookrightarrow First character capitalized
***.format()** - see Format Toolbox!
method: (1) substitution (2) pure format
 (1) 'string {sub0}{sub1}'.format(0, 1)
 a = "Give {0} a {1}".format('me','kiss')
 (2) '{:format_spec}'.format(value)
function: format(value, format_spec)
 format_spec: [[fill] align] [sign] [# - alt form]
 [0 - forced pad] [width] [,] [.precision] [type]
 x = format(12345.6789, "=+12,.2f") \hookrightarrow + 12,345.68
f-string: print(f"{'Charge \$'}{9876.543:,.2f}")
 \hookrightarrow Charge \$ 9,876.54 NEW in version 3.6
.ljust(width [, fillchar]) or **.rjust(same args)**
.lower() \hookrightarrow text converted to lowercase
.strip([chars]), lstrip(), rstrip() \hookrightarrow 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() \hookrightarrow upper \rightarrow lower & vice versa
.title() \hookrightarrow titlecased version - words cap'ed
.upper() \hookrightarrow text converted to uppercase
.zfill(width) - left fill with '0' to len width
.zip(iterables) - merges to list of tuples

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 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 funcName(args):
return(variable object) - return
 the value a function derives - or
yield(gen); yield returns a **gen-**
erator whose sequential results are
 triggered by **next**
global x creates global variable -
 defined inside a function
nonlocal a variable in a nested
 function is good in outer function
lambda unnamed inline function
 lambda [parameters]: expression
 z= lambda x:(x**2); print(z(5)) \hookrightarrow 25

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 object 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.

Miscellaneous

pass (placeholder - no action)
del deletes variables, data containers,
 items in iterables: del mylist[x]
ITERABLE: a data container with changeable items
breakpoint enters debugger if called
with wrapper ensures **_exit_** method
eval(expression) \hookrightarrow value after eval
bool(expression) \hookrightarrow T/F (F is default)
callable(object) \hookrightarrow True if callable
help(object) invokes built-in help
 system, (for interactive use)
id(object) \hookrightarrow unique object identifier
 Note: about a dozen functions not shown here

Selected Escape Characters

Nonprintable characters represented
 with backslash notation; ('r' (raw)
 ignores esc chars before a string literal)
 \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, many more ...

Data Containers Methods / Operations

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

L / T / D / S / F instances of:

list, tuple, dictionary, set, frozen set

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)**; get values **x,x,...=T**
[i:j]; reverse order **T[::-1]**; sorted (**T**,
reverse=True/False); clear values **T=()**

DICTIONARIES: create: **D={k:v, k:v,...}**,
=dict.fromkeys(L/F [,1 value]), **=dict**
(L) requires list of 2 tuples, =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()**

SETS: (no duplicates) create: **S=set(L/T/**
F), **S={x,x,x}**, **S='string'** unique letters;
Test and return T/F (sets & frozensets):
S.isdisjoint(S2) common items?
S.issubset(S2) or **S <=** contained by
S < S1 set is a proper subset
S.issuperset(S2) or **S >= S2** contains
S > S1 set is a proper superset
Change set data (sets & frozensets):
S.union(S2) or **S=S1|S2[...]** merge
S.intersection(S2) or **S & S1** intersection
of S & S1 ex: **S3 = S1.intersection(S2)**
S.difference(S2) or **S-S2** unique in S
S.symmetric_difference(S2) or **S^S2**
elements in either but not both
Change set data only (not frozensets)
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.add(element); **S.remove(element)**
KeyError if missing
S.discard(element)

FROZENSETS: immutable after creation;
create: **S=frozenset([iterable])** only
See Test and return methods listed above and
change of data methods as listed above.

Comments, corrections and suggestions appreciated:
oakey.john@yahoo.com www.wikipython.com

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, default])

min(iterable [,key, default])

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 with-
out keys are sorted alphabetically, numbers high
to low; **key ex:** **print(sorted(list, key=len))** sorts by
length of each str value; more examples: **key=**
alist.lower, or **key = lambda tupsort: tupitem[1]**
type(iterable) ↳ a datatype of any object
zip() creates aggregating iterator from multiple
iterables, ↳ iterator of tuples of ⁱth 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
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 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**

Line 1: (required in red, optional in green) inheritance creates a "derived class"

command key word colon
class myClassName (inheritance):
your class name-class definition header
Class creates a brand new namespace
and supports **two operations**: attribute
reference and instantiation

Next Lines: (statements) **usually (1)**
a docstring, like **"""Docstring example"""** **(2)**
instantiation, using a **special method:**
__init__(self, arguments) which is
autoinvoked when a class is created;
arguments are passed when a class
instantiation is called:

def __init__(self, passed arguments):
variable name assignments, etc.
(3) function definitions, local
variable assignments

class mammalia(object):
def __init__(self, order, example):
self.ord = order
self.ex = example
self.cls = "mammal"
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** ***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'

Creating a Function:

(required in red, optional in green)

Line 1:

command key word arguments
Def name (input or defined params):
your new function's name colon
↳ All subsequent lines must be indented

Line 2: a **docstring** (optional)

Line 2 or 3 to ?: code block

Usual line last: return(expression
to pass back) ↳ keyword to pass result
BUT... a generator can be passed
using **yield**: for example:

aword = "reviled"
def makegen(word):
marker = len(word)
for letter in word:
yield (word[marker-1: marker])
marker=marker-1
for letter in makegen(aword):
print(letter)
↳ d
e
v
i
l
e
d

re: format: (1) the old string % syntax will
eventually be **deprecated**: **print("%s.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**

Python Documentation: Tables & Lists

Functions * **boldface** not covered in this toolbox

abs()
all()
any()
ascii()
bin()
bool()
breakpoint()
bytearray()
bytes()

callable()
chr()
classmethod()
compile()
complex()
delattr()
dict()
dir()
divmod()

enumerate()
eval()
exec()
filter()
float()
format()
frozenset()
getattr()
globals()

hasattr()
hash()
help()
hex()
id()
input()
int()
isinstance()
issubclass()
iter()
len()

list()
locals()
map()
max()
memoryview()
min()
next()
object()
oct()
open()
ord()

pow()
print()
property()
range()
repr()
reversed()
round()
set()
setattr()
slice()
sorted()

staticmethod()
str()
sum()
super()
tuple()
type()
vars()
zip()
__import__()

Comparisons

Operation	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

Sequence Operations (4.6.1)

x in s
True if an item of s is equal to x, else False

x not in s
False if an item of s is equal to x, else True

s + t the concatenation of s and t

s * n or n * s
equivalent to adding s to itself 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[:]**)

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 contents repeated 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>

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

Bitwise Operations on Integers

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

comments and suggestions appreciated:
john@johnoakey.com

Numeric Type 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 with real part re, imaginary part im. defaults to zero.
c.conjugate()	conjugate of the 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

notes: <https://docs.python.org/3.6/library/stdtypes.html>

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 string).
'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 is converted, results in a '%' character in the result.

Keywords

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

(keywords = reserved words)

Operator Precedence

Lambda (Multiplication, matrix multiplication, division, floor division, remainder) **+x, -x, ~x**

if – else (Positive, negative, bitwise NOT) ****** (exponentiation)

or/and/not x **await x** (Await expression)

in, not in, is, is not, **x[index], x[index:index], x** (arguments..., x.attribute)

<, <=, >, >=, !=, == (subscription, slicing, call, attribute reference)

|/^/&

<<, >>

+, -

***, @, /, //, %**

Open File Modes

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, fails if it already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newlines mode (deprecated)

Built-in Constants

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

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 + number (or empty string)

'+' A sign character ('+' or '-') will precede the conversion (overrides a "space" flag).

Built-in Types

numerics, sequences, mappings, classes, instances, exceptions

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 (1,3)

\xhh Character with hex value hh (2,3)

www.wikipython.com

The real power of Python is its transformer-like ability to add functions and abilities to fit just about any conceived programming need. This is done through the importation of specialized **MODULES** that integrate with, and extend, Python; adding abilities that become part of the program. About 230 of these modules are downloaded automatically when Python is installed. If you can't find what you need in this "Standard Library", there are over another 1,000,000 packages contributed by users in the PyPi online storage waiting for your consideration. A few highlights of the modules in the "The Python Standard Library" and a couple of others in PyPi are noted below. Find PyPi at: <https://pypi.org/>

The Python Standard Library

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
collections — Container datatypes
array — Efficient arrays of numeric values

Numeric and Mathematical Modules - 7 modules including:

numbers — Numeric abstract base classes
math — Mathematical functions
decimal — Decimal fixed point and floating-point arithmetic

random — Generate pseudo-random numbers
statistics — Mathematical statistics functions

Functional Programming Modules - 3 modules:

File and Directory Access - 11 modules including:
pathlib — Object-oriented filesystem paths
os.path — Common pathname manipulations
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 including:

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 including:

os — Miscellaneous operating system interfaces
time — Time access and conversions
curses — Terminal handling for character-cell displays

Concurrent Execution - 10 modules including:

threading — Thread-based parallelism
multiprocessing — Process-based parallelism

Interprocess Communication and Networking - 9 modules:

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 Python modules

Python Runtime Services - 14 modules including:

sys — System-specific parameters and functions
sysconfig — Provide access to Python's configuration 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:

Miscellaneous Services - 1 module:

MS Windows Specific Services - 4 modules including:

winsound — Sound-playing interface for Windows

Unix Specific Services - 13 modules:

Superseded Modules - 2 modules:

Undocumented Modules - 1 module:

Cherrypicked Useful Standard Library Module Methods

calendar: many many functions; ex:
weekdays = ['M', 'Tu', 'W', 'Th', 'F', 'S', 'S']
print('birth day is a: ' + weekdays[calendar.weekday(1948, 1, 19)])
↳ birth day is a: M

copy: .copy(x), .deepcopy(x)

datetime: .date(year, month, day),
.date.today(), .datetime.now(),
.timedelta(days or seconds), ex:
start = datetime.date(2019, 1, 1)
duration = datetime.timedelta(days=180)
enddate = start + duration
print(enddate) ↳ 2019-06-30 *also in
PyPi see new python-dateutil module

decimal: accounting level precision,
from decimal import *
.Decimal(value="0", context=None) ex:
from decimal import *
import math
print(math.sqrt(2), '\n', Decimal(2).sqrt()) ↳
1.4142135623730951
1.414213562373095048801688724

math: .ceil(x), .fsum(iterable), .sqrt(x),
.log(x[,base]), .factorial(x), .floor(), .log(x[,base]), log1p(x), .sqrt(x), all trig
and hyperbolic functions constants: .pi, .e

pathlib: new in 3.5, Unless you understand the "PurePath" class, you want to use "concrete paths" and should import using "from pathlib import Path"; this is the assumption in the following where p = Path:
p.cwd() current directory; p.home();
p.exists(str); p.is_dir(); p.is_file();
p.iterdir() ↳ iterates directory paths

for file in p.iterdir(p.cwd()):
print(file) ↳ all files in working dir
p.mkdir(mode=0o777, parents=False, exist_ok=False) create new directory
FileExistsError if it already exists
p.open(mode='r', buffering=-1, encoding=None, errors=None, newline=None)
p.read_text(); p.rename(target);
p().resolve(strict=False) - make absolute path; p.glob(pattern) - creates
iterator for files filtered by pattern, "*" ↳ all dir and subdirs, "**.*" ↳ all files in path
dir, "**/*/*" ↳ all dir and their files
p.rglob(pattern) - like ** in front of .glob; p.rmdir() - remove empty directory; p.write_text(data,
encoding=None, errors=None) - open,
write, close - all in one fell swoop

os: os.environ['HOME'] home directory,
.chdir(path) change working dir, .getcwd()
current working dir, .listdir(path),
.mkdir(path), .remove(), .curdir,
note: os.path is a different module

random: .seed([x]), .choice(seq),

.randint(a, b), .random() - floating point
[0.0 to 1.0], reuse seed to reproduce value

sys: .exit([arg]), .argv, .exe_info(),
.getsizeof(object [,default]), .path,
.version, __stdin__, __stdout__

string: constants: ascii_letters,
ascii_lowercase, ascii_uppercase, digits,
hexdigits, octdigits, punctuation,
printable, whitespace

statistics: .mean(), .median(), .mode(),
.pstdev(), .pvariance(), p is for
population

time: sleep(secs), localtime(), clock(),
asctime(struct_time tuple)

wave: .open(file, mode='rb' or 'wb')
read or write, read_object.close(),
write_object.close()

**pickle tarfile shelve sqlite json
filecmp fileinput zipfile filecmp**

see **Data on Disk Toolbox**

Complex modules where single method examples are not useful:

tkinter: best gui but equivalent to learning Python twice - see 10 page
tkinter toolbox on www.wikipython.com

re: exigent find & match functions

collections: use mostly for named
tuples and ordered dictionaries

array: very fast, efficient, single type

turtle: intro graphics based on tkinter

Raspberry Pi Aficionados

Rpi.GPIO - module to control
Raspberry Pi GPIO channels - see GPIO
toolbox on www.wikipython.com,
download module from: <https://pypi.org/search/?q=rpi.gpio>

Selected Other PYPI Frequently Downloaded Packages

pip, pillow, numpy, python-dateutil,
doctils, pyasn1, setuptools (also see
pbr), jmespath 0.9.3, cryptography,
ipaddress, pytest, decorator pyarsing,
psutil, flask, scipy, scikit-learn (requires
3.5, Numpy and SciPy), pandas, django,
cython, imagesize, pyserial, fuzzywuzzy,
multidict, yarl

**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, contributions, ads, &
nobody looking for a job - secure site & downloads."