



Programmation Python

2ème Année du Cycle Préparatoire au
Cycle Ingénieur d'Etat

2024/2025



Dep. Génie Informatique

Pr. EL OMARI Achraf
Pr. SERRAR Abderrahim



- Les programmes Python sont **3 à 5 fois plus courts** que ceux écrits en C ou C++, par exemple.
- **Supporte la programmation Orienté objet**
- **Libre** : utilisation sans restriction dans les projets commerciaux ;
- **Interprété** : Python est doté d'un interpréteur : Il lit le code source, le convertit en une forme intermédiaire appelée **bytecode**, puis exécute ce bytecode, souvent dans une machine virtuelle (PMV). Cela rend le code source flexible et portable.
- **Un typage dynamique** : On a pas besoin de déclarer explicitement les types des variables. Les types sont déduits au moment de l'exécution, ce qui rend le code plus court et plus flexible.



Algorithme

Algorithme Surface_cercle ;

Constante PI=3,14 ;

r, s : réel ;

Début

AFFICHER ("Entrer r :");

Lire (r);

s ← PI*r*r ;

AFFICHER ("Surface du cercle (de rayon", r,

") : ", s);

Fin.

Programme Python

Constante

PI = 3.14

Demander à l'utilisateur le rayon du cercle

r = input("Entrer r : ") # Lire r (text)

r = float(r) # Convertir r en réel (float)

Calculer la surface du cercle

s = PI * r * r

Afficher la surface du cercle

print("Surface du cercle (de rayon ", r, ") : ", s)

Opérateurs numériques	Opérateurs alphanumériques	Opérateurs relationnels	Opérateurs logiques
Algorithmique			
<code>+, -, *, ./, ^, //, %</code>	<code>+</code>	<code>>, <, =, ≤, ≥, !=</code>	<code>Et, Ou, Non</code>
Programmation Python			
<code>+, -, *, /, **, //, %</code>	<code>+</code>	<code>>, <, ==, <=, =>, !=</code>	<code>and, or, not</code>

Exemple

Écrire le programme python qui permet de lire :

- Le montant hors taxes d'un article : `Montant_HT` (un réel)
- Le taux de TVA : `Taux_TVA` (un réel)
- Le nombre d'articles : `Nb_Articles` (un entier)

Puis, le montant total toutes taxes comprises `Total_TTC` sachant que :

`Total_TTC = Nb_articles * Montant_HT * (1 + Taux_TVA)`

Algorithme

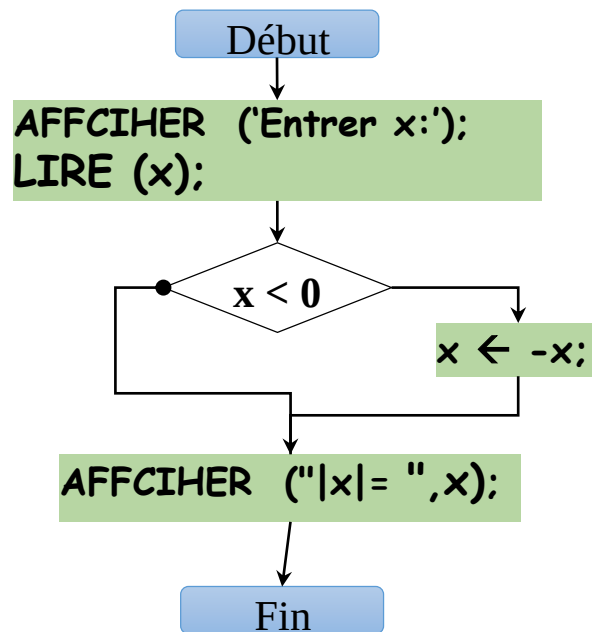
```
ALGORITHME TotalTTC;  
    m_HT, Total_TTC, taux_TVA : RÉEL ;  
    nb_articles : entier;  
  
Début  
    AFFCIHER ("Donner le nombre d'articles")  
  
    LIRE(nb_articles)  
  
    AFFCIHER ("Donner le montant hors taxe")  
  
    LIRE(montant_HT)  
  
    AFFCIHER ("Donner le taux de TVA")  
  
    LIRE(taux_TVA)  
  
    Total_TTC ← nb_articles * montant_HT * (1 + taux_TVA)  
  
    AFFCIHER ("Le montant total avec TVA :", total_TTC)  
  
Fin.
```

Programme Python

```
# Demander le nombre d'articles  
nb_articles = input("Donner le nombre d'articles : ")  
nb_articles = int(nb_articles)  
  
# Demander le montant hors taxe  
montant_HT = input("Donner le montant hors taxe : ")  
montant_HT = float(montant_HT)  
  
# Demander le montant hors taxe  
taux_TVA = input("Donner le montant hors taxe : ")  
taux_TVA = float(taux_TVA)  
  
# Calculer le total TTC  
total_TTC = nb_articles * montant_HT * (1 + taux_TVA)  
  
# Afficher le montant total TTC  
print("Le montant total avec TVA :", total_TTC)
```

Un programme python qui trouve la valeur absolue d'une variable x.

Organigramme



Algorithme

```
ALGORITHME  valeur_absolue;
x : RÉEL ;
Début

  AFFICHER  ("Entrer x :");
  LIRE  (x);

  SI x < 0 ALORS
    x ← -x;
  FINSI

  AFFICHER  ("|x| = ", x);

Fin.
```

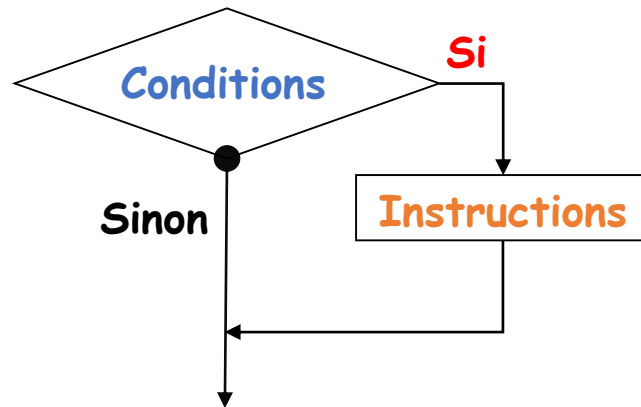
Python

```
# Demander à l'utilisateur d'entrer x
x = float(input("Entrer x : "))

# Verifier le signe de x
if x < 0:
    x = -x

# Afficher la valeur absolue de x
print("|x| =", x)
```

Organigramme



Algorithme

SI Conditions ALORS
Instructions;
FINSI

Python

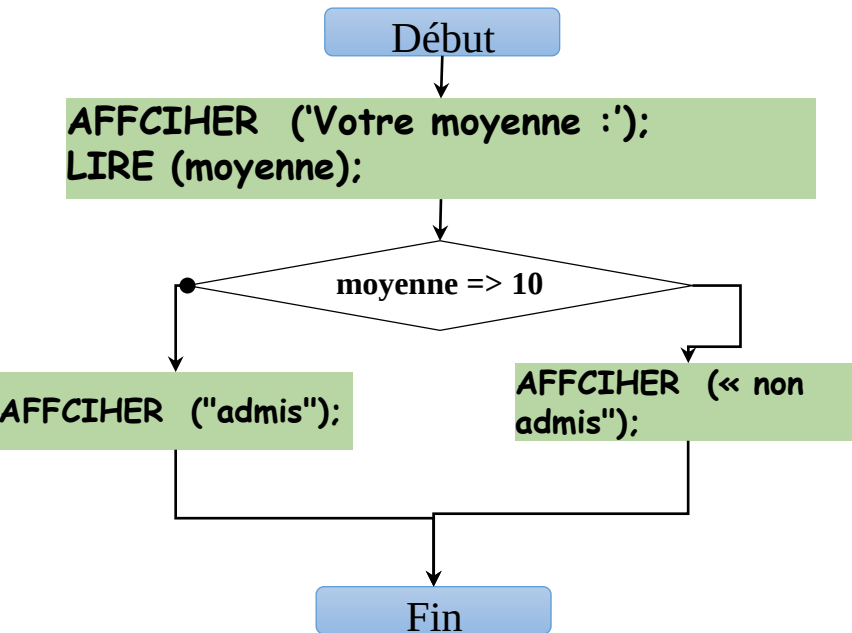
if Conditions:
Instructions

Si la condition est vraie alors le bloc d'instructions sera exécuté, sinon il sera ignoré.

Exemple : Un programme python qui trouve la valeur absolue d'une variable x.

Exemple : Écrire un programme qui décide si un étudiant est admis ou non en fonction de sa moyenne générale.

Organigramme



Algorithmme

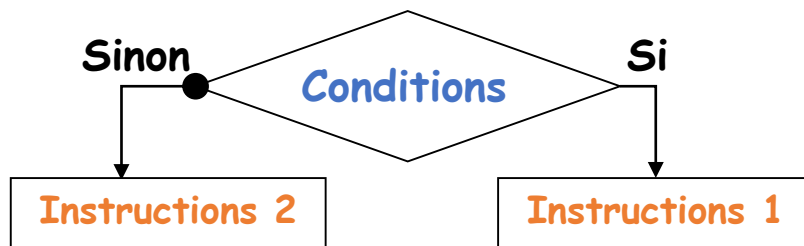
```
ALGORITHME Moyenne_decision;
moyenne : RÉEL ;
Début
    AFFCIHER ("Votre moyenne :");
    Lire (moyenne);
    SI moyenne >= 10 ALORS
        AFFCIHER ("admis");
    SINON
        AFFCIHER ("non admis");
    FINSI
Fin.
```

Python

```
# Demander à l'utilisateur d'entrer sa moyenne
moyenne = float(input("Votre moyenne : "))

# Vérifier si la moyenne est supérieure ou égale à 10
if moyenne >= 10:
    print("admis")
else:
    print("non admis")
```

Organigramme



Algorithme

```
SI Conditions ALORS  
    Instructions 1;  
SINON  
    Instructions 2;  
FINSI
```

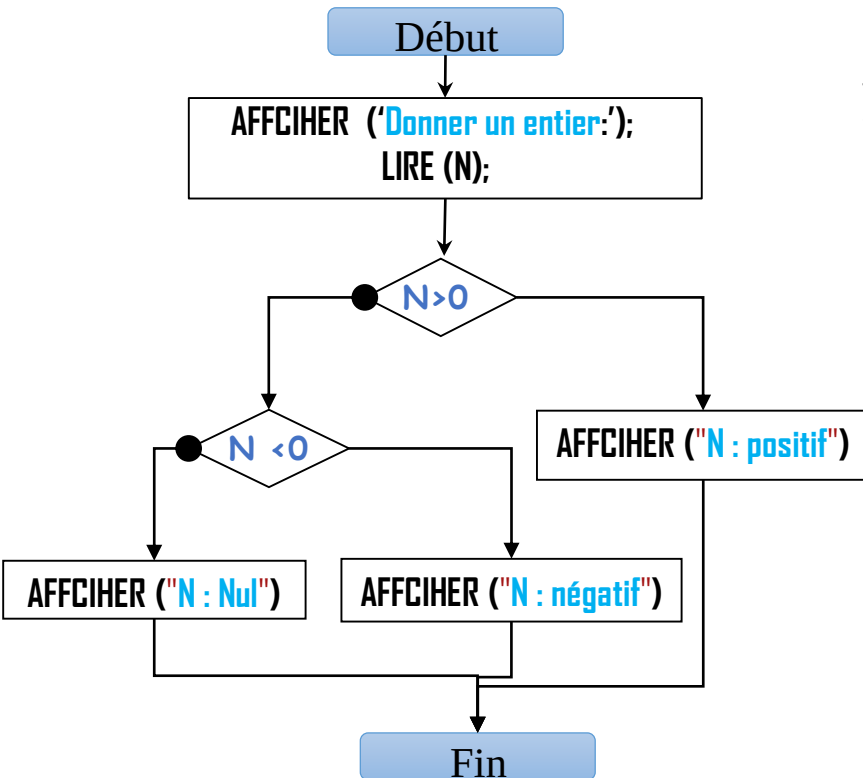
Python

```
if Conditions:  
    Instructions1  
else:  
    Instructions2
```

Exemple : Écrire un programme qui décide si un étudiant est admis ou non en fonction de sa moyenne générale.

Exemple : Un algorithme qui demande un entier à l'utilisateur, et l'informe ensuite s'il est positif, négatif ou nul.

Organigramme



Algorithme

```

ALGORITHME  Nature_entier;
N : ENTIER ;
Début
    AFFICHER ("Donner un entier:");
    LIRE (N);
    SI N > 0 ALORS
        AFFICHER ("N : positif");
    SINON
        SI N < 0 ALORS
            AFFICHER ("N : négatif");
        SINON
            AFFICHER ("N : Nul");
        FINSI
    FINSI
Fin.
    
```

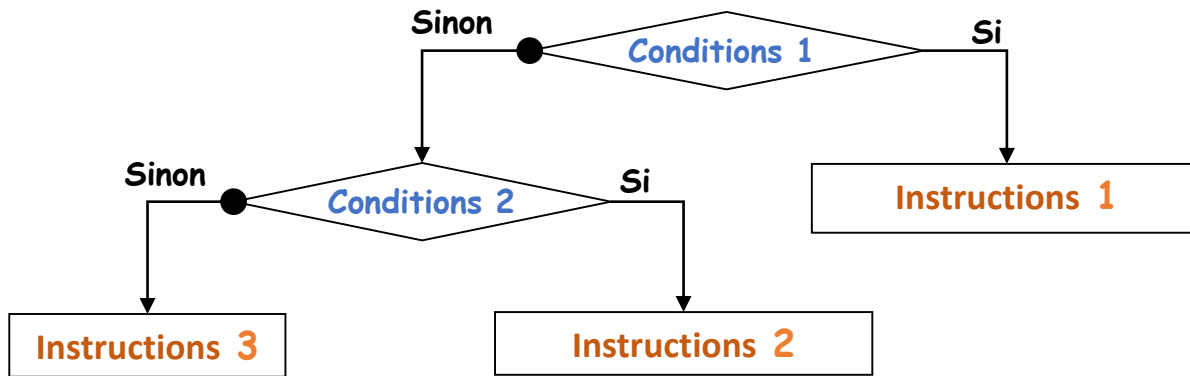
Python

```

# Demander à l'utilisateur de saisir un entier
N = int(input("Donner un entier : "))

# Vérifier si N est positif, négatif ou nul
if N > 0:
    print("N : positif")
elif N < 0:
    print("N : négatif")
else:
    print("N : Nul")
    
```

Organigramme



Algorithme

```
SI Conditions1 ALORS  
    Instructions1 ;  
SINON SI Conditions2 ALORS  
    Instructions2 ;  
SINON  
    Instructions3 ;  
FINSI  
FINSI
```

Python

```
if Conditions1:  
    Instructions1  
elif Conditions2:  
    Instructions2  
else:  
    Instructions3
```

Exemple : Un algorithme qui demande un entier à l'utilisateur, et l'informe ensuite s'il est positif, négatif ou nul.

Exemple : Un algorithme et le programme Python permettant de convertir les chiffres de 1 à 5 en lettres.

Algorithme

ALGORITHME chiffres_en_lettres;

N : ENTIER ;

Début

AFFCIHER ("Donner un caractère entre 1 et 5:");

LIRE (**N**);

SELON N

1 : AFFCIHER ("Un");

2 : AFFCIHER ("Deux");

3 : AFFCIHER ("Trois");

4 : AFFCIHER ("Quatre");

5 : AFFCIHER ("Cinq");

SINON : AFFCIHER ("Chiffre non supporté");

FinSELON

FIN.

Python

```
N = int(input("Donner un caractère entre 1 et 5 :"))
```

```
match N:
    case 1:
        print("Un")
    case 2:
        print("Deux")
    case 3:
        print("Trois")
    case 4:
        print("Quatre")
    case 5:
        print("Cinq")
    case _:
        print("Chiffre non supporté")
```

Algorithme

SELON choix

Choix1 :

Instructions1 ;

Choix2 :

Instructions2 ;

Choix3 :

Instructions3 ;

SINON :

instructions

FinSELON

Python

match choix:

case Choix1 :

Instructions1

case Choix2 :

Instructions2

case Choix3 :

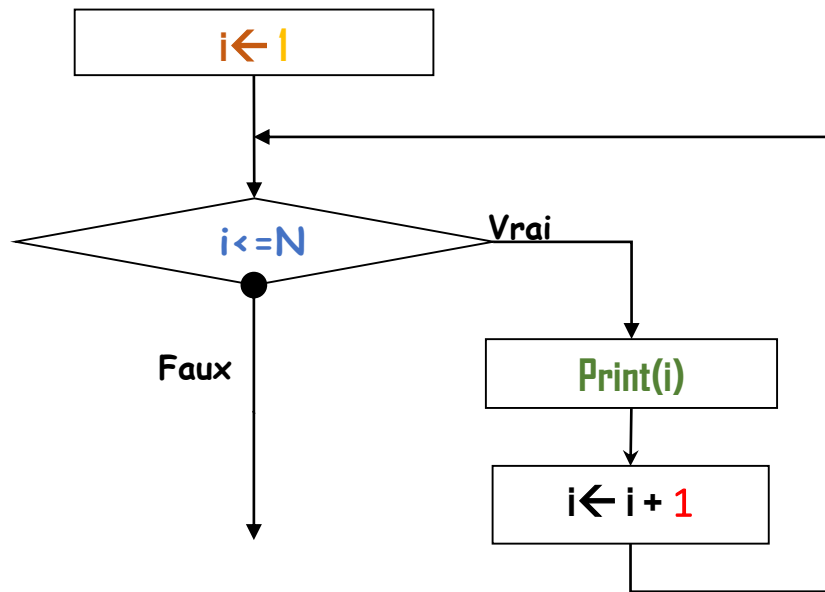
Instructions3

case _ : Instructions

La fonctionnalité match Cette fonctionnalité n'a été introduite que dans Python 3.10 et les version ultérieures,

Exemple de problèmes : Écrire un programme Python qui demande à l'utilisateur de saisir un entier N et affiche tous les nombres entiers de 1 à N.

Organigramme

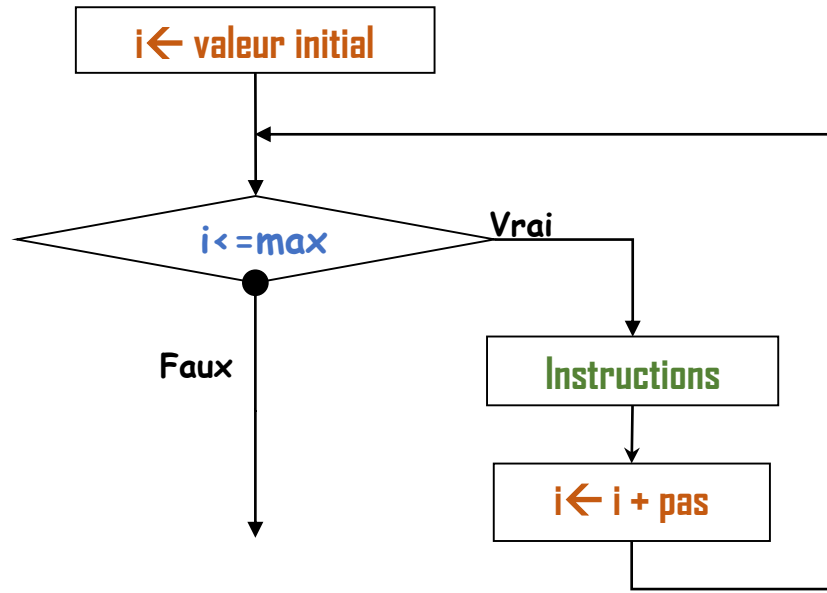


Python

```
# Demander à l'utilisateur de saisir un entier N
N = int(input("Entrez un nombre entier N : "))

# La boucle for pour afficher les entier entre 1 et N
for i in range(1, N + 1, 1):
    print(i)
```

Organigramme



Python

```
for i in range(valeur_initiale, max, pas):  
    instructions
```

La boucle for est généralement utilisée lorsque le nombre d'itérations est connu à l'avance ou lorsque vous devez itérer sur une séquence (comme une liste, tableau, ...).

Exemple : Écrire un programme Python qui calcule la somme des N premiers nombres entiers positifs, où N est un nombre entier fourni par l'utilisateur. (exemple pour N=5 → somme : $1+2+3+4+5 = 15$)

```
# Demander à l'utilisateur de saisir un nombre entier N
N = int(input("Entrez un nombre entier N : "))

# Initialiser la somme à 0
somme = 0

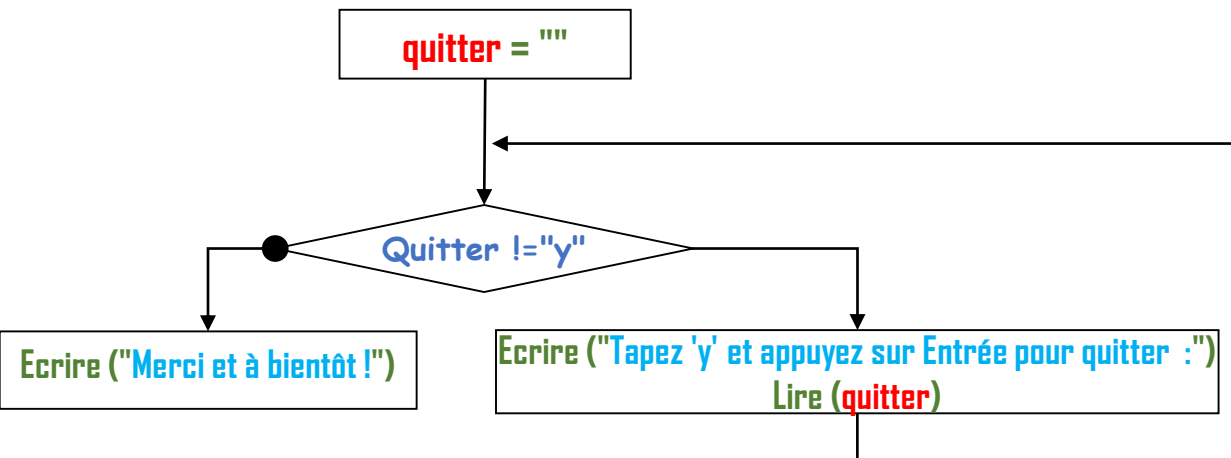
# Utiliser une boucle for pour calculer la somme des N premiers nombres entiers
for i in range(1, N + 1):
    somme += i

# Afficher le résultat
print(f"La somme des {N} premiers nombres entiers est : {somme}")
```

```
Entrez un nombre entier N : 10
La somme des 10 premiers nombres entiers est : 55
```

Problème : un programme Python qui invite l'utilisateur à décider du moment de quitter le programme. L'utilisateur doit pouvoir continuer à interagir avec le programme jusqu'à ce qu'il choisisse explicitement de le terminer en entrant le caractère « y ».

Organigramme



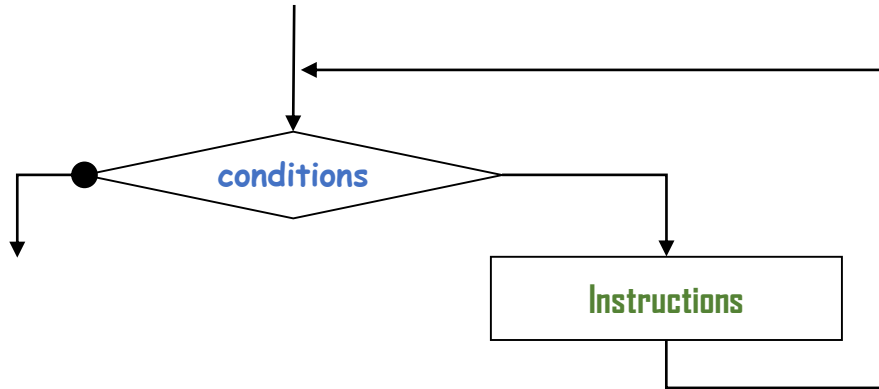
Python

```
quitter = ""

while quitter.lower() != "y":
    quitter = input("Tapez 'y' et appuyez sur Entrée pour quitter : ")

# Message de remerciement affiché à la sortie du programme
print("Merci et à bientôt !")
```

Organigramme



Algorithme

TANT QUE **condition**
instructions
FINTQ

Python

While **conditions** :
instructions

La boucle while s'emploie quand le nombre exact d'itérations reste indéterminé avant l'exécution. Elle réitère les instructions contenues dans son bloc tant que la condition évaluée demeure vraie.

Problème: Écrire un programme dans lequel l'utilisateur devine un nombre secret prédéfini par le programme. Après chaque tentative, le programme indique si l'utilisateur a deviné correctement ou non.

Algorithme

```
ALGORITHM Jeu_Devinette;  
Constante nombre_secret = 7  
N: ENTIER;  
Début  
    FAIRE  
        AFFCIHER ("Entrez un nombre : ")  
        LIRE (N) ;  
    TANTQUE (N != nombre_secret)  
        AFFCIHER ("nombre secret trouvé.")  
FIN.
```

Python

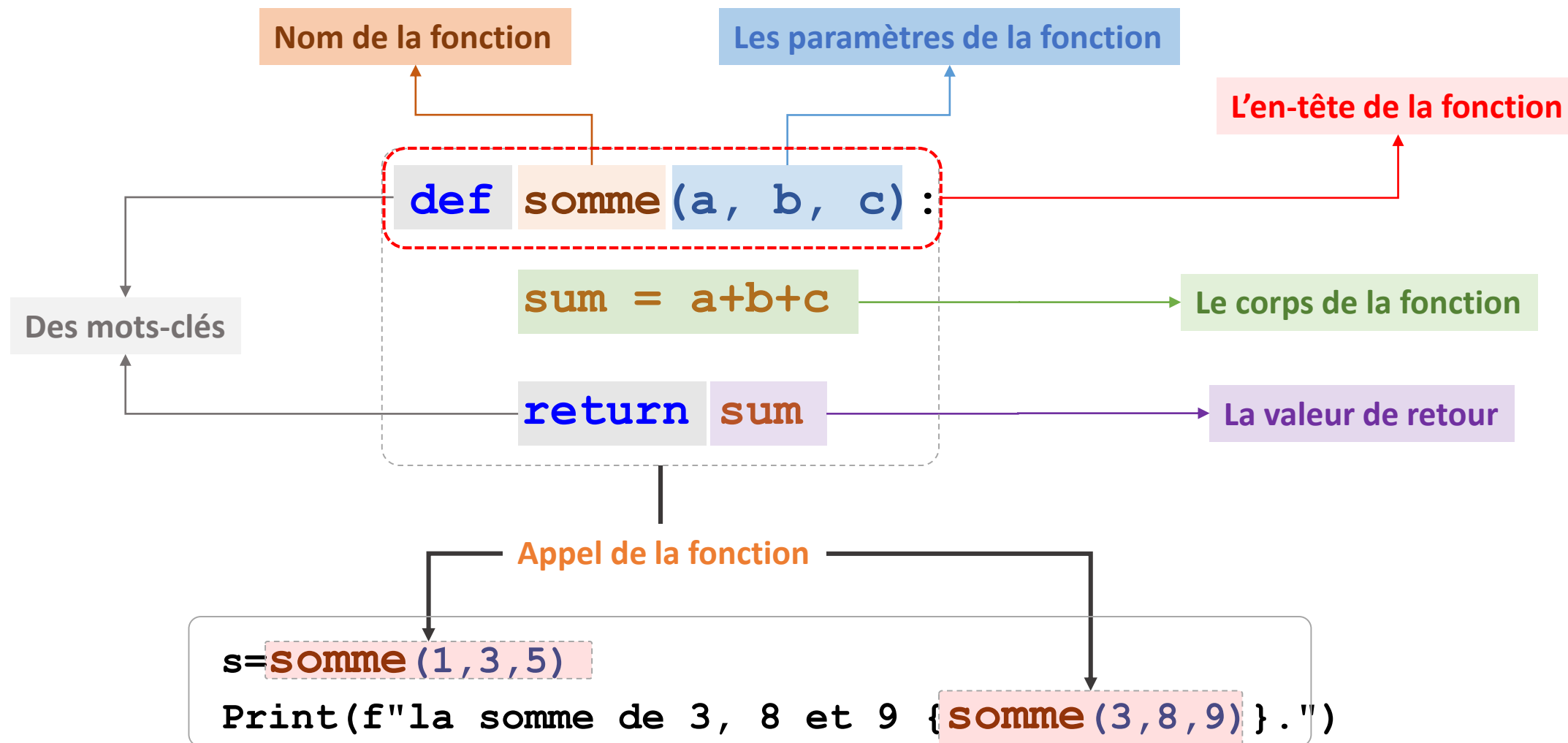
```
nombre_secret = 7  
while True:  
    n = int(input("Entrez un nombre : "))  
    if n == nombre_secret:  
        print("Félicitations ! nombre secret trouvé.")  
        break # Sortie de la boucle si le nombre est correct
```

La boucle Répéter... jusqu'à n'existe pas en PYTHON mais on peut la simuler en utilisant la boucle while et l'instruction break.

```
while True:  
    Instruction  
    if condition:  
        break
```

Problème: Écrire un programme python qui calcule la factorielle des entiers 3, 13, 25 et 40.

```
def factorielle(n):  
    fact = 1  
    for i in range(1, n + 1):  
        fact *= i  
    return fact  
  
# Appel de la fonction factorielle  
fact_3=factorielle(3)  
fact_5=factorielle(5)  
fact_9=factorielle(9)  
  
# Calcul et affichage de la factorielle pour chaque entier  
print(f"La factorielle de 3 est {fact_3}.")  
print(f"La factorielle de 5 est {fact_5}.")  
print(f"La factorielle de 9 est {fact_9}.")
```



Pourquoi Utiliser des Fonctions ?

Réutilisabilité : Les fonctions peuvent être appelées plusieurs fois avec différents arguments.

Modularité : Cela aide à organiser le code en unités logiques

Maintenance : vous n'avez besoin de modifier le code qu'à un seul endroit.

Clarté : L'utilisation de fonctions avec des noms descriptifs rend le code principal plus lisible. Au lieu de voir des formules mathématiques complexes éparpillées dans le code, vous voyez des appels de fonction clairs qui indiquent ce qu'ils font.

1. Variables locales

Les variables locales sont définies à l'intérieur d'une fonction et ne sont accessibles qu'à l'intérieur de cette fonction. Une fois que l'exécution de la fonction est terminée, ces variables sont détruites.

Exemple :

```
def ma_fonction():  
    x = 10 # Variable locale  
    print("Dans la fonction, x =", x)  
  
ma_fonction() # Affiche : Dans la fonction, x = 10  
# print(x) # Cela lèverait une erreur : NameError: name 'x' is not defined
```

2. Variables globales

Les variables globales, en revanche, sont définies en dehors de toute fonction et peuvent être accessibles depuis n'importe quelle fonction dans le même fichier. Elles conservent leur valeur tant que le programme est en cours d'exécution.

Exemple :

```
y = 20 # Variable globale  
  
def ma_fonction():  
    print("Dans la fonction, y =", y)  
  
ma_fonction() # Affiche : Dans la fonction, y = 20  
print("En dehors de la fonction, y =", y) # Affiche : En dehors de la fonction, y = 20
```

En Python, vous pouvez définir des valeurs par défaut pour les paramètres d'une fonction. Cela signifie que si l'utilisateur ne fournit pas d'argument pour ce paramètre lors de l'appel de la fonction, la valeur par défaut sera utilisée. Cela peut rendre vos fonctions plus flexibles et faciles à utiliser.

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

```
# Appel de la fonction avec un message par défaut
saluer("Alice") # Affiche : Bonjour, Alice!
```

```
# Appel de la fonction avec un message personnalisé
saluer("Bob", "Salut") # Affiche : Salut, Bob!
```

*# Avoir **pip** (gestionnaire de paquets Python). Vérifiez avec :*

```
pip --version
```

ou bien

```
python -m pip --version
```

*# Mettre à jour **pip** (si partiellement installé)*

```
pip install --upgrade pip
```

ou bien

```
python -m pip install --upgrade pip
```

*# Vérifiez si **pip** est maintenant bien installé / mis à jour :*

```
pip --version
```

ou bien

```
python -m pip --version
```

Lister les paquets installés :

```
pip list
```

*# Installation de **Numpy** :*

```
python -m pip install --user numpy  
import numpy
```

```
print(numpy.__version__) # ou bien  
print(numpy.version.version)
```

*# Mettre à jour **Numpy** :*

```
python -m pip install --upgrade numpy
```

*# Désinstaller **Numpy** :*

```
python -m pip uninstall numpy
```

Installation de librairies courantes :

```
python -m pip install --user scipy matplotlib ipython  
jupyter pandas sympy pytest scikit-learn tensorflow
```

```
18
19 import numpy as np
20
21 # Création d'un tableau NumPy
22 arr = np.array([1, 2, 3])
23 print(arr)
24
25 # Opérations vectorielles
26 print(arr * 2) # [2 4 6]
27 print(np.sqrt(arr)) # [1. 1.41 1.73]
28
```

Les **listes** sont des structures de données qui permettent de stocker une collection d'éléments. Elles sont très flexibles et peuvent contenir des éléments de types différents (nombres, chaînes, objets, etc.)

Caractéristiques des listes

Ordonnées : Les éléments d'une liste ont un ordre défini. L'ordre est basé sur l'emplacement des éléments dans la liste.

Mutables : Vous pouvez modifier une liste après sa création (ajouter, supprimer, modifier des éléments).

Indexées : Les éléments d'une liste peuvent être accédés par leur index, qui commence à 0.

Supportent des types variés : Une liste peut contenir différents types de données (nombres, chaînes, objets, etc.).

Exemple

```
# Création d'une liste vide
```

```
liste_vide = []
```

```
# Création d'une liste avec différents types d'éléments
```

```
liste_variee = [42, "texte", 3.14, True, None]
```

Création d'une liste

```
ma_liste = [1, 2, 3, 4]
```

1. Ajout d'un élément

```
ma_liste.append(5)
```

2. Insertion d'un élément à un index spécifique

```
ma_liste.insert(2, 2.5) # Insérer 2.5 à l'index 2
```

3. Insertion d'un élément à la fin de la liste

```
ma_liste.append(4)
```

4. Suppression d'un élément par valeur

```
ma_liste.remove(4) # Supprime la 1ère occurrence de 4
```

5. Supprime et retourne de l'élément à l'index 2

```
ma_liste.pop(2)
```

```
element_supprime = ma_liste.pop(2)
```

6. Tri de la liste

```
ma_liste.sort()
```

7. Inversion de la liste

```
ma_liste.reverse()
```

8. Longueur de la liste

```
longueur = len(ma_liste)
```

9. Vérification d'appartenance

```
est_dans_liste = 4 in ma_liste
```

10. Parcours des éléments de la liste

```
for element in ma_liste:
```

```
    print(element)
```

11. Parcours des éléments de la liste avec indice

```
for index, element in enumerate(ma_liste):
```

```
    print(f"{index} et (element)")
```

- Un dictionnaire est **une collection de paires clé-valeur**, où chaque **clé est unique**.
- Les dictionnaires **sont mutables**, permettant d'ajouter, de supprimer ou de modifier des éléments.
- Les **clés** doivent être de **types immuables** (comme des chaînes, des nombres ou des tuples).
- Les **valeurs peuvent** être de **n'importe quel type**, offrant une grande flexibilité dans la structuration des données.

```
# Création d'un dictionnaire vide
dictionnaire_vide = {}

# Création d'un dictionnaire avec différents types d'objets
dictionnaire_varie = {
    "nom": "Hamza",      # Chaîne de caractères
    "age": 20,           # nombre
    "est_etudiant": True, # Booléen
    "notes": [85, 90, 78], # Une Liste
    "adresse": {         # Dictionnaire imbriqué
        "ville": « CasaBlanca",
        "code_postal": 12400 }
}
```

**dans un dictionnaire Python,
vous pouvez utiliser des clés de
différents types. Les clés
peuvent être de n'importe quel
type immuable.**

Les opérations sur les dictionnaires

Création d'un dictionnaire

```
mon_dictionnaire = {  
    "nom": "Youssef",  
    "age": 30,  
    "ville": "Rabat" }
```

1. Accéder à une valeur par sa clé

```
nom = mon_dictionnaire["nom"]
```

2. Ajouter une nouvelle paire clé-valeur

```
mon_dictionnaire["profession"] = "Ingénieur"
```

3. Modifier une valeur existante

```
mon_dictionnaire["age"] = 31
```

4. Supprimer un élément par clé avec del

```
del mon_dictionnaire["profession"]
```

5. Vérification de l'existence d'une clé

```
est_dans_dictionnaire = "nom" in mon_dictionnaire
```

6. Longueur du dictionnaire

```
longueur = len(mon_dictionnaire)  
print("Longueur du dictionnaire :", longueur)
```

7. Parcourir les clés et les valeurs

```
for cle, valeur in mon_dictionnaire.items():  
    print(f"{cle} : {valeur}")
```