Python

m2iformation.fr

Sommaire

Les structures itératives

- Il existe en Python deux façons de faire des **boucles/structures d'itération**. Les instructions du bloc seront exécutées à chaque **itération** de celle-ci.
  - La boucle "**while**" (Tant que ...) qui sera exécutée **tant que la condition spécifiée est vraie**
  - La boucle "**for...in...**" (Pour chaque... dans...) qui sera exécutée **pour chaque élément** d'un ensemble de type **conteneur ou interval** (fonctionne aussi avec les str). Elle met chaque élément un à un dans une **variable**.

```
for _ in range(0, 10):
  print("Je me répète !")

for element in [0, 1, 2, 3, 4, 5]:
  print(element)

for item in range(1, 11)
  print("Je suis l'itération n° : ", item)
```

- Lorsque l'on ne se sert jamais de la variable en question, le norme est de la nommer par "_"

Les structures itératives

- Lorsque l'on utilise une structure itérative, on peut également utiliser des **mots clés** durant l'itération, tels que :
  - "**continue**" : On passe à **l'itération suivante** en se replaçant au **début de la boucle**. On **ignore** alors tout ce qui aurait dù se dérouler **après le mot-clé**.
  - "**break**" : On **sort immédiatement de la boucle** sans effectuer les instructions après le mot-clé et dans les itérations suivantes.

```
while True:
  valeur = input("Saisir STOP pour arrêter le programme :")
  if valeur == "STOP":
    break
  elif valeur.upper() == "Stop":
    print("EN UPPERCASE")
    continue
  else:
    pass # Ce bloc est inutile, pass ne fait rien.
```

Workshop N°3

1. **Surveillance de Serveurs**: Vous devez vérifier l'état de fonctionnement de plusieurs serveurs. Pour cet exercice, nous simulerons la vérification de 5 serveurs.

- **Consignes**:
  - Utilisez une boucle pour simuler la vérification de chaque serveur.
  - Pour chaque serveur, demandez à l'utilisateur de saisir l'état du serveur (actif/inactif).
  - À la fin, affichez le nombre de serveurs actifs et inactifs.

Workshop N°3

2. **Vérification des Unités de Stockage avec Tentatives Limitées**: Un data center doit régulièrement vérifier l'état de santé de ses unités de stockage pour prévenir les défaillances. Chaque unité de stockage doit passer une série de tests. Si un test échoue, le système doit retenter jusqu'à un maximum de 3 fois avant de marquer l'unité comme nécessitant une inspection manuelle.

- **Consignes**:
  - Simulez la vérification de 3 unités de stockage en utilisant une boucle pour chaque unité.
  - Pour chaque unité, le système tente de passer un test (vous pouvez simuler cela en demandant à l'utilisateur si le test a réussi ou échoué).
  - Si un test échoue, le système retente jusqu'à 3 fois au total. Si les tests échouent après 3 tentatives, l'unité est marquée pour inspection manuelle.
  - Après avoir traité les 3 unités, affichez combien nécessitent une inspection manuelle.

Workshop N°3

3. **Surveillance des Performances de Réseau avec des Seuils**: Dans un data center, il est crucial de surveiller la performance du réseau pour s'assurer qu'elle répond aux exigences. Vous devez simuler un système de surveillance qui vérifie périodiquement la latence du réseau. Si la latence dépasse un seuil spécifique, des actions doivent être prises.

- **Consignes**:
  - Utilisez une boucle pour simuler la surveillance continue de la latence du réseau.
  - À chaque itération, demandez à l'utilisateur de saisir la latence actuelle du réseau.
  - Si la latence dépasse 100 ms, affichez un avertissement et demandez si des actions correctives ont été prises (l'utilisateur répond par oui ou non).
  - Si des actions correctives ont été prises et la latence revient en dessous de 100 ms dans les tentatives suivantes, continuez la surveillance; sinon, après 3 avertissements consécutifs, le système doit signaler un problème majeur et s'arrêter.
  - La surveillance s'arrête également si l'utilisateur saisit "stop" à tout moment.

Qu'est-ce qu'un conteneur ?

- Un **conteneur** est un **objet** permettant de stocker d'autres **objets**.
- Pour une **liste**, il est possible de stocker plusieurs **valeurs** au sein de la même variable.
- Les **conteneurs** sont **dynamiques**, ils peuvent contenir **plusieurs types de données** (int, str, float, list, object...)
- Pour plusieurs conteneurs, on peut **accéder aux valeurs** par utilisation d'un **index** ou d'une **clé**.

```
ma_liste = [1, 2, 3, 4, 5]
print(ma_liste) # [1, 2, 3, 4, 5]

mon_dict = {'Key1' : 123,'Key2' : '456', 'Key3' : [7, 8, 9] }
print(mon_dict) # {'Key1' : 123,'Key2' : '456', 'Key3' : [7, 8, 9] }

ma_tuple = (1, 'blabla', 3.14)
print(ma_tuple) (1, 'blabla', 3.14)

ma_liste = [1, 2, 3, 4, 5]
print(ma_liste[2]) # 3

mon_dict = {'Key1' : 123,'Key2' : '456', 'Key3' : [7, 8, 9] }
print(mon_dict["key2"]) # 456
```

Les listes

- La **liste** est le type de **conteneur** le plus utilisé.
- Elle permet de **manipuler** facilement ses données via l'utilisation de ses **méthodes**.
- Les **méthodes** des listes les plus utilisées sont les suivantes :
  - **sort()** : trie les éléments de la liste
  - **append(element)** : ajouter un élément à la fin de la liste.
  - **extend(list)** : ajouter une liste à la fin de la liste.
  - **pop(index)** : retirer un élément de la liste à l'index donné.
  - **remove(element)** : retirer le premier élément de la liste qui correspond
  - **count(element)** : nombre d'occurrences d'un élément.
  - **index(element)** : index de la première occurrence.

```
ma_liste = []
print(ma_liste) # []

ma_liste = [1, 2, 3]
print(ma_liste) # [1, 2, 3]

ma_liste = [2, 1, 3]
print(ma_liste) # [2, 1, 3]

ma_list.sort()
print(ma_liste) # [1, 2, 3]

ma_list.append(4)
print(ma_liste) # [1, 2, 3, 4]

ma_list.extend([5, 6])
print(ma_liste) # [1, 2, 3, 4, 5, 6]

ma_list.remove(4)
print(ma_liste) # [1, 2, 3, 5, 6]

ma_list.pop(2)
print(ma_liste) # [1, 2, 5, 6]
```

l'itération sur une liste

- **l'itération** est la capacité de **parcourir** (via généralement une boucle) une **série de valeurs** contenues dans un conteneur afin de les afficher ou d'en modifier les valeurs de façon séquentielle.
- Pour parcourir une liste, on utilise généralement une boucle **for**, telle que :

```
ma_liste = [1, 2, 3, 4, 5]
print(ma_liste) # [1, 2, 3, 4, 5]

for item in ma_liste:
  print(item)

'''
  1
  2
  3
  4
  5
'''
```

Les tuples

- Le tuple permet de **regrouper** des données, on appelle ça du **packing/construction**
- Les données sont **non-modifiables** et identifiées par leurs **indices/index**
- Syntaxes de définition :
  - **mon_tuple = ()** ou **mon_tuples = tuple ()** ou **mon_tuple=(1,2,3)** ou **mon_tuple=1,2,3**
  - Avec les mêmes méthodes que pour les listes, on pourra itérer sur les tuples et récupérer les valeurs à des index précis.
  - Les opérations sur un tuple :
    - **len()tuple** => nombre d'éléments d'un tuple.
    - **tuple.count(element)** => nombre d'occurrences d'un élément dans le tuple.
    - **tuple.index(element)** => index de la première occurrence de l'élément.
  - Avec une assignation à plusieurs variables, python propose **l'unpacking**.
  - exemple : var1, var2 = (1, 2)

Les sets

- Un **set** est un ensemble d'éléments **uniques** et **ordonnés**, les doublons sont **impossibles**, les valeurs doivent donc être **immutable**.
- Lors de **l'ajout** ou du **retrait** d'un élément d'un set, le conteneur se voit ainsi automatiquement **réordonné**. L'ordre définit par python n'est pas toujours très sensé...
- Lorsque l'on **cast** une list en set, on obtient une série d'éléments **sans doublons** qui ne peuvent plus être modifiés via leur **index** (les sets ne permettant pas la modification des éléments via cette méthode).
- Les sets contiennent des méthodes semblables aux listes mais n'en disposent pas de beaucoup.

```
mon_set = {1, 2, 3, 5, 5, 6}
print(mon_set)  # {1, 2, 3, 5, 6}
mon_set.add(4)
print(mon_set)  # {1, 2, 3, 4, 5, 6}
mon_set.pop()
print(mon_set)  # {2, 3, 4, 5, 6}
# mon_set[2] = 5 n'est pas possible pour un set
```

```
ma_list = [1, 1, 24, 3, 10, 3, 4, 5, 5, 54, 5, 6]
print(ma_list)  # [1, 1, 24, 3, 10, 3, 4, 5, 5, 54, 5, 6]
mon_set_2 = set(ma_list)
print(mon_set_2)  # {1, 3, 4, 5, 6, 10, 54, 24}
```

Méthodes des sets

- **add(element)** : ajout d'élément.
- **update(set)** : fusion de 2 sets.
- **remove(element)** : supprime l'élément s'il est présent, sinon erreur.
- **discard(element)** : supprime l'élément s'il est présent, sinon ne fait rien.
- **isdisjoint(set)** : si aucun élément n'est commun entre les deux.
- **issubset(set2)** : si le set est compris dans le set2 .
- **issuperset(set2)** : si les set2 est compris dans le set.
- On retrouve aussi les méthodes d'**union |**, **d'insertion &**, de **différence -** et de **différence symétrique ^**.

Les dictionnaires

- un **dictionnaire** est un conteneur se servant d'une association de **clés** et de **valeur**.
- Il est possible d'accéder aux valeurs qui le constituent via l'utilisation de la clé associée entre crochets. Avec le mot clé **del**, on peut supprimer une entrée
- Certaines méthodes du dictionnaire produisent des types spéciaux qu'il faudra **cast en list**.
  - **.values()** : récupère les valeurs.
  - **.keys()** : récupère les clés.
  - **.items()** : récupère des tuples (clés, valeurs).
- Via **L'unpacking** des tuples et la méthode **.items()**, il est possible d'afficher les informations du dictionnaire plus facilement.

```python
mon_dict = {'k1': 'valeur un', 'k2': 258963, 'k3': 3.14, 'k4': {1: 'blabla'}}
print(mon_dict)  # {'k1': 'valeur un', 'k2': 258963, 'k3': 3.14, 'k4': {1: 'blabla'}}
print(mon_dict['k3'])  # 3.14
print(mon_dict['k4'][1])  # blabla
```

```python
print(mon_dict.values())  # dict_values(['valeur un', 258963, 3.14, {1: 'blabla'}])
print(mon_dict.keys())  # dict_keys(['k1', 'k2', 'k3', 'k4'])

print(mon_dict.items())  # dict_items([('k1', 'valeur un'), ('k2', 258963), ('k3', 3.14), ('k4', {1: 'blabla'})])
```

```python
for key, value in mon_dict.items():
    print(f"Key : {key}, Value : {value}")  # Key : k1, Value : valeur un
```

Les dictionnaires

- Pour parcourir un dictionnaire, on utilise généralement une boucle for, on peut également accéder en complément de la valeur. Les clés des dictionnaires sont forcément de types immutables

```
mon_dict = {'key1': 123, 'key2': '456', 'key3': [7, 8, 9]}
print(mon_dict)  # {'key1': 123, 'key2': '456', 'key3': [7, 8, 9]}
for key, value in mon_dict.items():
    print(f"{key}: {value}")
```



```
{'key1': 123, 'key2': '456', 'key3': [7, 8, 9]}
key1: 123
key2: 456
key3: [7, 8, 9]
```

list/set/dict comprehension

Avancé

- Il est possible de **générer et d'itérer** sur des conteneurs à l'aide de la **comprehension**
- Pour la **liste comprehension**, la syntaxe est la suivante, un **itérable** est un objet sur lequel on peut itérer:
  - var = [expression for element in iterable]
- Il est aussi possible d'ajouter un filtre avec un if après l'itérable (équivalent à la fonction filter)

```
liste_d = [x for x in range(1, 11) if x % 2 == 0]
print(liste_d)

# équivalent
liste_a = []
for x in range(1, 11):
    if x % 2 == 0:
        liste_a.append(x)
print(liste_a)
```

```
# list comprehension avec les carrés de 0 à 9
ls = [x**2 for x in range(10)]

# dict comprehension avec lettre et leur valeur ascii
dic = {chr(n): n for n in range(65, 91)}
print(dic)

# tuple comprehension avec reduction d'une chaine
chaine = "abracadabra"
s = {char for char in chaine}
print(s)
```

Mutable / Immutable

- La **mutabilité** est la capacité d'une variable à être **modifiée**
- Il ne faut pas la confondre avec la **réassignation**, qui stocke simplement une autre variable à un autre emplacement mémoire.
- Les types non-mutables/immutable sont les **bool, str, int, bytes, range, tuple et frozenset**
- A contrario les list, dict et set sont par exemple mutables, il est possible de les modifier.
- Il faut cependant faire attention à leur utilisation au sein d'une fonction visant à les altérer, car leur valeur pourrait changer sans qu'on ne veuille.
- L'emplacement mémoire d'une variable mutable ne change pas après sa modification.

```python
mon_nombre = 5
print(id(mon_nombre))  # 2358276981104
print(mon_nombre)  # 5

mon_nombre += 2
print(id(mon_nombre))  # 2358276981168
print(mon_nombre)  # 7
```

```python
ma_liste = [1, 2, 3]
print(id(ma_liste))  # 2408503638912
print(ma_liste)  # [1, 2, 3]

ma_liste.append(4)
print(id(ma_liste))  # 2408503638912
print(ma_liste)  # [1, 2, 3, 4]
```

Workshop N°4

1. **Gestion des tâches de maintenance**: Apprendre à utiliser les listes pour gérer une file d'attente de tâches de maintenance dans un data center.

- **Consignes:**
  - Demandez à l'utilisateur de saisir des tâches de maintenance prévues pour la journée.
  - Stockez ces tâches dans une liste.
  - Affichez ensuite la liste complète des tâches, puis simulez l'exécution de chaque tâche une par une en les supprimant de la liste après leur "exécution".
  - Après chaque suppression, affichez la liste mise à jour des tâches restantes.

Workshop N°4

2. Atelier sur les Tuples: Suivi des serveurs: Utiliser les tuples pour stocker et accéder aux informations immuables sur les serveurs d'un data center.

- **Consgines:**
  - Créez un tuple contenant des informations sur un serveur, par exemple: nom, IP, type de serveur (web, base de données, etc.), et état (actif, inactif).
  - Demandez à l'utilisateur de saisir ces informations pour plusieurs serveurs et stockez-les dans une liste de tuples.
  - Parcourez cette liste pour afficher les informations de chaque serveur.

Workshop N°4

3. Atelier sur les Dictionnaires: Inventaire des ressources: Apprendre à utiliser les dictionnaires pour gérer un inventaire des ressources informatiques dans un data center.

- **Consignes:**
  - Créez un dictionnaire pour stocker l'inventaire, où les clés sont les types de ressources (ex: serveurs, routeurs, commutateurs) et les valeurs sont le nombre d'unités disponibles.
  - Permettez à l'utilisateur de mettre à jour cet inventaire en ajoutant ou en retirant des unités.
  - Affichez l'inventaire complet à la fin de l'exercice.

Workshop N°4

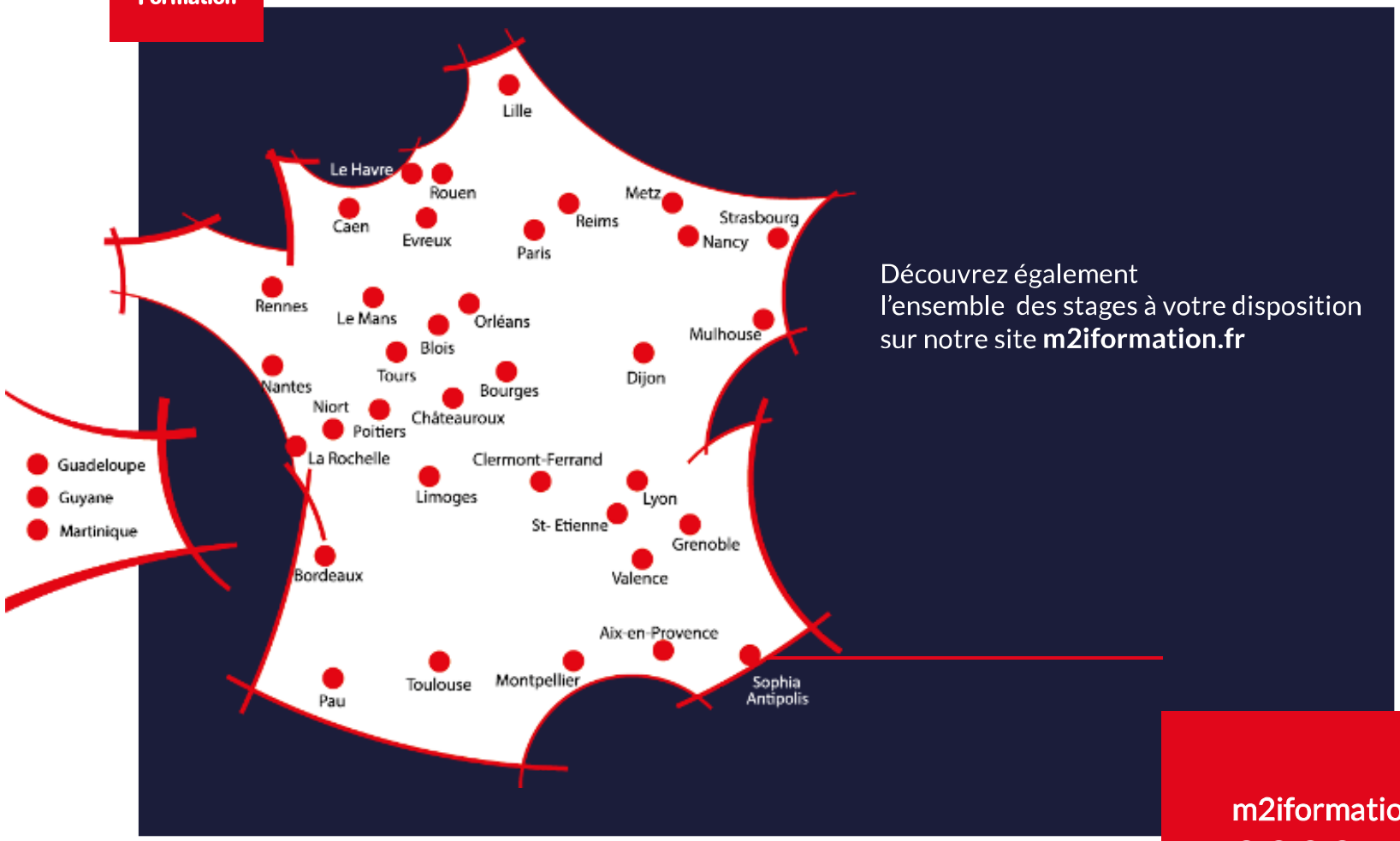
4. Atelier sur les Ensembles: Surveillance de l'accès réseau: Utiliser les ensembles pour surveiller et gérer les accès uniques au réseau d'un data center.

- **Consignes:**
  - Imaginez que chaque accès au réseau est enregistré avec un identifiant unique.
  - Utilisez un ensemble pour stocker ces identifiants au fur et à mesure qu'ils sont enregistrés.
  - Permettez à l'utilisateur de saisir de nouveaux identifiants et vérifiez si l'accès a déjà été enregistré.
  - Affichez le nombre total d'accès uniques à la fin.

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