

	Informatique	Programmation	
	Python Structures de contrôle		

1. Utiliser l'interpréteur ou éditer un script ?

Pour un programme composé d'une simple séquence d'instructions, le terminal peut suffire.

Pour les programmes plus complexes utilisant des structures de contrôle, il est préférable d'avoir une vision d'ensemble du programme sous la forme d'un script.

2. Le bloc d'instruction dans la structure de contrôle

Une structure de contrôle (Si, Tant que,...) contient un bloc d'instruction.

Dans certains langages de programmation, le bloc d'instruction est délimité par :

- Des accolades dans les langages C, C++, C#,... ;
- Les mots-clés begin et end dans les langages Pascal et Delphi.

Dans le langage Python, c'est l'indentation qui permet d'identifier le bloc d'instruction.

L'indentation est le décalage du texte vers la droite.

Pour un bloc d'instruction, toutes les lignes doivent être parfaitement alignées !

3. L'instruction conditionnelle if

L'instruction qui s'écrit **Si** en pseudo-code se traduit par **if**

```
if x<0:
    x=0
    print("x doit être positif ou nul")
```

Caractère deux points obligatoire !

Indentation

« Sinon » se traduit par **else :** (avec le caractère deux points).

```
if x%2 == 0 :
    print("x est pair")
else:
    print("x est impair")
```

si le reste de la division entière par 2 est nul

Syntaxe plus compacte :

```
plus_grand = a if a>b else b
```

opérateur ternaire

L'expression conditionnelle peut contenir un opérateur logique.

```
if c>='0' and c<='9':
    print ("Le caractère c est un chiffre")
```

and est l'opérateur logique ET

Le mot-clé not permet d'inverser la condition.

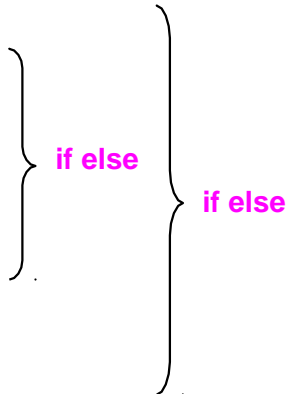
```
x = 3
if isinstance(x, int):
    print("x est un entier")
```

```
x = 1.5
if not isinstance(x, int):
    print("x n'est pas un entier")
```

La fonction **isinstance(objet, classinfo)** est une fonction intégrée qui renvoie un booléen indiquant si l'objet est du type de la classinfo.

Il est possible d'imbriquer deux instructions conditionnelles.

```
# résolution d'une équation du second degré
# a * x**2 + b * x + c = 0
d = b**2 - 4*a*c                # d est le discriminant
if d != 0:
    if d > 0:
        print "Deux racines réelles"
        x1 = (-b + sqrt(d))/(2*a)
        x2 = (-b - sqrt(d))/(2*a)
    else:
        print("Pas de racines réelles")
else:
    print("Une racine réelle")
    x = -b/(2*a)
```



Pour mettre les trois cas possibles au même niveau d'indentation, on peut utiliser le **mot-clé elif** qui est une **contraction de else: if**

```
if d > 0 :                        # cas 1
    print ("Deux racines réelles")
    x1 = (-b + sqrt(d))/(2*a)
    x2 = (-b - sqrt(d))/(2*a)
elif d < 0 :                      # cas 2
    print ("Pas de racines réelles")
else :                           # autres cas
    print("Une racine réelle")
    x = -b/(2*a)
```

L'instruction conditionnelle multiple « Selon ... Faire » se traduit par if elif elif...

4. Les boucles

4.1. La boucle while :

L'instruction qui s'écrit **Tant que ... Faire** en pseudo-code se traduit par **while**

La boucle while permet de répéter un bloc d'instructions tant qu'une expression booléenne est vraie.

Exemple : recherche de la puissance de 2 immédiatement supérieure à un nombre entier.

```
x = 21
p = 1                                # au départ p = 2 puissance 0
while p < x:
    p = p*2
print (p)                            # pour x = 21 on obtient p = 32
```

Autre exemple : demander à l'utilisateur de saisir un nombre entier compris entre 0 et 255.

```
print ("Saisir un nombre entier compris entre 0 et 255 :")
x = int(input())                    # convertir en int la chaîne de caractères saisie
while x < 0 or x > 255:              # or est l'opérateur logique OU
    print ("Le nombre doit être compris entre 0 et 255 :")
    x = int(input())
```

4.2. La boucle for :

L'instruction qui s'écrit **Pour** en pseudo-code se traduit par **for**

La boucle for permet :

- De répéter un bloc d'instructions un nombre déterminé de fois ;
- De parcourir un objet itérable (liste, tableau, chaîne de caractères, etc.) afin d'effectuer un traitement sur chacun de ses éléments.

❑ Répéter n fois un bloc d'instructions :

Exemple : afficher les puissances de 2 de 2^0 à 2^7 .

```
n = 8                                # nombre d'itérations souhaitées
for k in range(n):
    print ("2 puissance " + str(k) + " = " + str(2**k))
```

Cela affiche « 2 puissance 0 = 1 », « 2 puissance 1 = 2 »,... jusqu'à « 2 puissance 7 = 128 ». `str(k)` permet de convertir l'entier k en chaîne de caractère.

L'opérateur **+** permet de **concaténer (mettre bout à bout)** des chaînes de caractères.

range(n) faire varier le nombre k de 0 à n-1, soit n itérations dans la boucle for.

❑ Réaliser un traitement sur tous les éléments d'un objet itérable :

« Itérable » signifie « que l'on peut parcourir ».

Exemple : calculer la moyenne des nombres contenus dans une liste.

```
uneliste = [1, 5, -2, 8]
somme = 0
for n in uneliste:
    somme = somme + n                # calcul de la somme des éléments
moyenne = somme/len(uneliste)      # division par le nombre d'éléments
```

Remarque : les quatre lignes pourraient être avantageusement remplacées par :

```
moyenne = sum(uneliste)/len(uneliste)
```

❑ Remplir une liste à l'aide d'une boucle for :

Exemple : remplir une liste avec les puissances de 2 de 2^0 à 2^7 .

```
puis2 = [2**n for n in range(8)]
```

Le code à l'intérieur des crochets se lit : **2 puissance n pour n allant de 0 à 7.**

On obtient la liste [1, 2, 4, 8, 16, 32, 64, 128].

4.3. Boucles imbriquées :

Exemple : dessiner un rectangle composé de caractères *.

```
largeur, hauteur = 4, 6          # initialisation de 2 variables
print()                          # sauter une ligne
for i in range(hauteur):         # pour chaque ligne
    ligne = ""                   # commencer avec une ligne vide
    for j in range(largeur):     # pour chaque colonne
        ligne = ligne + "*"      # ajouter une *
    print(ligne)
```