Algorithmie et programmation – feuille 2

Nous avons vu précédemment comment créer et appeler des variables pour les utiliser et afficher quelque chose à l'écran. Avant d'en apprendre plus sur les fonctions – elles peuvent faire bien plus que simplement afficher du texte – nous allons étudier un autre type de variables : les listes.

Listes

Parfois, lorsqu'on on a beaucoup de valeurs à calculer, créer beaucoup de variables ne serait pas pratique. On peut alors créer des listes qui regroupent des valeurs. On crée une liste comme on crée une variable, mais en utilisant des crochets plutôt que des parenthèses.

Une liste est une chaîne de variables dont chaque élément peut être appelé indépendamment des autres dans un programme. Les éléments d'une liste sont compris entre des crochets, et séparés par des virgules.

Une fois qu'on a crée la liste, on peut appeler un ou plusieurs de ses éléments dans une fonction en indiquant le rang de la valeur qui nous intéresse. Attention : le premier rang est le rang 0.

Code	Résultat
`liste=["a","b","c","d","e","f","g","h","i","j","k"]` `print(liste[0])`	`>>>` `a`

On note que, dans ce cas, il a fallu encadrer chaque lettre de l'alphabet avec des guillemets pour que le programme comprenne que ce sont des lettres, pas des variables.

Appeler le contenu d'une liste.

Définir une liste comprenant chacun des caractères qui forment votre prénom. A l'aide de la fonction **print()**, afficher chaque lettre, de la dernière à la première, sur autant de lignes que votre nom comprend de lettres.

Exemple : `>>>`

```
N
E
I
```

(suite)

```
M
A
D
```

Il existe des outils de manipulation de listes qui permettent de créer, modifier et ordonner des listes numériques. Nous nous limiterons, en plus de l'appel d'éléments spécifiques, à la mesure d'une liste.

Pour mesurer une liste, on utilise la fonction **len()** (*lenght* en anglais). Cette fonction retourne le nombre d'éléments dans une liste. Pour l'appliquer à une liste, on inscrit le nom de la liste entre les parenthèses. Pour conserver ce nombre dans une variable, on attribue la fonction **len()** à la variable.

Code	Résultat

`liste=[0,1,2,3]`
`x = len(liste)`
`print(x)`
	`>>>`
	`4`

Mesurer une liste.

En dessous du code que vous avez déjà entré, attribuer à x la longueur de la liste contenant les lettres de votre nom. Afficher à l'aide de la fonction **print()** et sous forme de phrase complète, le nombre de lettres que comporte votre prénom.

Exemple :	`>>>` `Mon nom comporte 6 lettres.`

On note que pour ce dernier exercice, on a utilisé la variable x qu'on avait crée en page 2. Nous n'avons pas changé le code, donc cette variable est toujours sur la quatrième ligne, définie comme étant égale à un chiffre bien particulier. La valeur de x change pendant la lecture du code : elle devient le nombre de lettres dans un prénom. C'est tout à fait normal : x avait une valeur lorsqu'on lui a demandé de l'écrire une première fois, et a changé de valeur avant qu'on lui demande de l'écrire une seconde fois.

Les instructions sont lues par l'interpréteur ligne par ligne. Toute modification d'une variable se produit l'une à la suite de l'autre

Fonctions

Une fonction est une commande ou une suite de commandes qui prend un ou plusieurs arguments (des variables, des nombres, du texte...) et donne une sortie. C'est exactement le même concept qu'en mathématiques – sauf qu'on n'est pas limité aux nombres. En effet, les variables utilisées peuvent être des nombres, du texte, etc.

On défini une fonction en Python à l'aide de la commande **def**. Ensuite, on appelle la fonction dans le code pour en afficher le résultat. Évidemment, si on appelle pas la fonction, le programme ne l'utilisera pas !

Code	Résultat

`nom = "Damien Devaux"`
`def maFonction(x):`
`    print("Bonjour, mon nom est",x)`
`maFonction(nom)`
	`>>>`
	`Bonjour, mon nom est`
	`Damien Devaux`

Dans cet exemple, on définit une variable *nom*, puis on crée une fonction qui crée une variable *x* dans laquelle elle stocke la variable qui lui est donnée – on appelle cette variable un argument. La fonction fait quelque chose avec cette variable. Plus tard dans le code, on appelle la fonction en lui donnant en argument la variable qu'on a déjà défini.

On peut utiliser une même fonction avec plusieurs variables :

Code	Résultat

`nom = "Devaux"`
`prenom = "Damien"`
`def maFonction(x):`
`    print("Bonjour, mon nom est",x)`
`maFonction(nom)`
`maFonction(prenom)`
	`>>>`
	`Bonjour, mon nom est`
	`Damien`
	`Bonjour, mon nom est`
	`Devaux`

On note que :

- la définition d'une fonction inclut son nom, des parenthèses avec un ou des arguments (séparés par une virgule) et deux points
- les commandes à l'intérieur de la fonction sont indentées (touche tab)
- on appelle la fonction en lui indiquant quelle variable prendre comme argument

Lorsqu'on appelle une fonction dans le code, on doit appeler autant de variables que la fonction a d'arguments. Les arguments sont séparées par des virgules.

Créer une fonction avec deux arguments.

Écrire une fonction qui affiche votre nom et votre âge. Cette fonction aura deux arguments (par exemple, les variables *x* et *y*) séparés par une virgule, et vous devrez définir deux variables (par exemple *nom* et *age* – attention à ne pas utiliser d'accents dans les noms de vos fonctions et variables).

Exemple : `>>>`

```
Bonjour, mon nom est Damien Devaux et j'ai 45 ans.
```

On peut utiliser une fonction pour effectuer une tâche répétitive – cela permet de raccourcir le code et de gagner du temps. Cette fonctionnalité devient utile avec les formules.

Par exemple, supposons que nous devions programmer un ordinateur pour qu'il calcule une vitesse. Des détecteurs enregistrent une position, puis une deuxième position après un temps *t*.

On sait que la vitesse est égale à la distance divisée par le temps écoulé. La distance est la différence entre la position finale et la position initiale. On a donc besoin de quatre variables pour calculer la vitesse :

- le temps écoulé (nous l'appellerons *temps*)
- la position initiale (nous l'appellerons *positionInitiale*)
- la position finale (nous l'appellerons *positionFinale*)
- la distance parcourue (nous l'appellerons *distance*)

Dans la fonction, nous allons d'abord calculer la distance parcourue (*positionFinale* moins *positionInitiale)*, puis calculer la vitesse (*distance* divisée par *temps*). Nous utiliserons enfin la fonction **print()** pour afficher le résultat dans une phrase complète.

Code	Résultat

`temps=0`
`positionInitiale=0`
`positionFinale=0`
`def vitesse(positionInitiale,positionFinale,temps):`
`  distance=positionFinale-positionInitiale`
`  sortie=distance/temps`
`  print("La vitesse est de",sortie,"mètres par seconde.")`
`vitesse(100,120,1)`
	`>>>`
	`La vitesse est de 20.0 mètres par seconde.`

On note qu'il n'est pas nécessaire d'attribuer une valeur lorsqu'on déclare une variable. En fait, il n'est même pas nécessaire de déclarer les variables avant la fonction, mais c'est une bonne habitude à prendre pour pouvoir contrôler rapidement toutes les variables qui existent dans un code.

En effet, on peut créer des variables à l'intérieur d'une fonction. Dans ce code, la variable *sortie* est créée à l'intérieur de la fonction pour conserver, puis restituer le résultat du calcul. On aurait pu effectuer le calcul dans la fonction **print()**, mais la clarté dans le code est aussi une bonne habitude à prendre.

L'avantage d'une fonction se révèle lorsqu'on doit faire la même opération plusieurs fois. Une fois définie, on peut l'utiliser autant de fois que l'on veut.

Code	Résultat

`def vitesse(positionInitiale,positionFinale,temps):`
`  distance=positionFinale-positionInitiale`
`  sortie=distance/temps`
`  print("La vitesse est de",sortie,"mètres par seconde.")`
`vitesse(100,120,1)`
`vitesse(100,130,1)`
`vitesse(120,140,2)`
	`>>>`
	`La vitesse est de 20.0 mètres par seconde.`
	`La vitesse est de 30.0 mètres par seconde.`
	`La vitesse est de 10.0 mètres par seconde.`

Créer une fonction pour effectuer un calcul plusieurs fois.

A la suite du code que vous avez déjà écrit, sur le modèle de l'exemple ci-dessus, écrire une fonction qui donne la vitesse d'un objet qui tombe après une, cinq et dix secondes. On sait que la vitesse est l'accélération multipliée par le temps. L'accélération due à la gravité sur Terre est égale à 9,81 $m.s^{-2}$.

Exemple : >>>

```
     La vitesse est de 9.81 mètres par seconde après 1
seconde(s).
     La vitesse est de 49.050000000000004 mètres par
seconde après 5 seconde(s).
     La vitesse est de 98.10000000000001 mètres par
seconde après 10 seconde(s).
```

Comme vous l'avez probablement deviné, si on parle maintenant de fonctions, on en a utilisé une depuis le tout début de cette leçon : la fonction **print()**. Comme nous l'avons vu, cette fonction affiche à l'écran les arguments qu'on lui donne, que ce soit une chaîne de caractères entre guillemets ou le contenu d'une ou plusieurs variables.

Jusqu'à maintenant, nos programmes ne faisaient qu'afficher le résultat d'un calcul. On aimerait à l'avenir qu'ils fassent un peu plus que ça. Pour clarifier notre code, on devrait séparer les actions du programme, et diviser les tâches entre fonctions.

Dans l'exemple précédent, la fonction calculait et affichait le résultat. On peut modifier le code pour qu'elle ne fasse que calculer le résultat. On ajoute à la fonction la commande **return** pour qu'elle fasse le calcul. Ensuite, il faudra créer une variable et lui assigner le résultat de la fonction avant d'afficher le contenu de cette variable à l'écran.

Code	Résultat

`def vitesse(positionInitiale,positionFinale,temps):`
`  distance=positionFinale-positionInitiale`
`  return distance/temps`
`sortie = vitesse(100,120,1)`
`print("La vitesse est de",sortie,"mètres par seconde.")`
	`>>>`
	`La vitesse est de 20.0 mètres par seconde.`