

(x->y) indicates argument->return value

Python CORE DATA TYPES

INTEGER (int)

initialize an int

```
var = 10
```

convert to an int (str/float->int)

```
var = int('55') # 55
```

```
var2 = int(55.5) # 55
```

FLOAT (float)

initialize a float

```
var = 5.5
```

convert to a float (int/str->float)

```
var = float(55) # 55.0
```

```
var2 = float('55.5') # 55.5
```

int AND float MATH

```
mysum = 6 + 3
```

```
myprod = 6 * 3
```

```
mydiv = 5 / 3.0
```

```
mypow = 5 ** 3 # exponent
```

```
mymod = 6 % 4 # modulus: 2
```

STRING (str)

initialize a str (single/double quotes)

```
var = 'hello'
```

```
var2 = "hello"
```

initialize a str (triple quotes)

```
var = """a
```

```
single
```

```
string"""
```

convert any object to a str (obj->str)

```
var = str(55) # '55'
```

+: concatenate strs (str->str)

```
var1 = 'hello'; var2 = 'hi'
```

```
cvar = var1 + var2 # 'hellohi'
```

***: str repetition (str,int->str)**

```
var = '$'
```

```
rvar = var * 5 # '$$$$$'
```

slice: retrieve a substring from this str

```
line = 'the string'
```

```
slc1 = line[0:3] # 'the'
```

```
slc2 = line[0:7:2] # 'tesr'
```

count(): #occurrences of string in this str

```
var = 'hello'
```

```
numfound = var.count('l') # 2
```

endswith(): True if this str ends with a str

```
var = 'hello'
```

```
if var.endswith('o'): # True
    print("var ends with an 'o'")
```

find(): index pos of str in this str (str->int)

```
var = 'hello'
```

```
index = var.find('l') # 2
```

format(): insert values into a str (obj->str)

```
var = 'hello'
```

```
var2 = 'world'
```

```
frd = '{}', '{}!'.format(var, var2)
```

isalpha(): True if all alpha chars (str->bool)

```
var = 'hello'
```

```
if var.isalpha(): # True
    print('var is all letters')
```

isdigit(): True if all digits (str->bool)

```
var = '12345'
```

```
if var.isdigit(): # True
    print('var is all digits')
```

join(): join a list of str to str (list->str)

```
els = ['a', 'b', 'c', 'd']
```

```
jstr = ':'.join(els) # 'a:b:c:d'
```

lower(): return a str lowercased (str->str)

```
var = 'HELLO'
```

```
lvar = var.lower() # 'hello'
```

replace(): replace str in this str (str->str)

```
var = 'hello'
```

```
rvar = var.replace('l', 'y')
```

```
# 'heyyo'
```

rstrip(),lstrip(),strip(): strip chars (str->str)

```
line = 'this line of text,\n'
```

```
# no arg: strip whitespace
```

```
line = line.rstrip()
```

```
print(line) # 'this line of text,'
```

```
# with char arg: strip char
```

```
line = line.rstrip(',')
```

```
print(line) # 'this line of text'
```

```
# strip any chars included in str
```

```
line = line.rstrip('.,;!?') # any
```

```
line = line.lstrip() # strip left
```

```
line = line.strip() # strip both
```

split(): split a str into list of str (str->list)

```
line = 'this:that:other'
```

```
els = line.split(':')
```

```
# ['this', 'that', 'other']
```

```
# no arg: split on whitespace
```

```
line2 = 'this that other'
```

```
els = line2.split()
```

```
# ['this', 'that', 'other']
```

splitlines(): split str on newline (str->list)

```
text = """this represents
```

```
a file
```

```
with newlines
```

```
attached"""
```

```
lines = text.splitlines()
```

```
# ['this represents', 'a file',
```

```
'with newlines', 'attached']
```

upper(): return a str uppercased (str->str)

```
var = 'hello'
```

```
uvar = var.upper() # 'HELLO'
```

LIST (list)

initialize a list

```
x = ['a', 'b', 'c']
```

loop through / iterate through a list

```
for el in x: # el is element of x
    print(el)
```

check for membership in a list (obj->bool)

```
if val in x:
    print('val is in x')
```

subscript: retrieve an element by index

```
item = x[0] # 'a'
```

slice: retrieve a sub-list from this list

```
slice1 = x[0:2] # ['a', 'b']
```

```
slice2 = x[0:4:2] # ['a', 'c']
```

append(): add to end of list (no retrn val)

```
x.append('d')
```

pop(): remove element from this list

```
el = x.pop(0) # remove first
```

```
el2 = x.pop() # remove last
```

TUPLE (tuple)

initialize a tuple

```
x = ('a', 'b', 'c')
```

loop through / iterate through a tuple

```
for el in x:
```

```
    print(el)
```

check for membership in tuple (obj->bool)

```
if val in x:
```

```
    print('val is in x')
```

slice: retrieve a sub-tuple from this tuple

```
x = ('a', 'b', 'c')
```

```
newtuple = x[0:2] # ('a', 'b')
```

SET (set)

initialize a set

```
xset = {'a', 'b', 'c'}
```

```
xset = set(['a', 'b', 'c'])
```

loop through / iterate through a set

```
for el in xset:
```

```
    print(el)
```

check for membership in a set (obj->bool)

```
if val in xset:
```

```
    print('val is in x')
```

add(): add to a set (no return value)

```
xset.add('d')
```

difference(): items in set not in other

(->set)

```
dset = xset.difference(othrset)
```

intersection(): items in both (set->set)

```
iset = xset.intersection(othrset)
```

union(): items in either or both (set->set)

```
uset = xset.union(othrset)
```

DICTIONARY (dict)

initialize a dict

```
dd = {'a':1, 'b': 2, 'c': 3}
```

loop through / iterate through a dict

```
for key in dd:
```

```
    print(key, '=', dd[key])
```

check key membership (obj->bool)

```
if 'c' in x:
```

```
    print('c', 'is in dict')
```

add key / value pair

```
dd['d'] = 4
```

get value based on key

```
val = dd['a']
```

get value based on key, or default value if key is missing

```
val = dd.get('z', None)
```

del: remove key/value

```
del dd['a']
```

items(): get iteration of pairs as 2-item

tuples (dict->list)

```
for key, val in dd.items():
```

```
    print(key, '=', val)
```

keys(): get iteration of keys in dict

(dict->list)

```
for key in dd.keys():
```

```
    print(key)
```

values(): get iteration of values

(dict->list)

```
for val in dd.values():
```

```
    print(val)
```

BOOLEAN (bool)

```
initialize
x = False
y = True

test for True
if x:
    print("'x' is True")
```

NULL VALUE (None)

```
initialize
x = None

test for None
if testval is None:
    print('testval is None')
```

FILE

```
open a file (str->file)
fh = open('thisfile.txt')

open with implicit file closing
with open('thisfile.txt') as fh:
    text = fh.read()
## (file is implicitly closed)

readlines(): file as list of lines(file->list)
lines = fh.readlines()

read(): file as single string (file->str)
text = fh.read()

close(): close a file (no return val)
fh.close()

write() with 'w': write to a file
fh = open('thisfile.txt', 'w')
fh.write('a line of text\n')

write() with 'a': append to a file
fh = open('thisfile.txt', 'a')
fh.write('an appended line\n')
```

BUILT-IN FUNCTIONS

```
enumerate(): integers paired with
sequence elements (seq->list of tuples)
x = ['a', 'b', 'c']
for num, element in enumerate(x):
    print(num, element)
# 0 a 1 b 2 c (each on a line)

exit(): terminate program execution
exit(0) # indicates no error
exit('error message') # w/error

len(): length of a sequence (seq->int)
var = len('hello') # 5
var2 = len(['a', 'b', 'c']) # 3
var3 = len({'a':1, 'b':2}) # 2

max(): max value in a sequence (seq->int)
a_max = max([1, 2, 3, 4]) # 4

min(): min value in a sequence (seq->int)
a_min = min([1, 2, 3, 4]) # 1

print(): echo text, usually to screen
print('this') # adds a newline
print('hello', end='') # no newline
print('this', 'that') # this that
# (adds space between items)

range(): generate iterable of ints (int->iter)
for num in range(5):
    print(num) # 0 1 2 3 4
for num in range(1,6):
    print(num) # 1 2 3 4 5

input(): take keyboard input (str->str)
str_a = input('type something:')
```

```
repr(): return a "true" string reprsntation
str_b = repr(myobj)

round(): round a float (float->float)
num = 1.66666667
rnum = round(num, 2) # 1.67

sorted(): return a sorted list (seq->list)
s1 = sorted(mylist)
s2 = sorted(mylist, reverse=True)
s3 = sorted(mylist, key=sfunc)

sum(): (seq-> int or float)
as1 = sum([1, 2, 3, 4.0]) # 10.0

type(): type of object (int,str,etc.) (->type)
print(type(var)) # <class 'int'>
```

DEFINING A FUNCTION

```
define function with positional args
def dothis(arg1, arg2):
    return arg1 + arg2

define func with optional/default args
def dothis(opt1=None, opt2=None):
    if opt1 and opt2:
        return opt1 + opt2
```

STATEMENTS

conditionals

```
# comparison operators:
# <, >, !=, ==, <=, >=
if var >= var2:
    print("it's true")

using not
if not var == var2:
    print("it's not equal")

using and
if var == var2 and var > var3:
    print("both are true")

using or
if var == var2 or var > var3:
    print("one or both are true")

if/elif/else
if var > var2:
    print('var is greater')
elif var < var2:
    print('var is less than var2')
else:
    print('var is equal to var2')
```

while

```
numeric criteria
var = 1
while var <= 10:
    print(var)
    var = var + 1

endless loop using break
while True:
    print('cycling...')
    if input('quit?') == 'y':
        break
```

list comprehension (seq->list)

```
filtering
newlist = [ x for x in oldlist
            if x > 5 ]

transforming
newlist = [ x * 2
            for x in oldlist ]
```

```
filtering and transforming
newlist = [ x*2 for x in oldlist ]

lambda
x = sorted(y, key=lambda x: x[0])
```

MODULES

```
import a module
import modname
print(modname.var1)

import a module and rename
import modname as mod
print(mod.var1)

import a module's variables
from modname import var1, var2
print(var1); print(var2)
```

CLASSES

```
create a class
class MyClass(object):
    def __init__(self, arg):
        self.attr = arg
    def instfunc(self, arg):
        self.attr = self.attr+arg
```

FILES/DIRS/COMMAND LINE

```
os.listdir(): list a directory (str->list)
import os
for item in os.listdir('mydir'):
    print(item)

os.path.join(): join a path (e.g. dir/dir/file)
import os
print(os.path.join('mydir', fnme))

get arguments to a program (->list)
import sys
args = sys.argv[1:]
```

EXCEPTIONS

```
trap an exception
try:
    print(mylist[5])
except IndexError, e:
    print(e)

raise an exception
raise ValueError('wrong value!')

assert a truth
assert var == 5
```

OTHER

```
account for an empty block
if True:
    pass
else:
    # do something else

declare a var global in a func
x = 0
def myfunc():
    global x
    x = x + 1
```

Copyright 2017 David Blaikie
davidpython.com

Last updated 2017-05-16