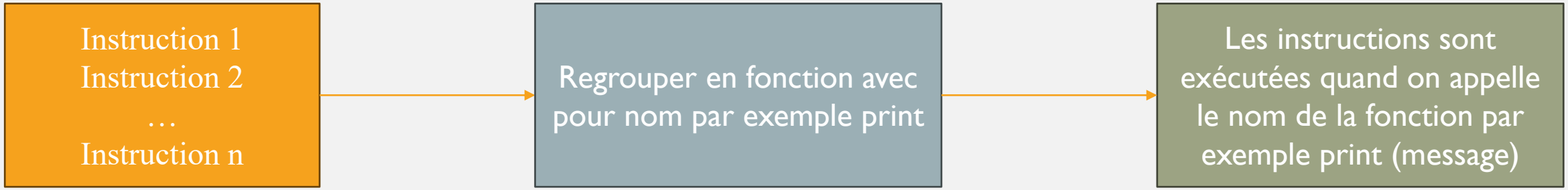
PROGRAMMATION PYTHON

Chapitre 5: Fonctions

Sommaire

Introduction

Les fonctions permettent de regrouper plusieurs instructions dans un bloc qui sera appelé grâce à un nom. Par exemple **print** est une fonction qui contient un groupe d'instructions pour afficher un message à l'écran.



On appelle module un groupe de fonctions stockées dans un ou plusieurs fichiers par exemple les fonctions mathématiques peuvent être placées dans un module dédié aux mathématiques.

Python propose des modules et fonctions de base comme le module **math, collections…** Dans un projet on aura besoin d'écrire des modules et fonctions spécifiques selon nos besoins et notre organisation d'où l'objet de ce chapitre.

Syntaxe de fonction

Une fonction en Python s'écrit dans un fichier **.py** avec la syntaxe suivante:

```
def nom_de_la_fonction(paramamètre1, paramètreN):
    Instruction 1
    Instruction 2
    Instruction 3
```

- **def:** mot clé obligatoire qui est l'abréviation de « define » (définir, en anglais) qui permet de spécifier à Python que l'on veut définir une fonction.
- Le nom de la fonction qui doit respecter les mêmes règles de nommage qu'une variable.
- La liste des paramètres qui seront fournis lors d'un appel à la fonction. Les paramètres sont séparés par une virgule et la liste est encadrée par des parenthèses. Une fonction peut avoir un ou plusieurs paramètres par contre les parenthèses sont obligatoires même si la fonction n'attend aucun paramètre.
- Les deux points sont obligatoires ils spécifient la fin de la définition de la fonction.
- Avant l'écriture des instructions, il faut un décalage appelé indentation pour séparer les instructions de la fonction du reste du code.

Après définition pour exécuter la fonction il suffira d'appeler le nom de la fonction pour qu'elle exécute ses instructions.

Exemple de fonctions

Création d'une fonction **hello** qui ne prend aucun paramètre et qui affiche le message **Bonjour tout le monde.**

```
>>> def hello():
...     print("Bonjour tout le monde")
...
...
>>> # Exécution
>>> hello()
Bonjour tout le monde
>>> hello()
Bonjour tout le monde
```

Modifions la fonction pour prendre un paramètre chaine de caractères appelé **nom** et afficher le message **Bonjour nom_paramètre !**

```
>>> def hello(nom):
...     print(f"Bonjour {nom} !")
...
...
>>> # Exécution
>>> hello("Christophe")
Bonjour Christophe !
>>> hello("Oscar")
Bonjour Oscar !
```

Paramètres d'une fonction

Lors de l'appel d'une fonction on dispose de deux manières pour spécifier les paramètres, **les paramètres non nommés et les paramètres nommés.**

Dans le cas où l'on utilise plusieurs paramètres sans les nommer il faut respecter l'ordre d'appel des paramètres sinon le programme fournira un résultat différent.

```
>>> def nomComplet(nom, prenom):
...     print(f"Le nom est {nom} et le prénom {prenom}.")
...
...
>>> # Appel en respectant l'ordre
>>> nomComplet("Doé", "John")
Le nom est Doé et le prénom John.
>>> # Appel en ne respectant pas l'ordre
>>> nomComplet("John", "Doé")
Le nom est John et le prénom Doé.
```

Si on veut appeler les paramètres en désordre il faut préciser le nom lorsqu'on appelle de la fonction.

```
>>> def nomComplet(nom, prenom):
...     print(f"Le nom est {nom} et le prénom {prenom}.")
...
...
>>> # Appel en désordre
>>> nomComplet(prenom="John", nom="Doé")
Le nom est Doé et le prénom John.
>>> nomComplet(nom="Doé", prenom="John")
Le nom est Doé et le prénom John.
```

Paramètres avec valeurs par défaut

On peut fournir à une fonction un ou plusieurs paramètres avec des valeurs par défaut. Lors de l'appel de la fonction si une valeur est spécifiée au paramètre cette valeur sera prise lors de l'exécution sinon la valeur par défaut est utilisée. Un paramètre avec une valeur par défaut est appelé **paramètre optionnel.**

```
>>> def hello(nom="Christophe"):
...     print("Bonjour", nom, "!")
...
...
>>> # Appel sans spécifier le paramètre
>>> hello()
Bonjour Christophe !
>>> # Appel en spécifiant le paramètre
>>> hello("Toto")
Bonjour Toto !
>>> hello(nom="tout le monde")
Bonjour tout le monde !
```

```
>>> def nomComplet(prenom, nom="Doé"):
...     print(f"Votre nom est {nom} et prénom {prenom} !")
...
...
>>> # Appel sans le paramètre optionnel
>>> nomComplet("John")
Votre nom est Doé et prénom John !
>>> nomComplet(prenom="John")
Votre nom est Doé et prénom John !
>>> # Appel avec le paramètre optionnel
>>> nomComplet("John", "Toto")
Votre nom est Toto et prénom John !
```

Quand une fonction contient des paramètres optionnels et obligatoires, il faut définir les paramètres optionnels après les paramètres obligatoires à la fin sinon on a une erreur.

```
def nomComplet(nom="Doé", prenom):
    print(f"Votre nom est {nom} et prénom {prenom} !")

SyntaxError: non-default argument follows default argument
```

Documenter une fonction

Quand on exécute dans l'interpréteur Python la fonction **help(nom_fonction)** on obtient un résultat comme:

```
help(print)
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

Le résultat affiché est la documentation sur la fonction **print** qui explique le rôle de la fonction et comment l'utiliser.

Pour documenter nos propres fonctions Python il faut respecter la syntaxe suivante:

```
def nom_de_la_fonction(paramamètre1, paramètreN):
    """
        Documentation de la fonction à l'intérieur les guillemets triples.
    """
    Instruction 1
    Instruction 2
    Instruction 3
```

La chaîne de la documentation est appelée **docstring** et sera affichée toutefois qu'on invoquera le nom de la fonction avec **help.**

Exemple de fonctions avec documentation

Création d'une documentation de la fonction **hello.**

```
def hello(nom="Christophe"):
    """
        Fonction qui permet d'afficher le message "Bonjour {nom en paramètre}" par défaut le message affiché sera "Bonjour Christophe".
    """
    print(f"Bonjour {nom}")


help(hello)
Help on function hello in module __main__:

hello(nom='Christophe')
    Fonction qui permet d'afficher le message "Bonjour {nom en paramètre}" par défaut le message affiché sera "Bonjour Christophe".
```

Une documentation peut être longue ou courte tout dépend de celui qui écrit la documentation de la fonction. Cependant il est conseillé de bien documenter nos fonctions pour faciliter la compréhension et l'utilisation par d'autres.

Exemple de fonctions avec documentation

Création d'une documentation évoluée de la fonction **hello.**

```
def hello(nom="Christophe"):
    """
        Nom
        -----
        hello

        Paramètre
        ---------
        nom (string optionnel): le nom de la personne par défaut sera 'Christophe'

        Description
        -----------
        Fonction qui permet d'afficher le message "Bonjour {nom en paramètre}" par défaut le message affiché sera "Bonjour Christophe".

        Exemple
        -------
        hello() => Affichera "Bonjour Christophe"
        hello("Toto") => Affichera "Bonjour Toto"
    """
    print(f"Bonjour {nom}")
```

```
help(hello)
Help on function hello in module __main__:

hello(nom='Christophe')
    Nom
    -----
    hello

    Paramètre
    ---------
    nom (string optionnel): le nom de la personne par défaut sera 'Christophe'

    Description
    -----------
    Fonction qui permet d'afficher le message "Bonjour {nom en paramètre}" par défaut le message affiché sera "Bonjour Christophe".

    Exemple
    -------
    hello() => Affichera "Bonjour Christophe"
    hello("Toto") => Affichera "Bonjour Toto"
```

L’instruction return

On dispose de deux types de fonctions:

- Des fonctions qui ne renvoient rien appelées **méthodes** dans certains langages comme la fonction **print** qui ne fait qu’afficher un message à l’écran.
- Des fonctions qui renvoient une valeur après exécution de leurs instructions comme la fonction **input** qui renvoie la valeur saisie au clavier par l’utilisateur.

Pour renvoyer une valeur dans une fonction en Python il faut utiliser le mot clé **return** suivie de la valeur qu’on veut renvoyer.

```
def nom_de_la_fonction(paramamètre1, paramètreN):
    """
        Documentation de la fonction à l'intérieur les guillemets triples.
    """
    Instruction 1
    Instruction 2
    Instruction 3

    return resultat
```

Exemple

```
def addition(nombre1, nombre2):
    """
        Fonction qui fait donne le résultat de l'addition de deux nombres.
    """
    resultat = nombre1 + nombre2
    return resultat

addition(2,3)
5
n1=2
n2=2
somme = addition(n1, n2)
print(f"{n1} + {n2} = {somme}")
2 + 2 = 4
```

Toute instruction de la fonction qui doit s’exécuter après exécution de return n’est pas exécutée car return met fin à l’exécution de la fonction.

Les fonctions lambda

Le mot clé **lambda** permet de créer des fonctions extrêmement courtes. Pour utiliser **lambda,** il faut utiliser la syntaxe suivante:

lambda arg1, arg2,… : instruction de retour

La fonction **lambda** doit être assignée à une variable on parle de **fonction anonyme** car on stocke ici une fonction dans une variable sans la définir avec le mot clé **def**.

```
>>> carre = lambda x: x * x
>>> carre(2)
4
>>> carre(3)
9
>>> # Equivalent avec def
>>> def carre2(x):
...     return x * x
...
>>> carre2(2)
4
>>> carre2(3)
9
```

Il faut préférer des fonctions **lambda** au lieu d'une fonction avec **def** quand la fonction ne contient qu'une seule instruction.

Exercices

Une fonction en Python s’écrit dans un fichier **.py** avec la syntaxe suivante:

FIN CHAPITRE 5