

## Reserve Words

### Comparsion / Conjunction

**True, ==** (is same as), **False, none** (i.e., null), **and, not, or, in** list/tuple/string/dictionary/set **is or is not ==** compare  $\hookrightarrow$  'True' or 'False'

### Definition

**class** create class: class ClassName: see below

**def** create function: def FName(args):

**del** variables, data containers, items in iterables: del mylist[x]

**ITERABLE:** a data container with changeable items

### Module Management

**import** connects module, ex: import math

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

### Miscellaneous

**pass** (placeholder – no action)

**with** wrapper ensures **\_exit\_** method

### Functions

**def, return(obj), yield, next** def creates; inside functions **yield** is like **return** but returns a generator whose sequential results are triggered by **next**; **global x** declares global var in function **non local** a variable inside a nested function is good in the outer function

**lambda** unnamed

inline function with no return statement

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

### Error Management

**raise** forces a ZeroDivisionError

**try except else finally assert**

used in error handling blocks

**try:** code with error potential

**except:** do this if you get the error

**else:** otherwise do this code

**finally:** do this either way

**assert:** condition=False raises **AssertionError**

### Looping

**while** (some statement is true)

**for** alist=["Be", "my", "love"]

for wordnum in range(0, len(alist)):

print(wordnum, alist[wordnum]) #slice

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

See data container functions

**break** ends the smallest loop it is in;

**continue** ends current loop iteration

### Decision Making

**if elif else**

def if\_example(anint):

if anint == 1:

print('One')

elif anint == 2:

print('Two')

else:

print('Some other')

### The Ternary if Statement

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

### Multi-line Statements \

Not needed within [], {}, or ()

**Multiple Statements on a Line ;** not with statements starting blocks like **if**

**CLASS:** Your own data container. **DEFINE DESIGN:**

class Name (inheritance object)

def \_\_init\_\_(self, mandatory variables,...)

accessname = mandatory variable ...repeat as necessary

Other functions: "getattr(accessname, self)", or "return self.variable"

**CREATE INSTANCE:**

MyInstanceName = ClassName(self)

**ACCESS INSTANCE DATA:**

Print(MyInstanceName.accessname in get function)

## Major Built-In Functions

**String Handling** ( $\hookrightarrow$ =converts/returns)

**str(object)**  $\hookrightarrow$  string value of object

**repr(object)**  $\hookrightarrow$  printable representation string

**ascii(str)**  $\hookrightarrow$  like repr but escape non-ascii

**eval(expression)**  $\hookrightarrow$  value after evaluation

**chr(i)**  $\hookrightarrow$  character of Unicode [ chr(97) = 'a']

**ord(str)**  $\hookrightarrow$  value of Unicode character

**input(prompt)**  $\hookrightarrow$  user input as a string

**len(—)**  $\hookrightarrow$  length of str, items in list/dict/tuple

**slice selection** **[[start[:]] [[:stop] [:step]]**

$\hookrightarrow$  a new string object created by the selection

**str.join('string seperator', [string list])**

**format(value [,format\_spec])**  $\hookrightarrow$  value

in a formatted string—**extensive and complex**

- 2 syntactical structures (1) simple format only:

**format(number/string, 'format string')**

(2) format and/or substitution: **'{:order or format string}'.format(objects);**

format string attributes/required order:

**[[fill] align] [sign] [#-alt form] [0 forc-**

**ed pad] [width] [,] [.precision] [type]**

Key **types:** 'f'/'F'~ fixed point, default 6; 'g'/'G'~

general; 'e'/'E'~ exponential; %~percent; 'c'~

Unicode char; ex: **format(number, '0+=+20,.3f')**

$\hookrightarrow$  +000,000,012,345.679

Substitution using format():

**"{variable to output} | {numeric format}..." .format**

**('string' or numeric values...)**

'{0[x]}' selects the xth value in a tuple which format

names: ex: print ('{0[x]}' .format(mytuple))

Can format dates with help of datetime module. SEE

WWW.WIKIPYTHON.COM : OUTPUT TOOLBOX

### Number Handling

**abs(x)**  $\hookrightarrow$  absolute value of x

**bin(x)**  $\hookrightarrow$  integer to binary bin(5)= '0b101'

(one 4, no 2's, one 1) bin(7)[2:] = '111'

**divmod(x,y)** takes two (non complex)

numbers as arguments,  $\hookrightarrow$  a pair of numbers -

quotient and remainder using integer division.

**float(x)**  $\hookrightarrow$  a floating point number from an

integer or string A='1.1'; print(float(A)\*2)  $\hookrightarrow$  2.2

**hex(x)**  $\hookrightarrow$  integer to hex string hex(65536)  $\hookrightarrow$  0x10000

or hex(x)[2:]='10000' also **oct(x)**  $\hookrightarrow$  int to octal

**int(x)**  $\hookrightarrow$  integer from a decimal, 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 re-

turns the nearest integer. Round(3.14159,4)=3.1416

### Miscellaneous Functions

**bool(x)**  $\hookrightarrow$  true/false,  $\hookrightarrow$  false if x is omitted

**callable(object)**  $\hookrightarrow$  true if object is callable

**help(object)** invokes built-in help system,

(for interactive use)

**id(object)**  $\hookrightarrow$  unique object integer identifier

**print(\*objects, sep=' ', end='\n', file=**

**sys.stdout, flush=False)** prints objects

separated by sep, followed by end;

### File open (and methods)

wholeFilePath = "C:\\file\\test\\mytest.txt"

fObj=open(file[, mode], buffering)) basic modes:

r, r+, w, w+, a ...more file object methods:

.read(size), .readline(), .readlines(),

.write(string), .close(), .splitlines

([keepends]), list(openfile)

with open("C:\\Python351\\Jack.txt", "r+") as sprattfile:

sprattlist=sprattfile.read().splitlines() \*<- removes '\n'

print(sprattlist)

$\hookrightarrow$  ['Jack Spratt', 'could eat ', 'no fat.', 'His Wife', 'could

eat', 'no lean.']\*The WITH structure auto closes the file.

## Operators

**Math:** = (= also can value swap; a,b=a), +, -, \*

, /, // (floor or truncated division - no remainder),

\*\* (exponent), % (mod or modulo returns the

remainder) x = 8%3; print(x)  $\hookrightarrow$  2

**Boolean/Logical:** and, or, not

**Comparison:** == (same as), <, <=, >, >=, is, is

not, !=(is not equal); operators can be chained

**Sequence Variable Opers** + concatenation, \*

repetition, s[i] slice, s[i:j:k] range slice

from,to,step -> starts at 0

**Membership:** in, not in

**Identity:** is/is not checks for objects being the

same object

**Bitwise:** & (and), | (or), ^ (xor 1 not both), ~ flips last bit

<< (shift left), >>(shift right) >>> bin(0b0101 <<1)  $\hookrightarrow$  '0b1010'

**Assignment:** (execute & assign) =, /=, AND

assignment operators [+=, +=, \*=, /=, \*\*=, %=] (only

+ & - work for strings)

r'str' raw string suppresses ESC chrs)

### Other Functions

vars(), dir(), super(), globals(), memoryview(),

setattr(), bytearray(), classmethod(), zip(),

locals(), \_\_import\_\_(), object(), hasattr(),

issubclass(), isinstance(), compile(), hash(),

complex(), bytes(), exec(), setattr(), property

(), getattr(), staticmethod() for a few not

covered here see [www.wikipython.com](http://www.wikipython.com)

## String Methods

**.find(sub[, start[, end]])**

$\hookrightarrow$  First char BEFORE sub is found or -1 if

not found ex: aword = "python"; print

(aword.find("th"))  $\hookrightarrow$  2

**.capitalize()**  $\hookrightarrow$  first character cap'ed

**.lower()**  $\hookrightarrow$  a copy of the string with all

text converted to lowercase.

**.center(width[, fillchar])**

string is centered in an area given by

width using fill character 'fillchar'

**.ljust(width [, fillchar])** or **.rjust()**

**.count(sub[, start[, end]])**

number of substrings in a string

**.isalnum()** **.isnumeric()** **.isalpha**

**.isdigit()** **.isspace()** **.islower()**

**.isupper** **.isprintable()** may be null

$\hookrightarrow$  true if all char meet condition and

variable is at least one char in length

**.replace(old, new[, count])**

$\hookrightarrow$  a copy of the string with substring old

replaced by new. If opt argument 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]. Return -1 on failure.

**.strip([chars])**  $\hookrightarrow$  a copy of the string

with the leading and trailing characters

removed. The chars argument is a string

specifying the set of characters to be

removed. If omitted or None, the chars

argument removes whitespace.

**.zfill(width)**  $\hookrightarrow$  a copy of the string

left filled with ASCII '0' digits to make a

string of length width. A leading sign

prefix ('+'/'-') is handled by inserting the

padding after the sign character rather

than before. Original string is returned if

width is less than or equal to len(str).

**str.split()** - returns list of words extract

-ed by an intervening space

## Data Containers Methods / Operations

In notes below: l[]/k-> index; x->item or object; L/T/D/S->name of list, tuple, dictionary, or set.

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

**TUPLES:** create - `(x,[x],(x),...)` *objects can*  
*include lists and other tuples*; `+=` *add items*;  
`+=(x,)` *add single item*; `tuple[i:j]` *start is 0,*  
*end j-1*; `x,x,...=T[i:j]` *retrieve values*;  
`i=T.index(x[,at or after index i [,before  
index j ]]); for int in T; v=iter(T) creates  
iteration generator; next(v) next iteration;  
len(T); .count(x); .max(T); .min(T); x in T;  
sorted(T, reverse=False); T[::-1]; T=()  
clears all values; del T; T=tuple(somelist)  
creates a tuple from a list`

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

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

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

## Escape Characters

Nonprintable characters represented with  
backslash notation: `r` ignores esc chars; Ex:  
`print('test1\ntest2')` vs. `print(r'test1\ntest2')`  
`\a` bell or alert, `\b` Backspace, `\s` Space,  
`\cx` or `\C-x` Control-x, `\e` Escape, `\M-\C-x`  
Meta-Control-x, `\f` Formfeed, `\n` Newline,  
`\t` Tab, `\v` Vertical tab, `\x` Character x, `\r`  
Carriage return, `\nnn` Octal notation, where  
range of n is 0-7 `\xnn` Hexadecimal  
notation, n is in the range 0-9, a-f, or A-F

## Data Container Functions

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

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

**type([iterable])**  
*a datatype of any*  
*object*

**max(type)** **min(type)** - *not for tuples*  
**sum(iterable [, start])** *must be all numeric,*  
*if a=[8,7,9] then sum(a) returns 24*

**sorted(iterable [,key=],[reversed])**

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

**reverse()** *reverses in place*; `mylist.reverse()`  
**reversed()** *reverses access order—list or tuple*

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

```
    print(i)
for i in reversed(alist):
    print(i)
```

Cy  
Bo  
Amy  
Amy  
Bo

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

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

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

0 Amy  
1 Bo  
2 Cy

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

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

Amy  
Bo  
Cy  
listend

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

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

```
for i in alist:
```

```
    print(next(itermap))
```

**filter(function, iterable)** *iterator for element of*  
*iterable for which function is True.*

**getattr(obj, 'name' [, default])**  
**setattr(object, 'name', value)**

## List Comprehensions

make a new list with item exclusions and modifications  
from an existing list: brackets around expression fol-  
lowed by 0 to many **for** or **if** clauses; can be nested  
`NewList = [[modified]item for item in OldList if some-`  
`conditional-item-attribute of (item)]` or if modifying x  
only, ex: `up1list=[x+1 for x in ylist]`

**\*args and "kwargs":** *are used to pass an*  
*unknown number of arguments to a function.*

*\*args is like a list, \*kwargs is a keyword->value*  
*pair, but keyword cannot be 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' )

## Useful Modules

Python Standard Library Module

Index with links:

<https://docs.python.org/3.5/library>  
**math** like Excel math functions `ceil`  
`(x)` `fsum(iterable)`, `sqrt(x)`, `log`  
`(x[,base])`, `pi`, `e`, `factorial(x)`  
**random** `seed([x])`, `choice(seq)`,  
`randint(a, b)`, `randrange(start,`  
`stop [, step])`, `random(x)` -  
*floating point [0.0 to 1.0]* **sys**  
`stdin` *standard input*, `stdout` *std*  
*output*, `exit(error msg)`  
**datetime** `date.today()`,  
`datetime.now()`, `time asctime`  
`(t)`, `clock()`, `sleep(secs)`  
**calendar**—a world of date options  
`>>> c = calendar.TextCalendar`  
`(calendar.SUNDAY)`  
`>>> c.pmonth(2016, 9)`

September 2016

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

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

**tkinter** Python's

*defacto GUI*; also see **ttk**; **tix**;  
*Older version was: Tkinter (capital T)*  
**os** *deep operating system access*  
`open(name[,mode[, buffer-`  
`ing]])` *modes: 'r' reading, 'w' writ*  
*-ing, 'a' appending, binary append*  
*'b' like 'rb' array work with*  
*mathematical arrays*; **tarfile/zip-**  
**file** - *file compression*; **wave** -  
*interface to wav format*; **RPI.GPIO**  
- *control Raspberry Pi pins via*  
*Python*; **csv** *import comma sep vals*

## re-Regular Expressions module

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

**import re** *if not found 'None'; attrbs error*  
`teststring = "Taking Math 101 is fun!"`  
`mypat = re.compile(r'd+', flags=0)`

```
myso = mypat.search(teststring)
print (myso)
print ('group()', myso.group())
print ('start()', myso.start())
print ('end()', myso.end())
print ('span()', myso.span())
...or don't compile it...
```

```
print(re.search(r'd', teststring).start())
```

**Special characters** `^ $ * + ? { } [ ] \ | ( )`

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

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

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