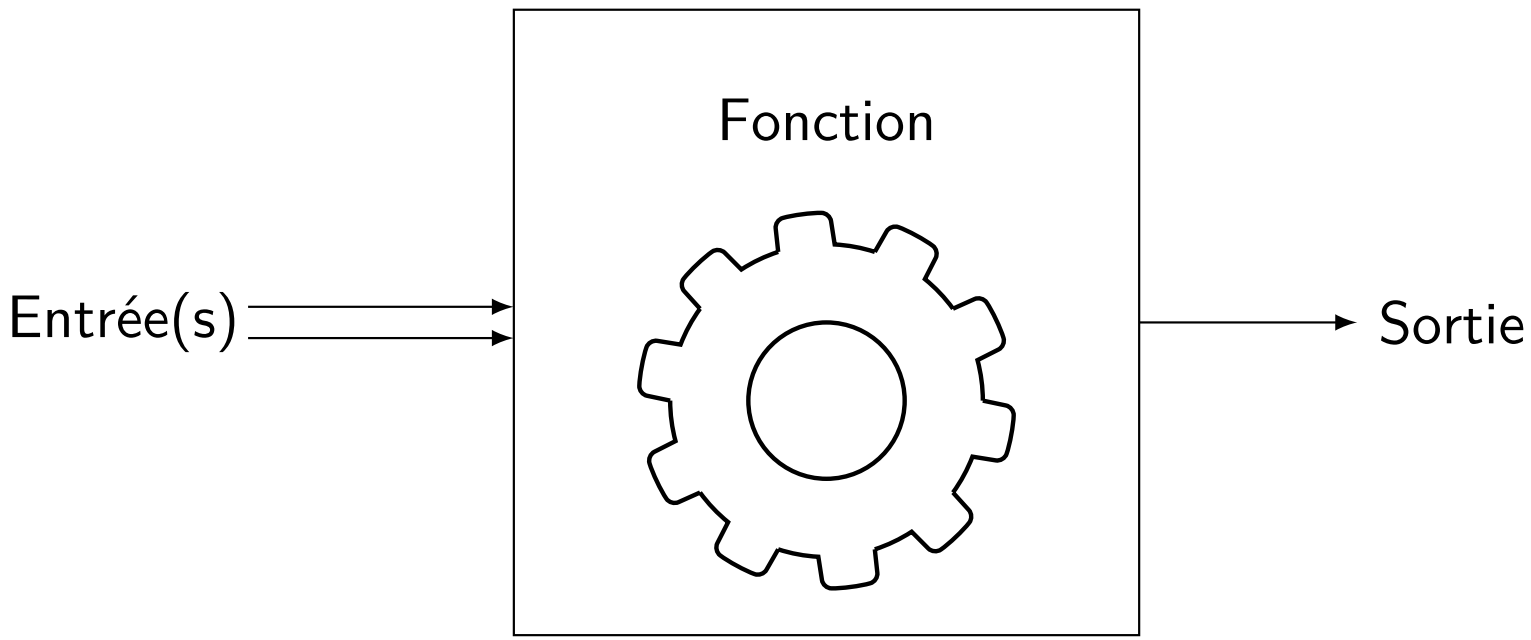
Fonctions

Informatique pour tous

Fonction

Définition

Une fonction (en informatique) a un **nom**, demande des éventuelles **entrées** (ou **arguments**), exécute des instructions et renvoie éventuellement une **sortie**.



Factorielle

Question

Comment calculer $n!$ $(= 1 \times 2 \times 3 \times ... \times n)$ en Python ?

Factorielle

Question

Comment calculer $n!$ $(= 1 \times 2 \times 3 \times ... \times n)$ en Python ?

```
f = 1
for i in range(2, n + 1):
    f = f * i
```

Factorielle

Si on a souvent besoin de calculer une factorielle, il peut être fastidieux de réécrire une boucle à chaque fois...

Factorielle

Si on a souvent besoin de calculer une factorielle, il peut être fastidieux de réécrire une boucle à chaque fois...

On peut définir une fonction factorielle :

```
def fact(n):
    res = 1
    for i in range(2, n + 1):
        res = res * i
    return res
```

Fonction factorielle

```
def fact(n):
    res = 1
    for i in range(2, n + 1):
        res = res * i
    return res
```

- `def` indique que l'on est en train de définir une fonction.

Fonction factorielle

```
def fact(n):
    res = 1
    for i in range(2, n + 1):
        res = res * i
    return res
```

- `def` indique que l'on est en train de définir une fonction.
- `fact` est le **nom** de la fonction.

Fonction factorielle

```
def fact(n):
    res = 1
    for i in range(2, n + 1):
        res = res * i
    return res
```

- `def` indique que l'on est en train de définir une fonction.
- `fact` est le **nom** de la fonction.
- `n` est un **argument** (l'entrée) de `fact`.

Fonction factorielle

```
def fact(n):
    res = 1
    for i in range(2, n + 1):
        res = res * i
    return res
```

- `def` indique que l'on est en train de définir une fonction.
- `fact` est le **nom** de la fonction.
- `n` est un **argument** (l'entrée) de `fact`.
- `return` **arrête** la fonction et **renvoie** une valeur (la sortie). Ici, `res` est la valeur renvoyée.

Fonction factorielle

Une fois la fonction connue de Python, on peut l'utiliser de la façon suivante :

```
In [25]: fact(3)
Out[25]: 6

In [26]: fact(4) == 24
Out[26]: True

In [27]: fact(2 + 2)
Out[27]: 24

In [28]: fact(3) + fact(4)
Out[28]: 30

In [29]: fact(fact(3))
Out[29]: 720
```

Fonction

Principaux intérêts des fonctions en informatique :

1. Réutiliser du code plutôt que de le réécrire à chaque fois : gain de temps.

Fonction

Principaux intérêts des fonctions en informatique :

1. Réutiliser du code plutôt que de le réécrire à chaque fois : gain de temps.
2. Avoir un code plus facile à comprendre : on peut souvent comprendre le rôle d'une fonction grâce à son nom ou son éventuel commentaire explicatif.

Lorsque l'on vous demande d'écrire une fonction dans une question de DS/concours, il faut souvent la réutiliser ensuite !

Fonctions mathématiques

On peut écrire les fonctions mathématiques, par exemple $f : x \longmapsto x^3 - 2\sqrt{x} + 1$:

```
def f(x):
    return x**3 - 2*x**0.5 + 1
```

Plusieurs arguments

Une fonction peut avoir un nombre quelconque d'arguments, qui doivent être séparés par des virgules.
Exemple : calculer la distance entre les points de coordonnées `x1`, `y1` et `x2`, `y2`.

Plusieurs arguments

Une fonction peut avoir un nombre quelconque d'arguments, qui doivent être séparés par des virgules.
Exemple : calculer la distance entre les points de coordonnées `x1`, `y1` et `x2`, `y2`.

```
def dist(x1, y1, x2, y2):
    return ((x1 - x2)**2 + (y1 - y2)**2)**0.5
```

Plusieurs arguments

Une fonction peut avoir un nombre quelconque d'arguments, qui doivent être séparés par des virgules.
Exemple : calculer la distance entre les points de coordonnées `x1`, `y1` et `x2`, `y2`.

```
def dist(x1, y1, x2, y2):
    return ((x1 - x2)**2 + (y1 - y2)**2)**0.5
```

```
In [31]: dist(1, 0, 0, 1)
Out[31]: 1.4142135623730951
```

⚠ Il faut donner tous les arguments à la fonction.

Plusieurs arguments

Une fonction peut avoir des arguments qui ne sont pas forcément des nombres.
Ex : on peut regrouper les coordonnées d'un point dans un 2-uplet.

Plusieurs arguments

Une fonction peut avoir des arguments qui ne sont pas forcément des nombres.
Ex : on peut regrouper les coordonnées d'un point dans un 2-uplet.

```
def dist(p1, p2):
    return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)**0.5
```

```
In [35]: dist((1, 0), (0, 1))
Out[35]: 1.4142135623730951
```

Aucun argument

Une fonction peut éventuellement ne pas avoir d'argument :

```
In [39]: def nombre_or():
    ...:     return (1 + 5**0.5)/2
    ...:

In [40]: nombre_or()
Out[40]: 1.618033988749895
```

⚠ Il faut quand même mettre des parenthèses pour appeler la fonction !

return

Une fonction peut retourner autre chose qu'un nombre.

```
def pair(n):
    return n % 2 == 0

if pair(26):
    # instructions
```

return

3 nombres x, y, z forment un triplet pythagoricien si $x^2 + y^2 = z^2$.

Question

Écrire une fonction déterminant si 3 nombres forment un triplet pythagoricien.

return

3 nombres x, y, z forment un triplet pythagoricien si $x^2 + y^2 = z^2$.

Question

Écrire une fonction déterminant si 3 nombres forment un triplet pythagoricien.

```
def pythagoricien(x, y, z):
    return x**2 + y**2 == z**2
```

return

Question

Écrire une fonction renvoyant le milieu de deux points.

return

Question

Écrire une fonction renvoyant le milieu de deux points.

```
def milieu(p1, p2):
    return ((p1[0] + p2[0])/2, (p1[1] + p2[1])/2)
```

```
In [53]: milieu((0, -1), (1, 0))
Out[53]: (0.5, -0.5)
```

Pas de return

Une fonction peut éventuellement ne pas avoir de `return` (elle s'arrête lorsque le bloc d'instructions est terminé).
Ex : une fonction pour sauter des lignes.

```
def sauter_lignes(n):
    for i in range(n):
        print("\n")

print("bla")
sauter_lignes(5)
print("bla")
```

`\n` est un caractère spécial pour sauter une ligne.

Pas de return

Une fonction peut éventuellement ne pas avoir de `return` (elle s'arrête lorsque le bloc d'instructions est terminé).
Ex : une fonction pour sauter des lignes.

```
def sauter_lignes(n):
    for i in range(n):
        print("\n")

print("bla")
sauter_lignes(5)
print("bla")
```

`\n` est un caractère spécial pour sauter une ligne.

En fait, une fonction sans `return` renvoie `None`.

Plusieurs returns

Une fonction peut contenir plusieurs `return`, mais qu'un seul sera exécuté (puisque la fonction s'arrête dès le premier `return` rencontré).

La fonction valeur absolue :

```
def abs(x):
    if x < 0:
        return -x
    else:
        return x
```

Plusieurs returns

Une fonction peut contenir plusieurs `return`, mais qu'un seul sera exécuté (puisque la fonction s'arrête dès le premier `return` rencontré).

La fonction valeur absolue :

```
def abs(x):
    if x < 0:
        return -x
    else:
        return x
```

(`abs` existe déjà en Python)

Résumé

Définition générale d'une fonction :

```
# avant la fonction

def nom_fonction(arg1, arg2, ...):
    # bloc d'instructions
    # avec d'eventuels
    # return

# après la fonction
```

Résumé

Définition générale d'une fonction :

```
# avant la fonction

def nom_fonction(arg1, arg2, ...):
    # bloc d'instructions
    # avec d'eventuels
    # return

# après la fonction
```

Appel de la fonction :

```
nom_fonction(a1, a2, ...)
```

Coefficient binomial

On peut utiliser une fonction à l'intérieur d'une fonction.
Par ex., calculer un coefficient binomial, défini par :

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

```
def binomial(k, n):
    return fact(n) / (fact(k) * fact(n-k))
```

Variable locale

Une variable définie à l'intérieur d'une fonction est dite **locale** à cette fonction.
Elle est accessible seulement à l'intérieur de cette fonction.

Variable locale

```
def fact(n):
    res = 1
    for i in range(2, n+1):
        res = res * i
    return res

print(res)
```

Variable locale

```python
def fact(n):
    res = 1
    for i in range(2, n+1):
        res = res * i
    return res

print(res)
```

```
  File "/home/qfortier/ownCloud/CPGE/
_Fonctions/functions.py", line 9, in
    print(res)
NameError: name 'res' is not defined
```

`res` est local à `fact` : elle ne peut pas être utilisée en dehors.

Variable locale

```
a = 3

def f():
    a = 7

f()
print(a) # affiche 3
```

Variable locale

```python
a = 3

def f():
    a = 7

f()
print(a) # affiche 3
```

`a = 7` créé une nouvelle variable locale.

Variable globale

Au contraire, une variable accessible partout dans le code est dite **globale**.
Pour utiliser une variable globale dans une fonction, il faut le préciser avec le mot clé `global` :

```
a = 3

def f():
    global a
    a = 7

f()
print(a) # affiche 7
```

Cependant, il faut éviter autant que possible l'utilisation de variable globale : elles rendent le code plus difficile à comprendre.

Copie des arguments

⚠ Une variable de type **int**, **bool**, **float** ou **string** passée en argument d'une fonction est **copiée** : elle est donc différente de la variable initiale !

Copie des arguments

```
def f(x):
    x = 2

y = 1
f(y)
print(y)
```

`print(y)` affiche 1

L'instruction `x = 2` n'a pas modifié `y`

Fonction comme argument

Il est possible d'avoir une fonction comme argument d'une autre fonction.

Exemple : calculer $\sum_{k=0}^{n} f(k)$.

```
def somme(f, n):
    s = 0
    for k in range(n + 1):
        s = s + f(k)
    return s
```

Fonction comme argument

Il est possible d'avoir une fonction comme argument d'une autre fonction.

Exemple : calculer $\sum_{k=0}^{n} f(k)$.

```
def somme(f, n):
    s = 0
    for k in range(n + 1):
        s = s + f(k)
    return s
```

```
In [11]: def carre(x):
    ...:     return x * x
    ...:

In [12]: somme(carre, 4)
Out[12]: 30

In [13]: 1 + 2*2 + 3*3 + 4*4
Out[13]: 30
```

Module

Python possède de nombreux modules, qui sont des ensembles de fonctionnalités autour du même « thème ».

Module

Python possède de nombreux modules, qui sont des ensembles de fonctionnalités autour du même « thème ».

Pour utiliser un module on peut écrire :

```
import nom_module
```

Il faut alors préfixer toutes les fonctions du module par « `nom_module.` »

Module

Par exemple, le module `numpy` regroupe des fonctions mathématiques pour le calcul scientifique, et beaucoup de fonctions mathématiques comme `cos`, `exp`, `log`...

```
import numpy

print(numpy.cos(numpy.pi/3))
# on obtient 0.5
```

Module

Il est possible d'importer un module en spécifiant un autre préfixe plus court :

```
import numpy as np

print(np.exp(np.log(6)))
# on obtient 6.0
```