
TRAVAUX PRATIQUES n° 5

Fonctions

Dans ce TP, nous allons apprendre à écrire des fonctions réutilisables, puis les combiner pour calculer des dates du calendrier. L'objectif est de bien distinguer `return` et `print`, et de réutiliser une fonction déjà écrite plutôt que de recopier du code.

Il est important de tenir compte des bonnes pratiques de programmation suivantes :

★ **print ou return ?**

- Si l'énoncé vous demande de **renvoyer** un résultat, il faudra utiliser un `return` et non `print`.
- Si l'énoncé vous demande d'**afficher** un résultat, il faudra utiliser un `print` et non `return`.
- En cas de doute, on utilise un `return`.

★ **Éviter la redondance**

L'un des pires ennemis du programmeur est la redondance du code : on doit éviter le plus possible d'écrire plusieurs fois le même code. L'un des avantages principaux des fonctions en informatique est de résoudre ce problème : lorsqu'un morceau de code est destiné à être utilisé plusieurs fois, on l'écrit une seule fois dans une fonction que l'on réutilisera partout où l'on en a besoin.

★ **Tester les fonctions**

Après avoir codé une fonction, on doit s'assurer de son bon fonctionnement. Il faut :

- vérifier qu'il n'y a pas d'erreur à l'exécution ;
- tester la cohérence des résultats sur des exemples dans le shell (pas d'`input`).

I. Premières fonctions

Exercice 1 (*Échauffement : print ou return ?*) :

- 1) a) Exécuter la fonction `f` puis la tester dans le shell :

```
1 def f(x):  
2     return x**2 + 3*x - 1
```

Pour tester une fonction, on tape dans le shell `f(x)` avec différentes valeurs pour `x`. Par exemple `f(2)`.

- b) Exécuter la fonction `g` puis la tester dans le shell :

```
1 def g(x):  
2     print(x**2 + 3*x - 1)
```

- c) Taper ensuite dans le shell les instructions suivantes :

```
>>> f(4)  
>>> g(4)  
>>> x = f(4)  
>>> x  
>>> y = g(4)  
>>> y  
>>> x + y
```

d) À quoi sert l'instruction `return` dans la fonction `f` ? Quelle est la différence avec le `print` dans la fonction `g` ? Quel est le meilleur choix ?

2) Exécuter les fonctions `discriminant` et `solutions` puis les tester dans le shell :

```
1 from math import sqrt
2
3 def discriminant(a, b, c):
4     delta = b**2 - 4 * a * c
5     return delta
6
7 def solutions(a, b, c):
8     delta = discriminant(a, b, c)
9     if delta > 0:
10         x1 = (-b - sqrt(delta)) / (2 * a)
11         x2 = (-b + sqrt(delta)) / (2 * a)
12         return [x1, x2]
13     elif delta == 0:
14         x0 = -b / (2 * a)
15         return [x0]
16     else:
17         return []
```

On remarquera les points suivants :

- ★ Les paramètres d'une fonction ne sont pas fixés à l'écriture de la fonction : ils restent quelconques.
- ★ On n'utilise plus `input` pour donner des valeurs aux paramètres : on appelle la fonction avec des valeurs entre parenthèses.
- ★ Une fonction peut en utiliser une autre : `solutions` réutilise `discriminant`.

Exercice 2 : Écrire une fonction `factorielle(n)` prenant un argument $n \in \mathbb{N}$ et qui renvoie $n!$.

```
>>> factorielle(0)
1
>>> factorielle(10)
3628800
```

Exercice 3 : Écrire une fonction `somme_carres(n)` prenant un argument $n \in \mathbb{N}$ et qui renvoie

$$\sum_{k=1}^n k^2.$$

```
>>> somme_carres(0)
0
>>> somme_carres(100)
338350
```

Exercice 4 : On considère la suite $(u_n)_{n \geq 0}$ définie par

$$u_0 = 3 \quad \text{et} \quad \forall n \in \mathbb{N}, \quad u_{n+1} = 2u_n - 1.$$

Écrire une fonction `suite_u(n)` prenant un argument $n \in \mathbb{N}$ et qui renvoie u_n .

```
>>> suite_u(0)
3
>>> suite_u(10)
2049
```

II. Mini-projet : calendrier

Année bissextile

Une année $n \geq 1$ (l'an 0 n'existe pas) est **bissextile** si et seulement (n est divisible par 4 mais pas par 100) ou (n est divisible par 400). Ainsi 2000 et 2012 étaient bissextiles mais 2100 ne le sera pas.

Exercice 5 : Écrire une fonction `bissextile(annee)` prenant un argument `annee` $\in \mathbb{N}^*$ et qui renvoie `True` si l'année est bissextile et `False` sinon.

Bonus : faire une fonction en une seule ligne !

```
>>> bissextile(2024)
True
```

Exercice 6 : Écrire une fonction `nb_jours_annee(annee)` prenant un argument `annee` $\in \mathbb{N}^*$ et qui renvoie le nombre de jours de l'année.

On rappelle qu'il faut utiliser la fonction `bissextile`.

```
>>> nb_jours_annee(2024)
366
```

Exercice 7 : Écrire une fonction `nb_jours_mois(mois, annee)` prenant un argument `mois` $\in \llbracket 1, 12 \rrbracket$ et un argument `annee` $\in \mathbb{N}^*$ et qui renvoie le nombre de jours du mois dans l'année.

```
>>> nb_jours_mois(2, 2023)
28
>>> nb_jours_mois(2, 2024)
29
```

Exercice 8 : Écrire une fonction `nb_jours_ecoules(jour, mois, annee)` prenant un argument `jour` $\in \llbracket 1, 31 \rrbracket$, un argument `mois` $\in \llbracket 1, 12 \rrbracket$ et un argument `annee` $\in \mathbb{N}^*$ et qui renvoie le nombre de jours écoulés depuis le 1^{er} janvier de l'an 1 jusqu'à cette date.

Indication : sommer d'abord le nombre de jours des années complètes de 1 à `annee - 1`, puis le nombre de jours des mois complets de 1 à `mois - 1` dans l'année `annee`, puis ajouter `jour`.

```
>>> nb_jours_ecoules(24, 11, 2023)
738848
>>> nb_jours_ecoules(1, 12, 2023)
738855
```

Exercice 9 : Écrire une fonction `difference_de_jours(date1, date2)` prenant deux arguments `date1` et `date2` sous forme de liste de 3 entiers (par exemple `[1, 10, 2021]`) et qui renvoie le nombre de jours écoulés entre ces deux dates.

On rappelle que pour accéder aux éléments d'une liste, on utilise la syntaxe `liste[indice]`. Par exemple si `date = [24, 11, 2023]` alors `date[0] = 24`, `date[1] = 11` et `date[2] = 2023`.

Combien de jours avez-vous vécu ?

```
>>> difference_de_jours([1, 1, 2000], [24, 11, 2023])
8728
```

Exercice 10 : Écrire une fonction `date(n)` prenant un argument $n \in \mathbb{N}^*$ et qui renvoie la date, sous forme de liste de 3 entiers, pour laquelle le nombre de jours écoulés depuis le 1^{er} janvier de l'an 1 est n .

À quelle date se sera-t-il écoulé un million de jours depuis le 1^{er} janvier de l'an 1 ?

```
>>> date(738848)
[24, 11, 2023]
```

Exercice 11 : Sachant que le 1^{er} janvier de l'an 1 était un lundi, écrire une fonction `jour_de_la_semaine(jour, mois, annee)` prenant un argument `jour` $\in \llbracket 1, 31 \rrbracket$, un argument `mois` $\in \llbracket 1, 12 \rrbracket$ et un argument `annee` $\in \mathbb{N}^*$ et qui renvoie le jour de la semaine correspondant à cette date.

```
>>> jour_de_la_semaine(24, 11, 2023)
'Vendredi'
```

Exercice 12 (*Paraskevidékatriaphobe s'abstenir*) :

Écrire une fonction `prochain_vendredi_13(jour, mois, annee)` prenant un argument `jour` $\in \llbracket 1, 31 \rrbracket$, un argument `mois` $\in \llbracket 1, 12 \rrbracket$ et un argument `annee` $\in \mathbb{N}^*$ et qui renvoie la date du prochain vendredi 13.

```
>>> prochain_vendredi_13(24, 11, 2023)
[13, 9, 2024]
```