8. Les fonctions : *def ...*

▶ Cours

• Dans un programme, il est possible d'écrire des *sous-programmes* ou *fonctions* : ce sont des parties de codes qui sont exécutées lorsqu'on fait appel à elles.

• Une fonction porte un nom, et peut avoir besoin pour fonctionner de variables qu'on appelle les *paramètres* de la fonction.

• On définit une fonction avant de pouvoir l'appeler à partir d'un autre ultérieur du programme.

• La syntaxe générale de définition d'une fonction est la suivante :

```
def nom_de_la_fonction(paramètre1, paramètre2, etc.) :
    instruction1
    instruction2
    instruction3
    etc.
```

Si la fonction n'a pas besoin de paramètres (on dit aussi dans ce cas qu'elle ne prend pas de paramètres) alors on ne met rien entre les parenthèses : `def nom_de_la_fonction() :`

• Comme pour les variables, le nom d'une fonction ne peut pas contenir d'espaces ou de tirets de la touche 6, ni commencer par un chiffre.

• Par exemple, en testant le programme suivant :

```
somme_carres(a,b) :
    s = a**2+b**2
    return(s)

hyp2=somme_carres(3,5)
print(hyp2)
```

, on verra qu'il affiche 34 (car $3^2 + 5^2 = 34$).

L'instruction `return` indique la valeur retournée par la fonction : c'est le résultat de la fonction.

L'instruction `return` n'est pas obligatoire, on ne la met que si la fonction est sensée retourner une valeur.

▶ Exercices

1 On définit la fonction `exercice1` de la façon suivante :

```
def exercice1(a) :
    return a**2-a+1
```

1. Combien cette fonction a-t-elle de paramètres ? .....

2. Que sera-t-il affiché si on tape les commandes suivantes dans le mode direct de Python ?

```
>>> exercice1(2)      .....
>>> exercice1(5)      .....
>>> exercice1(2.1)    .....
```

2 Écrire une fonction `vitesse` qui prend deux paramètres `distance` (supposé être en km) et `temps` (en heures) et qui retourne la vitesse moyenne en km/h.

3 On considère la fonction suivante :

```
import random
def exercice3() :
    return random.randint(1,6)
```

Dans ce script, la commande `import random` met à disposition du programmeur des fonctions additionnelles de Python, parmi lesquelles `randint`. Cette fonction permet d'obtenir un nombre entier aléatoire pris entre les deux nombres donnés en paramètres de `randint`.

Parmi les nombres suivants, lesquels ne peuvent pas être retournés par la fonction `exercice3` ?

2 **3,1** **7** **1,412** **6** **1** **0**

4 Écrire une fonction `solde` qui prend en paramètres un prix et un taux de pourcentage, et qui retourne le prix soldé.

5 La population d'un village était de 3000 habitants en 2017. En moyenne, la ville perd 2 % de ses habitants chaque année.

1. Écrire une fonction `population` qui prend en paramètre une durée en années, et qui calcule la population du village après cette duréee.

Vérifiez en exécutant votre fonction que l'instruction `population(2)` renvoie la valeur 2881,2.

2. Écrire un programme qui détermine au bout de quelle durée la population sera inférieure ou égale à 1500 habitants, et affiche cette durée.

6 Une patinoire propose deux formules de tarification :

• Formule A : chaque entrée coûte 5,25 €.

• Formule B : on paye un abonnement à l'année de 12 €, et chaque entrée coûte alors 3,50 €.

1. Écrire deux fonctions `tarifA` et `tarifB` pour calculer le prix à payer en fonction du nombre d'entrées.

2. Utiliser ces fonctions pour déterminer au bout de combien d'entrées la formule B est la plus avantageuse.

9. Les chaînes de caractères

▶ Cours

• En informatique, le stockage des données repose sur la circulation du courant électrique. Des transistors sont utilisés pour cela : ce sont de minuscules interrupteurs électriques, dont on peut changer l'état électriquement aussi. Un transistor est soit dans l'état ouvert, soit dans l'état fermé

• Un bit est la conceptualisation de ce transistor : un bit vaut 0 ou 1, suivant que le transistor qu'il représente est ouvert ou fermé. 0 ou 1 sont les deux seules valeurs qu'un bit peut prendre : l'information est donc codée en binaire.

• La mémoire de l'ordinateur est constituée d'octets. Ce sont des groupes de 8 bits ; la valeur d'un octet va donc de 0 (les 8 bits sont à 0) jusqu'à 255 (cas où les 8 bits sont à 1).

• Les caractères sont eux aussi codés en binaires : il existe des tables de conversion, qui indique quel caractère correspond à quel code. Par exemple, dans la table de codage ASCII, un octet de valeur 65 correspond à la lettre majuscule A, 66 correspond à B, 67 à C, etc.

• Comme 255 caractères n'étaient pas suffisant pour pouvoir coder tous les alphabets et les symboles connus, un comité a établi Unicode comme remplacement des tables de codage multiples qu'il y avait auparavant.

Le principe est le même : des octets codent les caractères, mais la table de codage est beaucoup plus grand qu'avant.

• Une chaîne de caractères est un groupe d'octets qui est interprété comme une succession de caractères.

Par exemple, la suite d'octets 65 66 67 68 69 code la chaîne "ABCDE" ; inversement, le mot "CHAINE" sera stocké dans la mémoire de l'ordinateur sous la forme de la suite d'octets

• Python convertit automatiquement les chaînes de caractères en octets.

Par exemple, avec la commande `s="CHAINE"`, Python crée en mémoire la suite d'octets 67, 72, 65, 73, 78, 69.

Mais l'instruction `print(s)` n'affichera pas la suite d'octets, elle affichera la chaîne de caractères, car Python fait dans ce cas la correspondance entre les octets et les caractères

• Pour déterminer la longueur d'une chaîne de caractères, on utilise la commande `len`. Si `s="CHAINE"`, alors la commande `len(s)` renverra 6 comme valeurs (il y a 6 caractères dans s).

On peut mettre des espaces dans une chaîne de caractères. Chaque espace compte comme un caractère.

• Il est possible de faire des opérations avec les chaînes de caractères :

Concaténation : +	Répétition : *	Égalité : ==	Différence : !=	Comparaison lexicographique	

`>>> x="auto"`
`>>> y="bus"`
`>>> x+y`
`'autobus'`
	`>>> x="bla"`
	`>>> x*3`
	`'blablabla'`
		`>>> x="chien"`
		`>>> x=="chien"`
		`True`
		`>>> x=="chine"`
		`False`
		`>>> x=="CHIEN"`
		`False`
			`>>> x="chien"`
			`>>> x !="chine"`
			`True`
			`>>> x !="CHIEN"`
			`True`
				`>>> x="fiole"`
				`>>> y="folie"`
				`>>> x<y`
				`True`
					`>>> x="foule"`
					`>>> y="fou"`
					`>>> x<y`
					`False`

• On peut extraire un caractère d'une chaîne, en fonction de sa position de la chaîne. Pour cela, on met le nom de la variable, suivi entre crochets du numéro du caractère qu'on veut (les caractères sont numérotés à partir de 0).

Par exemple avec la chaîne s égale à : "

B	o	n	j	o	u	r		à		t	o	u	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13

"

Dans ce cas, `s[0]` correspond au 'B', `s[1]` au premier 'o', `s[2]` au 'n', etc.

• On peut aussi prendre des « tranches » d'une chaîne, c'est-à-dire des parties ou sous-chaînes.

Pour cela, on met entre crochet le rang de départ puis un deux-points suivi du rang final.

Par exemple, avec la chaîne `s="Bonjour à tous"`, la commande `s[0 :3]` correspond à "Bon" et `s[4 :9]` correspond à "our à".

• On peut utiliser des nombres négatifs, cela permet de repérer les caractères depuis la fin de la chaîne : `s[-1]` correspond au 's' final, `s[-2]` au 'u' en avant-dernière position, `s[-3]` au 'o', etc.

• On peut utiliser la boucle `for` pour un chaîne de caractères, pour parcourir les caractères de la chaîne un par un.

Par exemple, avec

```
s="Bonjour à tous"
for c in s :
␣␣print(c)
```

, on verra s'afficher un à un les caractères de s.

▶ Exercices

1 La chaîne de caractères x est définie par l'instruction `x="Cet exercice est très intéressant␣!"`.

Complétez les cases du tableau ci-dessous.

Question	Quelle est la longueur de la chaîne ?				Quel est le dernier caractère ?	
Affichage			'x'			
Instruction correspondante		x[14]		x[13 :17]		x[7]+x[17 :20]

2 La chaîne de caractères abc est définie par la commande `abc='trois lettres'`. Complétez les affichages obtenus en réponse aux instructions ci-dessous :

```
>>> print(abc) .....
>>> print('abc') .....
>>> '1'+'2'+'3' .....
>>> 1+2+3 .....
>>> abc+'d' .....
```

3 On a commencé un programme qui demande une phrase à l'utilisateur, puis qui compte combien il y a d'espaces dans cette phrase.

Complétez ce programme et testez-le :

```
phrase = input("Tapez votre phrase␣:␣")
n=0
for c in ..... :
␣␣if c == ..... :
␣␣␣␣n=.....
print("Il y a", n, "espaces dans votre phrase")
```

4 Inspirez-vous du programme de l'exercice précédent pour créer un script qui compte la fréquence de 'e' (majuscules ou minuscules) dans une phrase.

Attention, il faut compter les 'é' 'è' 'ê' et 'ë' comme des 'e' !

Approfondissement : Calculez la fréquence de chacune des voyelles, voire la fréquence de chacune des lettres présente dans le texte.

5 Les fonctions `upper()`, `lower()` et `capitalize()` s'appliquent à une chaîne de caractères.

1. Complétez les exemples suivants pour voir ce qu'elles font.

```
>>> p="Il FAit bEAu auJOUrd'hui␣!
>>> p.upper() .....
>>> p.lower() .....
>>> p.capitalize() .....
```

2. Complétez ce script pour faire la fiche d'identité d'une personne, en mettant son nom en majuscules et son prénom avec une lettre capitale :

```
nom = input("Quel est votre nom ? ")
prenom = input("Quel est votre prénom ? ")
age = input("Quel est votre âge ? ")
print("NOM :", ..... )
print("Prénom :", ..... )
print("Âge :", ..... )
```