Algorithmie et programmation – feuille 3

Nous avons vu jusqu'à maintenant comme effectuer un calcul et afficher un résultat à l'écran. Bien sûr, un programme peut faire bien plus que ça. En fait, non seulement un programme exécute des calculs, mais il vérifie et compare aussi des variables et décide de l'action suivante en fonction du résultat.

Pour comparer des variables, on utilise des opérateurs de comparaison et des opérateurs logiques. On a déjà vu, sans les nommer, deux autres types d'opérateurs : l'opérateur d'affectation =, et les opérateurs mathématiques.

Les opérateurs de comparaison ou les opérateurs logiques ne donnent que deux résultats : *true* (vrai) ou *false* (faux). A partir de ces résultats, le programme décide de l'action à suivre.

Opérateur d'affectation	
=	Assigne une valeur ou une chaîne de caractères à une variable
Opérateurs de comparaison	
x < y	x est strictement inférieur à y
x > y	x est strictement supérieur à y
x <= y	x est inférieur ou égal à y
x >= y	x est supérieur ou égal à y
x == y	x est égal à y
x != y	x est différent de y
Opérateurs logiques	
A and B	Si A donne *false*, alors l'opération donne *false*. Sinon, l'opération B est évaluée.
A or B	Si A donne *true*, alors l'opération donne *true*. Sinon, l'opération B est évaluée.
not B	Si B donne *true*, alors l'opération donne *false*, et inversement.

Les opérateurs logiques et les opérateurs de comparaison permettent au programme de décider d'une marche à suivre.

Voyons voir comment les opérateurs logiques répondent à l'information qu'on leur donne : supposons que nous leur soumettons un nombre, et qu'ils comparent ce nombre à un autre.

Code	Résultat

`x = 10`
`y = 8`
`if x<y:`
`  print(x,"est plus petit que",y)`
`if x>y:`
`  print(x,"est plus grand que",y)`
`if x<=y:`
`  print(x,"est plus petit ou égal à",y)`
`if x>=y:`
`  print(x,"est plus grand ou égal à",y)`
`if x==y:`
`  print(x,"est égal à",y)`
`if x != y:`
`  print (x,"n'est pas égal à",y)`
	`>>>`
	`10 est plus grand que 8`
	`10 est plus grand ou égal à 8`
	`10 n'est pas égal à 8`

La commande *if* (si, en anglais) utilise un opérateur pour comparer deux variables. On écrit *if*, la première variable, l'opérateur de comparaison, la deuxième variable, puis deux points. Les commandes et fonctions à l'intérieur d'un *if* sont indentées.

Rappel : une indentation se fait avec la touche *tab*.

Ce petit programme compare les deux variables et affiche une phrase si la condition énoncé par *if* est rencontrée (si elle renvoie *true*). Nous avons écrit six conditions, mais il n'a affiché que trois phrases. On pourrait vouloir que le programme soit plus explicite et fasse autre chose quand la condition n'est pas remplie.

Pour ce faire, on peut insérer une commande *else* (sinon, en anglais) dans la première condition.

La commande *else* s'écrit après une commande *if* et son contenu, sans indentation, et suivie de deux points. Le contenu de la commande *else* s'écrit à la ligne, avec une indentation.

Comparer deux variables à l'aide des commandes if… else...

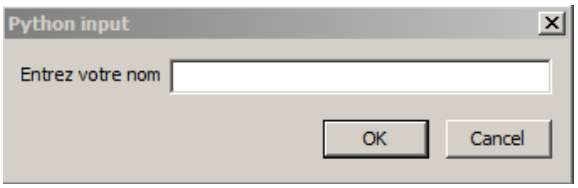
Pour réviser, écrire une fonction qui affiche votre nom.

Ensuite, créer deux variables et leur assigner deux valeurs.

Enfin, utiliser les commandes *if* et *else* pour comparer les deux variables à l'aide d'un opérateur de comparaison de votre choix. Si la condition est remplie, afficher un texte. Si elle n'est pas remplie, afficher un autre texte.

Exemple :

```
>>>
Damien Devaux
5 n'est pas égal à 9 .
```



A ce stade, il serait intéressant de ne pas définir toutes les variables mais de laisser l'utilisateur de notre programme décider de leurs valeurs. La fonction **input()** permet d'ouvrir une boîte de dialogue (*prompt* en anglais) dans laquelle l'utilisateur entre la valeur d'une variable. Elle prend comme argument du texte entre guillemets qui s'affiche dans la boîte de dialogue.

Code	Résultat
`nom =input("Entrez votre nom")` `print(nom)`	`>>>` `Damien Devaux`

Utiliser la fonction input()

Modifier votre code pour qu'une boîte de dialogue apparaisse et demande votre nom. Ne pas changer le reste du code.

La fonction input() permet d'ouvrir une boîte de dialogue dans laquelle l'utilisateur peut entrer le contenu d'une variable.

Maintenant, nous pouvons utiliser les données fournies par l'utilisateur dans notre programme. Or, comme tout bon programmeur le sait, laisser un utilisateur participer à notre programme est le début d'une longue série de problèmes…

Pour voir le problème, nous pouvons créer une calculatrice simple qui ajoute deux nombres fournis par l'utilisateur. En exécutant le programme, et en entrant les nombres 4 et 6 dans les boîtes de dialogue, on obtient le résultat suivant :

Code	Résultat

`x = input("Entrer un nombre")`
`y = input("Entrer un deuxième nombre")`
`print(x+y)`
	`>>>`
	`46`

Le résultat est 46, et non 10 comme on aurait pu s'y attendre. En lui donnant les valeurs, on a omis d'indiquer à l'ordinateur que les variables entrées par l'utilisateur étaient des nombres.

int()	*Signed integer*	*Nombre entier positif ou négatif*
float()	*Floating-point number*	*Nombre décimal positif ou négatif*
str()	*String*	*Chaîne de caractères*
list()	*List*	*Liste de valeurs*

Il existe plusieurs types de variables, et on peut convertir certains types de variables en d'autres types en utilisant des fonctions intégrées à Python. Par défaut, toute valeur entrée par l'utilisateur est considérée comme une chaîne de caractères (*string* en anglais).

Si nous appliquons aux variables les fonctions listés ci-dessus, on peut modifier leur type. Par exemple, modifions le code de notre calculatrice simple :

Code	Résultat

`x = int(input("Entrer un nombre"))`
`y = int(input("Entrer un deuxième nombre"))`
`print(x+y)`
	`>>>`
	`10`

On note qu'on peut mettre une fonction à l'intérieur d'une autre fonction. Le sens de lecture va de l'intérieur vers l'extérieur : ici, on prend la valeur donnée par l'utilisateur, puis on la converti en une variable de type *int* au moyen de la fonction **int()**.

> **Une fonction peut accepter une autre fonction à la place d'un argument. Dans ce cas, la valeur retournée par la fonction à l'intérieur est utilisée comme argument.**

Maintenant, rien ne nous dit que l'utilisateur ne va pas entrer un nombre décimal. On peut alors utiliser la fonction **float()** qui dit au programme que le nombre entré est décimal. En entrant comme valeurs 4,6 et 3,4 on obtient :

Code	Résultat

`x = float(input("Entrer un nombre"))`
`y = float(input("Entrer un deuxième nombre"))`
`print(x+y)`
	`>>>`
	`8.0`

Le résultat est un nombre décimal, puisque les deux nombres donnés étaient des décimaux. Si on souhaite obtenir uniquement la partie entière, on peut insérer la fonction **int()** dans la fonction **print()**.

Code	Résultat

`x = float(input("Entrer un nombre"))`
`y = float(input("Entrer un deuxième nombre"))`
`print(int(x+y))`
	`>>>`
	`8`

Convertir des variables

Changer votre code pour qu'un utilisateur puisse entrer les nombres à comparer lui-même.

Jusqu'à maintenant, on a écrit une ligne de code pour chaque chose que le programme doit faire. Lorsqu'on aura à traiter de grandes quantités de variables, cette manière de faire deviendra longue et répétitive. Il serait donc intéressant de demander au programme de faire plusieurs fois la même chose avec un certain nombre de variables. C'est à cela que les boucles servent.

La boucle *while* (*tant que* en anglais) permet de répéter une action jusqu'à ce qu'une condition soit atteinte. Elle est souvent utilisée avec un « compteur », une ligne de code qui incrémente une variable jusqu'à ce que cette dernière atteigne une valeur égale à la condition.

Code	Résultat

`x = 0`
`while x <= 2:`
`    print(x)`
`    x = x+1`
	`>>>`
	`0`
	`1`
	`2`

Ici, la commande *while* vérifie que x soit inférieur ou égal à 2, affiche la valeur de x, puis augmente x de 1. Une fois la condition atteinte, le programme s'arrête.

Utiliser une boucle

Écrire une boucle while (sans oublier de définir la variable avant) qui compte de trois en trois jusqu'à 90.

---

1. Bien sûr, on peut faire autre chose qu'imprimer *x* chaque fois que la boucle se répète. Par exemple, on peut demander au programme de faire une opération un certain nombre de fois avec des valeurs trouvées dans une liste et d'imprimer tous les résultats.

Mise en application

Nous avons vu comment créer et modifier des variables et des listes, comment créer et utiliser des fonctions, comparer deux variables et décider d'une action à prendre, et répéter une même action un certain nombre de fois.

Maintenant, vous allez créer un programme qui utilise toutes ces compétences pour calculer la vitesse d'un objet lorsqu'il tombe d'une certaine hauteur toutes les secondes jusqu'à ce qu'il arrive sur le sol.

Toute masse sur Terre ressent une force due à la gravité $\mathbf{F_g}$, égale au produit de la masse **m** par la gravité **g**.	$F_g = m\,g$
Si on applique une force à une masse, et en l'absence de toute autre force, cette dernière accélère à une accélération égale au ratio de la force appliquée sur la masse.	$a = \frac{F}{m}$

La force appliquée à la masse ici est celle due à la gravité. Substituant, on obtient :

$$a = \frac{mg}{m} \quad \rightarrow \quad a = g$$

L'accélération d'un objet qui tombe est donc égale à la gravité, peu importe sa masse.

Une accélération est un changement de vitesse sur un changement de temps.	$g = \frac{\Delta v}{\Delta t}$

Comme on sait que l'accélération est simplement la gravité, on peut écrire :

$$\Delta v = g\,\Delta t$$

Comme un changement de vitesse est égal à la différence entre une vitesse initiale et une vitesse finale après un certain temps :

$$v_f - v_i = g\,\Delta t \quad \rightarrow \quad \boxed{v_f = v_i + g\,\Delta t}$$

En utilisant cette dernière formule, créer un programme qui utilise des variables pour la vitesse finale, la vitesse initiale, la gravité (9,81 m/s²) et un changement en temps de une seconde pour calculer la vitesse de l'objet toute les secondes pendant une chute de dix secondes.