Algorithmique et programmation structurée en Python

m2iformation.fr

Sommaire

Qu'est-ce qu'un algorithme ?

Les algorithmes jouent un rôle central dans la résolution de problématiques dans de nombreux domaines, y compris mais sans s'y limiter à l'informatique, l'ingénierie, les mathématiques, la finance et la médecine.
Un algorithme peut être considéré comme une série d'instructions ou un ensemble de règles bien définies conçues pour effectuer une tâche spécifique ou résoudre un problème donné.

Rôle des algorithmes dans la résolution de problématiques :

1. **Abstraction et Modélisation :** Les algorithmes aident à modéliser des problèmes complexes du monde réel en termes de procédures étape par étape, rendant les solutions plus compréhensibles et plus faciles à analyser.
2. **Efficacité :** Ils permettent de trouver des solutions efficaces qui minimisent le temps d'exécution et l'utilisation des ressources, comme la mémoire. L'efficacité est particulièrement cruciale pour les applications traitant de grandes quantités de données ou nécessitant des calculs rapides.
3. **Précision et Fiabilité :** Un algorithme bien conçu garantit que la même entrée produira toujours la même sortie, assurant la fiabilité et la prévisibilité des systèmes informatiques.
4. **Innovation :** La recherche et le développement d'algorithmes nouveaux et améliorés stimulent l'innovation technologique, permettant de résoudre des problèmes auparavant considérés comme insolubles.

Lien avec les différents langages de programmation :

1. **Implémentation :** Les algorithmes doivent être implémentés dans un langage de programmation pour être exécutés par les ordinateurs. Chaque langage a ses propres caractéristiques et structures de données, influençant la manière dont un algorithme est écrit et exécuté.
2. **Universalité :** Bien que les algorithmes soient conceptuellement indépendants du langage de programmation, leur implémentation dépend de la syntaxe et des fonctionnalités spécifiques du langage choisi. Cependant, un algorithme peut généralement être traduit et adapté à différents langages.
3. **Optimisation :** Certains langages sont mieux adaptés à certaines tâches en raison de leur conception. Par exemple, Python est souvent privilégié pour le prototypage rapide et le traitement de données, tandis que des langages comme C ou Rust sont choisis pour les applications nécessitant une haute performance.
4. **Évolution des langages :** L'évolution des langages de programmation est souvent influencée par le besoin de soutenir de manière plus efficace l'implémentation d'algorithmes complexes. Les nouvelles structures de données, paradigmes de programmation, et améliorations de performance sont régulièrement introduits pour faciliter la tâche.

Présentation de Python et ses versions

- Le langage de programmation Python a été créé en 1989 par Guido Van Rossum
- La dernière version de Python est la version 3. Plus précisément, la version 3.12 qui a été publiée en Octobre 2023.
- La version 2 de Python est obsolète et n'est plus maintenue depuis le 1er Janvier 2020
- La Python Software Foundation est l'association qui organise le développement de Python et anime la communauté de développeurs et d'utilisateur.

Présentation de Python et ses versions

- Python est :
  - Multiplateforme
  - Un langage de haut niveau
  - Un langage interprété
  - Orienté objet
- Usage de Python :
  - Scripts pour automatiser des tâches
  - Analyses de données
  - Développement web ect...

Environnement de développement

- Par défaut, Python est déjà installé sur Linux et MacOS
- Pour Windows, il faudra le télécharger à cette adresse :
  - Download Python | Python.org
  - /!\ Attention, penser à cocher les bonnes cases à l'installation (notamment l'ajout au PATH sur windows)
- Pip est le gestionnaire de paquets de Python et qui est systématiquement présent depuis la version 3.4.

Environnement de développement

- Un script python est un fichier avec l'extension .py
- Pour démarrer un script python, on utilise la commande py (Windows) ou python/python3 (MacOS et Linux) et le nom du script
- Exemple
  - python mon_script.py

Environnement de développement

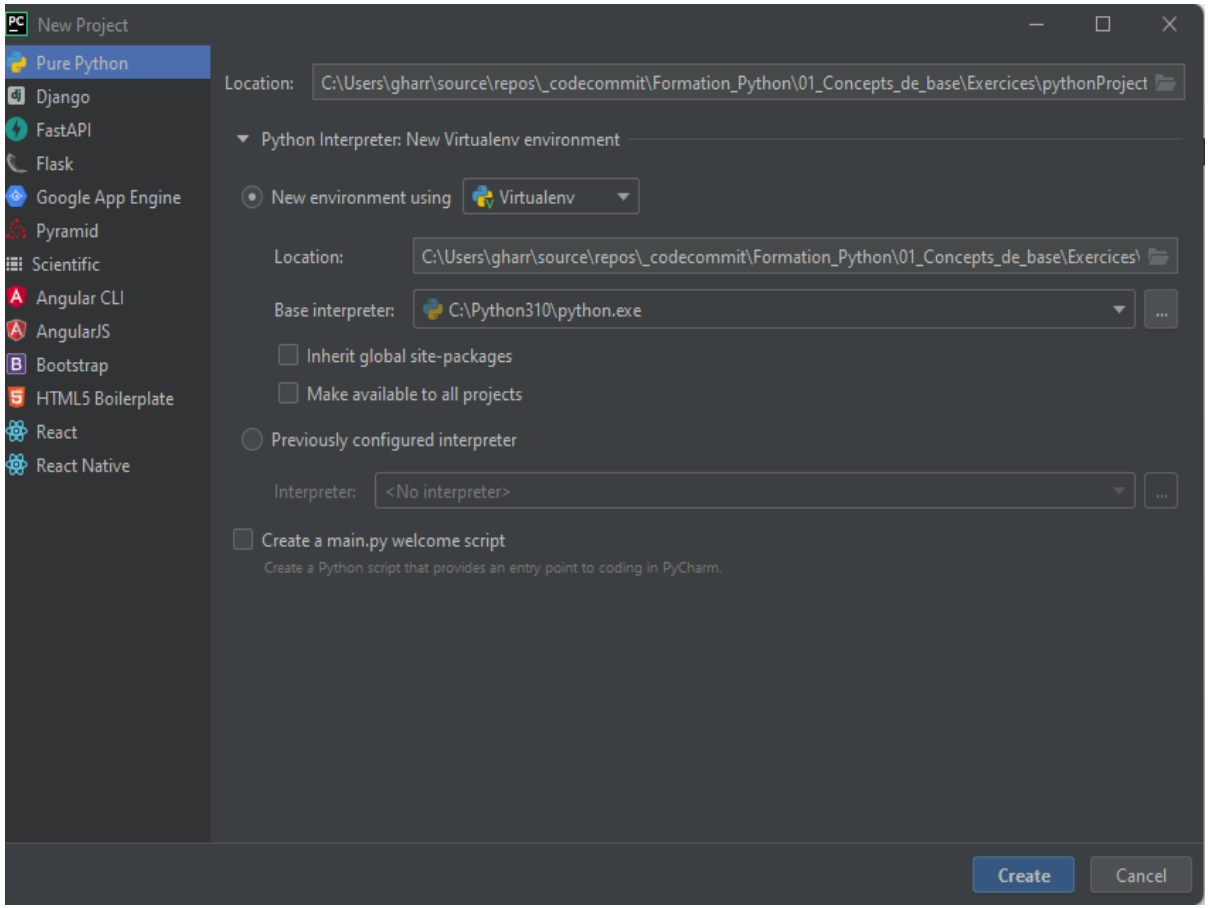
- L'interpréteur de Python est l'application qui permet de convertir les instructions python en un langage compréhensible par l'ordinateur.
- Il peut être utilisé pour exécuter une instruction.
- Pour ouvrir l'interpréteur, il suffit de taper dans un terminal (powershell, bash,...) py ou python3.
- Pour quitter l'interpréteur, Ctrl + D ou la fonction exit() de python.

Environnement de développement (IDE)

- Un IDE (Integrated Development Environment) est un environnement de développement intégré, il offre un ensemble d'outils tel que :
  - Un éditeur de code
  - Un compilateur
  - Un débogueur ect...
- Il convient d'installer l'un des IDE suivant :
  - Pycharm (JetBrains, conçut spécialement pour python, avec une version payante améliorée).
  - Visual Studio Code (Microsoft, utile dans de nombreux domaines et avec de nombreuses extensions disponibles)
- Lors de l'installation, il peut être utile pour les 2 logiciels de cocher les cases permettant de faire "Ouvrir le projet avec ..." et l'ajout au PATH

Pycharm

- Lors du lancement de Pycharm, appuyez sur **New Project**, puis sélectionnez l'emplacement de stockage du projet Python sur votre ordinateur via l'input **Location**.
- Vous pouvez ensuite spécifier l'environnement virtuel de lancement de l'interpréteur Python, comme cela est marqué dans l'exemple ci-après (celui-ci étant ici configuré
  sur **Virtualenv** , l'environnement virtuel de base).
- Un environnement virtuel permet d'installer des modules et des packages de façon spécifique au projet sans les
  installer de façon globale (sur la machine en elle-même).
- D'autres paramètres sont disponibles, mais nous n'en ferons pas usage dans ce cours. Appuyez simplement sur **Create** dans le but de valider la création du projet Python.

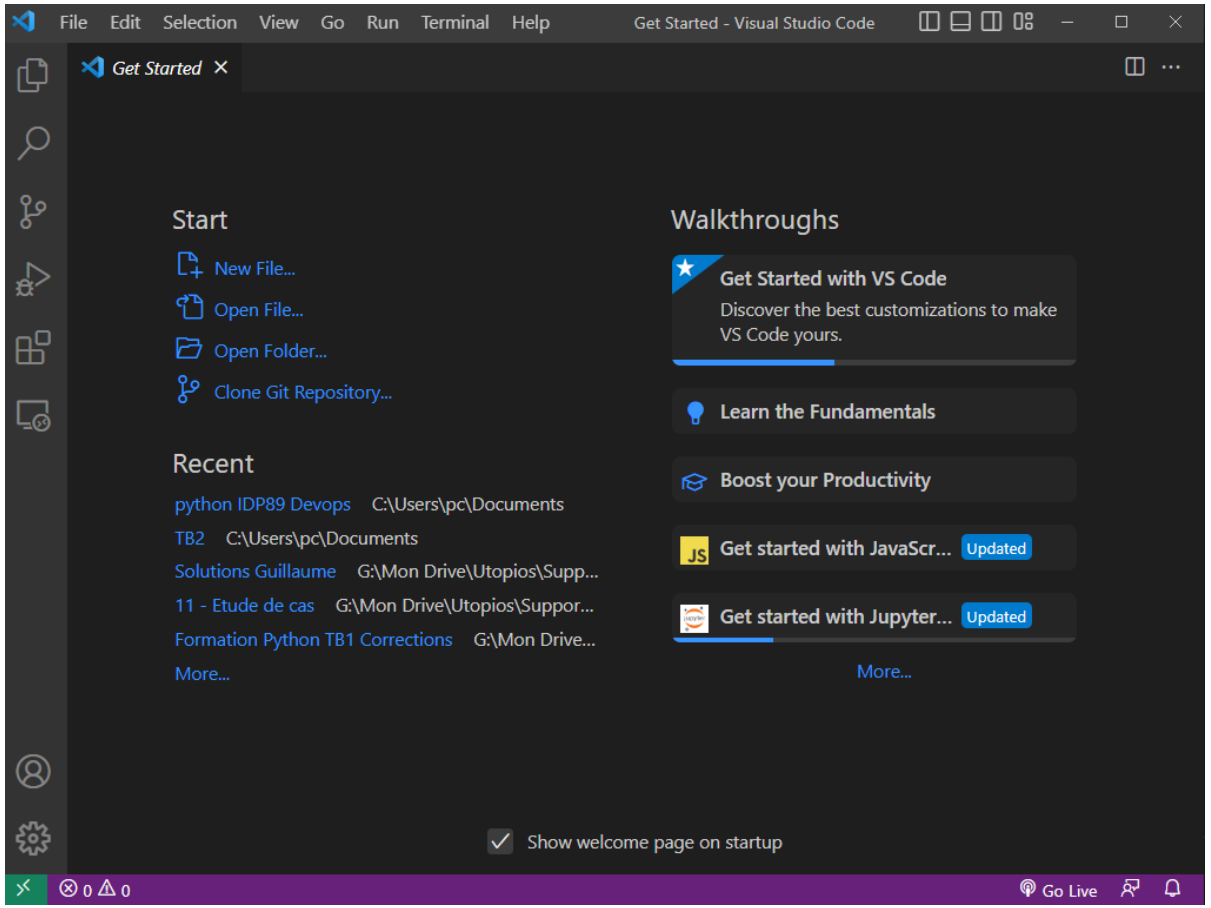


Pycharm

- Pycharm génère un environnement virtuel stocké dans venv (packages et modules installés)
- Il permet d'ajouter des scripts python dans notre dossier. (click droit, add python file)
- Il permet l'exécution d'un script par un **clic droit + run, CTRL + SHIFT + F10** ou encore en ajoutant une configuration de lancement (en haut à droite).

Visual Studio Code

- Dans le menu latéral, on trouve l'explorateur, il permet d'avoir un aperçu des fichiers du dossier ouvert.
- Lorsque l'on se situe sur un fichier python, un bouton "play" est affiché en haut à droite, il lance un terminal avec une commande équivalente à **py mon_script.py**



Les normes et conventions en Python

- La syntaxe python est soumise à des conventions.
- Un bon développeur s'assurera qu'il les suit en écrivant ses scripts python.
- Elles permettent de normaliser le langage et de faciliter la lecture.
- Python suit la norme PEP8
- Les IDE ont souvent un auto-formatage qui respectera une grande partie de ces règles. Les raccourcis pour formater automatiquement changent selon le logiciel.

Les variables

Les **variables** ont pour but de **stocker** des informations dans la mémoire vive de l'ordinateur.
Les **variables** peuvent être de plusieurs types, qui sont parmi les plus fréquentes :

- Les variables de type **numériques** servant à stocker des nombres. Ces variables peuvent être de plusieurs sous-types :
  - les integers/**int**, qui servent à stocker des nombres **entiers**
  - les **float**s, servant à stocker des **nombres à virgule** flottante (décimaux)
- Les chaines de caractères strings/**str** qui permettent de stocker du **texte**.
- Les **bool**éens, permettant de stocker des valeurs binaires (Vraie=**True** ou Fausse=**False**)
- Le vide, **None** en python, est un type à part qui ne représente 'Rien', il n'est **ni un 0, ni un False**

```
mon_int = 514 # Variable de type integer
mon_float = 3.14 # Variable de type float
mon_complex = 547J # Variable de type complex
mon_bool = True # Variable de type booléens
ma_string = "Je suis une chaîne de caractère" # Variable de type string.
```

Les variables

- L'opérateur '=' en python est l'opérateur **d'assignation/affectation**, il permet d'affecter à une variable une valeur donnée.
  Ex: **ma_variable = 3** permet d'assigner la valeur entière 3 à la variable 'ma_variable'.
- Autrement, lorsque l'on mettra **ma_variable** dans une instruction Python, on récupérera la valeur contenue dans cette variable
  Ex: **ma_variable2 = ma_variable + 1** permet d'assigner la valeur entière 4 à la variable 'ma_variable2'.

Le mot clé del

- Le mot clé **del** permet de **supprimer de la donnée** en python
- Il peut être utile lorsque l'on cherche à **libérer de la mémoire** ou à se débarrasser de certaines variables/fonctions/classes/...

Un peu de lexique

- Une **fonction** est un morceau de code déjà écrit, il prend des **paramètres** et retourne une **valeur**.
- On pourra **exécuter** ce morceau de code en ajoutant des parenthèses avec les **valeurs passées en paramètres** après le nom de la fonction.
- Une **méthode** est similaire à une fonction, la différence est qu'elle **s'applique** à une valeur/un objet donné, on ajoutera un "." après celui-ci suivi du **nom de la méthode** et des **parenthèses**.
- Lorsque l'on utilise ces 2 concepts, les **valeurs** que l'on met **entre les parenthèses** seront appelées **paramètres** ou **arguments**.

```
# Exemple de fonction
print("Un message à afficher")

# Exemple de méthode
test_maj = "test".upper()
```

Un peu de lexique

- Les mots **console** et **terminal** reviennent à peu près à la même chose, il s'agit d'une **fenêtre** permettant la **communication** entre l'utilisateur et le programme par du **texte**.
- Un **script python** est un fichier contenant des instructions.
- Un **programme** ou une **application** correspond au processus,résultat de l'interprétation du script par l'ordinateur.
- un **module python** est le contenu de l'interprétation d'un script python à l'exécution, nous y reviendrons plus tard...

Récupérer et afficher des valeurs

- Pour qu'il y ait un **dialogue** possible entre l'utilisateur et l'ordinateur (application console), on a recours à ce qui s'appelle des affichages et des **récupérations de valeurs**.
- **L'affichage** se fait par la fonction **input()**, qui affichera les valeurs passées en paramètre sur le terminal.
- **La récupération** se fait par la fonction **input()**, elle récupère une saisie de l'utilisateur sous forme de **str**. il est possible d'y ajouter une chaîne à afficher en paramètre.

```
ma_recuperation = input("Veuillez entrer une valeur : ")
print("Vous avez entrer comme valeur : ", ma_recuperation)
```

- Ces deux processus amènent rapidement à **deux composantes essentielles** de la programmation console :
  - **L'affichage** : le **formatage** des str.
  - **la récupération** : le **casting** des variables.

Récupérer et afficher des valeurs

- Si l'on veut **présenter** de façon claire des **informations** à l'utilisateur, il faut souvent se servir du **formatage**. Il existe plusieurs méthodes.
  - La méthode **.format()** qui est liée au type string. Pour s'en servir, il suffit de placer **deux accolades {}** dans une chaîne de caractère qui serviront **d'emplacement** prévus pour **l'afficher** des variables paramètres de la méthode format. Ces marqueurs peuvent être **numérotés** (en commençant par 0) pour indiquer quel paramètre de la méthode sera affiché. On pourra ajouter un formatage spécifique qui est le même que pour les f-strings (cf. diapo suivante).

```
nombre_A = 3.1415926553
print("nombre_a = {0:0.2f}".format(nombre_A)) # nombre_a = 3.14

nombre_B = 2
nombre_C = 21
print("{0:2d} {1:3d} {2:4d}".format(nombre_B, nombre_B**2), nombre_B**3) # 2 4 8
print("{0:2d} {1:3d} {2:4d}".format(nombre_C, nombre_C**2), nombre_C**3) # 21 441 9261
```

Récupérer et afficher des valeurs

- Le %-**formating**, maintenant déprécié et utilisé de nos jours dans de rares cas (formatage de dates, requêtes SQL)
- les **f-strings** sont des chaînes de caractères pour **formater directement le texte** en y incluant entre accolades les variables :

```
print(f"La valeur de nombre_a vaut {nombre_A:0.2f}") # nombre_a = 3.14
```

Fonctionnement des f-strings

Avancé

- Lorsque l'on utilise un f-string, on retrouve souvent ce genre de syntaxe :

```
variable = 55.2091
f"{variable:^7.2f}"
```

- Il s'agit en réalité d'un formatage rapide et simple d'utilisation pour les valeurs utilisées par le f-string.
- Ici la valeur sera :
  - f : avec un formatage dédié aux réels
  - ^: centrée
  - 7: Dans un espace de 7 caractères minimum au total (virgule, décimales, ...) on ajoutera le nombre d'espaces nécessaire si pas assez de caractères.
  - 2: avec toujours 2 chiffres après la virgule (arrondi si besoin)
  - Documentation sur les f-strings

Les raw-strings

- Similaire au f-strings, il existe aussi en python les **raw-strings**.
- Ce sont des chaînes dans lesquelles les **caractères spéciaux** comme le **backslash ** ne sont pas interprétés.
- Ils facilitent la saisie des chemins de fichier ou de regex par exemple.

```
print("\n{1}")
print(f"\n{1}")
print(r"\n{1}")
```

Cast des variables

- Lorsque l'on utilise un langage de programmation, on a fréquemment besoin de **passer d'un type de variable vers un autre.** Pour ce faire, on sert de ce qui s'appelle le "**cast**" (En français **transtypage**). Pour réaliser un **transtypage**, il faut utiliser **la fonction de cast** qui porte **le nom du type vers lequel on veut passer**

```
ma_string = "599.98"
mon_prix = float(ma_string)
print(f"Ma string vaut {ma_string} et est un type {type(ma_string)}")
# Ma string vaut 599.98 et est de type <class 'str'>

print(f"Mon prix vaut {mon_prix} et est un type {type(mon_prix)}")
# Mon prix vaut 599.98 et est de type <class 'float'>
```

- De plus, lorsque l'on cherche à obtenir un nombre de l'utilisateur, on peut également directement caster l'input de la sorte :

```
mon_nombre = int(input("Veuillez entrer un nombre : ")) # 25

print(f"Mon nombre vaut {mon_nombre} et est un type {type(mon_nombre)}")
# Mon prix vaut25 et est de type <class 'int'>
```

- Attention ! Le casting peut être la source de nombreux problèmes générant ce que l'on appelle des **exceptions** ! Nous verrons comment traiter les exceptions plus tard.

Cast en booléens

- Lorsque l'on fait un **cast** en **bool**, python applique certaines règles:
  - Les valeurs **None, False, 0, 0.0** et "" donnent forcément **False**
  - **Toutes les autres valeurs donnent True**
- En réalité toute valeur correspondant au vide pour son type sera considérée comme False dans une condition (cf : partie block conditionnels)

Les opérateurs arithmétiques

- Il existe plusieurs types **d'opérateurs**: unaires, binaires, arithmétiques, logiques, d'affectation, relationnels, ect... Il est possible d'utiliser les **opérateurs arithmétiques** sur les variables pour les manipuler.
- Il est possible de les **combiner** avec **l'opérateur d'assignation** pour effectuer l'opération directement sur une variable donnée.

```
mon_resultat = 4 + 4 # on affecte à la variable mon_resultat 4 + 4 soit 8
mon_resultat = mon_resultat - 4 # on affecte à la variable mon_resultat sa valeur - 4 soit 4
mon_resultat *= 2 # On multiplie la valeur de la variable par 2
mon_resultat /= # On divise (division entière)la valeur de la variable par 2
mon_resultat /= # On divise (division décimale)la valeur de la variable par 2
mon_resultat **= 3 # On passe la variable à la puissance 3
mon_resultat %= 3 # On consert le reste de la division de la variable par 3
```

- Il faut également savoir que l'opérateur **d'addition +** permet à deux variables de type **string** de s'ajouter l'une à la suite de l'autre, ce qui s'appelle la **concaténation**. Celui de **multiplication *** entre un **string** et un **int** permettra de faire ce qu'on appelle la **réplication**

Workshop N°1

1. **Calcul de la consommation énergétique**: Calculer la consommation énergétique totale d'un data center sur une journée donnée.

- **Consignes:**
  - Demandez à l'utilisateur d'entrer la consommation énergétique moyenne (en kW) de son data center par heure.
  - Demandez ensuite combien d'heures le data center a été en fonctionnement durant la journée.
  - Calculez la consommation énergétique totale de la journée.
  - Affichez le résultat en kW.

Workshop N°1

2. **Estimation des coûts de refroidissement**: Estimer le coût quotidien du refroidissement nécessaire pour maintenir le data center à une température optimale.

- **Consignes:**
  - Demandez à l'utilisateur d'entrer le coût par kWh (kilowatt-heure).
  - Demandez à l'utilisateur d'entrer la consommation énergétique moyenne du système de refroidissement par heure.
  - Demandez le nombre d'heures de fonctionnement du système de refroidissement dans la journée.
  - Calculez le coût total du refroidissement pour la journée.
  - Affichez le coût total.

Workshop N°1

3. **Simulation de panne de courant**: Calculer le temps de fonctionnement restant du data center en cas de panne de courant, en utilisant les batteries de secours.

- **Consignes:**
  - Demandez à l'utilisateur d'entrer la capacité totale des batteries de secours (en kWh).
  - Demandez ensuite la consommation énergétique moyenne du data center par heure (en kW).
  - Calculez combien de temps (en heures) le data center peut continuer à fonctionner en utilisant uniquement les batteries de secours.
  - Affichez le temps de fonctionnement restant.

Workshop N°1

5. **Vérification de l'état de maintenance d'un serveur**: Déterminer si un serveur est actuellement en maintenance ou non, sans utiliser de structures conditionnelles.

- **Consignes:**
  - Demandez à l'utilisateur si le serveur est en maintenance. L'utilisateur doit répondre par "oui" ou "non".
  - Convertissez la réponse de l'utilisateur en un booléen (True pour "oui", False pour "non").
  - Affichez un message indiquant si le serveur est en maintenance ou non, en utilisant directement la valeur booléenne dans la phrase.

Les opérateurs relationnels

- Il est également possible de comparer des valeurs via les opérateurs relationnels qui sont :
  - ">" : Supérieur à
  - "<" : Inférieur à
  - ">=" : Supérieur ou égal à
  - "<=" : Inférieur ou égal à
  - "==" : Égal à
  - "!=" : Différent de
- Le résultat des opérations utilisant ces opérateurs sera **bool**éen (**True/False**) permettant de savoir si le résultat de la comparaison est vrai ou faux.

Les opérateurs logiques

- Les valeurs de type booléennes peuvent être manipulées via les opérateurs logiques. En python, les 3 principaux sont **NOT, AND et OR**. On peut donner leurs résultats possibles sous forme de tables de vérités. Il existe notamment d'autres opérateurs.
- Ces opérateurs vont aussi donner comme résultat une variable de type **bool**éen.

```
variable_bool_A = True
variable_bool_B = False

variable_bool_C = variable_bool_A and variable_bool_B # False

  variable_bool_D = variable_bool_A or variable_bool_B # True
```

Table de vérité de ET		
a	b	a ET b
0	0	0
0	1	0
1	0	0
1	1	1

Table de vérité de OU		
a	b	a OU b
0	0	0
0	1	1
1	0	1
1	1	1

Autres opérateurs

- On retrouve d'autres types d'opérateurs notables :
  - **D'identité** : "**is**" et "**is not**" permettront de vérifier le type d'une variable (similaire à la fonction **isinstance()**)
  - **D'appartenance** : "**in**" et "**not in**" permettront de savoir si une variable est membre d'une autre.
  - Liés au **binaire** : les opérateurs **&, |, ^,~,<< et >>** permettent de travailler avec les valeurs binaires de nos variables, /!\ il ne faut pas confondre avec les opérateurs logiques AND, OR et NOT.

Les structures conditionnelles

- Si l'on veut permettre à nos programme de prendre **des chemins différents** de manière **conditionnelle** (saisies, calculs, ...) on a recours aux **structures conditionnelles**
- Pour réaliser une structure conditionnelle, on se sert des **clauses/mots clés** suivants :
  - **if** : initialiser la structure de contrôle, on donne une **condition** et on **exécute** le bloc d'instruction si elle est **vraie**.
  - **elif** : ajouter une **autre condition** dans le cas où la précédente **n'aura pas été validée**. On peut enchaîner ainsi **autant de structures elif que l'on souhaite**.
  - **else** : toujours la **dernière partie** d'une structure de contrôle, son bloc est exécuté dans le cas où **aucune des conditions précédentes n'a été validée**.

```
mon_age = int(input("Veuillez donner votre âge : "))

if mon_age >= 21 :
  print("Vous êtes majeur aux USA")
elif mon_age >= 18 :
  print("Vous êtes majeur en France")
else:
  print("Vous êtes mineur")
```

- Les clauses **elif et else** sont facultatives dans la structure conditionnelle, on en mets que si on en a réellement besoin.

Instruction pass

- L'instruction **pass** est une instruction à part de python
- Elle ne fait **absolument rien !**
- Il est régulier que l'on s'en serve de manière provisoire dans un bloc (if, else, for, fonction...) où l'on ne compte pas mettre d'instructions pour le moment.

match case

- Lorsque l'on travaille avec **la structure conditionnelle**, il est fréquent que l'on ait beaucoup de **elif** qui utilisent **la même variable**.
- Depuis la version **3.10** de python, il existe une nouvelle structure, le **pattern matching** ou structure **match...case**.

```
# Sans match case
if var == 1:
  print("une")
elif var == 2:
  print("deux")
else:
  print("ni une, ni deux")

# Avec match case
match var:
  case 1:
    print("une")
  case 2:
    print("deux")
  case _:
    print("ni une, ni deux")
```

Les ternaires

Avancé

- En python, il est possible d'utiliser ce qu'on appelle les **ternaire**, il s'agit d'une **expression** comportant une **condition**. On peut le comparer à une **structure conditionnelle** if.
- Il se structure comme suit :

```
temp = int(input("Saisir la température de l'eau : "))
etat = "solide" if temp < 0 else ("liquide" if temp <= 100 else "gazeux")
```

Workshop N°2

1. **Allocation de bande passante**: Allouer la bande passante à différents services dans un data center en fonction de leur priorité.

- **Consignes**:
  - Demandez à l'utilisateur d'entrer la bande passante totale disponible dans le data center (en Gbps).
  - Pour chaque service (par exemple, Web, Email, Base de données), demandez à l'utilisateur d'entrer la priorité du service (sur une échelle de 1 à 3, 1 étant le plus haut).
  - Demandez ensuite la demande de bande passante pour chaque service (en Gbps).
  - En fonction de la priorité, allouez la bande passante disponible aux services, en commençant par le service de priorité 1. Si la bande passante totale est insuffisante, répartissez-la proportionnellement en fonction de la priorité.
  - Affichez la bande passante allouée à chaque service.

Workshop N°2

2. **Classification des incidents**: Classer les incidents reportés dans un data center selon leur niveau de gravité en utilisant l'expression match.

- **Consignes:**
  - Demandez à l'utilisateur de saisir un niveau de gravité pour un incident (par exemple, "faible", "modéré", "élevé").
  - Utilisez une expression match pour associer le niveau de gravité à une action prédéfinie.
  - Affichez l'action à entreprendre selon le niveau de gravité de l'incident.

Workshop N°2

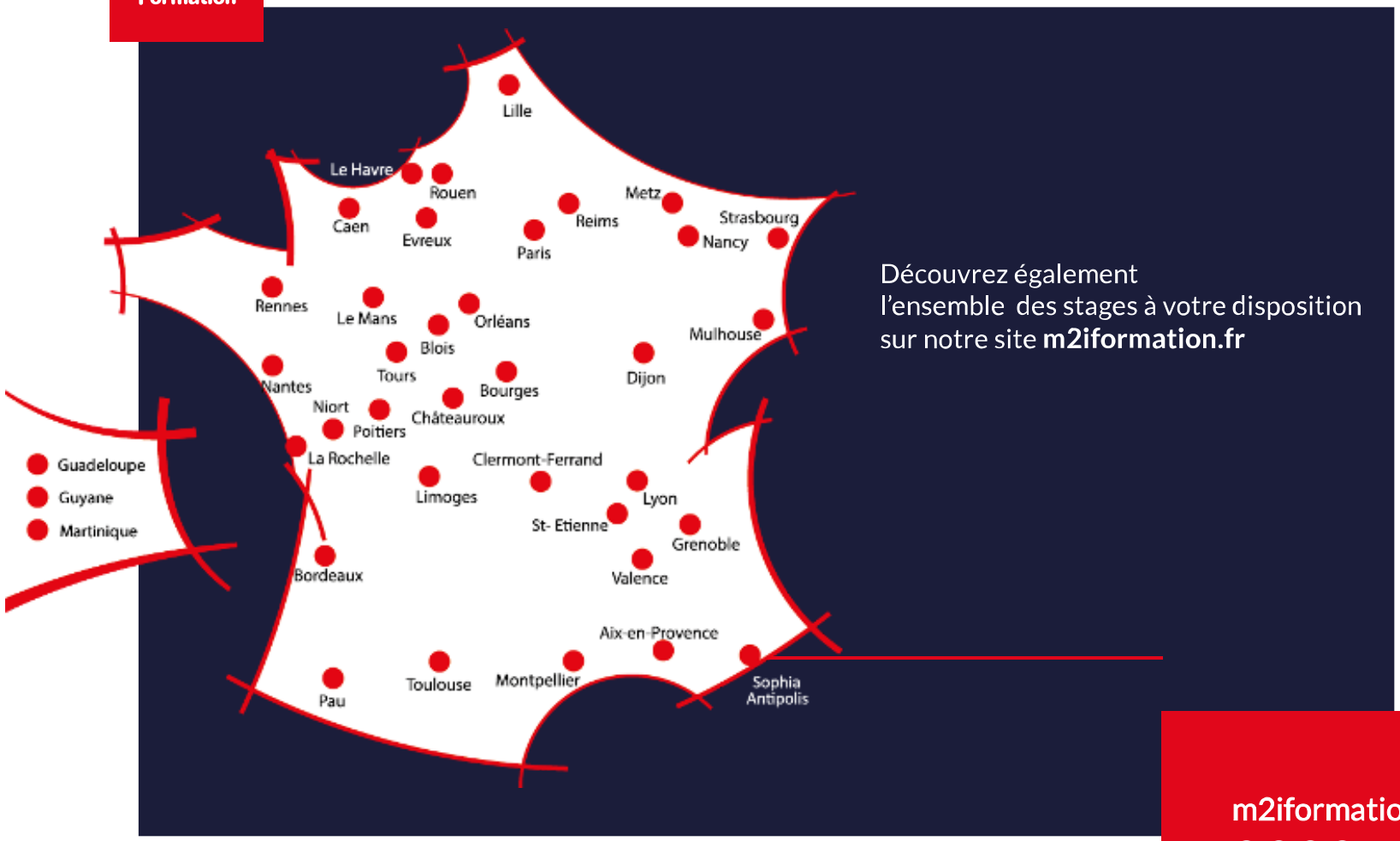
3.**Allocation de ressources basée sur la disponibilité**: Décider de l'allocation des ressources à un processus basé sur la disponibilité des ressources en utilisant un opérateur ternaire.

- **Consignes:**
  - Demandez à l'utilisateur le nombre total de ressources disponibles et le nombre de ressources demandées par un processus.
  - Utilisez un opérateur ternaire pour allouer les ressources si la demande est inférieure ou égale à la disponibilité, sinon refusez la demande.
  - Affichez le résultat de l'allocation.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr