Programmation structurée et fonctions

INFORMATIQUE COMMUNE - TP n° 1.2 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ indenter et enchaîner correctement les blocs Python
- ☞ coder une structure alternative et une condition
- ☞ 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

Au lycée, il est probable que vous ayez programmé des scripts. C'est une première étape. Le code 1 est un exemple de script. Le code 2 est un exemple de script qui utilise une fonction.

Code 1 – Exemple de script

```
x = 3
sqrx = x * x
print("Squaring 3 gives ", sqrx)
```

Code 2 – Exemple de fonction utilisée dans un script

```
def square(a):  # a est un paramètre formel
    x = a * a  # corps de la fonction
    return x  # valeur renvoyée (paramètre de sortie)

nine = square(3)  # 3 est un paramètre effectif
print("Squaring three gives ", nine)  # Le carré de 3 c'est 9...
```

Pour les concours, il est nécessaire de savoir programmer des fonctions. Celle-ci permettent de factoriser le code, de le scinder en petites unités et d'éviter les répétitions et donc les erreurs.

Ⓡ **On fera attention à ne pas utiliser de `print` dans les fonctions et réserver les `print` au programme principal. Une fonction permet de calculer un résultat qui dépend de paramètres d'entrée et renvoie le résultat de ce calcul (mot-clef `return`).**

B Fonctions avec alternatives

B1. **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 une fonction qui renvoie le tarif d'entrée de la piscine. Cette fonction a pour signature `prix(age: int, reduit: bool)-> float`. Les paramètres de la fonction permettent de tenir compte de l'âge du client et de la réduction éventuelle dont il peut bénéficier.

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

```
for age in range(4):
    print("Test age --> ", age)
    assert prix(age, False) == 0.0
    assert prix(age, True) == 0.0

for age in range(4, 18):
    print("Test age --> ", age)
    print(age)
    assert prix(age, False) == 2.1
    assert prix(age, True) == 2.1

for age in range(18,111):
    print("Test age --> ", age)
    assert prix(age, False) == 4.2
    assert prix(age, True) == 2.75
```

Solution :

Code 3 – Piscine

```
def prix(age, reduit):
    if age < 4:
        return 0.0
    elif 3 < age < 18:
        return 2.1
    else:
        return 2.75 if reduit else 4.2

# Programme princpal de test
for age in range(4):
    print("Test age --> ", age)
    assert prix(age, False) == 0.0
    assert prix(age, True) == 0.0

for age in range(4, 18):
    print("Test age --> ", age)
    print(age)
    assert prix(age, False) == 2.1
    assert prix(age, True) == 2.1
```

```
for age in range(18,111):
    print("Test age --> ", age)
    assert prix(age, False) == 4.2
    assert prix(age, True) == 2.75
```

B2. **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 une fonction qui renvoie la fréquence de passage des bus en fonction de l'heure et du jour de la semaine. Lorsque le bus ne circule pas, deux solutions sont possibles :

1. lever une exception, --→ HORS PROGRAMME c'est-à-dire interrompre l'exécution du programme en signalant qu'on ne peut pas renvoyer de résultat car les bus ne circulent pas. C'est la bonne pratique. Il suffit ensuite de gérer cette exception.
2. renvoyer `None`, qui est l'objet Python qui représente le rien. C'est une solution plus laxiste, typique de Python.

Vous implémenterez la **deuxième** solution.

(b) Tester le programme à l'aide du code suivant :

```
days = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "
    Dimanche"]
hours = [h for h in range(24)]

for day in days:
    for hour in hours:
        print(day, hour, "h --> Le bus passe toutes les ", frequence(day,
            hour), " minutes.")
```

Solution :

Code 4 – Bus

```
def frequence(jour, heure):
    if jour == "Samedi" or jour == "Dimanche":
        return 30
    elif 1 <= hour <= 4:
        return None
        #raise Exception("Le bus ne circule pas !") # hors programme
    else:
        if 7 <= heure < 9 or 16 <= heure < 18:
            return 5
        else:
            return 10

# TESTS
days = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "
    Dimanche"]
hours = [h for h in range(24)]

for day in days:
```

```
        for hour in hours:
            print(day, hour, "h --> Le bus passe toutes les ", frequence(day,
                hour), " minutes.")
```

C Fonctions, boucles et importations

C3. **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`, écrire une fonction de signature `u(n : int)-> float` qui renvoie la valeur u_n.

(b) Écrire une fonction de signature `v(n : int)-> float` qui renvoie la valeur v_n.

(c) On souhaite à présent comparer les deux suites. Créer une fonction de signature `diff(n : int )-> float` qui renvoie la différence de ces deux suites pour un indice n donné.

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

Solution :

Code 5 – Suites

```
from math import sqrt


def u(n):
    return 2 ** sqrt(n)


def v(n):
    return sqrt(2 ** n)


def diff(n):
    return u(n) - v(n)


for i in range(16):
    print("u_", i, " - v_", i, " = ", diff(i))
```

C4. **Sommes**

(a) Écrire une fonction qui renvoie le résultat de $\sum_{k=1}^{10^n} \frac{1}{k}$. Cette fonction a pour siignature `somme(n : int)-> 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 signature `diff(n : int)-> float`. Tester la fonction dans un programme principal en utilisant une boucle pour n variant de 1 à 7 **inclus**. 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

```
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)


for i in range(1, 9):
    print("#", i, " --> ", diff(i))
```

C5. **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 une fonction Python de signature `champernowne(n : int)-> str` qui renvoie le nombre de Champernowne sous la forme d'une chaîne de caractères. Par exemple, `champernowne(7)` renvoie `"0.1234567"`.

(b) Combien de décimales faut-il calculer pour voir apparaître la séquence "66666" dans la constante de Champernowne ? (procéder par force brute)

Solution : 1891

(c) Pourquoi appelle-t-on ce nombre un nombre univers ?

Solution : Parce que tous les livres possibles sont écrits dedans, puisqu'on peut trouver n'importe quel motif dedans pourvu qu'on aille assez loin dans les décimales…

Solution :

Code 7 – Champernowne

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

```
            i += 1
        return "0." + s[:n]


    def trouve_motif(motif):
        m = len(motif)
        s = ""
        i = 0
        while True: # ATTENTION : à ne pas écrire inconsciemment...
            s += str(i)
            mi = len(str(i))
            for k in range(mi):
                #print(i, k, s[len(s) - m - k:len(s) - k])
                if s[len(s) - m - k:len(s) - k] == motif:
                    return i, len(s) - 2 - k, "0." + s
            i += 1


    # TEST

    # version très peu efficace...
    i = 0
    while not ("66666" in champernowne(i)):
        i = i + 1
    print("Found --> ", i-1, champernowne(i))

    print(trouve_motif("66666"))
    print(trouve_motif("2025"))
```

D Console art : dessiner avec des boucles et des fonctions

Un caractère (une lettre) est codé en machine par un certain nombre prédéterminé figurant dans la table ASCII. On peut connaître le code associé à un caractère à l'aide de la commande `ord`. Par exemple, `ord('a')` renvoie la valeur 97 en décimal, ce qui, codé en binaire sera stocké en mémoire sous la forme `1100001`. Certains caractères sont spéciaux, par exemple le caractère de contrôle pour revenir à la ligne ou celui pour effectuer une tabulation. Le retour à la ligne (Line Feed ou LF) est noté `"\n"` et il faut noter qu'il ne représente qu'un seul caractère.

D6. **Console art.**

On cherche à dessiner des motifs carrés de dimension quelconque sur la console Python avec des caractères. Pour cela, on va créer des fonctions Python dont le paramètre est la taille du carré `n` de type `int` et qui renvoie le motif sous la forme d'une chaîne de caractère.

(a) Créer une fonction de signature `line(n :int)-> str` qui renvoie une chaîne de caractères représentant une ligne constituée de n fois le motif `%~`. Par exemple pour n = 20 :

%~

(b) Créer une fonction de signature `block(n : int)-> str` qui renvoie une chaîne de caractères représentant un carré de côté n délimité par des étoiles contenant des points. Par exemple pour

$n = 5$:

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

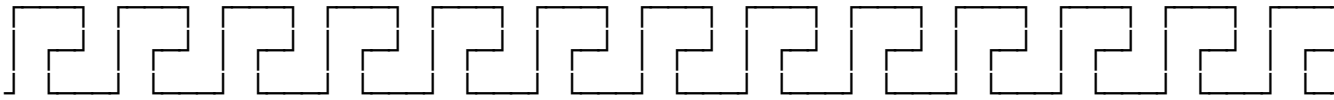
(c) Créer une fonction de signature `square_diag(n : int)-> str` qui renvoie une chaîne de caractères représentant les deux diagonales du carré avec des étoiles et le reste du carré par des points. Par exemple pour $n = 5$:

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

(d) Créer une fonction de signature `losange(n : int)-> str` qui renvoie une chaîne de caractères représentant 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 $n = 5$:

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

(e) Créer une fonction de signature `greek(n: int)-> str` qui renvoie une frise du motif grec repérésenté ci-dessous. Le paramètre dentrée `n` est le nombre de répétition du motif. Par exemple pour n = 13 :



On pourra utiliser les caractères suivants :

```
h_line = '─'
v_line = '│'
top_left = '┌'
top_right = '┐'
bottom_left = '└'
bottom_right = '┘'
```

Solution :

Code 8 – Console Art et procédures

```
def line(n):
```

```
        s = ""
        for c in range(n):
            s = s + "%~"
        s = s + "\n"
        return s


    def block(n):
        s = ""
        for row in range(n):
            for col in range(n):
                if row == 0 or row == n - 1 or col == 0 or col == n - 1:
                    s = s + "* "
                else:
                    s = s + ". "
            s = s + "\n"
        return s


    def square_diag(n):
        s = ""
        for row in range(n):
            for col in range(n):
                if row - col == 0 or row + col == n - 1:
                    s = s + "* "
                else:
                    s = s + ". "
            s = s + "\n"
        return s


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


    def greek(n):
        # h_line = ─''
        # v_line = │''
        # top_left = ┌''
        # top_right = ┐''
        # bottom_left = └''
        # bottom_right = ┘''
        # cross = ┼''
        s = ""
        # première ligne
```

```
        for k in range(n):
            s += "┌──┐ "
        s += "\n"
        # deuxième ligne
        for k in range(n):
            s += "│ ┌┘ "
        s += "\n"
        # troisième ligne
        for k in range(n):
            s += "┘ └──"
        s += "\n"
        return s


print(line(20))
print(block(5))
print(square_diag(5))
print(losange(5))
print(greek(13))
```