RÉSUMÉ 8 - OUTILS DE BASE

Lien vers l'activité : Outils de base
www.infoforall.fr - Dernière modif. : 26 10 2020

8.1 - INSTRUCTION CONDITIONNELLE

BOOLÉEN

Un **booléen** ne peut valoir que **True** (pour VRAI) ou **False**(pour FAUX).

DÉROULEMENT D'UNE SÉQUENCE IF - ELIF - ELSE

if veut dire SI. Sa condition est la première qui sera évaluée. Si elle est `True`, on exécute son bloc. **Les autres blocs ne seront même pas testés**.

elif veut dire SINON SI. Il s'agit d'un bloc qui sera exécuté si la condition est vraie ET que les conditions du dessus n'ont pas été validées.

else veut dire SINON. Il s'agit du bloc qui sera exécuté si aucune des conditions précédentes n'a été évaluée à `True`.

```
1   nombre = 12
2   if nombre == 0 :
3       message = "Zéro !"
4   elif nombre % 5 == 0 :
5       message = "Multiple de 5"
6   elif nombre % 2 == 0 :
7       message = "Multiple de 2"
8   elif nombre % 6 == 0 :
9       message = "Multiple de 6"
10  else :
11      message = "Ni 0, ni multiple de 2, 5 ou 6"
12  # On reprend l'exécution séquentielle ici
```

Ici, on va juste exécuter la ligne 7 car la ligne 6 donne une condition évaluée à `True`.
Même si la condition de la ligne 8 donnerait également `True`, l'interpréteur va juste passer les lignes jusqu'à la ligne 11 et voir ce qui se trouve en ligne 12.

Problème : cela veut dire que le test pose problème pour détecter les multiples de 6. Il va s'arrêter sur les multiples de 2 puisque 6 est un multiple de 2 !

TEST D'ÉGALITÉ

Attention en Python, il faut utiliser un double egal `code` puisqu'un égal simple traduit l'affectation.
Si on veut tester si le contenu de a est identique au contenu de b : `a == b`

AUTRES TESTS

- a Strictement supérieur à b : `a > b`
- a Supérieur ou égal à b : `a >= b`
- a Strictement Inférieur à b : `a < b`
- a Inférieur ou égal à b : `a <= b`
- a Différent de b : `a != b`

8.2 - MODULE RANDOM

Ce module permet de générer des nombres pseudo-aléatoires.
Voici comment créer une variable contenant un nombre compris entre 1 et 100 :

```
1  import random
2  mystere = random.randint(1,100)
```

Ligne 1 : on importe les codes contenus dans le module random.
En ligne 2, on va chercher le code de la fonction randint contenu dans le module random.
On fournit les **arguments** 1 et 100 pour signaler qu'on veut tirer un nombre dans l'intervalle [1 ; 100].
On stocke le résultat aléatoire dans une variable nommée **mystere**.

8.3 - BOUCLE NON BORNÉE : WHILE (TANT QUE)

On utilise une boucle non bornée lorsqu'on ne sait pas à l'avance combien de fois nous voulons réaliser une séquence d'instructions.
While veut dire **tant que**.
Cela veut dire que **tant que** la condition fournie est évaluée à **True**, l'interpréteur va réaliser la séquence indentée en boucle.

```
1  reponse = "N"
2  nombre = 0
3  while reponse == 'N' :
4      nombre = nombre + int(input("Quel nombre voulez-vous rajouter ? "))
5      reponse = input("Voulez-vous sortir du programme ? (O/N): ")
6  print(f"Au total, on obtient {nombre}")
```

Si la condition est évaluée à **False** lors d'un passage à la ligne 3 de la condition, l'interpréteur va repartir directement à la ligne 6.
Si la condition est évaluée à **True** lors d'un passage à la ligne 3 de la condition, l'interpréteur va exécuter les lignes 4, 5 et revenir en boucle à la ligne 6 .
Ici, on initialise une variable **reponse** à "N".
De cette façon, nous sommes certain de rentrer au moins une fois dans la boucle.
Tant que l'utilisateur répond par "N", on recommence la boucle.
Dès qu'il répond par autre chose, la condition de la ligne 3 devient fausse et on partira directement en ligne 6.

8.4 - BOUCLE BORNÉE FOR / POUR

On utilise une telle boucle lorsqu'on veut exécuter une séquence d'instructions en boucle en connaissant à l'"avance le nombre de fois que la boucle devra s'exécuter.

```
1  monTexte = "Nouveauté"
2  for x in range(5) :
3      print(x)
4      print(monTexte)
5  # On est sorti de la boucle en arrivant ici
```

Ici, la boucle va s'exécuter **5 fois** avec des valeurs successives de la variable **x** valant `0 1 2 3 4`.
On a donc bien 5 valeurs mais la valeur finale est 4 puisque la valeur initiale est 0.

DESCRIPTION DE RANGE

```
1  for nombre in range(10, 101, 2) :
2      print(nombre, end=" - ")
```

On transmet d'abord la valeur initiale : ici 10. Puis la valeur finale (non incluse) : ici 101. On ira donc jusqu'à 100 au maximum. Enfin, le pas entre chaque boucle. Ici, on rajoute 2 à chaque tour. 10, 12, 14, 16 ... 98, 100.
On peut ne transmettre que le deuxième. Dans ce cas, le départ est 0 et le pas vaut 1. On peut ne transmettre que le départ et la valeur de fin. Dans ce cas, le pas vaut 1.

8.5 - PERMUTATION

```
1  a,b = b,a # Version pythonnesque
2  temp = a # Version NSI, compatible avec les autres langages
3  a = b
4  b = temp
```

8.6 - AFFECTATION MULTIPLIE

```
1  a = b = c = 5 # Version Pythonnesque
2  a = 5 # Version NSI, compatible avec les autres langages
3  b = 5
4  c = 5
```

8.7 ET / OU

ET : VRAI UNIQUEMENT SUR TOUT EST VRAI

Valeur de a	Valeur de b	a ET b
VRAI	VRAI	**VRAI**
FAUX	VRAI	**FAUX**
VRAI	FAUX	**FAUX**
FAUX	FAUX	**FAUX**

En Python : `a and b`.

OU : VRAI SI L'UNE DES EXPRESSIONS EST VRAIE

Valeur de a	Valeur de b	a OU b
VRAI	VRAI	**VRAI**
FAUX	VRAI	**VRAI**
VRAI	FAUX	**VRAI**
FAUX	FAUX	**FAUX**

En Python : `a or b`