Faculté des Sciences
كلية العلوم
Université Ibn Tofaïl
جامعة ابن طفيل

Département d'Informatique
Filière MIP - Semestre 2
2024-2025

Module : Informatique II (Algorithmique II / Python)

Chapitre 2

Fonctions et Procédures

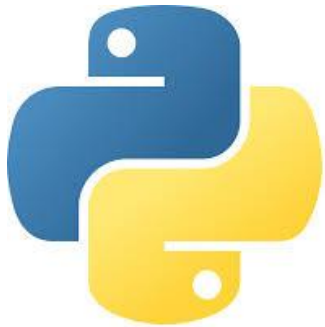
```python
#Définition des fonctions
def info():
    """ Informations """
    print("Touche q pour quitter")
    print("Touche Enter pour continuer")

def tirage_de():
    """ retourne un nombre entier aléatoire entre 1 et 6 """
    import random   #Charge le module random
    valeur = random.randint(1,6)
    #Utilise la fonction randint du module random
    return valeur

#Début du programme
info()
while True:
    choix=input()
    if choix == 'q':
        break
    print("Tirage :", tirage_de())
```

Pr. Khadija Louzaoui

Fonctions et Procédures En Python



Introduction aux fonctions et Procédures en Python

- En Python, une fonction est un bloc de code réutilisable qui effectue une tâche spécifique. Elle peut prendre des paramètres en entrée et retourner une valeur en sortie.
- Une procédure est simplement une fonction qui ne retourne pas de valeur explicite (None par défaut).
- Pourquoi utiliser des fonctions ?
  - Réutilisation du code : Évite la duplication du code.
  - Modularité : Divise un programme en petits blocs de code compréhensibles.
  - Facilité de maintenance : Permet de modifier une seule fonction au lieu de plusieurs parties du code.

Déclaration et Appel d'une Fonction en Python

❑ En **Python**, on définit une fonction avec le mot-clé "**def** ".

```
def nom_de_la_fonction(param1, param2):
    # Corps de la fonction
    # Code à exécuter
    return resultat  # Optionnel
```

Exemple une fonction qui retourne une valeur :

```
def addition(a, b):
    return a + b

# Appel de la fonction
resultat = addition(3, 5)
print(resultat)  # Affiche 8
```

- **def addition(a, b)**: Définit une fonction avec deux paramètres a et b.
- **return a + b**: Retourne la somme des deux nombres.
- **addition(5, 3)**: Appelle la fonction et stocke le résultat.

Déclaration et Appel d'une Fonction en Python

❑ Les annotations de type en Python sont utilisées pour spécifier les types attendus des arguments d'une fonction ainsi que le type de retour.

Exemple

```
def addition(a:int, b:int)->int:
    return a + b
```

- **a: int** et **b: int** indiquent que les arguments **a** et **b** doivent être des **entiers**.
- **-> int** indique que la fonction retourne un **entier**.

Paramètres et arguments

- **Paramètres :** Ce sont les variables définies dans la fonction.
- **Arguments :** Ce sont les valeurs passées aux paramètres de la fonction lors de son appel.

Exemple

```
def addition(a, b):
    return a + b

# Appel de la fonction
resultat = addition(3, 5)
print(resultat)  # Affiche 8
```

- **Paramètres : a, b**
- **Arguments : 3, 5**

Paramètres et arguments

Paramètres avec valeurs par défaut

Vous pouvez définir des paramètres avec des valeurs par défaut.

Exemple

```
def saluer(nom="inconnu"):
    print("Salut,",nom,"!")

saluer("Ahmed")  # Affiche "Salut, Ahmed !"
saluer()         # Affiche "Salut, inconnu !"
```

Si aucun argument n'est fourni, **nom** prend la valeur "**inconnu**".

```
def saluer(nom):
    print("Salut,",nom,"!")

saluer("Ahmed")  # Affiche "Salut, Ahmed !"
saluer()       # Affiche "Salut, inconnu !"
```

```
Salut, Ahmed !

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[24], line 5
      2     print("Salut,",nom,"!")
      4 saluer("Ahmed")  # Affiche "Salut, Ahmed !"
----> 5 saluer()

TypeError: saluer() missing 1 required positional argument: 'nom'
```

Paramètres et arguments

Paramètres nommés

Les **paramètres nommés** (ou **Keyword Arguments**) en Python permettent de spécifier les arguments d'une fonction en utilisant leur nom, plutôt qu'en suivant strictement leur ordre. Cela rend le code plus lisible et permet de ne passer que certains arguments, avec des valeurs par défaut.

Exemple

```
def saluer(nom, message="Bonjour"):
    print(message,",",nom,"!")

saluer("Ahmed")              # Utilisation du paramètre par défaut pour "message"
saluer("Marwa", message="Salut") # Utilisation d'un argument personnalisé pour "message"
```

```
Bonjour , Ahmed !
Salut , Marwa !
```

Paramètres et arguments
Paramètres nommés

```
def saluer(nom, message="Bonjour"):
    print(message,",",nom,"!")

saluer("Ahmed")                # Utilisation du paramètre par défaut pour "message"
saluer("Marwa", message="Salut") # Utilisation d'un argument personnalisé pour "message"
```

```
Bonjour , Ahmed !
Salut , Marwa !
```

- message="Bonjour" est un paramètre avec une valeur par défaut. Si aucun argument n'est passé pour message, Python utilise la valeur par défaut "Bonjour".
- Lorsque vous appelez la fonction, vous pouvez spécifier l'argument dans un ordre différent ou ne pas spécifier certains paramètres, en laissant Python utiliser les valeurs par défaut.

Paramètres et arguments
Paramètres nommés

Exemple

```
def afficher_info(prenom, age, ville="Rabat"):
    print(f"{prenom} a {age} ans et vit à {ville}.")

# Utilisation de paramètres nommés :
afficher_info(age=25, prenom="Ahmed", ville="Kénitra")
afficher_info(prenom="Marwa",age=18)
```

```
Ahmed a 25 ans et vit à Kénitra.
Marwa a 18 ans et vit à Rabat.
```

Paramètres et arguments

Arguments variables (*args et ** kwargs)

En Python, il existe deux types d'arguments variables qui permettent à une fonction de recevoir un nombre variable d'arguments :

- ***args :** permet de passer un nombre variable d'arguments non nommés (positionnels).
- ****kwargs :** permet de passer un nombre variable d'arguments nommés (avec clé-valeur).

Paramètres et arguments

Arguments variables (*args et ** kwargs)

***args** permet de passer un nombre indéfini d'arguments positionnels à une fonction. Ces arguments sont capturés sous forme de **tuple** à l'intérieur de la fonction.

Exemple

```
def somme(*args):
    return sum(args)

print(somme(1, 2, 3))          # 6
print(somme(4, 5, 6, 7,8,9))      # 39
```

*args capture tous les arguments passés à la fonction sous forme de **tuple**. On les manipuler comme un tuple classique, ici en les additionnant avec sum().

Paramètres et arguments
Arguments variables (*args et ** kwargs)

Un **tuple** en Python est une structure de données qui ressemble à une liste, mais avec une différence importante : les éléments d'un tuple sont **immuables**, c'est-à-dire qu'on ne peut pas les modifier après leur création.

```
mon_tuple = (1, 2, 3, "Python")
print(mon_tuple)
```

```
(1, 2, 3, 'Python')
```

Paramètres et arguments
Arguments variables (*args et ** kwargs)

****kwargs** permet de passer un nombre indéfini d'arguments nommés (c'est-à-dire des arguments sous forme de paires clé-valeur). Ces arguments sont capturés sous forme de **dictionnaire**.

Exemple

```
def afficher_info(**kwargs):
    for cle, valeur in kwargs.items():
        print(f"{cle}: {valeur}")

afficher_info(nom="Ahmed", age=25, ville="Kénitra")
```

```
nom: Ahmed
age: 25
ville: Kénitra
```

kwargs capture les arguments passés sous forme de paires clé-valeur et les transforme en **dictionnaire

Paramètres et arguments
Arguments variables (*args et ** kwargs)

En Python, un **dictionnaire** est une collection non ordonnée d'éléments qui sont stockés sous forme de paires clé-valeur. Voici un exemple de syntaxe de dictionnaire :

```
mon_dictionnaire = {
    "nom": "Ahmed",
    "âge": 30,
    "profession": "Professeur"
}
```

Dans cet exemple :
"nom", "âge", et "profession" sont les clés.
"Ahmed", 25, et "Professeur" sont les valeurs associées à ces clés.

Paramètres et arguments

Arguments variables (*args et ** kwargs)

Une fonction peut accepter à la fois des arguments positionnels et des arguments nommés. Si vous utilisez ***args** et ****kwargs**, assurez-vous que *args vient avant **kwargs.

```
def presentation(*args, **kwargs):
    print("Arguments positionnels:", args)
    print("Arguments nommés:", kwargs)

presentation(1, 2, 3, nom="Ahmed", age=25)
```

```
Arguments positionnels: (1, 2, 3)
Arguments nommés: {'nom': 'Ahmed', 'age': 25}
```

Vous pouvez mélanger des arguments positionnels, des arguments nommés, *args et **kwargs dans une même fonction, mais vous devez respecter un ordre spécifique : les arguments positionnels viennent en premier, suivis de *args, puis des arguments nommés et enfin de **kwargs.

Portée des Variables (Local et Global)

La portée d'une variable désigne la zone dans laquelle elle est accessible. En général, les variables définies à l'intérieur d'une fonction sont locales et ne sont pas accessibles à l'extérieur.

Exemple

```
def ma_fonction():
    x = 10  # Variable locale
    return x

print(ma_fonction())  # Affiche 10
print(x)  # Erreur: x n'est pas défini ici
```

```
10

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[54], line 6
      3     return x
      5 print(ma_fonction())  # Affiche 10
----> 6 print(x)

NameError: name 'x' is not defined
```

Portée des Variables (Local et Global)

Utiliser une variable globale dans une fonction

Exemple

```
x = 5  # Variable globale

def modifier_x():
    global x
    x = 10  # Modification de la variable globale

modifier_x()
print(x)  # Affiche 10
```

```
10
```

Attention ! Modifier des variables globales directement est déconseillé.

Fonction Lambda (Anonyme)

Les fonctions **lambda** sont des fonctions **anonymes** en **une seule ligne.**

Python permet également de créer des fonctions anonymes avec le mot-clé "**lambda**".

Exemple

```
addition = lambda x,y: x + y
print(addition(2,3))  # Affiche 5
```

```
carre = lambda x: x ** 2
print(carre(4))  # Affiche 16
```

Fonctions avec Retour Multiple

Une fonction peut retourner une valeur ou plusieurs valeurs sous forme de tuple.

Exemple

```
def calculs(a, b):
    return a + b, a - b, a * b

somme, difference, produit = calculs(5, 3)
print(somme, difference, produit)  # Affiche 8, 2, 15
```

Procédure (Fonction sans Retour)

❑ Une procédure est une fonction qui n'a pas de valeur de retour. En Python, il n'y a pas de distinction formelle entre une fonction et une procédure : une fonction peut être considérée comme une procédure si elle ne retourne rien.

Exemple

```
def dire_bonjour(nom):
    print(f"Bonjour, {nom} !")

dire_bonjour("Ahmed")
```

```
Bonjour, Ahmed !
```

Cette fonction affiche un message mais ne retourne rien.