

Reserve Words

Comparison / Conjunction

False, True, note caps; **==** (is same as); **none** (i.e., null); **and**; **not**; **or**; **in** list/tuple/string/dictionary/set; **is** or **is not** == comparison ↪ 'True' or 'False'

Definition

class create class: class className: see below
def creates a function: def funcName(args):
del deletes 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: See Page 2

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 creates global var in a function
non local a variable inside a nested function is good in the outer function

lambda unnamed

inline function, no return needed

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

Error Management

raise forces a specified exception
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 expression:

```
alist=['Be','my','love']; x=iter(alist)
```

```
for i in range (len(alist)):
```

```
    print(i+1, next(x))
```

range (start, stop, [step])

See data container functions

break ends loop, skips else, for holds val

continue skips to next loop cycle

Decision Making

if elif else

```
def ifExample(MyInt):
```

```
    if MyInt == 1:
```

```
        print('One')
```

```
    elif MyInt == 2:
```

```
        print('Two')
```

```
    else:
```

```
        print('Some other')
```

```
ifExample(int(input("1 or 2: ")))
```

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**

Functions not covered here:

vars(), dir(), super(), globals(), memoryview(), setattr(), bytearray(), classmethod(), locals(), __import__(), object(), hasattr(), isinstance(), compile(), hash(), complex(), bytes(), exec(), delattr(), property(), getattr(), staticmethod()

for some of those not covered here see:

www.wikipython.com

Major Built-In Functions

String Handling (↪=converts/returns)

str(object) ↪ string value of object

repr(object) ↪ printable representation string

ascii(str) ↪ like repr but escape non-ascii

eval(expression) ↪ value after evaluation

chr(i) ↪ character of Unicode [chr(97) = 'a']

ord(str) ↪ value of Unicode character

input(prompt) ↪ user input as a string

len(—) ↪ length of str, items in list/dict/tuple

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

↪ a new string object created by the selection

str.join('string separator',[string list])

format(value [,format_spec]) ↪ 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 forced 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')**

↪ +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**

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

Also: format dates with help of datetime module. **SEE**

WWW.WIKIPYTHON.COM → **TB4: Formatting Options**

Number Handling

abs(x) ↪ absolute value of x

bin(x) ↪ 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, ↪ a pair of numbers -

quotient and remainder using integer division.

float(x) ↪ a floating point number from an

integer or string **A="1.1"; print(float(A)*2)** ↪ 2.2

hex(x) ↪ integer to hex string **hex(65536)** ↪ 0x10000

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

int(x) ↪ integer from a decimal, string, hex

pow(x,y [,z]) ↪ 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]) ↪ floating point

number rounded to digits; Without digits it re-

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

Miscellaneous Functions

bool(x) ↪ True/False, ↪ False if x is omitted

callable(object) ↪ True if object is callable

help(object) invokes built-in help system,

(for interactive use)

id(object) ↪ 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 helpful 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)

↪ ['Jack Spratt', 'could eat ', 'no fat.', 'His Wife', 'could

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

Operators

Math: = (= can also value swap; a, b = b, a), +, -, *, /, // (floor or truncated division - no remainder),

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

remainder) **x = 8%3; print(x)** ↪ 2

Boolean: True or False (1 or 0)

Logical: and, or, not **not(a [and/or] b)**

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

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

Membership: in, not in

Identity: is/is not checks for same object

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

<< (shift left), >>(shift right) >>> **bin(0b0101 <<1)** ↪ '0b1010'

Assignment: (execute & assign) =, //=, -=, +=, *=, /=,

**, %=

Sequence Variable Opers (for strings) +

concatenation, * repetition; **s[i]** single slice,

s[i:j:k] range slice from, to, step -> starts at 0,

end -count from 1; ie 1 more than qty needed

r'str' raw string/byte obj suppresses ESC chrs

Escape Characters

Nonprintable characters represented with
 backslash notation: **r** ignores esc chars;
\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, **\xxx** Hexadecimal notation, n
 is in the range 0-9, a-f, or A-F; **many more**

Helpful String Methods

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

↪ First char BEFORE sub is found or -1 if

not found ex: **print('Python'.find("th"))** ↪ 2

.rfind(sub[, start[, end]])

↪ the **highest index** in the string where

substring **sub** is found, contained within

slice **[start:end]**. Return -1 on failure.

.capitalize() ↪ first character cap'ed

.lower() ↪ a copy of the string with all

text converted to lowercase; **.upper()**

.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

Attributes: **isalnum, isalpha,**

isdecimal, isdigit, isidentifier,

islower, isnumeric, isprintable,

isspace, istitle, isupper - may be

null, ↪ True if all char meet condition

and variable is at least one char in length

.replace(old, new[, count])

↪ a copy of the string with substring old

replaced by new. If opt argument count is

given, only first count are replaced.

.strip([chars]) ↪ 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.

Also **lstrip / rstrip**

.split() - returns list of words extracted

by an intervening space.

str.join(iterable) - concatenates

strings in iterable; str is the separator

Others include: casefold, join,

encode, endswith, expandtabs,

format, format_map, index,

partition, maketrans, rindex,

rpartition, rsplit, splitlines

(keepends), title, startswith,

swapcase, translate, upper, zfill

Data Containers Methods / Operations

In notes below: (i)/j/k & an index; x & value or object; L/T/D/S & an 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)`; `.max(L)`; `.min(L)`; `.pop()`; `.pop(i)`; `.remove(x)`; Replace item `L[i]=x`, Replace multiple items `L[i:j]=[x,x,...]`; Retrieve index, 1st value of `x` `indexno=L.index(x[, at/after index i [,before index j]])`; `.reverse`; `.sort(key=none, reverse=False)`; Create iterative generator `V=iter(L)`, trigger iteration `next(V, default)`

List Comprehensions

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

`newLst = [[modified]item for item in OldLst if some-conditional-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]`

TUPLES: Add items `+=`; Add single item `+=(x)`; `.count(x)`; Create `T=(x,[x],(x),...)` members can be lists & other tuples, parens not required; Create tuple from a list `T=tuple(L)`; clear values `T=()`; `.del T`; Item index `i=T.index(x[,at or after index i [,before index j]])`; Iteration generator `v=iter(T)`, next iteration `next(v)`; `.len(T)`; `.max(T)`; Member `x in T`; `.min(T)`; Retrieve values `x,x,...=T[i:j]`; Slice `T[i:j]` start is 0, end `j-1`; `T[::-1]`; sorted (`T`, `reverse=False`)

DICTIONARIES: Create `D={k:v, k:v,...}`, `=dict.fromkeys(keys/list[,values])`; Add `D2 to D` `D.update(D2)`; `D.copy()`; `D.clear()`; `.del D`; Delete key and value `del D[k]`; `D.get(k[,x])` like `D[k]` but `D.get(k,x)` & x if no k; Iteration variable `V=iter(D)`, trigger iterations `next(v)`; `.len(D)`; Member `x in D`, `x not in D`, `D.has_key(x)`; `D.pop(k[,default])`; `D.popitem()`; Returns `D.items()`, `D.keys()`, `D.values()`; Returns the `v` mapped to `k` `D[k]`; `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: 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`, `S not in`; `.union(other sets)`; `.update(other sets)`

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

comments and suggestions appreciated:
john@johnnoakey.com

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(list(enumerate(alist)))`
[[0,'x'], (1,'y'), (2,'z')]

Use `enumerate` to make a dictionary: ex:
`mydict = dict(enumerate(mylist))`
Dictionaries enumerate keys & yield values unless values specified; `print(dict(enumerate(mydict.values())))` yields keys

type([iterable])

& a datatype of any object

max(type) **min(type)**

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() inverts list order; `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
Cy
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))

zip merges two iterables left to right
filter(function, iterable) iterator for element of iterable for which function is True.
getattr(object, 'name' [, default])
setattr(object, 'name', value)

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 & colon
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+"\n", Example: "+self.ex`
`print(info)`
`x = mammalia("Cetacea", "whales")`
`x.printInfo()`
`class/order: mammal/Cetacea, Example: whales`

Creating a Function: required components - red, optional - 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

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
l
i
v
e
r

***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 Module/Functions

Python Standard Library Module

See 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)`

calendar—a world of date options

`import calendar`
`mymo = calendar.TextCalendar()`
`mymo.setfirstweekday(calendar.SUNDAY)`
`mymo.pmonth(2018,7)`

July 2018

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	31				

Works best with a
mono-spaced font like
Consolas.

tkinter also see **ttk**; **tix**; see TB's on **wikipython**; **tkinter NOT Tkinter**
RPi.GPIO - control Raspberry Pi pins via Python; See also: **os** deep operating system access; **array** arrays; **tarfile/zip-file** - file compression; **wave** - interface to wav format; **csv** access data: comma separated values, so very very much more.

A note on format: (1) new f string options available in version 3.6 (2) the old string % syntax will eventually be deprecated: `print("$%.2f buys %d %sss"%(1.2,2,'hot dog'))` try it