Programmation structurée et fonctions

INFORMATIQUE COMMUNE - TP n° 2 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ indenter et enchaîner correctement les blocs Python
- ☞ coder une structure conditionnelle (et l'expression conditionnelle)
- ☞ coder une boucle avec un range sur l'intervalle d'entiers ouvert [0,n[
- ☞ coder une boucle tant que en précisant la condition de sortie
- ☞ passer d'un script à l'écriture d'une fonction
- ☞ coder et utiliser une fonction (paramètres formels, effectifs, retour)
- ☞ tester une fonction à l'aide d'un programme principal

A Des scripts aux fonctions

Ⓡ Pour résoudre les trois premiers exercices, on demande de créer :

1. dans un premier temps, un script qui affiche à l'écran le résultat,
2. puis, dans un deuxième temps, une fonction qui renvoie le résultat et un programme principal qui appelle cette fonction. Attention : ne pas utiliser de `print` dans les fonctions, réserver les `print` au programme principal.

Voici un exemple :

Code 1 – Exemple

```
def square(x):
    return x * x


if __name__ == "__main__":
    # SCRIPT VERSION
    x = 3
    sqrx = x * x
    print(f"Squaring {x} gives {sqrx}.")

    # FUNCTION VERSION
    print(f"Squaring {x} gives {square(x)}.")
```

On pourra dans la suite de ce TP **omettre** l'instruction `if __name__ == "__main__":` et considérer que la fin du script est le programme principal. L'essentiel étant de comprendre cette notion de programme principal, point d'entrée de l'exécution du code.

A1. **Tarifs de la piscine.** À Brest, les tarifs d'entrée à la piscine en vigueur sont :

- Ticket individuel : 4,20 €,
- Tarif réduit : 2,75 €,
- Enfant 4 - 17 ans : 2,10 €,
- Enfant 0 - 3 ans : gratuit.

(a) Écrire un script qui affiche le tarif d'entrée de la piscine de la manière suivante : `4 ans --> Tarif : 2.1 euros.`
L'age est une variable du programme définie ainsi :

```
1   age = 4
```

(b) Améliorer ce script afin de prendre en compte les réductions éventuelles. Pouvez-vous utiliser une expression conditionnelle pour prendre en compte cet élément ?

(c) Transformer ce script en une fonction Python. Le prototype de la fonction est `get_price(age, reduced=False)`. Le paramètre `age` est de type `int`, `reduced` est un paramètre optionnel de type `bool` et la fonction renvoie un type `float`.

(d) Tester la fonction à l'aide du programme principal suivant :

```
1   # valid parameters
2   print(f"3 ans --> Tarif : {get_price(3)} euros")
3   print(f"26 ans --> Tarif : {get_price(26)} euros")
```

(e) Modifier le programme principal afin de rendre les tests plus exhaustifs : tester des paramètres incongrus ou de type inadéquat.

Solution :

Code 2 – Piscine

```
1  def get_price(age, reduced=False):
2      if age < 4:
3          return 0.0
4      elif 3 < age < 18:
5          return 2.1
6      else:
7          return 2.75 if reduced else 4.2
8
9
10 if __name__ == "__main__":
11
12     # SCRIPT VERSION
```

```
    age = 4
    reduced = False
    if age < 4:
        print(f"{age} ans --> Tarif : 0.0 euros")
    elif 3 < age < 18:
        print(f"{age} ans --> Tarif : 2.1 euros")
    else:
        if reduced:
            print(f"{age} ans --> Tarif : 2.75 euros")
        else:
            print(f"{age} ans --> Tarif : 4.2 euros")

    # FUNCTION VERSION
    # valid parameters
    print(f"2 ans --> Tarif : {get_price(2)} euros")
    print(f"3 ans --> Tarif : {get_price(3)} euros")
    print(f"4 ans --> Tarif : {get_price(4)} euros")
    print(f"10 ans --> Tarif : {get_price(10)} euros")
    print(f"17 ans --> Tarif : {get_price(17)} euros")
    print(f"18 ans --> Tarif : {get_price(18)} euros")
    print(f"26 ans --> Tarif : {get_price(26)} euros")
    print(f"26 ans & not reduced --> Tarif : {get_price(26, False)}
        euros")

    # exotic parameters (tests for robustness)
    print(f"3 ans & reduced --> Tarif : {get_price(3, True)} euros")
    print(f"3 ans & not reduced --> Tarif : {get_price(3, False)}
        euros")
    print(f"10 ans & reduced --> Tarif : {get_price(10, True)} euros
        ")
    print(f"10 ans & not reduced --> Tarif : {get_price(10, False)}
        euros")

    # wrong parameters (tests for robustness)
    print(f"3.5 ans --> Tarif : {get_price(3.5)} euros")
    print(f"11.5 ans --> Tarif : {get_price(11.5)} euros")
    print(f"26.335 ans --> Tarif : {get_price(11.5)} euros")
```

A2. **Horaires des bus.** La fréquence de passage des bus varie au cours de la semaine : le week-end, ils passent toutes les 30 minutes. Les autres jours de la semaine, ils passent toutes les 10 minutes en heure creuse et toutes les 5 minutes en heure d'affluence (entre 7h et 9h puis entre 16h et 18h).

(a) Écrire un script qui calcule la fréquence de passage des bus en fonction de l'heure et du jour de la semaine.

(b) Transformer ce script en fonction Python. Le prototype de la fonction est `get_frequency(day, hour)`. `day` est de type `str`, `hour` est de type `int`. Cette fonction renvoie un type `int`.

(c) Tester la fonction à l'aide du programme principal suivant :

```
print(f"Frequency : {get_frequency("dimanche")}")
print(f"Frequency : {get_frequency("lundi", 12)}")
print(f"Frequency : {get_frequency("mardi", 8)}")
print(f"Frequency : {get_frequency("jeudi", 17)}")
```

(d) Si le paramètre `day` est un jour du week-end, le paramètre `hour` est optionnel. Modifier le programme afin de rendre le paramètre `hour` optionnel.

(e) Modifier le programme principal afin de rendre les tests plus exhaustifs : tester des paramètres incongrus ou de type inadéquat.

(f) Comment devrait se comporter la fonction `get_frequency` si les bus ne circulent pas entre 1h et 4h du matin ?

Solution :

Code 3 – Bus

```
def get_frequency(day, hour=0):
    if day == "Samedi" or day == "Dimanche":
        return 30
    # elif 1 <= hour <= 4:
    #    raise Exception("Le bus ne circule pas !")
    else:
        if 7 <= hour < 9 or 16 <= hour < 18:
            return 5
        else:
            return 10


if __name__ == "__main__":

    #  SCRIPT VERSION
    day = "Lundi"
    hour = 15
    if day == "Samedi" or day == "Dimanche":
        print(f"{day, hour} --> le bus passe toutes les 30 minutes."
            )
    elif 1 <= hour < 4:
        print(f"{day, hour} --> le bus ne circule pas.")
    else:
        if 7 <= hour < 9 or 16 <= hour < 18:
            print(f"{day, hour} --> le bus passe toutes les 5
                minutes.")
        else:
            print(f"{day, hour} --> le bus passe toutes les 10
                minutes.")

    #  FUNCTION VERSION

    days = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "
        Samedi", "Dimanche"]
```

```
    hours = [h for h in range(24)]

    for day in days:
        for hour in hours:
            print(f'({day, hour}) --> Le bus passe toutes les {
                get_frequency(day, hour)} minutes.')

    print(f'({"Mercredi", 7.75}) --> Le bus passe toutes les {
        get_frequency(day, hour)} minutes.')
```

A3. **Constante de Champernowne** La constante de Champernowne est un nombre réel univers qui se construit à partir de la suite croissante des entiers naturels :

$$C = 0,12345678910111213\ldots$$

(a) Écrire un script Python qui affiche les n premières décimales de ce nombre sous la forme suivante : `Champernowne constant 5 --> 0.12345`

(b) Transformer ce script en fonction Python. Le prototype est `champernowne(n)`. `n` est un type `int` et la fonction le nombre de Champernowne sous la forme d'une chaîne de caractère `str`.

(c) Tester la fonction dans un programme principal. Le test fera apparaître toutes le nombres comportant de 1 à 499 décimales.

Solution :

Code 4 – Champernowne

```
def champernowne(n):
    s = "0."
    i = 1
    while len(s) < n+1:
        s += str(i)
        i += 1
    return s


if __name__ == "__main__":

    # SCRIPT VERSION
    n = 50
    s = "0."
    i = 1
    while len(s) < n+2:
        s += str(i)
        i += 1
    print(f"Champernowne constant {n} --> {s}")
```

```
20
21      # FUNCTION VERSION
22      for i in range(1,500):
23          print(f"Champernowne constant {i} --> {champernowne(i)}")
```

B Des fonctions qui renvoient un résultat

R À partir de maintenant, on n'écrira plus que des fonctions pour résoudre les exercices. Les `print` se feront dans le programme principal sauf nécessité de débogage.

B1. **Suites** On s'intéresse aux suites $(u_n)_{n\in\mathbb{N}}$ définie par $u_n = 2^{\sqrt{n}}$, et $(v_n)_{n\in\mathbb{N}}$ définie par $v_n = \sqrt{2^n}$.

(a) À l'aide du module `math`, créer une fonction qui renvoi la valeur de la suite u_n à un indice donné. Le prototype de la fonction est `suite_u(n)` où `n` est un paramètre de type `int`. La fonction renvoie un type `float`.

(b) Créer une fonction qui renvoi la valeur de la suite v_n à un indice donné. Le prototype de la fonction est `suite_v(n)` où `n` est un paramètre de type `int`. La fonction renvoie un type `float`.

(c) On souhaite à présent comparer les deux suites. Créer une fonction qui renvoie la différence de ces deux suites pour un indice `n` donné. Cette fonction a pour prototype `diff(n)` où `n` est un paramètre de type `int`. Elle renvoie un type `float`.

(d) Dans le programme principal, en utilisant une boucle, faire afficher la différence des deux suites pour n variant de 0 à 15. Pouvez-vous justifier ces résultats ?

Solution :

Code 5 – Suites

```
1  from math import sqrt
2
3
4  def suite_u(n):
5      return 2 ** sqrt(n)
6
7
8  def suite_v(n):
9      return sqrt(2 ** n)
10
11
12 def diff(n):
13     return suite_u(n) - suite_v(n)
14
15
```

```python
if __name__ == "__main__":
    for i in range(15):
        print(f"u_{i} - v_{i} = {diff(i)}")
```

B2. **Sommes**

(a) Écrire une fonction qui renvoie le résultat de $\sum_{k=1}^{10^n} \frac{1}{k}$. Cette fonction a pour prototype `somme(n)` où `n` est un paramètre de type `int`. Elle renvoie un type `float`. Tester la fonction dans un programme principal.

(b) Écrire une fonction qui renvoie le résultat de $\left(\sum_{k=1}^{10^n} \frac{1}{k}\right) - \ln(10^n)$. Cette fonction a pour prototype `diff(n)` où `n` est un paramètre de type `int`. Elle renvoie un type `float`. Tester la fonction dans un programme principal en utilisant une boucle pour n variant de 1 à 7. Pourrait-on calculer cette différence pour des valeurs de n beaucoup plus grandes?

Solution : La durée du calcul de somme s'allonge exponentiellement avec n. On ne peut guère espérer calculer somme pour des n beaucoup plus grands dans un temps raisonnable.

Code 6 – Sommes

```python
from math import log


def somme(n):
    acc = 0
    for i in range(1, 10 ** n + 1):
        acc += 1 / i
    return acc


def diff(n):
    return somme(n) - log(10 ** n)


if __name__ == "__main__":
    for i in range(1, 7):
        print(f"#{i} --> {diff(i)}")
```

C Des procédures pour afficher

C1. **Console art.** On cherche à dessiner des motifs carrés de dimension quelconque sur la console Python avec des caractères ASCII. Pour cela, on va créer des fonctions Python dont

le paramètre est la taille du carré `n` de type `int` et qui dessine le motif sur la console. [1]

(a) Créer une fonction de prototype `block(n)` qui affiche un carré de côté n délimité par des étoiles contenant des points. Par exemple pour $n = 5$.

```
* * * * *
* . . . *
* . . . *
* . . . *
* * * * *
```

(b) Créer une fonction de prototype `square_diag(n)` qui trace les deux diagonales du carré avec des étoiles et le reste du carré par des points. Par exemple pour $n = 5$.

```
* . . . *
. * . * .
. . * . .
. * . * .
* . . . *
```

(c) Créer une fonction de prototype `losange(n)` qui trace un losange avec des étoiles et le reste du carré par des points. On veillera à ce que le paramètre d'entrée n soit impair. Les opérateurs division entière et modulo sont utiles! Par exemple pour $n = 5$.

```
. . * . .
. * . * .
* . . . *
. * . * .
. . * . .
```

Solution :

Code 7 – Console Art et procédures

```
def block(n):
    for row in range(n):
        for col in range(n):
            if row == 0 or row == n - 1 or col == 0 or col == n - 1:
                print("*", end=" ")
            else:
                print(".", end=" ")
        print()


def square_diag(n):
    for row in range(n):
        for col in range(n):
```

1. En fait, on devrait appeler ces fonctions des procédures car elles ne renvoient rien et ne font qu'afficher à l'écran. Python ne fait pas la distinction entre les deux.

```
            if row - col == 0 or row + col == n - 1:
                print("*", end=" ")
            else:
                print(".", end=" ")
        print()


def losange(n):
    assert n % 2 == 1
    for row in range(n):
        for col in range(n):
            if (row + col)%(n-1) == n // 2 or abs(row - col) == n //
                2:
                print("*", end=" ")
            else:
                print(".", end=" ")
        print()


if __name__ == "__main__":
    block(5)
    square_diag(5)
    losange(5)
```