1. Introduction

Le but de ce cours avec de nombreux exercices est d'introduire le langage de programmation Python.
Dans mon site Web, je référence quelques autres cours sur le langage Python :
http://www.juggling.ch/gisin/python/python.html

2. Première prise en main du langage Python

Exécutez le logiciel **IDLE**. Le "Python 3.6.x Shell" s'ouvre.
Tapez : `1 + 1` Le résultat s'affiche.
Tapez : `355 / 113` Le résultat s'affiche.
Tapez :

```
for i in range(1,11):
    print(i, i**2)
```

Terminez en tapant deux fois sur la touche "Enter"

3. Les variables

Il est indispensable de stocker des nombres, des chaînes de caractères (string) et divers structures dans des zones mémoire. Pour cela les **variables** sont utilisées. On leur donne un nom formé d'une lettre, suivit de 0, 1 ou plusieurs autres lettres, chiffres et du caractère _
On assigne une valeur à une variable à l'aide du symbole =
Tapez : `b = 5` # Rien ne s'affiche
Ce qui suit le # est un **commentaire**. Il est indispensable de **commenter vos programmes** !
Tapez : `print(b); b = b + 1; print(b);` # La valeur de la variable `b` a changé.
Tapez : `b += 1; print(b);` # La valeur de la variable `b` a changé.
Tapez : `c = b * b; print(c);`
Tapez : `c = c * b; print(c);`
Le symbole = est utilisé comme **opérateur d'affectation**. L'expression à droite de = est évaluée et le résultat est mis dans la variable qui se trouve à gauche de =
Tapez : `3*c = c + 14` # Vous constaterez que vous obtenez une erreur.
Tapez : `bonjour = "Hello"; print(bonjour)`
Tapez : `bonjour = bonjour + " tout le monde"; print(bonjour)`
Tapez : `bonjour += ". Comment allez-vous ?"; print(bonjour)`

Dans Python, certains mots sont **réservés** et ne peuvent pas être utilisés comme nom de variable.
Il y a 33 mots réservés que voici :

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

4. Les opérateurs

Voici une liste d'opérateurs dans Python. À vous de décrire ce qu'ils font :

```
2 + 3                                   9 - 5
5 - 9                                   7 * 8
5 + 3 * 4                               (5 + 3) * 4
25 / 7                                  25 // 7
25 % 7                                  (25 // 7) * 7 + 25 % 7
355 / 113                               355 // 113
355 % 113                               (355 // 113) * 113 + 355 % 113
2**4                                    3**4
2**10                                   4**3
"abc" + "def"                           5*"salut "
salut = 8                               5*salut
"abcde" - "a"
8 > 5                                   8 < 5
2 + 3 == 5                              2 + 3 != 5
2 + 3 >= 5                              2 + 3 <= 5
(8 > 5) and (8 < 5)                     (8 > 5) or (8 < 5)
2 << 5                                  128 >> 4
13 & 7                                  13 | 7
~13                                     13^7
```

Les opérateurs : `+ - * / // % **` sont des **opérateurs arithmétiques**.
Les opérateurs : `+ *` sont des **opérateurs sur des chaînes de caractères.**
Les opérateurs : `== != > < >= <= and or not is` sont des **opérateurs logique**.
Les opérateurs : `<< >> & | ~ ^` sont des **opérateurs sur les bits**. (bitwise operators)
`=` est un **opérateur d'assignation.**
`( ) [ ] { }` sont aussi considérés comme des opérateurs.
Il y a encore d'autres opérateurs moins importants pour nous.

5. Les types

Dans certains langages de programmation, le type des variables doit être défini avant leur utilisation.
En Python, le type des variables est défini automatiquement à leur assignation.
Testez les instructions suivantes pour comprendre leur signification.

```
type(20)                                type(20.0)
type(25 / 7)                            type(25 // 7)
type(25 % 7)                            type(27.5 % 5)
type(27.0 % 5)                          type(27 % 0.5)
type("hello")                           type("ab" + "cd")
type(5*"salut ")                        type(5*"salut " + "à tous")
e1 = 12                                 type(e1)
e1 = e1 / 5                             type(e1)
e2 = 111111111; type(e2)
e3 = e2 * e2; print(e3); type(e3)
e4 = e3 * e3; print(e4); type(e4)
```

Conversion de types

On peut convertir un type en un autre. Testez les types des variables suivantes :

```
a1 = int(3.14)                          a2 = float(3)
a3 = str(3)                             a4 = str(3.14)
a5 = "33"                               a6 = float(a5)
a7 = "3.14"                             a8 = float(a7)
a9 = 3.14                               a10 = "le nombre " +  str(a9) + " ~= pi"
a11 = int("3.14")                       a12 = int(float("3.14"))
```

6. L'éditeur IDLE

Rapidement, on désire mémoriser dans un fichier texte les instructions que l'on a tapé. Pour cela, lancez l'éditeur intégré **IDLE** depuis le menu "**File -> New File**"
Dans l'éditeur, retapez les instructions de conversions de types de la page précédente, avec des :
`print("a1 =", a1, type(a1))`
après chaque définition de variable, `a1` étant remplacé par `a2`, `a3`, `a4`, etc. convenablement.
Le début sera donc :

```
# À vous d'écrire un commentaire pertinent ici.
a1 = int(3.14)
print("a1 =", a1, type(a1))
a2 = float(3)
print("a2 =", a2, type(a2))
a3 = str(3)
print("a3 =", a3, type(a3))
```

Il est utile de taper les instructions les unes à la suite des autres, pour la lisibilité !
Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex06000_editeur_idle.py**
Exécutez-le en pressant la touche **F5**.

7. Les modules, les packages, les bibliothèques

Dans le "*Python 3.6.x Shell*", tapez : `sin(3)`

Vous obtenez une erreur, car ne nombreuses fonctions ne sont pas implémentés de façon standard dans Python. On peut ajouter des fonctions en important des modules.

Dans l'éditeur, créez un nouveau fichier et tapez :

```
# À vous d'écrire un commentaire pertinent ici.
from math import *
print(sin(90))
```

Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex07000_module_math.py**
Exécutez-le en pressant la touche **F5**.

? Les arguments des fonctions trigonométriques, sont-ils en degrés ou radians ?

Complétez votre code en tapant à la suite :

```
print(sqrt(25), sqrt(2))  # La fonction sqrt calcule ... à compléter
approx_e = 2 + 1/2 + 1/(2*3) + 1/(2*3*4) + 1/(2*3*4*5) + 1/(2*3*4*5*6)
print(approx_e, " ~=", e)
print("Erreur relative =", (approx_e - e) / e)
approx_pi = 355 / 113
print(approx_pi, " ~=", pi)
print("Erreur relative =", (approx_pi - pi) / pi)
```

Nous verrons par la suite plusieurs autres modules. Ils étendent les fonctionnalités de Python.
Il en existe des dizaines de milliers !!!

Dans le "*Python 3.6.x Shell*", tapez : `help("math")` et double-cliquez sur le message jaune.

Le module *random*, avec la fonction *randint(min, max)* est souvent utile.

8. La boucle while

Fréquemment, on veut répéter plusieurs fois une instruction, avec une légère variation entre chaque instruction.
Dans l'éditeur, créez un nouveau fichier et tapez :

```
# À vous d'écrire un commentaire pertinent ici.
from math import *

nMax = 10
somme=0
nn = 0
while nn < nMax:
    nn += 1
    somme = somme + 1/nn
    print("nn =", nn, " ; somme =", somme) # À mettre en commentaire par la suite.

print("nn =", nn, " ; somme =", somme)
print("gamma ~= ", somme - log(nMax))
```

Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex08000_boucle_while.py**
Exécutez-le en pressant la touche **F5**.

*? Que calcule cette "**boucle while**" ?*
? Dans quelle base la fonction log est-elle calculée ?

Vous pouvez jouer avec ce programme en augmentant l'exposant, mais pas de trop !

Par la suite, on utilisera de plus en plus l'éditeur de fichiers.

Exercice 8.1 :
Soit à la suite du programme précédent, soit dans un nouveau fichier, sans oublier d'écrire des commentaires,
calculez la somme : $1 + 1/2^2 + 1/3^2 + ... + 1/100000^2$

Vérifiez qu'elle se rapproche de $\frac{\pi^2}{6}$ *qui se note pi^2 / 6*

Exercice 8.2 :
Le calcul : `approx_e = 2 + 1/2 + 1/(2*3) + 1/(2*3*4) + 1/(2*3*4*5) + 1/(2*3*4*5*6)`
*est répétitif, donc la "**boucle while**" est faite pour cela.*
L'intérêt est d'aller plus loin que le calcul de 6 termes.
Soit à la suite du programme précédent, soit dans un nouveau fichier, + commentaires,
calculez la somme approx_e = 2 + 1/2! + 1/3! + 1/4! + ... + 1/nn!
nn étant donné. Exemples nn = 6 ou nn = 10.
kk! se calcule avec Python avec la fonction : factorial(kk)

Vérifiez que la somme se rapproche du nombre e.

Exercice 8.3 :
Saurez-vous calculer : approx_e_x = $1 + x + x^2/2! + x^3/3! + x^4/4! + ... + x^{nn}1/nn!$
nn et x étant donnés.

Vérifiez que la somme se rapproche du nombre e^x.

9. La boucle for

Une autre boucle est très utile, la "**boucle for**". Elle permet de passer en revue tous les éléments d'une chaîne de caractère, d'une liste ou d'autres structures.

Dans l'éditeur, créez un nouveau fichier et tapez :
 # À vous d'écrire un commentaire pertinent ici.

```
from math import *

print("----- Exercice 9.0a -----")
machaine = "Ceci est un petit texte"
for cc in machaine:
    print(cc)
```

Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex09000_boucle_for.py**
Exécutez-le en pressant la touche **F5**.

Ajoutez les lignes suivantes :

```
print("----- Exercice 9.0b -----")
strS = ""
for cc in machaine:
    strS = strS + cc + "; "
print(strS)
```

Exercice 9.1 :
Ajoutez du code pour qu'il écrive le texte de la variable "machaine" à l'envers, c'est-à-dire la première lettre en premier, puis l'avant-dernière, etc.
Cela donne : 'etxet titep ...'

Ajoutez les lignes suivantes pour comprendre ce qu'elles font :
Écrivez en commentaires ce qu'elles font.

```
print("----- Exercice 9.2a -----")
for nn in range(11):
    print(nn, end="; ")
print()  # Que se passe-t-il si on enlève ce "print()" ?

for nn in range(4, 10):
    print(nn, end="; ")
print()  # Que se passe-t-il si on enlève ce "print()" ?

for nn in range(10, 4, -1):
    print(nn, end="; ")
print()  # Que se passe-t-il si on enlève ce "print()" ?
print("Terminé")
```

Exercice 9.2 :
En utilisant une boucle "for" au lieu d'une boucle while, calculez les sommes :
s1 = 1 + 2 + 3 + 4
s2 = 1 + 2 + 3 + ... + 100
s3 = 1 + $1/2^2$ + $1/3^2$ + ... + $1/100000^2$

Vous rappelez-vous vers quelle nombre se rapproche la somme s3 ?

10. Le Test if ... elif ... else ...

Souvent, il faut tester si une condition est satisfaite ou non et le programme continuera différemment suivant le résultat du test. C'est l'utilité du "**Test if ... elif ... else ...**".

Dans l'éditeur, créez un nouveau fichier et tapez :

```
# À vous d'écrire un commentaire pertinent ici.
from math import *
print("----- Exercice 10.0a -----")
nb = 55
if (nb % 2 == 0):
    print(nb, "est un nombre pair")
else:
    print(nb, "est un nombre impair, ou ...")

print("----- Exercice 10.0b -----")
nb = 55
if (nb % 3 == 0):
    print(nb, "est divisible par 3")
elif ((nb-1) % 3 == 0):
    print(nb, "- 1  est divisible par 3")
elif ((nb-2) % 3 == 0):
    print(nb, "- 2  est divisible par 3")
else:
    print(nb, "est un nombre spécial, il y en a !")

print("----- Exercice 10.0c -----")
nb = 55
while nb > 0:
    print(nb)

    if (nb % 2 == 0):
        nb = nb // 2
    else:
        nb = (nb-1) // 2
```

Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex10000_test_if_elif_else.py**
Exécutez-le en pressant la touche **F5**.

Exercice 10.1 :
Écrivez le code pour qui indiquera si un nombre donné est divisible par 5 ou non.

Exercice 10.2 :
Écrivez le code pour qui indiquera si un nombre donné est divisible :
par 2 et/ou par 3 et/ou par 5 et/ou par 7.
C'est un début pour tester si un nombre entier est premier.

Exercice 10.3 :
a, b, c étant donnés, déterminez si le polynôme $ax^2 + bx + c$ *possède des racines et indiquez-les.*
sqrt(expr) retourne la racine carrée de expr.
Avant de d'afficher les réponses, afficher le polynôme.

L'instruction **elif** peut apparaître zéro, une ou plusieurs fois.
L'instruction **else** peut apparaître zéro ou une fois, elle est optionnelle.

11. L'entrée de données avec la fonction "input" et gestion d'erreur

Pour qu'un programme puisse interagir avec l'utilisateur, il doit non seulement pouvoir afficher des informations, mais également pouvoir recevoir des informations de l'utilisateur.

Dans l'éditeur, créez un nouveau fichier et tapez :

```
# À vous d'écrire un commentaire pertinent ici.
print("----- Exercice 11.0a -----")
nb = input("Tapez un nombre entier : ");
print("La moitié de ce nombre vaut :", nb/2)
```

Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex11000_input.py**
Exécutez-le en pressant la touche **F5**.

Vous obtenez une erreur ! Dans le Shell, tapez : `type(nb)`
Expliquez pourquoi l'instruction : `nb/2` provoque une erreur.

"**input**" **retourne toujours un type 'str'** (string).
Si on veut faire des opérations arithmétiques avec le nombre tapé, il faut le convertir avec **int** ou **float**.

Modifiez votre code pour avoir :

```
print("----- Exercice 11.0b -----")
strS = input("Tapez un nombre entier : ");
nb = int(strS)
print("La moitié de ce nombre vaut :", nb/2)
```

Exécutez le code et tapez autre chose qu'un nombre entier, ou un nombre entier suivit d'autres caractères. Constatez l'erreur.

Modifiez votre code pour avoir :

```
print("----- Exercice 11.0c -----")
while True:
    try:
        strS = input("Tapez un nombre entier : ");
        nb = int(strS)
        break
    except:
        print("Il faut taper un nombre entier, recommencez...")

print("La moitié de ce nombre vaut :", nb/2)
```

Cette fois, la séquence **try ... except ...** permet de gérer l'erreur pour n'accepter que des nombres entiers.

Il ne sera pas toujours nécessaire de gérer les erreurs, mais il faudra être conscient que le programme peut planter si l'on ne donne pas un nombre entier lorsque cela est demandé.

Exercice 11.1 :
Le problème de **Syracuse** est le suivant :
On part d'un nombre entier positif.
S'il est pair, on le remplace par sa moitié.
S'il est impair, on le remplace par son triple additionné de 1.
On recommence.
Malgré la simplicité de cet algorithme, on ne sait toujours pas en 2019 si le nombre 1 arrive dans la succession de nombres obtenus, quel que soit le nombre de départ. On sait que c'est le cas pour tous les nombres entre 1 et 10^{20}.
Écrivez le code qui, partant d'un nombre nb, écrit la suite des nombres obtenu par le problème de Syracuse. Il s'arrête si on tombe sur le nombre 1.
Par exemple, en partant de 7, on a la suite :
7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Exercice 11.2 :
Conversion d'un nombre entier positif en binaire.
Étant donné un nombre entier positif, écrire le programme qui stock dans une chaîne de caractères l'expression binaire de ce nombre. Affiche le nombre en fin de programme.

Exercice 11.3 :
Jeu de Nim. Un jeu à deux joueurs.
30 allumettes sont disposées sur une table au départ.
À tour de rôle, chaque joueur enlève 1, 2 ou 3 allumettes du tas.
Le joueur qui ne peut plus prendre d'allumettes a perdu (... ou gagné suivant la variante.)
Variante avec une complication :
on interdit à un joueur de prendre le même nombre d'allumettes que son prédécesseur.
Écrivez le code qui permette à deux joueurs de jouer à ce jeu.
Vous pouvez choisir la variante qui vous intéresse.

Il peut être intéressant de réfléchir à une stratégie gagnante, dans chaque variante.

Exercice 11.4 :
Jeu à un joueur : "**Devine mon nombre**"
L'ordinateur tire un nombre au hasard entre 10 et 99.
```
import random
nbCache = random.randint(10, 99)  #  Tire le nombre au hasard entre 10 et 99
```

Le joueur doit deviner le nombre tiré par l'ordinateur en faisant des essais. À chaque essai, le programme donne une réponse plus ou moins encourageante (glacé, froid, tiède, chaud, brûlant) en fonction de la valeur absolue de la différence entre le nombre donné par le joueur et celui tiré par l'ordinateur. Une fois le nombre de l'ordinateur trouvé, le nombre d'essais mis par le joueur doit-être affiché avec un petit commentaire.
Bonus : le joueur gagne un diamant de taille égale à son nombre d'essais !

```
    #
   ###
  #####
 #######
#########
 #######
  #####
   ###
    #
```

12. Les fonctions

Souvent on désire :
° réutiliser un bout de programme plusieurs fois ou
° isoler un bout de programme ou
° définir une fonction qui retourne un résultat en fonction de données.

Dans l'éditeur, créez un nouveau fichier et tapez :

```
# À vous d'écrire un commentaire pertinent ici.
# Tapez : help(livretDe7)  pour avoir de l'aide sur la fonction.

def livretDe7():
#===============
    ''' Affiche le livret de 7.'''
    nn=1
    while nn<11:
        print(nn*7,end=" ")
        nn+=1
    print()

print("----- Exercice 12.0a -----")
livretDe7()
```

Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex12000_fonctions.py**
Exécutez-le en pressant la touche **F5**.

Ajoutez le code suivant :

```
def livretDe(base):
#==================
    '''Affiche le livret du nombre base.'''
    n=1
    while n<11:
        print(n*base, end=" ")
        n+=1
    print()

print("----- Exercice 12.0b -----")
base = int(input("Quelle base ? "))
livretDe(base)
```

Paramètres facultatifs.

Ajoutez le code suivant :

```
def tableMulti(base, debut=2, fin=10):
#=====================================
    '''Affiche le livret du nombre base.
       debut et fin sont des paramètres facultatifs.'''
    n=debut
    while n<=fin:
        print(n*base, end=" ")
        n+=1
    print()

print("----- Exercice 12.0c -----")
base = int(input("Quelle base ? "))
tableMulti(base)
tableMulti(base, 5, 12)
```

Variables locales et variables globales.

Dans l'éditeur, créez un nouveau fichier et tapez :
 # À vous d'écrire un commentaire pertinent ici.

```
def Test_1():
#============
    '''Juste pour tester les variables locales et globales.'''
    print("Entrée dans 'Test_1()'")
    print("var1 =", var1)
    print("Sortie de 'Test_1()'")

print("----- Exercice 12.0d -----")
var1 = 11
print("var1 =", var1)
Test_1()

def Test_2():
#============
    '''Juste pour tester les variables locales et globales.'''
    print("Entrée dans 'Test_2()'")
    var1 = 22  # var1 devient une variable locale. On ne peut plus accéder à la
variable var1 globale.
    print("var1 =", var1)
    print("Sortie de 'Test_2()'")

print("----- Exercice 12.0e -----")
var1 = 11
print("var1 =", var1)
Test_2()
print("var1 =", var1)

def Test_3():
#============
    '''Juste pour tester les variables locales et globales.'''
    print("Entrée dans 'Test_3()'")
    global var1  # Pour pouvoir modifier une variable définie hors de la fonction
    print("var1 =", var1)
    var1 = 22  # Si la variable n'est pas déclarée global, cela génère une erreur.
    print("var1 =", var1)
    var2 = 33
    print("var2 =", var2)
    print("Sortie de 'Test_3()'")

print("----- Exercice 12.0f -----")
var1 = 11
print("var1 =", var1)
Test_3()
print("var1 =", var1) # La variable a été modifiée, car elle est déclarée "global"
#print("var2 =", var2) # génère une erreur, car var2 n'est pas défini hors de
```

Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex12001_variables_locales_globales.py**
Exécutez-le en pressant la touche **F5**.

Une variable définie dans une fonction, n'est visible qu'à l'intérieur de cette fonction.
On dit que cette variable est **locale** à la fonction.

Une variable définie dans le corps du programme, n'est visible à l'intérieur d'une fonction, uniquement si elle a été déclarée **global** dans la fonction. Dans ce cas, la variable est aussi modifiable dans la fonction.

Valeur retournée par une fonction

Dans l'éditeur, créez un nouveau fichier et tapez :
 # À vous d'écrire un commentaire pertinent ici.

```
def maFunc(x):
#=============
    '''Un exemple de fonction qui retourne une valeur.'''
    y = x**2 + 1
    return y

print("----- Exercice 12.02a -----")
y1 = maFunc(2)
print("f(2) =", y1)
y2= maFunc(2.5)
print("f(2.5) =", y2)
print("f(3) =", maFunc(3))
```

Sauvegardez votre fichier dans le dossier **adoc** sous le nom **ex12002_return.py**
Exécutez-le en pressant la touche **F5**.

Ajoutez le code suivant :

```
def SuiteJolie(strS):
#====================
    # ... À commenter ...
    strRep = ""
    cLast = " "
    cpt = 1
    for cc in strS:
        if (cLast == " "): # Cas du premier caractère
            cLast = cc
        elif (cc == cLast):
            cpt += 1
        else:
            # On change de caractère
            strRep += str(cpt) + cLast
            cLast = cc
            cpt = 1

    strRep += str(cpt) + cLast
    return strRep

strS = "1" ; print(strS)
for k in range(10):
    strS = SuiteJolie(strS);  print(strS)
```

Ici, l'utilisation de la fonction et de la chaîne de caractères retournée sert à clarifier le code.
Il n'est pas évident de comprendre ce que fait cette fonction.
Exécutez-la et essayez de comprendre la suite obtenue, elle est "logique".
Lisez-la à haute voix, cela peut aider à saisir la "logique" de la suite.

Exercice 12.1 :
Conversion d'un nombre entier positif en binaire.
Écrivez une fonction "dec2bin(nn)" qui pour un entier nn donné, retourne un nombre composé de 0 et de 1 qui correspond à la représentation binaire du nombre.
Vous pouvez aussi écrire la fonction bin2dec(nn) qui fait la conversion inverse, d'un nombre en base 2 vers un nombre en base 10.
Exemples : dec2bin(13) -> 1101 et bin2dec(1101) -> 13

Exercice 12.2 :
Test de divisibilité.
Écrivez une fonction "estDivisible(nn)" qui pour un entier nn donné, retourne True si le nombre nn est divisible par 2 ou 3 ou 5 ou 7 ou 11 ou 13, retourne False sinon.

Pour continuer des instructions à la ligne suivante, on peut terminer une ligne par un \

Testez votre code sur divers nombres, tels que 121, 91, 101 et 323.
À la suite, écrivez le code qui donne la liste des nombres premiers de 2 à 100.

Exercice 12.3 :
Teste si un nombre donné est premier.
Écrivez une fonction "estPremier(nn)" qui pour un entier nn donné, retourne True si le nombre nn est premier, retourne False sinon.

Testez votre code sur divers nombres, tels que 101, 121, 323 et 997.

Plus petit nombre premier plus grand qu'un nombre donné.
À la suite :
Écrivez une fonction "premierSuivant(nn)" qui pour un entier nn donné, retourne le premier nombre premier qui est plus grand que nn.
Exemple : premierSuivant(14) -> 17 ; premierSuivant(17) -> 19

Écrivez la liste des 10 nombres premiers qui suivent 10.
Écrivez la liste des 10 nombres premiers qui suivent 1000.

**Challenge : trouver une suite de 12 nombres consécutifs, non premiers.*

Avec de très nombreuses fonctions pré-définies, les **commentaires**, les **variables**, les **types de bases**, les **opérateurs**, les **boucles for** et **while**, le **test if** et les **fonctions** sont la base de tout langage de programmation.

Un type important qu'il reste à voir est la "**list**".
Un cas particulier de liste est le **tableau** ou "**array**" en anglais.