```
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
# ordre des opérations: parenthèses, **, {* / // %}, {+ -}, {== != <= < > >=}
min(3, 4), max(3, 4), abs(-10)            #==> 3, 4, 10
sum([1, 2, 3])  # [1, 2, 3] est une liste #==> 6

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

"quotes doubles: ', échappé \" \\ \'"     #==> quotes doubles: ', échappé " \ '
'quotes simples: c\'est "similaire"'      #==> quotes simples: c'est "similaire"

ord("A"), chr(66)                         #==> 65, 'B'
chaine = "bonjour"      # type 'str'
# les déclarations suivantes fonctionnent aussi pour les listes
len(chaine)                               #==> 7
chaine[0], chaine[3]    # caractères      #==> "b", "j"
chaine[1:3]             # sous-chaînes    #==> "on"
chaine[:2], chaine[2:]  # sous-chaîne g/d #==> "bo", "njour"
chaine[-1], chaine[-2:] # indices negatifs#==> "r", "ur"
"en" + "chaîne" + "ment " + str(123)      #==> "enchaînement 123"
"bon" * 2                                 #==> "bonbon"

ligneDEntreeCommeChaine = input()         #==> lire entrée (ou erreur EOF)
print("prend", 0, "arguments ou plus")    #==> prend 0 arguments ou plus
print("avec", "sep", "choisi", sep=".")   #==> avec.sep.choisi
print("pas de retour à la ligne", end="!")#==> pas de retour à la ligne!

not True, False or True, False and True   #==> False, True, False
# ordre des opérations: parenthèses, {== !=}, not, and, or

if conditionBooleenne:
   x                    # bloc indenté des instructions
   x                    # chaque ligne du bloc a la même indentation
elif autreCondition:    # peut avoir zéro, un, or plusieurs blocs elif
   x                    # bloc indenté des instructions
else:                   # optionnel
   x                    # bloc indenté des instructions

while conditionBooleenne:
   x                    # bloc indenté des instructions
   break                # sauter dehors (optionnel)
   continue             # commencer au début de l'iteration suivant (optionnel)

for variableDIndice in range(debut, finPlus):
   print(variableDIndice)           #==> debut, debut+1, ..., finPlus-1
# "for article in listeOuChaine:" boucle forall/foreach (pour tous/pour chaque)
# break, continue peuvent être utilisés dans les boucles for
```

```
def nomDeNouvelleFonction(argument1, argument2):
    x                     # bloc indenté des instructions
    return y              # (optionnel; s'il n'existe pas, None est retourné)

def memoriser(bar):       # l'écriture aux variables globales
    global enregistrerBar # après appeler foo(3), enregistrerBar = 3
    enregistrerBar = bar  # même en dehors de la portée de la fonction

# ces commandes "tranche" ont des analogues pour les listes et pour range()
"0123456789"[::2]         # tranche       #==> "02468"
"0123456789"[::-1]        # décroissant   #==> "9876543210"
"0123456789"[6:3:-1]                      #==> "654"

x += 1        # aussi -=, /=, *=, %=, **=, //=. Python n'a pas de "x++"
x, y = y, x   # affectations multiples
3 < x < 5     # comme "(3 < x) and (x < 5)". peut combiner {< <= > >= == != is}

import math               # import, pour obtenir tout avec une période
print(math.sqrt(2))
from math import sqrt     # import, pour obtenir une chose sans période
print(sqrt(2))
# aussi dans le module math: pi, exp, log, sin, cos, tan, ceil, floor, etc...

liste = ['zéro', 'un', 'deux']
liste.index('un')                         #==> 1
liste.index('trois')                      #==> cause une erreur
'trois' in liste, 'zéro' in liste         #==> False, True
liste.count('deux')                       #==> 1
del liste[1]     # liste devient ['zéro', 'deux']
"chaîne" in "super-chaîne"                #==> True
"super-chaîne".index("chaîne")            #==> 6

# encore des methodes pour listes: append(article), insert(article, indice),
# extend(liste), remove(valeur), pop(), pop(indice), reverse(), sort(), etc...

# quelques methodes pour chaînes: capitalize(), lower/upper(),
# islower/isupper(), isalpha/isdigit(), strip(), split(), splitlines(),
# center/ljust/rjust(largeur, carDeRemplissage), endswith/startswith(chaîne),
# find(chaîne), replace(ancien, nouvel), etc...

monListe = [11, 99]
enFaitLeMemeListe = monListe # pas une vraie copie! ne copie que la référence
enFaitLeMemeListe is monListe             #==> True
vraiCopie = monListe[:]      # ou list(monListe), copy.copy(monListe), deepcopy
vraiCopie is monListe                     #==> False
```