CHAPITRE 5
TESTS CONDITIONNELS ET INSTRUCTIONS DE RÉPÉTITION

Programmation en Python

Test conditionnel « If-Elif-Else »

- Un test conditionnel est un bloc d'instructions qui implémente le fait de dire :
  - si une condition booléenne est vraie, alors un certain bloc d'instructions sera exécuté ;
  - autrement, si une autre condition booléenne est vraie, alors un autre bloc d'instructions sera exécuté ;
  - etc. ;
  - sinon (i.e., si toutes les conditions booléennes ci-dessus sont fausses), alors un autre bloc d'instruction sera exécuté.

Test conditionnel « If-Elif-Else »

- En Python, la syntaxe d'un test conditionnel est la suivante :

```
if condition_1:
    bloc_istructiions_1
elif condition_2:
    bloc_instructions_2
...
elif condition_n:
    bloc_instructions_n
else:
    bloc_instructions_finales
```

Test conditionnel « If-Elif-Else »

- On peut mettre autant de conditions « elif » que désiré (et même aucunes).
- De même, la dernière condition « else » est facultative. Si elle est omise, Python l'interprète comme un « sinon, rien ne se passe ».
- Ne pas oublier les « : » à la fin des instructions « if », « elif » et « else », et les indentations des blocs d'instructions qui suivent ces instructions.

Test conditionnel « If-Elif-Else »


```
>>> a = -1
>>> if a < 0:
        print "a est négatif"

a est négatif
>>>
>>> a = 1
>>> if a < 0:
        print "a est négatif"

>>>
```

Test conditionnel « If-Elif-Else »


```
>>> a = -3
>>> if a < 0:
        print "a est négatif"
    else:
        print "a est positif ou nul"

a est négatif
>>> a = 2
>>> if a < 0:
        print "a est négatif"
    else:
        print "a est positif ou nul"

a est positif ou nul
```

Test conditionnel « If-Elif-Else »


```
>>> a = 5
>>> if a < 0:
        print "a est negatif"
    elif a == 0:
        print "a est nul"
    else:
        print "a est positif"

a est positif
>>>
```

Test conditionnel « If-Elif-Else »


```
>>> a = 7
>>> if a < 0:
        print "a est negatif"
    elif (a >= 0) and (a <=5):
        print "a est compris entre 0 et 5"
    elif (a >= 6) and (a <=10):
        print "a est compris entre 6 et 10"
    else:
        print "a est supérieur à 10"

a est compris entre 6 et 10
>>>
```

Programme (faire démo)

- On implémente un programme qui résout les équations du second degré (déjà vu au chapitre 3).
- On calcule les solution en fonction du signe du discriminent.

Programme (faire démo)


```
from math import * # importation de la librairie math

print "Soit l'équation du second degré a*x^2 + b*x + c:"

a = input("Entrer le coefficient a: ")
b = input("Entrer le coefficient b: ")
c = input("Entrer le coefficient c: ")

delta = b**2 - 4*a*c
```

Programme (faire démo)


```
if delta < 0:
        print "Les solutions sont complexes:"
        # création du nombre complexe aux = sqrt(delta)
        aux = 0 + (sqrt(-delta))*1j
        print (-b + aux) / (2*a) # 1ère solution complexe
        print (-b - aux) / (2*a) # 2ème solution complexe
elif delta == 0:
        print "Une seule solution:"
        print -b / (2*a)
else:
        print "Deux solutions:"
        print (-b + sqrt(delta)) / (2*a)
        print (-b - sqrt(delta)) / (2*a)
```

Boucle « for »

- En programmation, une boucle « for » est une instruction de répétition qui ré-exécute un même bloc d'instructions autant de fois qu'une variable met de temps pour s'incrémenter d'une valeur initiale A jusqu'à une valeur finale B.
- « For i = A to B, do... » se lit « pour i allant de A jusqu'à B, faire... ».
- En Python, la boucle « for » s'exécute généralement de paire avec la fonction `range()` (cf. chapitre 3).

Boucle « for »

- La fonction `range(start,stop,[step])` produit un objet qui retourne une liste de valeurs telle que :
  - la première valeur est start
  - les valeurs suivantes sont incrémentées par pas de step
  - et ce jusqu'à atteindre la valeur stop non incluse
  - si start est omis, Python considère que start = 0
  - si step est omis, Python considère que step = 1
- Par exemple `range(3,17,2)` représente la liste `[3,5,7,9,11,13,15]`.

Boucle « for »

- La syntaxe d'une boucle « for » est la suivante :

```
for var in range(start,stop[,step]):
    instructions...
```

  - Dans ce cas, la variable « var » prend comme valeurs les éléments successifs de l'objet « range(start,stop[,step]) »
  - le bloc d'instructions « instructions... » est ré-exécuté autant de fois que la variable « var » met de temps pour parcourir l'objet « range(start,stop[,step]) ».
- On dira : « pour var allant de start à stop par pas de step, faire... ».

Boucle « for »


```
>>> for i in range(1000):
        print i


0
1
2
3
4
5
6
...
```

Boucle « for »


```
>>> for i in range(7,1000):
        print i


7
8
9
10
11
12
13
...
```

Boucle « for »


```
>>> for i in range(7,1000,3):
        print i


7
10
13
16
19
22
25
...
```

Boucle « for »

- On peut également utiliser des boucles for pour parcourir des chaînes, des listes, des tuples et des dictionnaires.

Boucle « for » sur les chaînes

- La syntaxe d'une boucle « for » pour parcourir une chaîne de caractère est :

```
for var in chaine:
    instructions...
```

  - Dans ce cas, la variable « var » prend comme valeurs les caractères successifs de la chaîne « chaîne »
  - le bloc d'instructions « instructions... » est ré-exécuté autant de fois que la variable « var » met de temps pour parcourir l'objet « chaîne ».

Boucle « for » sur les chaînes

Programme

```
voyelles = "aeiouy"
for x in "il pleut":
    if x in voyelles:
        print x + " est une voyelle"
    elif x == " ":
        pass
    else:
        print x + " est une consonne"
```

Boucle « for » sur les chaînes

Exécution

```
i est une voyelle
l est une consonne
p est une consonne
l est une consonne
e est une voyelle
u est une voyelle
t est une consonne
```

Boucle « for » sur les listes ou tuples

- La syntaxe d'une boucle « for » pour parcourir une liste ou un tuple est :

```
for var in liste (ou tuple):
    instructions...
```

  - Dans ce cas, la variable « var » prend comme valeurs des éléments successifs de la liste « liste »
  - le bloc d'instructions « instructions... » est ré-exécuté autant de fois que la variable « var » met de temps pour parcourir l'objet « liste ».

Boucle « for » sur les listes ou tuples

Programme

```
# importation de la librairie random
from random import *

# la fonction randint(a,b) génère un entier
# tiré aléatoirement entre a et b compris
l = []
for i in range(1000): # boucle for sur objet range()
    l.append(randint(0,10))

somme = 0
for i in l # boucle for sur objet liste
    somme += i
print somme
```

Boucle « for » sur les listes ou tuples

Exécution (le programme génère une liste de 1000 entiers tirés aléatoirement entre 0 et 10 et calcule leur somme)

```
4955
```

Boucle « for » sur les listes ou tuples


```
>>> for word in ('cool', 'powerful', 'readable'):
        print 'Python is ' + word

Python is cool
Python is powerful
Python is readable
```

Boucle « for » sur les dictionnaires

- Dans le cas des dictionnaires, on peut utiliser des boucles « for » pour parcourir les clés, les valeurs, ainsi que les couples clé-valeur du dictionnaire.

Boucle « for » sur les dictionnaires

- La syntaxe d'une boucle « for » pour parcourir les clés d'un dictionnaire est :

```
for var in dico.keys():
    instructions...
```

  - Dans ce cas, la variable « var » prend comme valeurs les clés successives du dictionnaire « dico »
  - le bloc d'instructions « instructions... » est ré-exécuté autant de fois que la variable « var » met de temps pour parcourir toutes les clé de l'objet « dico ».

Boucle « for » sur les dictionnaires

- La syntaxe d'une boucle « for » pour parcourir les valeurs d'un dictionnaire est :

```
for var in dico.values():
    instructions...
```

  - Dans ce cas, la variable « var » prend comme valeurs les valeurs successives du dictionnaire « dico »
  - le bloc d'instructions « instructions... » est ré-exécuté autant de fois que la variable « var » met de temps pour parcourir toutes les valeurs de l'objet « dico ».

Boucle « for » sur les dictionnaires

- La syntaxe d'une boucle « for » pour parcourir les couples clé-valeur d'un dictionnaire est :

```
for var in dico.items():
    instructions...
```

  - Dans ce cas, la variable « var » prend comme valeurs les couples clé-valeur successifs du dictionnaire « dico », représentés comme des tuples à deux éléments
  - le bloc d'instructions « instructions... » est ré-exécuté autant de fois que la variable « var » met de temps pour parcourir toutes les couples clé-valeur de l'objet « dico ».

Boucle « for » sur les dictionnaires

Programme

```
from random import *
d = {}
for i in range(1000): # boucle for sur objet range
    d[i] = randint(0,10)
for k in d.keys(): # boucle for sur les clés du dico
    print k
for k in d.values(): # boucle for sur les valeurs du dico
    print k
# boucle for sur les couples clé-valeur du dico
for k in d.items():
    print k
```

Boucle « for » sur les dictionnaires

Exécution

```
0
1
2
...
8
2
0
...
(0,8)
(1,2)
(2,0)
...
```

Programme (faire démo)

- On fait un programme qui génère le cours d'une action qui part du niveau 80 et fluctue aléatoirement de +/- 2 durant 1000 pas de temps.
- Puis, on implémente une stratégie automatique d'achat et de vente en fonction de la fluctuation de ce cours.
  - si le cours passe en-dessous de 75, on achète
  - se le cours dépasse les 85, on vend

Programme (faire démo)


```
from random import * # importation de la librairie random

# on génère le cours d'une action de 1000 valeurs
# le cours varie de +/- 2 à chaque pas de temps

# on initialise le 1er élément de la liste
cours = [80]
# on crée les 999 éléments restants de la liste
for i in range(999):
    cours.append( cours[-1] + randint(-2,2) )
```

Programme (faire démo)


```
# on implémente une stratégie d'achat et de de vente
# on suppose qu'on a acheté les actions au départ
ordre = "achat"
for i in range(1000):
    # si l'action monte en-dessus de 85, on vend
    # (pour autant qu'on avait acheté auparavant)
    if (cours[i] >= 85 and ordre == "achat"):
        print "Vendre au temps", i
        ordre =  "vente"
    # si l'action baisse en-dessous de 75, on achète
    # (pour autant qu'on avait vendu auparavant)
    elif (cours[i] <= 75 and ordre == "vente"):
        print "Acheter au temps", i
        ordre = "achat"
```

Boucle « while »

- En programmation, une boucle « while » est une instruction de répétition qui ré-exécute un même bloc d'instructions tant qu'une condition booléenne est vraie.
- Autrement dit, la boucle « while » continue d'être exécutée tant que la condition booléenne reste vraie ; elle cesse donc d'être exécutée dès que la condition booléenne devient fausse.
- « While condition, do... » se lit « tant que la condition booléenne est vraie, faire... ».

Boucle « while »

- La syntaxe d'une boucle « while » est la suivante :

```
while condition:
    instructions...
```

  - le bloc d'instructions « instructions... » est ré-exécuté tant que la condition booléenne « condition » est vraie, i.e., a prend valeur `True`.
- On dira : « tant que la condition est vraie, faire... ».
- On peut également forcer l'interruption d'une boucle « while » avec l'instruction `break`.

Boucle « while »


```
>>> l = []
>>> x = 0
>>> while (x**2 <= 1000):
        l.append(x**2)
        x += 1


>>> print l
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169,
196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576,
625, 676, 729, 784, 841, 900, 961]
```

Programme (faire démo)

- On crée un programme qui permet de jouer à « pile ou face » contre l'ordinateur.
- L'ordinateur génère aléatoirement un jet de pièce.
- Tant qu'on trouve si c'est pile ou face, on peut rejouer...

Programme (faire démo)


```
# importation de la librairie random
from random import *
score = 0
reponse = "gagné"
while reponse == "gagné":
    x = randint(0,1)
    if x == 0:
        x = "pile"
    else:
        x = "face"
    entree = raw_input("pile ou face? ")
```

Programme (faire démo)


```
if entree == x:
    print "vous avez gagné"
    print "vous pouvez rejouer"
    score += 1 # incrémente le score
else:
    print "vous avez perdu"
    reponse = "perdu" # interrompt la boucle while
    print "votre score est:", score
```

Programme (faire démo)

- On crée un programme qui détermine si un nombre entré par l'utilisateur est premier ou non.
- On rappelle qu'un nombre premier est un nombre qui n'est divisible que par un et par lui-même.
- Ainsi, pour déterminer si n est premier, on teste si n possède d'autre diviseurs que 1 et lui-même.

Programme (faire démo)


```
# pour un nombre entier n entré par l'utilisateur,
# ce programme détermine si n est premier ou non

# importation de la librairie math
from math import *

n = input("Entrer un nombre entier positif: ")
# 0, 1 et 2 sont premiers
if (n == 0 or n == 1 or n == 2):
    premier = True
```

Programme (faire démo)


```
else:
    # au début, si n impair, premier = True
    # si n pair, premier = False
    premier = ((n % 2) == 1)
    diviseur = 3
    sqrtN = sqrt(n)
    # augmenter le diviseur tant qu'il est premier
    while (diviseur <= sqrtN and premier == True):
        premier = ((n % diviseur) != 0)
        # on incrémente le diviseur de 2 en 2
        # pour éviter les diviseurs pairs, déjà testé
        diviseur = diviseur + 2
```

Programme (faire démo)


```
if premier == True:
    print "Votre nombre est premier"
else:
    print "Votre nombre n'est pas premier"
```