GILSON RAPHAEL : INFORMATIQUE -> PYTHON

Faire le code -> compiler (traduction pour que la machine le comprenne : exécuter instruction élémentaire).
La traduction se fait automatiquement et immédiatement dans python.
Python monte en puissance (mais Java reste le leader)
On manipule des variables.
var = 5
var est un nombre quelconque -> on crée une petite case dans la mémoire dans laquelle on a mis une étiquette var
! sensible aux majuscules et minuscules !

- Class 'int' : la variable est un entier (un entier se met sur 8,16 ou 32 bits) ici se met sur 32 bits
- Class 'float' : la variable est un réel
- Class 'str' : chaine de caractères

On a 2 cases mémoires : une avec étiquette 5 et l'autre avec étiquette 8
Var=5
Var2=8
Var=var2 (égalité au sens informatique du terme : on prend ce qu'il y avait dans la boite var2 et on le met dans la boite var) on a écrasé ce qu'il y avait dans le première
Var -> 8

Var2= var2 + 4
Raccourci : var2+=4

Var2/=2
Var2
8

Si on une chaine de caractères et qu'on veut qu'il entre un des caractères, on entre la position de la lettre
N.B. : le premier est toujours notice 0 et le dernier n-1

Prendre un élément

```
>>> var3='Bersini'
>>> var3[0]
'B'
>>> var3[6]
'i'
```

Expression du reste

```
>>> 5%2
1
```

Type de variable de type composite : on stock plusieurs choses dans un sac
2 types de composites : les listes et les dictionnaires
On peut mettre des nombres ou autre chose dans la liste

Les listes

Exemple 1

```
>>> liste =[1,2,3,4]
>>> liste
[1, 2, 3, 4]
>>> liste[1]
2
```

Exemple 2 : la position

```
>>> liste=[1,2.3,"tom","paul"]
>>> liste
[1, 2.3, 'tom', 'paul']
>>> liste [2]
'tom'
>>> liste [2] [0]
't'
```

(2ème position de la liste)

(1ère lettre de tom)

Remarque : Ne fonctionne pas : tom+3=5
(dans d'autres programmes, compilateur traduit et connaît la syntaxe des langages et avertit que ca se plante lors de la traduction et non lors de l'exécution -> langage compilé-> utilisé pour des projets qui requièrent plus de robustesse et de fiabilité)
(en python il faut exécuter le programme pour qu'il se plante)

Exemple 3 : on rajoute un élément à la liste

On peut rajouter un mot ou insérer un mot dans la liste, on peut aussi sélectionner une partie de la liste.

```
>>> liste
[1, 2.3, 'tom', 'paul']
>>> liste.append ("Eric")
>>> liste
[1, 2.3, 'tom', 'paul', 'Eric']
>>> liste.insert(4,"Jean")
>>> liste
[1, 2.3, 'tom', 'paul', 'Jean', 'Eric']
```

(on insère Eric)

(on insère Jean à la position 4)

Exemple 4 : on sélectionne une partie de la liste

```
>>> tom="Solvay"
>>> tom
'Solvay'
>>> tom[0:2]
'So'
>>> liste[1:4]
[2.3, 'tom', 'paul']
>>> liste[1:]
[2.3, 'tom', 'paul', 'Jean', 'Eric']
>>> liste=["Eric","Jean","Paul","Sylvie","Florence"]
```

Exemple 5 : on sélectionne un élément de la liste

```
>>> liste
['Eric', 'Jean', 'Paul', 'Sylvie', 'Florence', 'Marie']
>>> liste[3]
'Sylvie'
```

Les dictionnaires

Dans le dico l'ordre n'a aucune importance, alors que dans la liste c'est l'ordre qui permet de retrouver un élément dans la liste.

Exemple 1

```
>>> dico={"Eric":10,"Jean":15,"Sylvie":18,"Paul":5}
>>> dico[0]
Traceback (most recent call last):
  File "<pyshell#77>", line 1, in <module>
    dico[0]
KeyError: 0
>>> dico["Eric"]
10
```

Exemple 2

```
>>> trad={"mouse":"souris","software":"logiciel","geek":"geek"}
>>> trad["mouse"]
```
(fonctionne dans ce sens là)
```
'souris'
>>> trad["souris"]
Traceback (most recent call last):
  File "<pyshell#81>", line 1, in <module>
    trad["souris"]
KeyError: 'souris'
```
(ne fonctionne pas)

Exemple 3 : numéro de téléphone

```
>>> tel={"Jean":["21-20","30-10"],"Marc":["10-10","25-54"]}
>>> tel["Jean"]
['21-20', '30-10']
>>> tel["Jean"][0]
'21-20'
```

Exemple 4 : Différence entre a qui est un nombre et b qui est une chaine de caractère

```
>>> a=1   (nombre)
>>> b="1"  (chaîne de caractères)
>>> a+=    (pas de sens)
SyntaxError: invalid syntax
>>> a+=1   (a du sens)
>>> b+=1  (pas de sens)
Traceback (most recent call last):
  File "<pyshell#89>", line 1, in <module>
    b+=1 (pas de sens)
TypeError: Can't convert 'int' object to str implicitly
```

Ce qui fait qu'un programme est intelligent c'est qu'il exécute des instructions.
Il y a 2 grandes familles d'instructions : les tests
Exécute les instructions l'une à la suite des autres.

On peut regrouper des instructions
On entre ca dans une nouvelle fenêtre : run -> save -> il exécute

```
a=5
toto="Bersini"
print(a)
print(toto)
```

IF/ELSE

On encode une valeur, on pose condition et puis on voit le résultat.
If (=si) est une condition booléenne. Ce qui suit le if est une condition qui ne peut être que vrai ou fausse.
Les deux points indiquent que c'est la fin de la condition
On passe à la ligne et on décale vers la droite (indentation).
Python va exécuter tout ce qui est aligné si la condition est vraie.
Else c'est si la condition est fausse : (esle=sinon)

Exemple 1

```
>>> a=5
>>> if a<10:
        print ("OK")
else:
        print ("pas OK")


OK
```

Exemple 2

```
>>> if a<5:
        print ("OK")
else:
        print ("pas OK")
        a=10


pas OK
```

Exemple 3

```
>>> a
10
>>> a=8
>>> if a<5:
        print ("OK")
else:
        print ("pas OK")
        a=10


pas OK
>>> a
10
```

Exemple 4

```
>>> a
10
>>> if a<5:
        print ("OK")
        print ("toto")
else:
        print ("pas OK")
        print ("titi")
        a=10


pas OK
titi
>>> a
10
```

Exemple 5

```
>>> a=3
if a<5:
    print ("OK")
    print ("toto")
else:
    print ("pas OK")
    a=8 (inutile)
print("pas OK") (n'est pas dans les conditions donc n'apparaît as d'office)
a=8
if a<5:
    print ("OK")
    print ("toto")
else:
    print("pas OK")
    a=8
```

Exemple 6

```
>>>
OK
toto
pas OK
pas OK
```

N.B. : == ca veut dire test d'égalité

IF dans IF

Exemple 1

```
>>> a=2
>>> if a >=2:
        print ("OK")
        if a > 0:
                print ("aussi")
                a=4
        else:
                print ("non")
        if a == 2:
                print ("c'est vrai")


OK
aussi
```

Exemple 2

```
>>> >>> a=2
>>> if a >=2:
        print ("OK")
        if a > 0:
                print ("aussi")
                if a == 2:
                        print ("c'est vrai")
                a=4
                print ("ici")
        else:
                print ("non")
```

COURS 2

Boucles : si l'ordi fait quelque chose d'intelligent, il fait des boucles.
Exemple de boucle :

Programme qui joue aux échecs : il boucle sur un tas de coups possibles et puis boucles sur toutes les réponses possibles de l'adversaire (14 niveau de profondeurs) -> évaluent plein de coups possibles avant de se décider.

C'est les boucles qui font les mathématiques.

Réalisation d'une boucle (2 manières)

1ère méthode (while : tant que)

Il faut une condition d'arrêt pour la boucle, la condition d'arrêt est ce qu'il y a entre ().

Exemple 1

```
>>> i=0
>>> while(i<5):
        print ("hello")
        i+=1
```
(permet d'empêcher qu'il répète hello à l'infini, c'est la condition d'arrêt)

```
hello
hello
hello
hello
hello
```

Exemple 2

```
>>> i=0
>>> while(i<5):
        print (i)
        i+=1
```
(attention : strictement inférieur)

```
0
1
2
3
4
```

Exemple 3 : inférieur ou égal

```
>>> i=0
>>> while(i<=5):
        print(i)
        i+=1
```
(attention à l'indentation de i)

```
0
1
2
3
4
5
```

Exemple 4

```
>>> i=0
>>> while(i<=5):
        print("solvay")
        i+=2


solvay
solvay
solvay
```

Exemple 5 : autre boucle (for)

```
>>> for i in range (0,3)
SyntaxError: invalid syntax
>>> for i in range (0,3):
        print("Solvay")


Solvay
Solvay
Solvay
```

Exemple 6 : for

On utilise le for pour les listes ou les dico généralement.

```
>>> liste=["Bob","Jack","Jim","Louis"]
>>> for x in liste:
        print(x)


Bob
Jack
Jim
Louis
```

Exemple 7 : moyenne

Attention len (liste) nous donne la longueur de la liste

Moy+=x permet de sommer les éléments de la liste

```
>>> liste=[10,15,16,20,0,5,18]
>>> moy=0 (on initialise la variable)
>>> len(liste) (on prend la longueur de la liste)
7
>>> for x in liste:
        moy+=x
```
(x est une variable qui va prendre successivement tous les éléments de la liste)

(on somme les éléments)

```
>>> moy
84
>>> moy/=len(liste) (on divise les éléments de la liste par la longueur de la liste)
>>> moy
12.0
```

Exemple 8 : dico

```
>>> moy=0
>>> dico={}
>>> dico ["Bob"] =0
>>> dico["jean"]=10
>>> dico["Sylvie"]=15
>>> dico["PAul"]=7
>>>
>>> dico
{'Bob': 0, 'jean': 10, 'PAul': 7, 'Sylvie': 15}
>>> nr = 0
>>> moy =0
>>> for x,y in dico.items():
        moy+=y
        if y <10:
                nr+=1


>>> nr
2
>>> moy/=len(dico)
>>> moy
8.0
```

Exemple 9

```
>>> liste
[10, 15, 16, 20, 0, 5, 18]
>>> i=0
>>> while (i<len(liste)):
        print (liste [i])
        i+=1


10
15
16
20
0
5
18
```

UTILISER UN FOR QUAND LISTE

Si on cherche un élément précis de la liste, on utilise les crochets : liste [0]

```
>>> liste[0]
10
```

2 boucles : on boucle sur les lettres et sur les nombres -> on obtient toutes les lettres.

```
>>> liste=["Jean","Paul","Anne","Gil"]
>>> for x in liste:
        for y in x:
                print(y)


J
e
a
n
P
a
u
l
A
n
n
e
G
i
l
```

DING DING BOTTLE

On veut faire apparaître les nombres, et quand multiple de 5 il faut que ca mette ding ding (et pas le nombre).

On va faire un while et aller jusqu'à 100

Le modulo = le reste de la division

Le modulo c'est le %

Attention ! mettre au dessus la condition restrictive !!!

```
>>> x=0
>>> while (x<100):
        if(x%5 == 0)and (x%7 == 0):
                print ("ding ding bottle")
        elif (x%7 == 0):
                print ("bottle")
        elif (x%5 == 0):
                print ("ding ding")
        else:
                print (x)
        x+=1
```

Fonctions :

```
>>> def f1 (x):
        return x*x*x

>>> f1 (3)
27
>>> def f2(x,y):
        return x*x*x + y*y

>>> f2 (3,2)
31

>>> def toto(titi):
        return titi*titi*titi

>>> toto (3)
27
```

```
>>> x=int()
>>> x
0
>>> def cube (x):
        return x*x*x

>>> cube (3)
27
>>> def cube (x):
        x*x*x


>>> cube (3)
>>>
```

Ne donne pas de valeur si on ne met pas de return.
Le retour de la fonction est très important.
On fait un cube.

```
>>> def cube (x):
        return x*x*x

>>> for x in liste:
        cube (x)


1
27
1000
3375
8000
1000000
```

On fait une factorielle. -> attention on prend -1

```
>>> def fact(n):
        if n==0:
                return 1
        else:
                return n*fact(n-1)

>>> fact(4)
24
```

On veut lire tous les mots à l'envers.

x c'est l'argument
on met « nom » car on veut inverser.

```
>>> def inverse (x):
        nom=""
        i=len(x)-1
        while (i>=0):
                nom+=x[i]
                i-=1
        return nom

>>> inverse("Bersini")
'inisreB'
```

Autre opération, on inverse tous les noms de la liste

```
>>> inverse(liste)
'zoeannejacquesJean'
>>> inverse(liste[2])
'enna'
>>> inverse(liste[1])
'seuqcaj'
>>> for x in liste :
        inverse(x)


'naeJ'
'seuqcaj'
'enna'
'eoz'
>>> liste
['Jean', 'jacques', 'anne', 'zoe']
```

x prend toutes les valeurs de la liste : Jean puis jacques etc

```
>>> liste
['Jean', 'jacques', 'anne', 'zoe']
>>> i=0
>>> for x in liste:
        liste [i]=inverse (x)
        i+=1


>>> liste
['naeJ', 'seuqcaj', 'enna', 'eoz']
```

On peut refaire cette opération et on retrouve la liste d'origine

```
>>> i=0
>>> for x in liste:
        liste [i]=inverse (x)
        i+=1


>>> liste
['Jean', 'jacques', 'anne', 'zoe']
```

Autre opération

```
>>> liste
['Jean', 'jacques', 'anne', 'zoe']
>>> i=0
>>> while(i<len(liste)):
        liste[i]=inverse(liste[i])
        i+=1


>>> liste
['naeJ', 'seuqcaj', 'enna', 'eoz']
```

Autre opération avec le return

```
>>> def (liste):

SyntaxError: invalid syntax
>>> def invListe(liste):
        while(i<len(liste)):
                liste[i]=inverse(liste[i])
                i+=1
        return liste

>>> liste
['naeJ', 'seuqcaj', 'enna', 'eoz']
```

On les remet dans l'ordre

```
>>> i=0
>>> while(i<len(liste)):
              liste[i]=inverse(liste[i])
              i+=1


>>> liste
['Jean', 'jacques', 'anne', 'zoe']
```

On prend une fonction simple
Attention la fonction travaille avec des variables locales.
La fonction va traiter localement le x.
Quand la fonction se termine elle se débarrasse de toutes les variables locales.

```
>>> def test (x):
        x+=1
        print (x)


>>> x=5
>>> test(5)
6
>>> x
5
```

Autre opération : y n'a pas de valeur (car pas de return)

```
>>> def text(x):
        x+=1
        print(x)


>>> u=5
>>> y=text(u)
6
>>> y
>>>
```

Y a une valeur si on met un return

```
>>> def text(x):
        x+=1
        return x

>>> u=5
>>> y=text(u)
>>> y
6
```

Print random -> nous renvoie à un nombre aléatoire.
Mais il faut faire un import

```
>>> import random
>>> print(random.randint(1,10))
9
```

Liste randomisée

```
>>> liste=["a","b","c","d","e"]
>>> random.shuffle(liste)
>>> liste
['c', 'b', 'e', 'a', 'd']
```

CLOCK

```
>>> from time import clock
>>> clock ()
74.094003
```