

**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'  
 (1 4, 0 2's, 1 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(x)[2:] = '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(2,7)=128, pow(2,7,3)=2  
**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):**  $\hookrightarrow$  a remainder  
**Boolean:** True, False (1 or 0)  
**Logical:** and, or, not modifies cmp  
**Comparison:** == (same as); != (is  
 not equal); <; <=; >; >=; is; is  
 not; all  $\hookrightarrow$  a Boolean value  
**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 at j-1, increment by count

## Decision Making

**if elif else:**  
 if somenum == 1:  
 print('One')  
 elif somenum == 2:  
 print('Two')  
 else:  
 print('Whatever!')

### 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](http://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  
 a 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** [[start[:]] [:]stop [:step]]  
 $\hookrightarrow$  a string object created by the selection  
**format() function and method - see below**  
**Methods** **Attribute Information:**  
**isalnum(), isalpha(), isdecimal(), isdigit**  
**( ), isidentifier(), islower(), isnumeric(),**  
**isprintable(), isspace(), istitle(),**  
**isupper()** may be null,  $\hookrightarrow$  True if all charact  
 -ers in a string meet the attribute condition  
 and variable is at least one char in length  
**.capitalize()**  $\hookrightarrow$  first character capitalized  
**.casefold()**  $\hookrightarrow$  casefold - caseless matching  
**.center(width[, fillchar])** string centered in  
 width area using fill character 'fillchar'  
**.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  
**.join([string list])** concatenates strings in  
 iterable - see .reverse example on page 2  
**.ljust(width [, fillchar])** or .rjust(same args)  
**.lower()**  $\hookrightarrow$  text converted to lowercase  
**.maketrans** see https://docs.python.org/3.6/library  
**.partition(sep)** split str at sep  $\hookrightarrow$  3-tuple of parts  
**.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 highest  
 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 first char  
 BEFORE sub is found, or -1 if not found  
 print('Python'.find("th"))  $\hookrightarrow$  2  
**.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'd  
**.translate(table)** map to translation table  
**.upper()**  $\hookrightarrow$  text converted to uppercase  
**.zfill(width)** - left fill with '0' to len width  
**\*.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]  
 val = 12345.67891; prt\_str=format(val, " =+12,.2f")  
 print(prt\_str)  $\hookrightarrow$  + 12,345.68

## Looping

**while** (some statement is True):  
 process data statements; **else:**  
 for expression to be satisfied:  
**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:** do this if you get the 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  
**yield(gen), next yield** returns a  
**generator** whose sequential  
 results are triggered by **next**  
**global x** creates global variable -  
 defined inside a function  
**nonlocal** a variable inside a nest-  
 ed function is good in outer function  
**lambda** unnamed inline function  
 z= lambda x:(x\*\*2) if x<4 else (x\*\*3)  
 print(z(3), z(5))  $\hookrightarrow$  9, 125

## 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  
**with** wrapper ensures **\_exit\_** method  
**eval(expresion)**  $\hookrightarrow$  value after eval  
**bool(x)**  $\hookrightarrow$  True/False, (False 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 2 dozen not shown**

### 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, **more**

## Data Containers Methods / Operations

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

**LISTS:** `.append(x)`; `.copy()`; create `L=[x,x,...]`; `L=[]`; `L=list(tuple)`; `.clear()`; `.count(x)`; `del L`; `.extend(x,x,...)`; determine membership if `x in L`; insert `(i,x)`; `len(L)`; `.pop()`; `.pop(i)`; `.remove(x)`; replace item `L[i]=x`; replace multiples `L[i:j]=[x,x,...]`; retrieve index, 1st value of `x` `indexno=L.index(x)`, at/after index `i` [, before index `j` ]; `L.reverse()`; `L.sort(key=None, reverse=False)`; create generator `V=iter(L)`, trigger iteration `next(V, default)`

### 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)
newList= [item*2 for item in atuple if item>0]
print(atuple, newList)  # (1, -2, 3, -4, 5) [2, 6, 10]
if modifying items only: uplist=[x+1 for x in L]
```

**TUPLES:** Add items `+=`; Add single item `+=(x,)`; `.count(x)`; create `T=(x,[x],(x),...)` can include lists, other tuples; create tuple from a list `T=tuple(L)`; `del T`; clear values `T=()`; index `i=T.index(x)`, at or after index `i` [, before index `j` ]; iteration generator `v=iter(T)`, next iteration `next(v)`; `len(T)`; retrieve values `x,x,...=T[i:j]`; slice `T[i:j]` start 0, end `j-1`; reverse order `T[::-1]`; sorted (`T`, `reverse=True/False`); join tuples `T1=T1+T2`

**DICTIONARIES:** create `D={k:v, k:v,...}`; `=dict.fromkeys(keys/list [,1 value])`; add `D2 to D` `D.update(D2)`; `D.copy()`; `D.clear()`; delete key/value `del D[k]`; `del D`; `D.get(k[,x])` like `D[k]` but `D.get(k,x)` if no `k`; iteration var `v=iter(D)`, trigger iterations `next(v)`; member `x in / not in D`; `D.pop(k[,default])`; `D.popitem()`; return views: `D.items()`, `D.keys()`, `D.values()`; returns `v` mapped to `k` `D[k]`; `len(D)`; change value `D[k]=new value`; `D.setdefault(k[,default])` if `k` is in the dictionary, return the key value, if not, insert it with default value and return the default

**SETS:** no duplicates create `S=set()`, `S={x,x,x}`, `S=set(L)` from list, `S='string'` unique letters; `.add(x)`; `.clear()`; `.copy()`; `del S`; `.difference(S2)`; `.discard(x)`; `.intersection set('abc').intersection('cbs')`; `.isdisjoint(S2)` True if no common items; contained by `.issubset(S2)` or `S<=S2` `y`; contains `.issuperset(S2)` or `S>=S2`, `S>S2`; `len(S)`; `.pop()`; `.remove()` `KeyError` if not present; iteration variable `v=iter(S)`; trigger iteration `next(v)`; member `S in/not in`; `S.union(other sets)`; `S.update(other sets)`

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

comments and suggestions appreciated:  
john@johnoakey.com

## Data Container Functions

**all(iterable)** True if all elements are True  
**any(iterable)** True if any element is True  
**\*all** and **any** are both FALSE if empty  
**enumerate(iterable, start = 0)** list

```
alist = ['x','y','z']
print(list(enumerate(alist)))
# [(0,'x'), (1,'y'), (2,'z')]
type([iterable])
# a datatype of any object
max(iterable [,key, default])
min(iterable [,key, default])
sum(iterable [, start]) must be all numeric,
if a=[8,7,9] then sum(a) returns 24
sorted(iterable [,key=],[,reversed])
```

`reversed` is Boolean with default False; strings without `key` sorted alphabetically, numbers high to low; `key` ex: `print(sorted(strs, key=len))` sorts by length of each str value; ex: `key= str.lower`, or `key= lambda tupsort: tupitem[1]`  
**reversed()** reverses access order—list or tuple

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

`[.reverse()]` inverts list order; `mylist.reverse()`  
**range([start,] stop [,step])**

```
alist=["Amy","Bo","Cy"]
for i in range(0,len(alist)):
    print(str(i), alist[i]) # note slice
iter and next(iterator [,default]) create
iterator with iter; fetch items with next; default
returned if iterator exhausted, or StopIteration
alist=["Amy","Bo","Cy"]; itemnum = iter(alist)
print(next(itemnum, "listend"))
print(next(itemnum, "listend"))
print(next(itemnum, "listend"))
print(next(itemnum, "listend"))
```

**map(function,iterable)** can take multiple iterables but function must take just as many  
`alist=[5,9,13,24]`  
`x = lambda z: (z**2 if z**2 < 150 else 0)`  
`itermap = map(x, alist)`

**filter(function, iterable)** iterator for element of iterable for which function is True  
`getattr(object, 'name' [, default])`  
`setattr(object, 'name', value)`  
**zip()** creates aggregating iterator from multiple iterables, iterator of tuples of `i`th iterable elements from each sequence or iterable

**CLASS:** "Your very own complex data object blueprint."

**Line 1:** (required in red, optional in green)  
command key word inheritance - creates a "derived class"  
**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.order = order
        self.ex = example
        self.cls = "mammal"
    def printInfo(self):
        info="class/order: " + self.cls + "/" + self.order + "\n", Example: " + self.ex
        print(info)
mam_instance = mammalia("cetacea", "whales") #create class obj
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'}`

## 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)
```

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

arg#1: B  
arg#2 is C  
arg#3 is T  
arg#4 is A  
formal arg: 1  
(dog, cat)  
(arg2, 'two')

## Useful Modules/Toolboxes

See Python Standard Library Module and [www.wikipython.com](http://www.wikipython.com) vetted module examples

<https://docs.python.org/3.5/library>

**math:** like Excel math functions `ceil(x)`, `fsum(iterable)`, `sqrt(x)`, `log(x[,base])`, `pi`, `e`, `factorial(x)`  
**random:** `seed([x])`, choice

(seq), `randint(a, b)`, `random()` - floating point [0.0 to 1.0] **sys** exit ([]), path, platform **datetime** `date.today()`, `datetime.now()`, **time** `localtime()`, `clock()`, `asctime (struct_time tuple)`, `sleep(secs)`  
**os** deep operating system access  
**tkinter** see toolbox on wikipython; note: **tkinter NOT Tkinter**  
**RPi.GPIO** - control Raspberry Pi pins via Python and new in 3.6:  
**filelib** - does it all for files

OTHER TOOLBOXES AVAILABLE from [WWW.WIKIPYTHON.COM](http://WWW.WIKIPYTHON.COM) ...and don't miss **tksidekick** - a companion program for tkinter! All downloads are from GitHub  
No registration, no cookies, no charges, no contributions, enjoy!

2 Notes on format: (1) the old string % syntax will eventually be deprecated: `print("$%.2f buys %d %ss"%(1.2, 2, "hot dog"))` try it (2) new f string options available in version 3.6 see [www.wikipython.com](http://www.wikipython.com) : format toolbox