```
3 * 4, 3 + 4, 3 - 4, 3 / 4               #==> 12, 7, -1, 0.75
3 ** 4, 3 // 4, 3 % 4                    #==> 81, 0, 3
4 > 3, 4 >= 3, 3 == 3.0, 3 != 4, 3 <= 4   #==> True, True, True, True, True
# order of operations: brackets, **, {* / // %}, {+ -}, {== != <= < > >=}
min(3, 4), max(3, 4), abs(-10)           #==> 3, 4, 10
sum([1, 2, 3])  # [1, 2, 3] is a list    #==> 6

type(3), type(3.0), type("myVariable")   #==> class 'int', class 'float',
                                         #    class 'str'
int("4"+"0"), float(3), str(1 / 2)       #==> 40, 3.0, '0.5'

"double quotes: ', escaped \" \\ \'"     #==> double quotes: ', escaped " \ '
'it\'s "similar" in single quotes '      #==> it's "similar" in single quotes

ord("A"), chr(66)                        #==> 65, 'B'
string = "hello"
# the following statements work for lists too
len(string)                              #==> 5
string[0], string[4]    # get characters  #==> "h", "o"
string[1:3]             # get a substring #==> "el"
string[:2], string[2:]  # l/r substrings  #==> "he", "llo"
string[-1], string[-2:] # negative indices#==> "o", "lo"
"con" + "cat" + "enation " + str(123)    #==> "concatenation 123"
"boo" * 2                                #==> "booboo"

getLineOfInputAsString = input()         #==> read input (or EOF error)
print("takes", 0, "or more arguments")   #==> takes 0 or more arguments
print("using", "custom", "sep", sep=".")  #==> using.custom.sep
print("no", "newline", end="bye")        #==> no newlinebye

not True, False or True, False and True  #==> False, True, False
# order of operations: brackets, {== !=}, not, and, or

if booleanCondition:
   x                      # indent the body block
   x                      # every line by the same amount
elif anotherCondition:    # can do zero, one, or several elif blocks
   x                      # multi-line block
else:                     # optional
   x                      # multi-line block

while booleanCondition:
   x                      # the body block
   break                  # jump out (optional)
   continue               # restart from top of next iteration (optional)

for indexVariable in range(low, highPlus):
   print(indexVariable)                  #==> low, low+1, ..., highPlus-1
# "for item in listOrString:" forall/foreach loops
# break, continue can be used in for loops
```

```
def nameOfNewFunction(argument1, argument2):
    x                      # the body block
    return y               # (optional; if you don't return, it returns None)

def remember(bar):         # writing to global variables
    global saveBar         # after calling foo(3), saveBar = 3
    saveBar = bar          # even outside of the function's scope

# these 'slice' commands have analogues for lists and range()
"0123456789"[::2]          # slices        #==> "02468"
"0123456789"[::-1]         # descending    #==> "9876543210"
"0123456789"[6:3:-1]                       #==> "654"

x += 1        # also -=, /=, *=, %=, **=, //=. Python has no C++-style "x++"
x, y = y, x   # multiple assignment
3 < x < 5     # same as "(3 < x) and (x < 5)". can chain {< <= > >= == != is}

import math                # import, to get everything with period
print(math.sqrt(2))
from math import sqrt      # import, to get one thing without period
print(sqrt(2))
# also in math module: pi, exp, log, sin, cos, tan, ceil, floor, and more

list = ['zero', 'one', 'two']
list.index('one')                          #==> 1
list.index('three')                        #==> causes an error
'three' in list, 'zero' in list            #==> False, True
list.count('two')                          #==> 1
del list[1]     # list becomes ['zero', 'two']
"string" in "superstring"                  #==> True
"superstring".index("string")              #==> 5

# more list methods: append(item), insert(index, item), extend(list),
# remove(value), pop(), pop(index), reverse(), sort(), and more

# some string methods: capitalize(), lower/upper(), islower/isupper(),
# isalpha/isdigit(), center/ljust/rjust(width, fillChar), strip(), split(),
# splitlines(), endswith/startswith(string), find(string), replace(old, new),
# and more

myList = [11, 99]
actuallyTheSameList = myList # not a true copy! just copies the reference
myList is actuallyTheSameList              #==> True
realCopy = myList[:]         # or list(myList), copy.copy(myList), deepcopy
realCopy is myList                         #==> False
```