Langage et programmation – Les fonctions

Contenus	Capacités attendues
Constructions élémentaires	Mettre en évidence un corpus de constructions élémentaires.
Diversité et unité des langages de programmation	Repérer, dans un nouveau langage de programmation les traits communs et les traits particuliers à ce langage.
Mise au point de programmes	Utiliser des jeux de tests
Utilisation de bibliothèques	Utiliser la documentation d'une bibliothèque
Spécification	Prototyper une fonction Décrire les préconditions sur les arguments Décrire les postconditions sur les résultats

Problématique :

Comment programmer une fonction ?

1 – Qu'est-ce qu'une fonction ?

Une fonction est un petit programme rendant un service déterminé. Elle peut avoir besoin de recevoir des entrées, que l'on appelle **paramètres** ; et elle peut calculer un résultat qui est renvoyé lorsqu'on invoque la fonction.

La fonction peut être utilisée dans les programmes au moyen d'appels de fonctions qui sont des demandes d'utilisation du service. Lors de l'exécution du programme, l'appel de la fonction consiste à exécuter la fonction.

Ces appels de fonction ont déjà été vus. Par exemple print() est une fonction permettant d'afficher une ou plusieurs valeurs. Lorsque l'on utiliser une bibliothèque, on appelle les fonction qui sont présente dans cette bibliothèque (comme la fonction randint() ou localtime()).

A – Sortie d'une fonction

En Python, une fonction possède zéro ou une seule sortie. La fonction print() est un exemple de fonction de renvoyant pas de résultat.

Les fonctions renvoyant un résultat renvoient une valeur Python : int, float, str, bool … Par exemple la fonction input() renvoie la chaîne de caractère (str) lue au clavier.

B – Entrée d'une fonction

Les entrées sont les paramètres. Pour une fonction donnée, il faut toujours donner le même nombre de paramètres dans tous les appels de fonction. De nombreuses fonctions attentent également des paramètres d'un type précis.

Exemple : la fonction max() prend en argument deux valeurs et renvoie la plus grande des deux. Tester :

```
max(5)
max(8,5)
max('6',8)
max('aba','axa')
```

Il est possible d'effectuer des opérations sur les paramètres dans l'appel de fonction ; ou encore d'utiliser le résultat d'une fonction dans l'entrée d'une autre fonction.

Exemple : Tester

```
x = 10
max(x,5)
max(x+5,3*3)
max(x,abs(-50))
```

C – Différence avec une fonction mathématique

- Un appel de fonction informatique ne calcule pas nécessairement une valeur. Exemple : print('ab')
- Les fonctions informatiques peuvent modifier la mémoire et l'environnement du programme.
- La valeur calculée par une fonction informatique ne dépend pas uniquement de ses paramètres mais aussi de l'état de la mémoire et de l'environnement (par exemple les saisies au clavier)

Exemple :

```
x = 10
print(x)
x = input(« Quel est ton nom ? »)
print(x)
```

D – Un peu de vocabulaire

- **Spécification d'une fonction / d'un programme** : description de la fonction / du programme en s'axant sur les besoins dont la fonction / programme répond.

  Exemple : spécification de la fonction max() : déterminer la valeur la plus grande de deux nombres

- **Préconditions sur les arguments** : décrit les conditions que doivent remplir les arguments d'une **fonction**.

  Exemple : préconditions sur les arguments de la fonction max() : int ou float.

- **Postconditions sur le résultat** : décrit les conditions que doivent remplir la sortie d'une **fonction**.

  Exemple : postconditions sur le résultat de la fonction max() : un des deux arguments, eux-mêmes int ou float.

2 – Définir une fonction

En Python, le type d'une fonction est function. La définition d'une fonction doit respecter les règles suivantes :

- La première ligne doit commencer par le mot **def suivi du nom de la fonction puis des parenthèses entourant les paramètres de la fonction séparés par des virgules**. On finit par deux points (:).
- Les lignes qui suivent représentent le bloc d'instruction et **doivent être indentées**.
- L'introduction **return** permet de retourner la valeur de la fonction.
- Le **retour à la ligne** signale la fin de la fonction.
- On **donnera toujours un commentaire** à la fonction pour bien préciser quel est son rôle et ce qu'elle renvoie.

Exemple :

```
def carré (a) :            # cette fonction donne en argument le carré du paramètre.
      return a*a
print(carré(2))     # Affiche le carré de 2.
```

Attention : l'instruction return, qui permet de retourner une valeur, interrompt également l'exécution de la fonction.

Remarque :

- lorsque la fonction renvoie un booléen, on dit que la fonction est **booléenne**.
- Il est possible d'affecter dans des variables les valeurs renvoyer par une fonction.
- Si on renseigne print au lieu de return, on obtient bien la valeur affichée, mais dans ce cas la valeur obtenue ne peut plus être utilisée dans un calcul. Si on souhaite utiliser la valeur calculée par une fonction il faut donc utiliser return.
- Une fonction qui ne renvoie rien est de type NoneType. En Python, une suite d'instructions qui ne renvoie rien est appelée une **procédure** et non une fonction. Ces deux types ont toutefois la même syntaxe.
- On appelle **fonction récursive des fonctions qui s'appellent elles-mêmes.**

Pour s'entraîner à construire des fonctions, nous allons utiliser le module turtle de Python. Il permet de dessiner sur une fenêtre graphique.

Les commandes de base sont :

forward(), backward(), left(), right()

up() : on lève le « crayon », goto(abscisse, ordonnée) : se déplacer vers le point, down() : on abaisse le « crayon »

Exemple : Dessiner un carré :

```
import turtle
turtle.pencolor(« purple ») # Couleur purple
turtle.width(2) # Epaisseur du crayon
for i in range (0,4) :
      turtle.forward(100) #déplacement vers l'avant de 100 pixels
      turtle.left(90) # Rotation vers la gauche de 90°
```

Applications :

A – Créer une fonction carre() permettant de dessiner un carre.

B – Créer une fonction rectangle(), triangle_isocele() et losange()

C – Créer une fonction permettant de dessiner trois carrés (on pourra utiliser la fonction créée en A).

D – Créer une fonction carreDeLongueur(n) prenant en paramètre d'entrée n la longueur du carrée.

E – La fonction suivante permet de se déplacer vers un point sans tracer sur la fenêtre :

```
def seDeplacerSansTracer(abscisse, ordonneee) :
      turtle.up()
      turtle.goto(abscisse,ordonnee)
      turtle.down()
```

A l'aide de la fonction précédente, créer une fonction permettant de tracer un carré de longueur, et d'origine x et y.

F – Créer une fonction permettant de dessiner un carré dans un carré.

3 – Fonction renvoyant un résultat

Certaines fonctions ont pour vocation de calculer une valeur.

```
def double (n) :
      return n*n
```

L'instruction return permet de définir le résultat de la fonction. Mais c'est également une instruction qui permet de stopper l'exécution de la fonction (lorsque le résultat peut être défini, la fonction a terminé son travail). Ainsi, l'exécution d'un return provoque la sortie de toutes les instructions de bloc qui le contiennent (donc la sortie des boucles notamment) et l'abandon des instructions en attente (effet court-circuit).

Exemple :

```
def tousInferieurA(borne, liste) :
      for n in liste :
            if n > borne :
                  return False
      return True
```

Tester la fonction pour la liste : mesAchats = [12,6,78,50,3,65] avec pour borne 80 puis 60.

4 – Fonctions et mémoire

Un appel de fonction conduit à allouer temporairement de la mémoire pour les paramètres et les variables définies dans le corps de la fonction, c'est-à-dire la suite d'instruction qu'elle contient. Les paramètres et ces variables n'existent que le temps d'exécuter un appel de fonction. S'il y a plusieurs appels différents, les espaces mémoires réservés seront différents et aucune valeur ne sera conservée entre deux appels de la fonction.

Toutes les variables auxquelles on effectue une valeur dans un fonction 'ont d'existence qu'à l'intérieur de la fonction. Ce sont des variables locales. Elles sont créées lors de l'exécution d'un appel de la fonction et détruites à la fin de l'exécution de cet appel. Il est dès lors impossible d'y faire référence à l'extérieur de la fonction.

Exemple :

```
def totalAchats(Achat_1, Achat_2) :
      total = 0
      total = Achat_1 + Achat_2
      return total
```

Tester : print(totalAchats(10, 40)) puis print(total)

Dans la fonction précédente, total est une variable locale de la fonction totalAchat. Cette variable n'a pas d'existence en dehors de la fonction. Il est possible d'un remédier en utilisant la commande global (son utilisation est toutefois à éviter pour limiter les interactions entre la fonction et le reste du programme) :

```
total = 0
def totalAchats(Achat_1, Achat_2) :
      global total
      total = Achat_1 + Achat_2
      return total
```

Tester : print(totalAchats(10,50)) puis print(total)

Applications :

G – On souhaite réaliser une fonction permettant d'afficher entre deux nombres, le plus petit.

1) Quel nom pourrait-on donner à cette fonction ?
2) Programmer en Python cette fonction
3) Faire un test avec 23 et 12
4) Modifier le programme, de façon à l'utiliser avec n'importe quel nombre.

H – Physique et informatique : créer une fonction qui prend en paramètre la masse de l'utilisateur en kg et renvoie son poids sur la Terre en N.

I – Physique et informatique : créer une fonction qui prend en paramètre la masse d'un objet en kg ainsi que sa vitesse en m/s et renvoie son énergie cinétique.

J – Mathématiques et informatique : Créer une fonction permettant de calculer le volume d'une sphère. Puis faire afficher ce volume en laissant à l'utilisateur le choix du rayon. On utilisera une valeur approchée de pi.

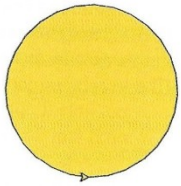
K – Chimie et informatique : créer un programme « dilution » contenant une ou plusieurs fonctions donnant le volume de solution mère à prélever connaissant la concentration en solution mère, la concentration en solution fille, et le volume de solution fille à produire. Ce programme écrira alors le protocole de la dilution avec les bonnes valeurs numériques.

L – Chimie et informatique : créer un programme « dissolution » sur le même principe que celui de la dilution.

Retour sur le module turle : pour remplir une forme avec une couleur, il faut utiliser la fonction fillcolor(couleur) qui permet de choisir la couleur de remplissage et d'encadrer les instructions correspondant à la construction de la forme avec begin_fill() et end_fill().

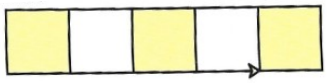
Exemple (construction d'un cercle jaune) :

```
fillcolor(« yellow »)
begin_fill()
circle(100)
end_fill()
```

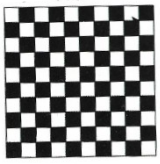


M – Utiliser cette méthode pour tracer un carré bleu de côté 20 en (0,0) puis un carré vert de côté 100 en (50,30)

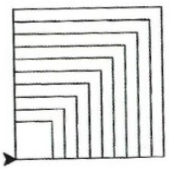
N – Ecrire une fonction qui trace une ligne de n carrés consécutifs et de même taille et dont on alterne les couleurs avec deux couleurs données. Les couleurs, la taille, le nombre de carrées et l'origine de la ligne doivent pouvoir varier. Voici un exemple de carrés jaunes et blanc :



O – Ecrire la fonction damier() qui trace un damier noir et blanc 10 x 10 cases. Les cases sont de côté 20 et le coin inférieur gauche du damier et le point (0,0)



P – Ecrire la fonction carresEmboites qui trace des carrés conformément à la figure suivante : Le nombre de carrés et le côté du plus petit carré doivent pouvoir varier.



Q – QCM

Question 7 Avec la définition de fonction `f` suivante en Python, quelle est la valeur retournée par l'appel `f(42, 21)` ?

```
def f(x,y):
    if x > y:
        return y,x
    else:
        return x,y
```

- ☐ (21,42)
- ☐ (21,21)
- ☐ (42,42)
- ☐ (42,21)

Question 9 Dans la définition suivante de la fonction `somme` en Python, quelle est l'instruction à ajouter pour que la valeur retournée par l'appel `somme([10,11,12,13,14])` soit 60 ?

```
def somme(tab):
    s = 0
    for i in range(len(tab)):
        ...
    return s
```

- ☐ `s = s + tab[i]`
- ☐ `s = tab[i]`
- ☐ `tab[i] = tab[i] + s`
- ☐ `s = s + i`

Question 32 La fonction suivante calcule la racine carrée du double d'un nombre flottant.

```
from math import sqrt

def racine_du_double(x):
    return sqrt(2*x)
```

Quelle est la précondition sur les arguments de cette fonction ?

- ☐ `x >= 0`
- ☐ `2 * x > 0`
- ☐ `x < 0`
- ☐ `sqrt(x) >= 0`

Question 33 Avec la définition de fonction `capital_double` suivante, que peut-on toujours affirmer à propos du résultat `n` retourné par la fonction ?

```
def capital_double (capital, interet):
    montant = capital
    n = 0
    while montant <= 2 * capital:
        montant = montant + interet
        n = n + 1
    return n
```

- ☐ `capital + n * interet > 2 * capital`
- ☐ `n = capital / interet`
- ☐ `capital * n * interet > 2 * capital`
- ☐ `n = 2 * capital / interet`

Question 34 Le programme Python suivant ne calcule pas toujours correctement le résultat de x^y. Parmi les tests suivants, lequel va permettre de détecter l'erreur ?

```
def puissance (x,y):
    p = x
    for i in range (y - 1):
        p = p * x
    return p
```

- ☐ `puissance(2,0)`
- ☐ `puissance(2,1)`
- ☐ `puissance(2,2)`
- ☐ `puissance(2,10)`

Question 37 La fonction suivante doit calculer la moyenne d'un tableau de nombres, passé en paramètre. Avec quelles expressions, faut-il compléter l'écriture pour que la fonction soit correcte ?

```
def moyenne(tableau):
    total = ...
    for valeur in tableau:
        total = total + valeur
    return total / ...
```

- ☐ 0 et `len(tableau)`
- ☐ 0 et `len(tableau) + 1`
- ☐ 1 et `len(tableau)`
- ☐ 1 et `len(tableau) + 1`

QCM 2.2

On considère le code suivant :

```
def f(l):
    for i in range(len(l)//2):
        l[i],l[-i-1] = l[-i-1],l[i]
```

Après les lignes suivantes :

```
l = [2,3,4,5,7,8]
f(l)
```

quelle est la valeur de `l` ?

Réponses

A [2,3,4,5,7,8]

B [5,7,8,2,3,4]

C [8,7,5,4,3,2]

D [4,3,2,8,7,5]

QCM 6.1

a et m étant des entiers strictement positifs la fonction suivante calcule a^m :

```
def puissance(a,m):
    p=1
    n=0
    while n < m:
        p = p * a
        #
        n = n + 1
    return p
```

À la ligne marquée d'un #, on a :

Réponses possibles

A $p = a^n$

B $p = a^{n-1}$

C $p = a^{n+1}$

D $p = a^m$

QCM 6.2

Voici une fonction Python de recherche d'un maximum :

```
def maxi(t):
    m = -1
    for k in range(len(t)):
        if t[k] > m:
            m = t[k]
    return m
```

Avec quelle précondition sur la liste `t`, la postcondition « `m` est un élément maximum de la liste `s` » n'est-elle pas assurée ?

A Tout élément de t est un entier positif ou nul

B Tout élément de t est un entier supérieur ou égal à −2

C Tout élément de t est un entier supérieur ou égal à −1

D Tout élément de t est un entier strictement supérieur à −2

QCM 6.3

La fonction `indice_maxi` ci-dessous doit rechercher l'indice de la valeur maximale présente dans une liste de nombres et le renvoyer. Dans le cas où cette valeur maximale est présente plusieurs fois, c'est le plus petit de ces indices qui doit être renvoyé.

```
def indice_maxi(liste):
    valeur_max = liste[0]
    indice = 0
    for i in range(len(liste)):
        if liste[i] > valeur_max:
            indice = i
return indice
```

Cette fonction a mal été programmée. On souhaite mettre en évidence son incorrection par un test bien choisi. Quelle valeur de test mettra l'erreur en évidence ?

Réponses possibles

A [1, 2, 3, 4]

B [4, 3, 2, 1]

C [1, 3, 3, 2]

C [1, 1, 1, 1]