Synthèse :

UAA2 - Programmation séquentielle

Table des matières :

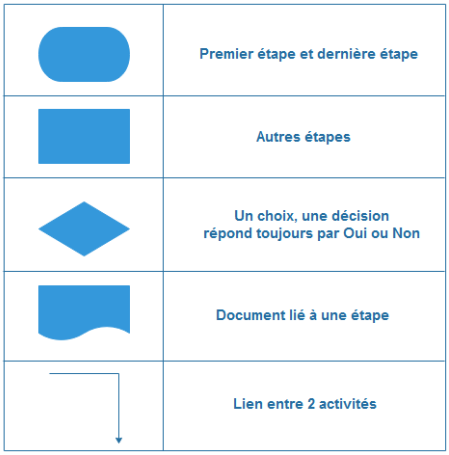
1.Les Logigrammes

Un logigramme, également appelé ordinogramme ou algorigramme, est une **représentation graphique d'un processus ou d'un algorithme**. Il s'agit d'un outil visuel qui permet de décomposer une tâche, une résolution de problème ou une prise de décision en une série d'étapes logiques et séquentielles.

Symboles et Conventions

La création d'un logigramme repose sur l'utilisation de symboles normalisés :

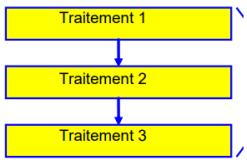
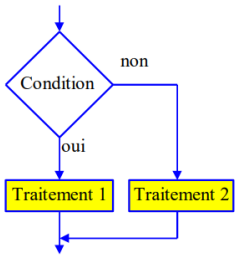
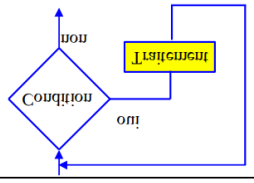
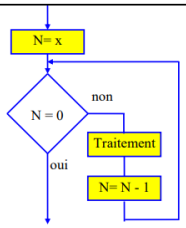
- **Ovale/rectangle arrondi** : Représente le début ou la fin du processus.
- **Rectangle** : Symbolise une action ou une opération à effectuer.
- **Losange** : Indique un point de décision ou une question nécessitant une réponse par oui ou par non.
- **Flèche** : Détermine le sens et l'ordre d'exécution des étapes.



Structures Fondamentales

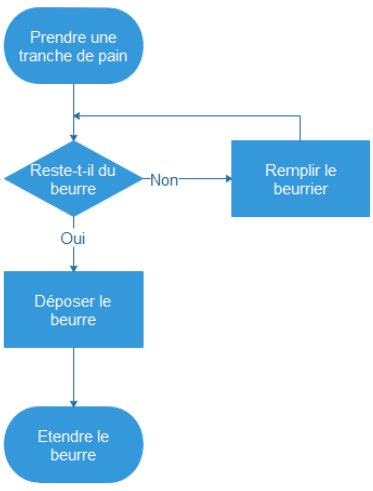
Les logigrammes peuvent être construits selon différentes structures :

- **Structure linéaire** : Les étapes sont exécutées **séquentiellement**, l'une après l'autre.
- **Structure alternative** : Le déroulement du processus **dépend d'une condition ou d'une décision** (si... alors... sinon...).
- **Structure répétitive** : Une ou plusieurs étapes sont **répétées jusqu'à** ce qu'une condition soit remplie (tant que... faire...).

Structure	Algorigramme	Algorithme
Structure linéaire La structure linéaire se caractérise par une suite d'actions à exécuter successivement dans l'ordre de leur énoncé.		**FAIRE « traitement 1 »** **FAIRE « traitement 2 »** **FAIRE « traitement 3 »**
Structure alternative ou conditionnelle Une structure alternative n'offre que deux issues possibles s'excluant mutuellement. Une condition est testée et en fonction du résultat du test soit le traitement 1, soit le traitement 2 est réalisé.		**SI « condition » vraie** **ALORS FAIRE** **« traitement 1 »** **SINON FAIRE** **« traitement 2 »** **FIN SI**
Dans cette structure on commence par tester la condition, si elle est vraie alors le traitement est exécuté. (boucle avec bifé-test) **Itérative** **Structure répétitive ou**		**FIN TANT QUE** **FAIRE « traitement »** **vraie** **TANT QUE « condition »**
Boucle avec comptage On initialise la variable N avec une valeur x. On teste si N est égal à 0, si ce n'est pas le cas, on exécute le traitement et on décrémente la variable N puis on teste à nouveau la variable N, et ainsi de suite jusqu'à ce que N=0.		**POUR N = x A 0 REPETER** **« traitement »** **FIN POUR**

Exemple Illustratif :

Un exemple **présentant le processus** pour faire une tartine de beurre.



2. La programmation Python

1. Affichage de textes, suite d'instructions

Affichage simple avec print()

- La fonction print() permet d'afficher du texte à l'écran.
- Le texte à afficher est placé entre guillemets "" à l'intérieur des parenthèses de la fonction.
- Exemple : **print("Bonjour le monde !")** affichera "Bonjour le monde !" à l'écran.

Suite d'instructions séquentielles

- En Python, les instructions sont exécutées les unes après les autres, de haut en bas dans le fichier.
- Chaque instruction est écrite sur une nouvelle ligne.
- L'ordre des instructions est crucial, car il détermine l'ordre d'exécution du programme.

Ajout de la bibliothèque "robot"

- Pour utiliser des fonctions comme avancer(), reculer(), droite(), gauche(), il faut d'abord importer la bibliothèque associée "robot"
- L'instruction en début de programme « **from robot import *** » permet d'accéder aux fonctions de cette bibliothèque.
- Challenge : https://www.france-ioi.org/algo/chapter.php?idChapter=841

2. Répétitions d'instructions

Les boucles for

En Python, on utilise les boucles for pour répéter un bloc d'instructions un certain nombre de fois. La structure de base est la suivante :

```
for loop in range(nombre_de_repetitions):
    print(« Hello »)
```

- range(nombre_de_repetitions) génère une séquence de nombres, et la boucle se répète autant de fois qu'il y a de nombres dans cette séquence.
- Les instructions à répéter sont indentées (décalées vers la droite) par rapport à la ligne for.

Indentation

L'indentation en Python est essentielle pour structurer le code et indiquer quels blocs d'instructions appartiennent à quelles structures de contrôle (boucles, conditions, fonctions, etc.).

- Pour augmenter le niveau d'indentation, on utilise la touche Tab.
- Pour diminuer le niveau d'indentation, on utilise la combinaison de touches Shift + Tab.

Contrôle de la fin de ligne avec print

La fonction print permet d'afficher du texte à l'écran. Par défaut, elle ajoute un saut de ligne à la fin de l'affichage. On peut modifier ce comportement en utilisant l'argument end :

```
print("Bonjour les ", end="")          # N'ajoute pas de saut de ligne à la fin
print("3ème TT")                       # Affichera en une ligne : « Bonjour les 3ème TT »
```

Boucles imbriquées

On peut imbriquer des boucles les unes dans les autres pour créer des structures répétitives plus complexes.

```
for loop in range(5):  # Boucle extérieure
    for loop in range(10):  # Boucle intérieure
        print("X", end="")
    print()  # Saut de ligne après chaque ligne de "X"
```

Dans cet exemple, la boucle intérieure (celle qui affiche les "X") est répétée 10 fois pour chaque itération de la boucle extérieure. Cela produit 5 lignes de 10 "X".

```
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
```

3. Commentaires

Les commentaires sont des notes que l'on ajoute dans le code pour l'expliquer. Ils sont ignorés par l'interpréteur Python. On utilise le symbole # pour commencer un commentaire :

```
# Ceci est un commentaire
print("Bonjour")  # Ceci est aussi un commentaire
```

Les commentaires sont utiles pour :

- Expliquer le fonctionnement du code
- Documenter les choix de conception
- Désactiver temporairement des parties du code

4. Les types d'opérateurs

1. Opérateurs arithmétiques

- Ce sont les opérateurs qui permettent de réaliser des calculs mathématiques, comme en mathématiques.
- Présentation des opérateurs :
    - + : Addition
    - - : Soustraction
    - * : Multiplication
    - / : Division (résultat à virgule)
    - // : Division entière (résultat arrondi)
    - % : Modulo (reste de la division entière)
    - ** : Exposant
- Exemples :
    - 5 + 2 = 7
    - 10 / 3 = 3.3333
    - 10 // 3 = 3
    - 10 % 3 = 1
    - 2 ** 3 = 8

L'ordre des opérations

Python respecte les priorités mathématiques habituelles :

1. **Parenthèses**
2. **Multiplication et division** (de gauche à droite)
3. **Addition et soustraction** (de gauche à droite)

Exemple :

```
resultat = 2 + 3 * 4  # resultat vaut 14 (3 * 4 est calculé en premier)

resultat = (2 + 3) * 4  # resultat vaut 20 (les parenthèses sont prioritaires)
```

2. Opérateurs de comparaison

- Ces opérateurs permettent de comparer deux valeurs et de déterminer si une condition est vraie ou fausse.
- Présentation des opérateurs :
    - == : égal à
    - != : différent de
    - > : supérieur à
    - < : inférieur à
    - >= : supérieur ou égal à
    - <= : inférieur ou égal à
- Exemples :
    - 5 > 3
    - 10 == 10
    - 'a' != 'b'
    - 10 >= 5

3. Opérateurs logiques

- Ces opérateurs permettent de combiner des conditions.
- Présentation des opérateurs :
    - and : ET logique - Il renvoie True seulement si les deux opérandes sont vraies.
    - or : OU logique - Il renvoie True si au moins une des deux opérandes est vraie.
    - not : NON logique - Il inverse la valeur de l'opérande.
- Exemples :
    - (5 > 3) and (10 < 20)
    - (5 > 3) or (10 > 20)
    - not (5 > 3)

5. Les Variables

On utilise des variables pour stocker des informations, comme des nombres, du texte, ou même des listes d'éléments. Pour comprendre, une variable, c'est comme une boîte avec une étiquette. L'étiquette, c'est le **nom de la variable**, et ce qu'il y a à l'intérieur, c'est sa **valeur**.

Comment créer une variable ?

En Python, créer une variable est très simple. Il suffit d'écrire le nom que tu veux lui donner, suivi du signe =, puis de la valeur que tu veux lui attribuer.

Exemple :

```
age = 12
nom = "Alice"
```

Dans cet exemple, on a créé deux variables :

- age qui contient la valeur 12 (un nombre)
- nom qui contient la valeur "Alice" (du texte, qu'on appelle une chaîne de caractères)

Les différents types de variables

Tout comme il existe différentes sortes de boîtes pour ranger différents objets, il existe différents **types de variables** pour stocker différents types d'informations :

- **Nombres entiers (int)** : pour stocker des nombres sans virgule (ex : 1, 5, -3)
- **Nombres décimaux (float)** : pour stocker des nombres avec virgule (ex : 3.14, -0.5)
- **Chaînes de caractères (str)** : pour stocker du texte, toujours entre guillemets (ex : "Bonjour", "Python est génial !")
- **Booléens (bool)** : pour stocker des valeurs de vérité, soit *True* (vrai), soit *False* (faux)

Utiliser les variables

Une fois que tu as créé des variables, tu peux les utiliser dans ton programme.

Exemple :

```
age = 12
nom = "Alice"

print("Bonjour", nom, "!")
print("Tu as", age, "ans.")
```

Ce programme affichera :

```
Bonjour Alice !
Tu as 12 ans.
```

Modifier la valeur d'une variable

La valeur d'une variable peut être modifiée à tout moment dans le programme.

Exemple :

```
age = 12
print("Tu as", age, "ans.")
age = age + 1
print("L'année prochaine, tu auras", age, "ans.")
```

Ce programme affichera :

Tu as 12 ans.

L'année prochaine, tu auras 13 ans.

Utiliser les variables dans les calculs

Tu peux utiliser des variables dans tes calculs pour les rendre plus dynamiques.

Exemple :

```
prix_unitaire = 5

quantite = 3

prix_total = prix_unitaire * quantite


print("Le prix total est de", prix_total, "euros.")
```

6. Les types

Conversion de type : int(), float(), str()

En Python, vous pouvez convertir des variables d'un type à un autre en utilisant les fonctions int(), float() et str().

- int() : Convertit une valeur en un entier.

```
nombre_entier = int(3.14)  # nombre_entier vaudra 3

nombre_entier = int("10")  # nombre_entier vaudra 10
```

- float() : Convertit une valeur en un nombre à virgule flottante.

```
nombre_decimal = float(5)  # nombre_decimal vaudra 5.0

nombre_decimal = float("3.14")  # nombre_decimal vaudra 3.14
```

- str() : Convertit une valeur en une chaîne de caractères.

```
chaine = str(42)  # chaine vaudra "42"

chaine = str(3.14)  # chaine vaudra "3.14"
```

La conversion de la valeur d'une variable en numérique est essentiel si vous souhaitez effectuer des opérations arithmétiques ou de comparaison entre les nombres.

7. Les entrées : input()

La fonction input() permet d'obtenir une entrée de l'utilisateur. Elle affiche un message à l'écran et attend que l'utilisateur saisisse quelque chose au clavier, puis appuie sur Entrée. La valeur saisie par l'utilisateur est renvoyée par la fonction sous forme de chaîne de caractères.

```
nom = input("Quel est votre nom ? ")

print("Bonjour,", nom + "!")
```

Dans le cas où l'utilisateur doit entrer un nombre, il faudra convertir la valeur entrée en numérique afin de pouvoir effectuer des calculs les lignes suivantes.

8. Les conditions : if, else, elif

Les instructions if, else et elif permettent d'exécuter du code conditionnellement, c'est-à-dire seulement si une certaine condition est vraie.

- if : Exécute un bloc de code si la condition est vraie.

```
age = 20

if age >= 18:

   print("Vous êtes majeur.")
```

- else : Exécute un bloc de code si la condition if est fausse.

```
age = 15

if age >= 18:

   print("Vous êtes majeur.")

else:

   print("Vous êtes mineur.")
```

- elif : Permet de tester plusieurs conditions.

```
note = 12

if note >= 16:

   print("Mention Très bien.")

elif note >= 14:

   print("Mention Bien.")
```

```
elif note >= 10:
    print("Mention Assez bien.")
else:
    print("Recalé.")
```

Remarques :

- Les conditions sont des expressions qui s'évaluent à True (vrai) ou False (faux).
- L'indentation est essentielle en Python pour définir les blocs de code.
- Vous pouvez imbriquer des instructions if, else et elif pour créer des structures conditionnelles plus complexes.

9. Boucles while

Les boucles while permettent d'exécuter un bloc de code de manière répétée tant qu'une condition donnée est vraie.

Syntaxe :

```
while condition:
    # Code à exécuter tant que la condition est vraie
```

Exemple :

```
compteur = 0
while compteur < 5:
    print(compteur)
    compteur = compteur + 1
```

Ce code affichera les nombres de 0 à 4. La boucle s'arrête lorsque la variable compteur atteint la valeur 5.

10. Fonctions

Les fonctions sont des blocs de code réutilisables qui effectuent une tâche spécifique. Elles permettent de structurer le code et de le rendre plus lisible et plus facile à maintenir.

Définition d'une fonction :

```
def nom_de_la_fonction(parametre1, parametre2, ...):
```

```
    # Code à exécuter
    return valeur_de_retour
```

- def est le mot-clé utilisé pour définir une fonction.
- nom_de_la_fonction est le nom que vous donnez à la fonction.
- Les paramètres sont des valeurs que vous passez à la fonction pour qu'elle puisse les utiliser.
- return est utilisé pour renvoyer une valeur de la fonction.

Appel d'une fonction :

```
nom_de_la_fonction(argument1, argument2, ...)
```

- Les arguments sont les valeurs que vous passez à la fonction lorsque vous l'appelez.

Exemple de fonction sans arguments :

```
def afficher_bonjour():
    print("Bonjour !")


afficher_bonjour()  # Affiche "Bonjour !"
```

Exemple de fonction avec arguments :

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


resultat = addition(5, 3)  # resultat vaut 8
```

Remarques :

- Les fonctions peuvent avoir des paramètres optionnels.
- Les fonctions peuvent renvoyer une valeur.
- Les fonctions peuvent être imbriquées les unes dans les autres.
- Les fonctions peuvent être utilisées pour décomposer un problème complexe en sous-problèmes plus simples.

11. Exercices récapitulatifs Python

1. Rappel du print()
   a. Dire : **Bonjour tout le monde !**

2. Utilisation d'une variable texte :
   a. Déclarer une variable : **Nom=«Toto»**
   b. Dire en utilisant la variable : **Bonjour [Nom], tu vas bien ?**

3. Utilisation d'une variable numérique :
   a. Déclarer une variable contenant le nombre 100 : **x=1**
   b. Ajouter **5** à la variable
   c. Dire : **Votre variable vaut : [x]**

4. Transformer un nombre sous format texte en format numérique :
   a. Si la variable contient le texte « 20 » en valeur : **X=«20»** et **Y=«20»**
   b. Dire : **[X]+[Y]** et analyser le résultat.
   c. Transformer en numérique la variable
   d. Dire : **[X]+[Y]** et analyser le résultat.

5. Utiliser une boucle for :
   a. Sur une seule ligne : afficher les 20 premiers nombres commençant à 1 et incrémenté de 1 la précédente valeur à chaque tour de boucle : **1 2 3 4 5 …**
   b. Sur une seule ligne afficher les 20 premiers nombres commençant à 1 et multipliez par 2 la précédente valeur à chaque tour de boucle : **1 2 4 8 16 32 ….**

6. Utilisation de l'entrée :
   a. Demandez le nom à l'utilisateur : **entrez votre nom svp.**
   b. Enregistrez la valeur dans une variable nom1
   c. Dire = **Bonjour [nom1] !**
   d. Demandez un nombre à l'utilisateur : **entrez un nombre svp.**
   e. Enregistrez dans une variable X
   f. **Transformer** le texte encodé par l'utilisateur en numérique ! via **int()**
   g. Ajoutez 10 au nombre enregistré dans la variable.
   h. Dire : **le nombre est : [X]**

7. Comparer des nombres via un print et marquez le résultat à coté :

a. 4==4
b. 4==5
c. 4 !=5
d. 4<5
e. 4<=5
f. 5<=5
g. 6<5

8. Utilisation du « if » et du « else » en comparant des valeurs numériques :
    a. Demandez deux nombres à l'utilisateur et enregistrer les dans deux variables n1 n2.
    b. Pensez à les convertir en numérique afin de pouvoir les comparer.
    c. Si n1<n2 écrire : **Le nombre [n1] est plus petit que le nombre [n2] !**
    d. Sinon écrire : **Le nombre [n2] est plus grand ou égale au nombre [n2] !**

9. Découverte du While :
    a. Initialiser une variable à **oui**.
    b. Intégrer le point 8 dans une boucle **while** tant que la variable est « **oui** ».
    c. Demander si l'utilisateur veut continuer le jeu en fin du bloc de la boucle.
    d. Si **oui** continuer la boucle…

10. Utilisation de plusieurs « if » et du « else » imbriqués en comparant du texte :
    a. Demandez à l'utilisateur s'il aime les pommes
    b. Si la réponse est oui dire : **tu aimes les pommes !**
    c. Sinon : demander s'il aime peut-être les poires
        i. S'il aime les poires dire : **Tu n'aimes pas les pommes mais tu aimes les poires !!!**
        ii. Sinon dire : **Tu n'aimes ni les pommes ni les poires !!!**

11. Découverte des fonctions :
    a. Créer la fonction addition()
    b. Créer la fonction addition avec deux arguments : addition(x1,x2)
    c. Créer une fonction qui renvoie un résultat.

12. Corrigé des exercices récapitulatifs

Rappel du print()

```
# Affiche le texte "Bonjour tout le monde !" à la console
print("Bonjour tout le monde !")
```

Utilisation d'une variable texte

```
# Déclare une variable nommée "Nom" et lui assigne la valeur "Toto"
Nom = "Toto"
# Affiche la phrase "Bonjour Toto, tu vas bien ?" en utilisant la variable "Nom"
print("Bonjour " + Nom + ", tu vas bien ?")
```

Utilisation d'une variable numérique

```
# Déclare une variable nommée "x" et lui assigne la valeur 1
x = 1
# Ajoute 5 à la valeur de la variable "x"
x = x + 5
# Affiche la phrase "Votre variable vaut : 6" en utilisant la valeur actuelle de "x"
print("Votre variable vaut :", x)
```

Transformation d'un nombre sous format texte en format numérique

```
# Déclare une variable "X" avec la valeur textuelle "20"
X = "20"
# Déclare une variable "Y" avec la valeur textuelle "20"
Y = "20"
# Affiche le résultat de la concaténation des deux variables textuelles "X" et "Y" ("2020")
print(X + Y)
# Convertit les variables "X" et "Y" en nombres entiers et affiche leur somme (40)
```

```
X=int(X)
Y=int(Y)
print(X + Y)
```

Utiliser une boucle for

```
# Affiche les 20 premiers nombres entiers, chacun étant incrémenté de 1 par rapport au précédent
X=1
for i in range(20):
   print(X, end=" ")
   X=X+1


print()  # Passe à la ligne suivante


# Affiche les 20 premiers termes d'une suite géométrique, chaque terme étant multiplié par 2 par rapport au précédent
nombre = 1
for i in range(20):
   print(nombre, end=" ")
   nombre = nombre * 2
```

Utilisation de l'entrée

```
# Demande à l'utilisateur d'entrer son nom et stocke la valeur dans la variable "nom1"
nom1 = input("Entrez votre nom svp : ")
# Affiche "Bonjour" suivi du nom entré par l'utilisateur
print("Bonjour", nom1 + " !")
```

Demander un nombre à l'utilisateur

```
# Demande à l'utilisateur d'entrer un nombre et stocke la valeur (convertie en entier) dans la variable "X"
X = int(input("Entrez un nombre svp : "))
# Ajoute 10 à la valeur de "X"
X = X + 10
# Affiche la phrase "Le nombre est : " suivie de la valeur actuelle de "X"
print("Le nombre est :", X)
```

Comparer des nombres

```
# Affiche des comparaisons entre des nombres et le résultat de ces comparaisons (Vrai ou Faux)
print("4 == 4 :", 4 == 4)  # Égalité
print("4 == 5 :", 4 == 5)  # Égalité
print("4 != 5 :", 4 != 5)  # Inégalité
print("4 < 5 :", 4 < 5)  # Strictement inférieur à
print("4 <= 5 :", 4 <= 5)  # Inférieur ou égal à
print("5 <= 5 :", 5 <= 5)  # Inférieur ou égal à
print("6 < 5 :", 6 < 5)  # Strictement inférieur à
```

Utilisation du "if" et du "else"

```
# Demande à l'utilisateur d'entrer deux nombres et les stocke (convertis en entiers) dans les variables "n1" et "n2"
n1 = int(input("Entrez le premier nombre : "))
n2 = int(input("Entrez le deuxième nombre : "))

# Compare "n1" et "n2" et affiche le résultat
if n1 < n2:
    print("Le nombre", n1, "est plus petit que le nombre", n2, "!")
else:
```

```
print("Le nombre", n1, "est plus grand ou égal au nombre", n2, "!")
```

Découverte du While

```
# Initialise la variable "reponse" à "oui" pour démarrer la boucle
reponse = "oui"
while reponse == "oui":
    # Demande à l'utilisateur d'entrer deux nombres et les stocke (convertis en entiers) dans les variables "n1" et "n2"
    n1 = int(input("Entrez le premier nombre : "))
    n2 = int(input("Entrez le deuxième nombre : "))

    # Compare "n1" et "n2" et affiche le résultat
    if n1 < n2:
        print("Le nombre", n1, "est plus petit que le nombre", n2, "!")
    else:
        print("Le nombre", n1, "est plus grand ou égal au nombre", n2, "!")
    # Demande à l'utilisateur s'il veut continuer et stocke sa réponse dans "reponse"
    reponse = input("Voulez-vous continuer le jeu ? (oui/non) : ")
```

Utilisation de plusieurs "if" et du "else" imbriqués

```
# Demande à l'utilisateur s'il aime les pommes et stocke sa réponse dans "reponse"
reponse = input("Aimez-vous les pommes ? (oui/non) : ")
if reponse == "oui":
    print("Tu aimes les pommes !")
else:
    # Si l'utilisateur n'aime pas les pommes, demande s'il aime les poires
    reponse = input("Aimez-vous les poires ? (oui/non) : ")
    if reponse == "oui":
```

```
    print("Tu n'aimes pas les pommes mais tu aimes les poires !!!")
  else:
    print("Tu n'aimes ni les pommes ni les poires !!!")
```

Découverte des fonctions

```
def addition():
  # Cette fonction additionne deux nombres demandés à l'utilisateur.
  a = int(input("Entrez le premier nombre : "))
  b = int(input("Entrez le deuxième nombre : "))
  somme = a + b
  print("La somme de", a, "et", b, "est :", somme)

# Appel de la fonction
addition()
```

```
def addition(x1, x2):
  # Cette fonction additionne les deux arguments passés en paramètre.
  somme = x1 + x2
  print("La somme de", x1, "et", x2, "est :", somme)

# Appel de la fonction avec des valeurs
addition(5, 3)
```

```
def addition(x1, x2):
  #Cette fonction additionne les deux arguments passés en paramètre et renvoie le résultat.
  somme = x1 + x2
  return somme
```

```
# Appel de la fonction et affichage du résultat
resultat = addition(7, 2)
print("La somme est :", resultat)
```

Explications :

- La première fonction addition() demande à l'utilisateur d'entrer deux nombres, les additionne et affiche le résultat.
- La deuxième fonction addition(x1, x2) prend deux arguments, les additionne et affiche le résultat.
- La troisième fonction addition(x1, x2) prend également deux arguments, les additionne et **renvoie** le résultat. Cela permet de stocker le résultat dans une variable et de l'utiliser plus tard.