Patrick Nsukami

1/12/24

Contents

Fonctions **3**

Pourquoi les fonctions **3**
- builtins . . . 3
- réutilisation . . . 3
- séparation, modularité . . . 3

Définition et appel de fonctions **4**
- Définition . . . 4
- Appel . . . 4

Le mot clé return **4**
- fonction . . . 4
- procédure . . . 5
- return . . . 5

Les paramètres **5**
- Paramètres positionnels . . . 5
- Paramètres optionnels: . . . 6
- Positional only parameters: . . . 6
- Keyword only parameters: . . . 7

Les paramètres spéciaux *args et **kwargs: **8**
- *args: . . . 8
- **kwargs: . . . 8

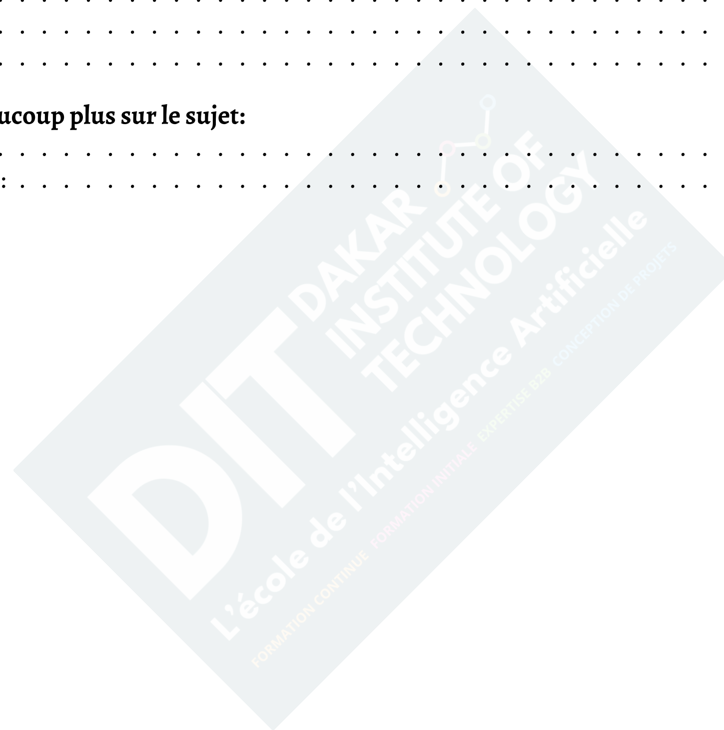
Docstrings: **8**

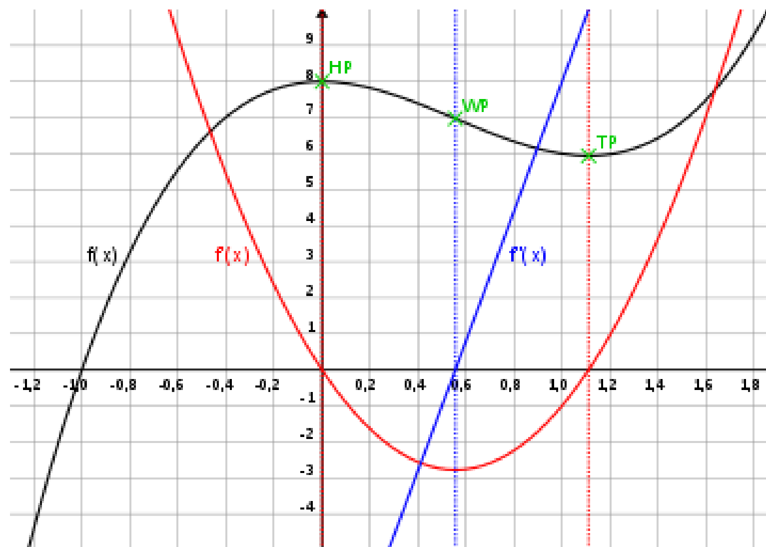
Fonctions anonymes: 8
lambda . . . 9
cas d'utilisation . . . 9
Attention . . . 9

Citoyens de première classe 9
stockage . . . 9
passage en tant qu'argument . . . 9
renvoi depuis une fonction . . . 9
definition ds une fonction . . . 9

Programmation fonctionnelle 9
Comprehension . . . 9
Fonctions d'ordre superieur: map, reduce, filter . . . 9
any, all . . . 9
itertools . . . 9
functools . . . 9

Pour en apprendre beaucoup plus sur le sujet: 9
Tutoriels à lire: . . . 9
Tutoriels à regarder: . . . 9





Fonctions

En programmation, une fonction est une suite d'instructions qui ensemble, effectuent une tâche bien précise. Une fonction peut potentiellement prendre des informations en entrée et potentiellement renvoyer un résultat en sortie.

Pourquoi les fonctions

Le rôle des fonctions est de permettre la modularisation, la réutilisation de code et la maintenance de nos programmes.

builtins

Les fonctions built-in ou fonction prédéfinies sont des fonctions disponibles à l'emploi au lancement de l'interpreteur. Pas besoin de bibliothèques à installer, pas besoin d'importer de modules.

Ces fonctions prédéfinies peuvent être utilisées pour écrire n'importe quel genre de programme sans avoir besoin de définir nos propres fonctions.

réutilisation

Une fonction peut être réutilisée, à plusieurs endroits du même programme, ou bien dans des programmes différents. Aulieu de réécrire les mêmes instructions plusieurs fois. réutiliser une fonction permet au développeur d'effectuer la même tâche rapidement. Parce qu'une fonction est réutilisable, cela nous fait gagner du temps, cela nous évite de faire des erreurs.

séparation, modularité

En divisant un programme en composants (en fonctions) plus courts et réutilisables, nous améliorons la compréhension, la collaboration et la maintenance de ce programme.

Il est beaucoup plus simple de debug et de comprendre un programme lorsque celui ci est organisé sous forme de fonctions, chacune d'elle avec un rôle spécifique.

DIT DAKAR INSTITUTE OF TECHNOLOGY L'école de l'Intelligence Artificielle

Définition et appel de fonctions

Nous rencontrerons des situations dans lesquelles aucune fonction prédéfinie ne résoudra notre problème. Nous rencontrerons des situations dans lesquels nous serons obligés de définir nos propres fonctions. Il existe un moyen de définir nos propres fonctions.

Définition

Avant de pouvoir *appeler* une fonction, il faut d'abord *définir* cette fonction. Voici la syntaxe pour définir une fonction:

```
# mot clé def suivi du nom de la fonction, suivi des paramètres potentiels
def function_name(param_1, param_2, param_n):
    # Corps de la fonction
    # Code devant être executé

    # valeur renvoyée potentiellement
    return output
```

Voici un exemple de fonction calculant le carré d'un nombre:

```
# définition
def square(number):
    return number ** 2
```

Appel

Nous ne pouvons appeler une fonction qu'après l'avoir définie. L'appel d'une fonction se fait en utilisant le nom de la fonction. Si la fonction possède des paramètres, il faudra appeler la fonction avec "des arguments":

```
# execution
result = square(5)

# affichage du résultat renvoyé
print(result)  # Output: 25
```

Le mot clé return

Il existe en programmation ce qu'on appelle des fonctions et ce qu'on appelle des procédures. Les deux concepts sont très similaires. Les deux entités se déclarent et s'executent de la même manière. Les deux representent un bloc de code, les deux entités effectuent chacune une certaine tâche. Cependant il existe une différence entre les deux. L'existence du mot clé return

fonction

Une fonction renvoie un résultat ...

```
# une fonction qui renvoie un entier
def bar():
    # remarquez la présence du mot clé return
    return 42
```

procédure

… Alors qu'une procédure ne fait rien d'autre qu'executer des instructions sans jamais renvoyer de résultat:

```
# une méthode (procédure est le mot le plus exact) (une fonction qui ne renvoie rien)
def bar():
    # remarquez l'absence du mot clé return
    f = open("./resultat", "w")
    f.write("42")
    f.close()
```

return

Le mot clé return peut apparaitre une ou plusieurs fois dans une fonction:

```
# remarquez l'apparition du mot clé return à 3 reprise
def f(x):
    if x < 0:
        return "negatif"
    elif x == 0:
        return "nul"
    else:
        return "positif"
```

Les paramètres

Les paramètres sont des variables définies dans l'entête de la fonction. Elles permettent à la fonction de recevoir des valeurs en entrée lors de l'appel de la dite fonction. Grâce aux paramètres, nous pourront écrire des fonctions qui sont beaucoup plus flexibles.

Paramètres positionnels

On appelle paramètre positionnel (ou paramètre requis), le paramètre qui recevra son argument en fonction de sa position.

1. Premier exemple:

```
# greet est une procédure qui possède un paramètre positionnel: name
def greet(name):
    print("Hello, " + name + "!")

# Lors de l'appel de la fonction,
# Alice est un argument qui sera passé au paramètre name "par position"
greet("Alice")
```

2. Deuxième exemple:

```
# add est une fonction qui possède deux paramètres positionnels
# et renvoie la somme de ces deux paramètres
def add(num1, num2):
    sum = num1 + num2
    return sum

# Lors de l'appel de la fonction,
# 10 et 5 sont les arguments passés à la fonction add, "par position"
# result est une variable contenant la valeur renvoyée par la fonction add
result = add(10, 5)
print(result) # Output: 15
```

3. Troisième exemple:

Les paramètres positionnels sont aussi appelés paramètres obligatoires. Lors de l'appel de la fonction, nous devons "obligatoirement" passer des arguments. Dans le cas contraire, nous recevrons un message d'erreur:

```
In [6]: add()
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 add()

TypeError: add() missing 2 required positional arguments: 'num1' and 'num2'
```

Paramètres optionnels:

Les paramètres sont dits optionnels lorsque durant l'appel de la fonction, nous ne sommes pas obligés de leurs passer une valeur:

3. Exemple montrant l'utilisation de paramètres optionnels:

```
# num2 est un paramètre optionel, remarquez sa valeur par défaut, 1
def multiply_numbers(num1, num2=1):
    product = num1 * num2
    return product

result1 = multiply_numbers(5) # par défaut, num2 est égal à 1

# l'argument passé pdt l'appel aura priorité sur la valeur par défaut
result2 = multiply_numbers(5, 3)
print(result1, result2)
```

Positional only parameters:

Les `positional-only parameters` ont été introduit dans la version 3.8 de Python. Ils nous permettent de définir des paramètres qui ne pourront recevoir leurs arguments que d'une seule manière: **par position**. Autrement dit, il ne sera **pas possible de nommer le paramètre lors de l'appel de la fonction.**

Jusqu'à présent, voici ce dont nous avons l'habitude:

```
In [1]: def hello(name): # name est un paramètre positionnel
   ...:     return f"Hello, {name}, how are you doing today ?"
   ...:

In [2]: hello("Nsukami")  # je passe l'argument par position
Out[2]: 'Hello, Nsukami, how are you doing today ?'

In [3]: hello(name="Nsukami") # je passe l'argument par nom, tout va bien également
Out[3]: 'Hello, Nsukami, how are you doing today ?'
```

Maintenant, définissons notre paramètre comme étant `positional only`:

```
In [5]: def hello(name, /): # remarquez le slash
   ...:     return f"Hello, {name}, how are you doing today ?"
   ...:

In [6]: hello("Nsukami") # appel par position
Out[6]: 'Hello, Nsukami, how are you doing today ?'

In [7]:

In [7]: hello(name="Nsukami") # tentative d'appel par le nom
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 hello(name="Nsukami")

TypeError: hello() got some positional-only arguments passed as keyword arguments: 'name'
```

> **Utilité:** Forcer l'utilisateur de votre programme à effectuer les appels de fonctions d'une manière bien spécifique. Cette fonctionnalité permet aussi de maintenir ce qu'on appelle la rétrocompatibilité d'un programme.

Keyword only parameters:

In Python, keyword-only parameters are a type of function parameter that can only be passed by using their name, and not by position. This means that when calling a function, you must explicitly specify the parameter name along with its corresponding value.

To define keyword-only parameters in Python, you can use the `*` symbol in the function signature. Any parameters after the `*` symbol will be keyword-only parameters.

Here's an example to illustrate the concept:

```
def greet(name, *, message):
    print(f"{message}, {name}!")

greet("Alice", message="Hello")  # Output: Hello, Alice!
```

In the above example, the `greet` function has a keyword-only parameter `message` which can only be passed using its name. When calling the function, you need to specify the parameter name (`message`) along with its value (`"Hello"`). If you try to pass the `message` parameter positionally, it will result in a `TypeError`.

Here's another example with multiple keyword-only parameters:

```
def calculate_total(*items, discount, tax_rate=0.1):
    total = sum(items)
    total -= total * discount
    total += total * tax_rate
    return total

print(calculate_total(10, 20, 30, discount=0.2, tax_rate=0.15))  # Output: 41.4
```

In this example, the `calculate_total` function takes a variable number of positional arguments (`*items`) and two keyword-only parameters (`discount` and `tax_rate`). The keyword-only parameters must be passed using their names (`discount` and `tax_rate`). In the function call, the `discount` is set to `0.2` and the `tax_rate` is set to the default value `0.1`.

Keyword-only parameters are useful when you want to enforce the use of parameter names for clarity, prevent accidental positional parameter passing, or when you want to provide default values for some parameters while requiring explicit naming for others.

> **Utilité:** Ici aussi, forcer l'utilisateur de votre programme à effectuer les appels de fonctions d'une manière bien spécifique. Cette fonctionnalité permet aussi de maintenir ce qu'on appelle la rétrocompatibilité d'un programme.

Les paramètres spéciaux *args et **kwargs:

*args:

**kwargs:

Docstrings:

On appelle Python docstring la chaine de caractère située immédiatement après l'entête de la fonction. Elle sert à "documenter" la fonction. Attention, les docstrings s'utilisent aussi pour les modules, les classes et les méthodes.

```
# remarquez les 3 doubles quotes au début puis à la fin
def function():
    """Do nothing, but document it.


    No, really, it doesn't do anything.
    """
    pass
```

Fonctions anonymes:

Les fonction anonymes.

lambda

cas d'utilisation

Attention

Attention aux fonctions anonymes.

Citoyens de première classe

stockage

passage en tant qu'argument

renvoi depuis une fonction

definition ds une fonction

On dit en Python que les fonction sont "first class citizen" ou citoyens de première classe.

Programmation fonctionnelle

Comprehension

Fonctions d'ordre superieur: map, reduce, filter

any, all

itertools

functools

Pour en apprendre beaucoup plus sur le sujet:

En espérant que ce cours vous aura appris quelque chose, retenez que le sujet est vaste et qu'il y a encore beaucoup à découvrir. Pour en apprendre d'avantage, nous vous recommendons vivement de consulter les ressources suivantes.

Tutoriels à lire:

- Functions
- Scripting
- More on functions
- Variable number of arguments
- Anonymous functions
- Docstring in Python

Tutoriels à regarder:

- Notebooks
- Functions
- Function call & argument passing
- Scope variables

"La vie est l'ensemble des fonctions qui résistent à la mort." Proverbe