2 - Contrôle du flux d'instructions

1. Les instructions simples et les séquences d'instructions
2. Les instructions conditionnelles
3. Les instructions de répétition
4. Les instructions imbriquées

2.1 - Instructions simples et séquences d'instructions

- Une instruction simple ou élémentaire est une action de base définie par le langage Python. Ce peut être une **évaluation d'expression**, une **affectation** ou une **instruction qui a une fonction très spécifique.**

- Une **évaluation d'expression** s'exprime par l'écriture de l'expression à évaluer. La valeur de l'expression est calculée et affichée sur la sortie standard.

2.1 - Instructions simples et séquences d'instructions

- Une **affectation** a la forme : *cible* = *expression* . La cible est souvent une variable mais ce peut être un objet plus complexe capable de recevoir et de stocker une valeur, tel par exemple une entrée d'un tableau. L'opérateur d'affectation = peut être précédé d'un opérateur permettant d'exprimer la nouvelle valeur d'une zone mémoire à partir de l'ancienne.

- La façon la plus simple de composer des instructions est de les organiser en **séquences**. Dans une séquence, deux instructions sont séparées par un point-virgule ou un passage à la ligne.

2.1 - Instructions simples et séquences d'instructions

```
>>> 2+3
5

>>> "la" == "le"
False

>>> x = 2 + 3
>>> x
5

>>> x *=2
>>> x
10

>>> y = [1, 2]
>>> y[0] = 7
>>> y
[7, 2]

>>> x = 7; x
7
```

2.2 - Instructions conditionnelles

- Une **instruction conditionnelle** permet d'effectuer un choix dans l'exécution d'instructions en fonction de **conditions**.

- La plus simple des instructions conditionnelles est l'instruction **if** qui a la syntaxe suivante :

  if *<condition> :*

  <bloc d'instructions>

- *<condition>* est une expression de type booléen et *<bloc d'instructions>* est une séquence d'instructions qui sont indentées de la même façon par rapport à la ligne où figure la condition.

- La sémantique de l'instruction est la suivante : si la valeur de *<condition>* est *True* , la séquence *<bloc d'instructions>* est exécutée, sinon aucune action n'est effectuée.

2.2 - Instructions conditionnelles

```
>>> x=0
>>> if x >=0:
        x
        x+1
        x-1

0
1
-1

>>> x= -1
>>> if x >=0:
        x
        x+1
        x-1
>>>


>>>
```

2.2 - Instructions conditionnelles

- L'instruction **if … else** a la syntaxe suivante :

  if *<condition> :*

  <bloc1 d'instructions>

  else *:*

  <bloc2 d'instructions>

- La sémantique de l'instruction est la suivante : si la valeur de *<condition>* est *True* , la séquence *<bloc1 d'instructions>* est exécutée, sinon c'est *<bloc2 d'instructions>* qui est exécutée.

2.2 - Instructions conditionnelles

```
>>> x = 7
>>> if x % 2 == 0 :
        print "x est pair"
else:
        print "x est impair"

x est impair

>>> x = 8
>>> if x % 2 == 0 :
        print "x est pair"
else:
        print "x est impair"

x est pair
>>>
```

2.2 - Instructions conditionnelles

- On peut introduire un test sur une suite de conditions alternatives en utilisant le mot-clé **elif** dans l'instruction **if … else** avec la syntaxe suivante :

```
if <condition 1> :
    <bloc 1 d'instructions>
elif <condition 2>:
    <bloc 2 d'instructions>
        .
        .
        .
elif <condition n>:
    <bloc n d'instructions>
else :
    <bloc n+1 d'instructions>
```

- La sémantique de l'instruction est la suivante : si la valeur de *<condition 1>* est *True* , la séquence *<bloc 1 d'instructions>* est exécutée, sinon si la valeur de *<condition 2>* est *True* , la séquence *<bloc 2 d'instructions>* et ainsi de suite. Si aucune des conditions de la suite n'est vraie, c'est *<bloc n+1 d'instructions>* qui est exécutée.

2.2 - Instructions conditionnelles

```
>>> heure = 15

>>> if heure > 6 and heure <= 13 :
        print "c'est le matin"
elif heure > 13 and heure <= 19 :
        print "c'est l'après-midi"
elif heure > 19 and heure <= 23 :
        print "c'est le soir"
else:
        print "c'est la nuit"


c'est l'après-midi
>>>
```

2.3 - Instructions de répétition

- Les instructions de répétition permettent de répéter un bloc d'instructions.

- L'instruction **for** a la syntaxe suivante :

  for *<controleur> in <seq>:*

  <bloc d'instructions>

- Dans cette instruction, *<controleur>* est une variable qui est introduite pour contrôler le nombre d'itérations et *<seq>* est un objet de type *sequence (string, list, tuple).*

- La sémantique de l'instruction est la suivante : la variable *<controleur>* prend successivement les différentes valeurs de *<seq>* dans l'ordre où elle apparaissent. A chaque instanciation, la séquence *<bloc d'instructions>* est exécutée.

2.3 - Instructions de répétition

```
>>> range(5)
[0, 1, 2, 3, 4]

>>> for n in range(5) :
    print "le carre de ", n, "est : ", n**2


le carre de  0 est :  0
le carre de  1 est :  1
le carre de  2 est :  4
le carre de  3 est :  9
le carre de  4 est :  16
>>>
```

2.3 - Instructions de répétition

- L'instruction **while** a la syntaxe suivante :

  while *<condition> :*

  <bloc d'instructions>

- Dans cette instruction, *<condition>* est une expression de type booléen.

- La sémantique de l'instruction est la suivante : l'expression *<condition>* est évaluée et tant que sa valeur est True, la séquence *<bloc d'instructions>* est exécutée.

- On utilise une instruction for quand les itérations sont connues à l'avance et on utilise une instruction while quand les itérations ne sont pas connues à l'avance mais qu'elles dépendent d'une condition.

2.3 - Instructions de répétition

```
>>> n = 0
>>> while n <= 5 :
    print "le carre de ", n, "est : ", n**2
    n += 1


le carre de  0 est :  0
le carre de  1 est :  1
le carre de  2 est :  4
le carre de  3 est :  9
le carre de  4 est :  16
le carre de  5 est :  25
>>>
```

2.3 - Instructions de répétition

- L'instruction **break** doit être utilisée dans le bloc d'instructions d'une boucle. Elle interrompt l'exécution de la boucle la plus interne dans laquelle elle se situe.

```
>>> for k in range(6) :
            for i in range(k+1) :
                if i > 1:
                    if i == k :
                        print k
                    elif  k % i == 0:
                        break


2
3
5
```

2.4 - Instructions imbriquées

- Les instructions composées peuvent être imbriquées les unes dans les autres

```
>>> entree = "La belle brise la glace."
>>> sortie = ""
>>> for caractere in entree :
    if caractere == " ":
        sortie += "\n"
    else :
        sortie += caractere


>>> print sortie
La
belle
brise
la
glace.
>>>
```

2.5 Exercices

1. Donner le résultat de l'exécution du programme ci-dessous lorsque l'on entre au clavier les valeurs 4, 2, 5 pour respectivement a, b, c. Même question lorsque l'on entre 2, 4, 5. En déduire le rôle de ce programme.

```
a = input("nombre1 : ")
b = input("nombre2 : ")
c = input("nombre3 : ")
if a >b :
    if a > c :
        print a
    else :
        print c
elif b >c :
    print b
else :
    print c
```

2.5 Exercices

2. Donner le résultat de l'exécution du programme ci-dessous lorsque l'on entre au clavier la valeur "arma" pour *mot1*. Même question lorsque l'on entre "erre". En déduire le rôle de ce programme.

```
mot1 = input("Entrez un mot !")
mot2 = ""
for c in mot1 :
    mot2 = c + mot2
print  mot1 == mot2
```

3. Donner le résultat de l'exécution du programme ci-dessous lorsque l'on entre au clavier la valeur 5 pour x. Même question lorsque l'on entre 6. En déduire le rôle de ce programme.

```
x = input("Entrez un nombre entier naturel ! ")
y =2
continuer = True
while y < x and continuer :
    z = y
    while z < x and continuer :
        z += y
        if z == x :
            continuer = False
    y += 1
print  continuer
```

2.5 Exercices

4. Ecrire un programme Python pour chacune des spécifications ci-dessous qui utilise un minimum de fonctions prédéfinies.

a) Afficher tous les caractères d'une chaîne de caractères en en mettant un par ligne.

b) Calculer la longueur d'une chaîne de caractères (parcourir la chaîne et incrémenter une variable de 1 à chaque pas).

c) Compter le nombre de « e », de « a » et de « i » dans une chaîne de caractères.

d) Afficher tous les mots d'une phrase en en mettant un par ligne (deux mots sont supposés être séparés exactement par un espace).