Les Structures de Contrôle en Python

if...else, match...case, for, while

Mamadou Moustapha DIALLO

L2 Informatique

muustafa.dllo@gmail.com

2025

Plan de la Présentation

Qu'est-ce qu'une Structure de Contrôle ?

- Les **structures de contrôle** permettent de modifier l'ordre d'exécution des instructions

Qu'est-ce qu'une Structure de Contrôle ?

- Les **structures de contrôle** permettent de modifier l'ordre d'exécution des instructions
- Elles permettent de :
  - Prendre des décisions (conditions)
  - Répéter des actions (boucles)
  - Organiser le flux du programme

Qu'est-ce qu'une Structure de Contrôle ?

- Les **structures de contrôle** permettent de modifier l'ordre d'exécution des instructions
- Elles permettent de :
  - Prendre des décisions (conditions)
  - Répéter des actions (boucles)
  - Organiser le flux du programme
- En Python, l'indentation définit les blocs de code

Qu'est-ce qu'une Structure de Contrôle ?

- Les **structures de contrôle** permettent de modifier l'ordre d'exécution des instructions
- Elles permettent de :
  - Prendre des décisions (conditions)
  - Répéter des actions (boucles)
  - Organiser le flux du programme
- En Python, l'indentation définit les blocs de code
- Quatre structures principales :
  - `if...else` : Conditions simples
  - `match...case` : Conditions multiples (Python 3.10+)
  - `for` : Boucles sur séquences
  - `while` : Boucles conditionnelles

La Structure if...else : Syntaxe de Base

Syntaxe :

```
if condition:
    # Instructions si vrai
    instruction1
    instruction2
else:
    # Instructions si faux
    instruction3
    instruction4
```

Exemple Simple :

```
age = 18
if age >= 18:
    print("Majeur")
    print("Peut voter")
else:
    print("Mineur")
    print("Ne peut pas voter")
```

Sortie : Majeur
Peut voter

Conditions avec elif

```
note = 15

if note >= 16:
    print("Tr s bien")
    mention = "TB"
elif note >= 14:
    print("Bien")
    mention = "B"
elif note >= 12:
    print("Assez bien")
    mention = "AB"
elif note >= 10:
    print("Passable")
    mention = "P"
else:
    print("Insuffisant")
    mention = "I"

print(f"Mention obtenue : {mention}")
```

Sortie : Bien

Opérateurs de Comparaison

Opérateurs de base :

- == : égal à
- != : différent de
- < : strictement inférieur
- <= : inférieur ou égal
- > : strictement supérieur
- >= : supérieur ou égal

Exemple :

```
1 x = 10
2 y = 5
3
4 if x > y:
5     print(f"{x} > {y}")
6 if x != y:
7     print("x et y sont
      diff rents")
8 if y <= x:
9     print("y est inf rieur ou
        gal      x")
```

Sortie : 10 > 5
x et y sont différents
y est inférieur ou égal à x

Opérateurs Logiques

Opérateurs :

- `and` : ET logique
- `or` : OU logique
- `not` : NON logique
- `in` : appartenance
- `not in` : non-appartenance

Exemple :

```
1  age = 20
2  permis = True
3  voiture = False
4
5  if age >= 18 and permis:
6      print("Peut conduire")
7
8  if voiture or permis:
9      print("A acc s un
       v hicule")
10
11 nom = "Alice"
12 liste_vip = ["Alice", "Bob", "
       Charlie"]
13
14 if nom in liste_vip:
15     print("Acc s VIP accord
       ")
```

Conditions Complexes : Exemple Pratique

```
# Syst me de connexion utilisateur
nom_utilisateur = "alice"
mot_de_passe = "secret123"
tentatives = 2
compte_bloque = False

if not compte_bloque:
    if nom_utilisateur == "alice" and mot_de_passe == "secret123":
        print("Connexion r ussie!")
        print("Bienvenue Alice!")
    elif tentatives > 0:
        print(f"Identifiants incorrects. {tentatives} tentative(s) restante(s)")
        tentatives -= 1
    else:
        print("Compte bloqu  apr s trop de tentatives")
        compte_bloque = True
else:
    print("Compte bloqu . Contactez l'administrateur.")
```

Valeurs de Vérité en Python

Valeurs considérées comme `False` :

- `False`
- `None`
- `0` (zéro)
- `""` (chaîne vide)
- `[]` (liste vide)
- `{}` (dictionnaire vide)
- `()` (tuple vide)

Exemple :

```
1  liste = []
2  if liste:
3      print("Liste non vide")
4  else:
5      print("Liste vide")
6
7  texte = ""
8  if texte:
9      print("Texte pr sent")
10 else:
11     print("Texte vide")
12
13 nombre = 0
14 if nombre:
15     print("Nombre non nul")
16 else:
17     print("Nombre nul")
```

Introduction à match...case

- Nouveauté de Python 3.10 (octobre 2021)

```
match variable:
    case valeur1:
        # Instructions pour valeur1
    case valeur2:
        # Instructions pour valeur2
    case _:  # cas par d faut (optionnel)
        # Instructions par d faut
```

Introduction à match...case

- Nouveauté de Python 3.10 (octobre 2021)
- Alternative élégante aux chaînes de `if...elif`

```
match variable:
    case valeur1:
        # Instructions pour valeur1
    case valeur2:
        # Instructions pour valeur2
    case _:  # cas par d faut (optionnel)
        # Instructions par d faut
```

Introduction à match...case

- Nouveauté de Python 3.10 (octobre 2021)
- Alternative élégante aux chaînes de `if...elif`
- Similaire au `switch...case` d'autres langages

```
match variable:
    case valeur1:
        # Instructions pour valeur1
    case valeur2:
        # Instructions pour valeur2
    case _:  # cas par d faut (optionnel)
        # Instructions par d faut
```

Introduction à match...case

- Nouveauté de Python 3.10 (octobre 2021)
- Alternative élégante aux chaînes de `if...elif`
- Similaire au `switch...case` d'autres langages
- Plus puissant avec le **pattern matching**

```
match variable:
    case valeur1:
        # Instructions pour valeur1
    case valeur2:
        # Instructions pour valeur2
    case _:  # cas par d faut (optionnel)
        # Instructions par d faut
```

Introduction à match...case

- Nouveauté de Python 3.10 (octobre 2021)
- Alternative élégante aux chaînes de `if...elif`
- Similaire au `switch...case` d'autres langages
- Plus puissant avec le **pattern matching**
- Syntaxe plus claire pour les conditions multiples

```
match variable:
    case valeur1:
        # Instructions pour valeur1
    case valeur2:
        # Instructions pour valeur2
    case _:  # cas par d faut (optionnel)
        # Instructions par d faut
```

match...case : Exemple de Base

Avec if...elif :

```
jour = "lundi"

if jour == "lundi":
    print("D but de semaine")
elif jour == "vendredi":
    print("Fin de semaine!")
elif jour in ["samedi", "
    dimanche"]:
    print("Weekend!")
else:
    print("Milieu de semaine")
```

Avec match...case :

```
jour = "lundi"

match jour:
    case "lundi":
        print("D but de
    semaine")
    case "vendredi":
        print("Fin de semaine!
    ")
    case "samedi" | "dimanche"
    :
        print("Weekend!")
    case _:
        print("Milieu de
    semaine")
```

Sortie : `Début de semaine`

Pattern Matching Avancé

```python
def analyser_donnee(donnee):
    match donnee:
        case int() if donnee > 0:
            return f"Entier positif: {donnee}"
        case int() if donnee < 0:
            return f"Entier n gatif: {donnee}"
        case int():
            return "Z ro"
        case str() if len(donnee) > 10:
            return f"Cha ne longue: {donnee[:10]}..."
        case str():
            return f"Cha ne courte: {donnee}"
        case list() if len(donnee) == 0:
            return "Liste vide"
        case list():
            return f"Liste de {len(donnee)}   lment   (s)"
        case _:
            return f"Type non g  r  : {type(donnee)}"

# Tests
print(analyser_donnee(42))              # Entier positif: 42
```

Correspondance de Structures

```python
def traiter_coordonnees(point):
    match point:
        case (0, 0):
            return "Origine"
        case (0, y):
            return f"Sur l'axe Y    y={y}"
        case (x, 0):
            return f"Sur l'axe X    x={x}"
        case (x, y) if x == y:
            return f"Sur la diagonale    ({x}, {y})"
        case (x, y):
            return f"Point g n ral    ({x}, {y})"

# Tests
print(traiter_coordonnees((0, 0)))    # Origine
print(traiter_coordonnees((0, 5)))    # Sur l'axe Y    y=5
print(traiter_coordonnees((3, 3)))    # Sur la diagonale
    (3, 3)
print(traiter_coordonnees((2, 7)))    # Point g n ral
    (2, 7)
```

Exemple Pratique : Calculatrice

```python
def calculatrice(operation, a, b):
    match operation:
        case "+":
            return a + b
        case "-":
            return a - b
        case "*":
            return a * b
        case "/" if b != 0:
            return a / b
        case "/" if b == 0:
            return "Erreur: Division par z ro"
        case "**":
            return a ** b
        case "%":
            return a % b
        case _:
            return f"Op ration '{operation}' non support e
    "

# Tests
```

La Boucle for : Principe

- Permet d'itérer sur des séquences (listes, chaînes, tuples...)

```
for element in sequence:
    # Instructions    r  p  ter
    instruction1
    instruction2
# Instructions apr s la boucle
```

La Boucle for : Principe

- Permet d'itérer sur des séquences (listes, chaînes, tuples...)
- Syntaxe : `for element in sequence:`

```
for element in sequence:
    # Instructions     r  p  ter
    instruction1
    instruction2
# Instructions apr s la boucle
```

La Boucle for : Principe

- Permet d'itérer sur des séquences (listes, chaînes, tuples...)
- Syntaxe : `for element in sequence:`
- La variable `element` prend successivement chaque valeur de la séquence

```
for element in sequence:
    # Instructions     r  p  ter
    instruction1
    instruction2
# Instructions apr s la boucle
```

La Boucle for : Principe

- Permet d'itérer sur des séquences (listes, chaînes, tuples...)
- Syntaxe : `for element in sequence:`
- La variable `element` prend successivement chaque valeur de la séquence
- Indentation obligatoire pour délimiter le bloc d'instructions

```
for element in sequence:
    # Instructions    r  p  ter
    instruction1
    instruction2
# Instructions apr s la boucle
```

Boucle for : Exemples Simples

Parcours d'une chaîne :

```
1 mot = "Python"
2 for lettre in mot:
3     print(f"Lettre: {lettre}")
```

Sortie :
```
Lettre: P
Lettre: y
Lettre: t
Lettre: h
Lettre: o
Lettre: n
```

Parcours d'une liste :

```
1 fruits = ["pomme", "banane", "
      orange"]
2 for fruit in fruits:
3     print(f"J'aime les {fruit}
      s")
```

Sortie :
```
J'aime les pommes
J'aime les bananes
J'aime les oranges
```

La Fonction range()

Syntaxes de range() :

- `range(n)` : 0 à n-1
- `range(début, fin)` : début à fin-1
- `range(début, fin, pas)` : avec un pas

Exemples :

```
1  # range(5) : 0, 1, 2, 3, 4
2  for i in range(5):
3      print(f"i = {i}")
4
5  # range(2, 6) : 2, 3, 4, 5
6  for i in range(2, 6):
7      print(f"i = {i}")
8
9  # range(0, 10, 2) : 0, 2, 4,
       6, 8
10 for i in range(0, 10, 2):
11     print(f"i = {i}")
```

Boucles for avec Indices : enumerate()

```python
# Acc s    l'index et    la valeur
animaux = ["chat", "chien", "oiseau", "poisson"]

# M thode 1 : avec range(len())
print("M thode 1 :")
for i in range(len(animaux)):
    print(f"Index {i}: {animaux[i]}")

# M thode 2 : avec enumerate() (plus pythonique)
print("\nM thode 2 (recommand e) :")
for index, animal in enumerate(animaux):
    print(f"Index {index}: {animal}")

# enumerate() avec un d but personnalis
print("\nAvec d but    1 :")
for numero, animal in enumerate(animaux, start=1):
    print(f"Animal #{numero}: {animal}")
```

Boucles Imbriquées

```python
# Table de multiplication
print("Table de multiplication (1    5) :")
for i in range(1, 6):
    print(f"\nTable de {i} :")
    for j in range(1, 11):
        resultat = i * j
        print(f"{i}    {j} = {resultat}")

# Parcours d'une matrice
matrice = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

print("\nParcours de la matrice :")
for ligne_index, ligne in enumerate(matrice):
    for col_index, valeur in enumerate(ligne):
        print(f"matrice[{ligne_index}][{col_index}] = {
    valeur}")
```

Instructions break et continue

break : Sort de la boucle

```
for i in range(10):
    if i == 5:
        print("Arr t    5")
        break
    print(f"i = {i}")
print("Fin de boucle")
```

Sortie :

```
i = 0
i = 1
i = 2
i = 3
i = 4
Arrêt à 5
Fin de boucle
```

continue : Passe à l'itération suivante

```
for i in range(6):
    if i % 2 == 0:  # Si pair
        continue
    print(f"Nombre impair: {i}
    ")
print("Fin de boucle")
```

Sortie :

```
Nombre impair: 1
Nombre impair: 3
Nombre impair: 5
Fin de boucle
```

La Clause else avec for

```python
# La clause else s'exécute si la boucle se termine normalement
# (pas avec break)

def chercher_nombre(liste, cible):
    for nombre in liste:
        print(f"Vérification de {nombre}")
        if nombre == cible:
            print(f"Trouvé {cible} !")
            break
    else:
        print(f"{cible} non trouvé dans la liste")

# Test 1 : nombre présent
nombres = [1, 3, 7, 12, 18]
chercher_nombre(nombres, 7)

print("\n" + "="*30 + "\n")

# Test 2 : nombre absent
chercher_nombre(nombres, 15)
```

Exemple Pratique : Analyse de Données

```python
# Analyse des notes d'une classe
notes = [12, 18, 9, 15, 14, 7, 16, 11, 13, 20, 8, 17]

# Calculs
total = 0
nb_notes = len(notes)
note_max = notes[0]
note_min = notes[0]
nb_admis = 0

for note in notes:
    total += note

    if note > note_max:
        note_max = note
    if note < note_min:
        note_min = note

    if note >= 10:
        nb_admis += 1
```

La Boucle while : Principe

- Répète un bloc d'instructions tant qu'une condition est vraie

```
while condition:
    # Instructions     r  p  ter
    instruction1
    instruction2
    # IMPORTANT: Modifier la condition !
# Instructions apr s la boucle
```

La Boucle while : Principe

- Répète un bloc d'instructions tant qu'une condition est vraie
- Syntaxe : `while condition:`

```
while condition:
    # Instructions     r  p  ter
    instruction1
    instruction2
    # IMPORTANT: Modifier la condition !
# Instructions apr s la boucle
```

La Boucle while : Principe

- Répète un bloc d'instructions tant qu'une condition est vraie
- Syntaxe : `while condition:`
- **Attention** : Risque de boucle infinie si la condition ne devient jamais fausse

```
while condition:
    # Instructions     r  p  ter
    instruction1
    instruction2
    # IMPORTANT: Modifier la condition !
# Instructions apr s la boucle
```

La Boucle while : Principe

- Répète un bloc d'instructions tant qu'une condition est vraie
- Syntaxe : `while condition:`
- **Attention** : Risque de boucle infinie si la condition ne devient jamais fausse
- Il faut modifier la condition dans le corps de la boucle

```
while condition:
    # Instructions     r  p  ter
    instruction1
    instruction2
    # IMPORTANT: Modifier la condition !
# Instructions apr s la boucle
```

while : Exemples de Base

Comptage simple :

```
compteur = 0
while compteur < 5:
    print(f"Compteur: {compteur}")
    compteur += 1  # ESSENTIEL!
print("Fini!")
```

Sortie :
```
Compteur: 0
Compteur: 1
Compteur: 2
Compteur: 3
Compteur: 4
Fini!
```

Décompte :

```
n = 5
while n > 0:
    print(f"D compte: {n}")
    n -= 1  # ESSENTIEL!
print("D collage!")
```

Sortie :
```
Décompte: 5
Décompte: 4
Décompte: 3
Décompte: 2
Décompte: 1
Décollage!
```

while vs for : Quand Utiliser Chacun ?

Utilisez `for` quand :

- Vous connaissez le nombre d'itérations
- Vous parcourez une séquence
- Vous voulez itérer sur des indices

```python
# Parcours d'une liste
for item in liste:
    print(item)

# Nombre d'it rations connu
for i in range(10):
    print(i)
```

Utilisez `while` quand :

- La condition d'arrêt n'est pas liée à un compteur
- Le nombre d'itérations est inconnu
- Vous attendez une condition externe

```python
# Saisie utilisateur
reponse = ""
while reponse != "oui":
    reponse = input("Continuer ? ")

# Condition complexe
while not fichier.eof():
    ligne = fichier.readline()
```

Exemple : Jeu de Devinette

```python
import random

# Le programme choisit un nombre entre 1 et 100
nombre_secret = random.randint(1, 100)
tentatives = 0
max_tentatives = 7

print("=== JEU DE DEVINETTE ===")
print("Je pense    un nombre entre 1 et 100.")
print(f"Vous avez {max_tentatives} tentatives!")

while tentatives < max_tentatives:
    tentatives += 1

    essai = int(input(f"Tentative {tentatives}: Votre nombre ? "))

    if essai == nombre_secret:
        print(f"Bravo! Trouv  en {tentatives} tentative(s)! ")
        break
```

while avec break et continue

```python
# Simulation d'un menu interactif
while True:  # Boucle infinie contr l e
    print("\n=== MENU ===")
    print("1. Afficher info")
    print("2. Calculer")
    print("3. Quitter")

    choix = input("Votre choix (1-3): ")

    if choix == "1":
        print("Informations affich es")
    elif choix == "2":
        a = float(input("Premier nombre: "))
        b = float(input("Second nombre: "))
        if b == 0:
            print("Division par z ro impossible!")
            continue  # Retour au d but du menu
        print(f"R sultat: {a / b}")
    elif choix == "3":
        print("Au revoir!")
        break  # Sortie de la boucle
```

while avec else

```python
# Recherche d'un   lment   avec limite de tentatives
def chercher_avec_limite(liste, element, max_essais):
    index = 0
    essais = 0

    while essais < max_essais and index < len(liste):
        essais += 1
        print(f"Essai {essais}: v rification index {index}"
    )

        if liste[index] == element:
            print(f"Trouv  {element}   l'index {index}")
            break
        index += 1
    else:
        # Ex cut  si while se termine sans break
        if index >= len(liste):
            print(f"{element} non trouv  dans la liste")
        else:
            print(f"Limite d'essais atteinte ({max_essais})"
    )
```

Éviter les Boucles Infinies

Problème : Boucle infinie

```
# ATTENTION: Boucle infinie!
i = 0
while i < 10:
    print(f"i = {i}")
    # Oubli de modifier i !
    # Cette boucle ne s'arr tera jamais
```

Solutions :

```
# Solution 1: Modifier la variable
i = 0
while i < 10:
    print(f"i = {i}")
    i += 1  # Modifier i

# Solution 2: Compteur de s curit
i = 0
compteur = 0
while i < 10 and compteur < 1000:
    print(f"i = {i}")
    i += 1
    compteur += 1
```

Exemple Avancé : Validation de Saisie

```python
def saisir_entier(message, min_val=None, max_val=None):
    """Fonction pour saisir un entier avec validation"""
    while True:
        try:
            valeur = int(input(message))

            # V rification des bornes
            if min_val is not None and valeur < min_val:
                print(f"Valeur trop petite! Minimum: {min_val}")
                continue

            if max_val is not None and valeur > max_val:
                print(f"Valeur trop grande! Maximum: {max_val}")
                continue

            return valeur  # Valeur valide, sortie de la boucle

        except ValueError:
```

Tableau Récapitulatif

Structure	**Usage**	**Avantages**	**Inconvénients**
`if...else`	Conditions simples	Simple, universel	Verbeux pour multiples conditions
`match...case`	Conditions multiples	Élégant, puissant pattern matching	Python 3.10+ uniquement
`for`	Itération sur séquences	Naturel pour collections	Nombre d'itérations doit être connu
`while`	Boucles conditionnelles	Flexible pour conditions complexes	Risque de boucle infinie

Bonnes Pratiques : Lisibilité

À éviter :

```
# Conditions trop complexes
if ((age >= 18 and permis ==
    True and
     voiture != None) or (age
    >= 21
     and moto == True)) and not
     (suspension == True):
     conduire()

# Variables peu explicites
for i in l:
     if i > x:
          r.append(i * 2)
```

Mieux :

```
# Conditions claires
peut_conduire_voiture = (age
    >= 18 and
                          permis
     and voiture)
peut_conduire_moto = (age >=
    21 and moto)
pas_suspendu = not suspension

if (peut_conduire_voiture or
     peut_conduire_moto) and
    pas_suspendu:
     conduire()

# Variables explicites
for nombre in liste_nombres:
     if nombre > seuil:
          resultats.append(
    nombre * 2)
```

Bonnes Pratiques : Performance

Moins efficace :

```
# Calcul r p titif dans la
    boucle
for i in range(len(liste)):
    if len(chaine) > 100:  #
    Recalcul !
        traiter(liste[i])

# Boucle while non optimis e
i = 0
while i < len(liste):
    element = liste[i]
    traiter(element)
    i += 1
```

Plus efficace :

```
# Calcul une seule fois
longueur_chaine = len(chaine)
for element in liste:
    if longueur_chaine > 100:
        traiter(element)

# Boucle for naturelle
for element in liste:
    traiter(element)
```

Exercices Pratiques

1. **Conditions :** Créez un programme qui détermine la saison selon le mois (1-12)
2. **Match-case :** Implémentez un convertisseur d'unités (km/miles, °C/°F)
3. **Boucle for :** Calculez la factorielle d'un nombre
4. **Boucle while :** Créez un jeu "Pierre-Papier-Ciseaux" contre l'ordinateur
5. **Projet :** Système de gestion de notes d'étudiants avec menu interactif

Critères d'évaluation :

- Choix approprié de la structure de contrôle
- Gestion des cas d'erreur
- Lisibilité et documentation du code
- Respect des bonnes pratiques

Résumé de la Présentation

- **if...else** : Base des conditions, indispensable pour toute logique décisionnelle

Point Clé

Le choix de la bonne structure de contrôle dépend du contexte et améliore significativement la qualité et la maintenance du code.

Résumé de la Présentation

- **if...else** : Base des conditions, indispensable pour toute logique décisionnelle
- **match...case** : Nouveauté Python 3.10, élégant pour les conditions multiples

Point Clé

Le choix de la bonne structure de contrôle dépend du contexte et améliore significativement la qualité et la maintenance du code.

Résumé de la Présentation

- **if...else** : Base des conditions, indispensable pour toute logique décisionnelle
- **match...case** : Nouveauté Python 3.10, élégant pour les conditions multiples
- **for** : Idéal pour parcourir des séquences et répétitions avec nombre connu

Point Clé

Le choix de la bonne structure de contrôle dépend du contexte et améliore significativement la qualité et la maintenance du code.

Résumé de la Présentation

- **if...else** : Base des conditions, indispensable pour toute logique décisionnelle
- **match...case** : Nouveauté Python 3.10, élégant pour les conditions multiples
- **for** : Idéal pour parcourir des séquences et répétitions avec nombre connu
- **while** : Puissant pour les boucles conditionnelles et interactions utilisateur

Point Clé

Le choix de la bonne structure de contrôle dépend du contexte et améliore significativement la qualité et la maintenance du code.

Résumé de la Présentation

- **if...else** : Base des conditions, indispensable pour toute logique décisionnelle
- **match...case** : Nouveauté Python 3.10, élégant pour les conditions multiples
- **for** : Idéal pour parcourir des séquences et répétitions avec nombre connu
- **while** : Puissant pour les boucles conditionnelles et interactions utilisateur
- **Bonnes pratiques** : Lisibilité, performance, et éviter les pièges courants

Point Clé

Le choix de la bonne structure de contrôle dépend du contexte et améliore significativement la qualité et la maintenance du code.

Pour Aller Plus Loin

Concepts Avancés à Découvrir :

- Compréhensions de listes : `[x*2 for x in liste if x > 0]`
- Générateurs : `(x*2 for x in liste)`
- Décorateurs pour contrôler l'exécution des fonctions
- Gestion d'exceptions avec `try...except`
- Programmation asynchrone avec `async/await`

Ressources Recommandées :

- Documentation officielle Python : `docs.python.org`
- PEP 8 : Guide de style Python
- Livres : "Python Crash Course", "Automate the Boring Stuff"
- Plateformes : Codecademy, HackerRank, LeetCode

Questions ?

Merci pour votre attention !

Contact :
muustafa.dllo@gmail.com
Tel : +221 77 193 52 60

Réseaux :
GitHub — LinkedIn