Python

m2iformation.fr

Sommaire

Les Fonctions

- En programmation, les **fonctions** sont très utiles pour réaliser plusieurs fois la même opération au sein d'un programme.
- Une fonction effectue une tâche. Pour cela, elle **reçoit** éventuellement des **arguments** et **renvoie** une **valeur** ou **None** (Rien).
- Chaque fonction effectue en général une tâche **unique et précise**. Si cela se complique, il est plus judicieux d'écrire **plusieurs fonctions** (qui peuvent éventuellement s'appeler les unes les autres)
- Les valeurs passées à l'exécution de la fonction s'appellent des **arguments**.
- Les variables entre parenthèses qui **contiendront** ces valeurs sont les **paramètres**.

Les Fonctions

- Pour **définir** une fonction, Python utilise le mot-clé **def**.
- Si on souhaite que la fonction **renvoie** quelque chose, il faut utiliser le mot-clé **return**.
- Le nombre **d'arguments** que l'on peut passer à une fonction est **variable** et dépends du **nombre de paramètres**.
- Une particularité des fonctions en Python est que vous n'êtes **pas obligé de préciser le type** des **paramètres**, dès lors que les **opérations** que vous effectuez avec ces eux sont **valides**. Python est en effet connu comme étant un langage au "**typage dynamique**".
- Il est possible de passer un ou plusieurs argument(s) de manière **facultatives** et de leur attribuer une valeur par **défaut**. Pour cela, on utilise le **=**.

```
def carre(nombre: int):
  return nombre**2

res = carre(2) # retourne 2

def carre(nombre=3):
  return nombre**2

res = carre() # retourne 9
```

Les Fonctions

- Lorsqu'on manipule des fonctions, il est essentiel de bien comprendre comment se comportent les variables. Une variable est dite **locale** lorsqu'elle est créée **dans une fonction**. Elle n'existera et ne sera visible que lors de l'exécution de la dite fonction.
- Une variable dite **globale** lorsqu'elle est créée dans le **programme principal**. Elle sera visible partout dans le programme.
- Lorsque l'on essaie de modifier une variable globale à l'intérieur d'une fonction, il sera obligatoire d'utiliser le mot clé **global**

```
a = 10
def fonction():
  global a
  a += 1
```

Workshop N°5

1. **Gestion des Utilisateurs du Data Center**

- **Consignes:**
  - Créez une fonction ajouter_utilisateur qui prend comme arguments le nom de l'utilisateur et son rôle (ex: "Administrateur", "Visiteur", "Opérateur") et l'ajoute à un dictionnaire global utilisateurs, où le nom est la clé et le rôle est la valeur.
  - Écrivez une fonction changer_role qui modifie le rôle d'un utilisateur existant. La fonction doit vérifier si l'utilisateur existe avant de changer le rôle.
  - Développez une fonction afficher_utilisateurs qui imprime tous les utilisateurs et leurs rôles.

Workshop N°5

2. Suivi des Tâches de Maintenance

- **Consignes:**
  - Définissez une liste globale taches_maintenance où chaque élément est un tuple contenant l'ID de la tâche, la description, et le statut ("En cours", "Terminé").
  - Créez une fonction ajouter_tache pour ajouter une nouvelle tâche à la liste, en s'assurant que l'ID est unique.
  - Implémentez une fonction mettre_a_jour_tache qui change le statut d'une tâche basée sur son ID.

Workshop N°5

3. **Analyse des Logs du Système**

- **Consignes**:
  - Imaginez que vous avez une liste logs de chaînes de caractères, où chaque entrée représente un message log du système avec son niveau de priorité (ex: "ERROR: Échec de connexion", "INFO: Maintenance prévue à 23h00").
  - Créez une fonction filtrer_logs qui accepte un niveau de priorité et retourne une nouvelle liste contenant uniquement les logs de ce niveau.
  - Écrivez une fonction compter_logs qui retourne le nombre de logs pour chaque niveau de priorité, en utilisant une compréhension de dictionnaire.

Les lambdas

- Les lambdas sont des fonctions simplifiées à l'extrême et anonymes.
- Elles n'ont pas de nom et s'utilisent en général comme arguments d'autres fonctions (cf diapo filter, map, reduce).
- Elles doivent rester très simples (généralement 1 instruction).

```
fct = lambda x : x**2

def fct2(x):
    return x**2

print(fct(2))
print(fct2(2))
```

Sorted, Filtred, Map et Reduce

Avancé

- Pour aller plus loin dans l'usage des listes, il existe certaines fonctions utiles.
- Ces 4 fonctions utilisent les **fonctions ou lambdas** et nous simplifient beaucoup le travail avec les listes.
  - **sorted** : trier la liste selon certains critères
  - **filter** : filtrer les éléments de la liste
  - **map** : créer une nouvelle liste avec tous les éléments transformés par une fonction
  - **reduce** : réduire la liste à une seule valeur.

Les *args et **kwargs

- En python, il est possible d'ajouter des paramètres spéciaux précédés avant leurs noms par une ou deux étoiles.
- Leurs noms sont conventionnés, il est important de les nommer arg(arguments) et kwargs(keyword arguments)
- Ils permettent d'avoir des fonctions au nombre d'argument variable.

```
def ma_fonction(argument_classique, argument_par_défaut="valeur par défaut", *args, **kwargs):
    print(argument_classique)
    print(argument_par_défaut)
    print(args)
    print(kwargs)
```

*args

- Le paramètre *arg se transformera en tuple qui aura tous les arguments supplémentaires non nommés en son sein, il sera possible d'y accéder par leur index[]

```
def ma_fonction_avec_args(*args):
    for arguments in args:
        print(arguments)

ma_fonction_avec_args(1, "5", True, "Salut", "\na\nb\nc", "Hello World !")
ma_fonction_avec_args()
ma_fonction_avec_args("aaa")
```

**kwargs

- Le paramètre **kwargs se transformera en un dictionnaire qui contiendra un ensemble clé-valeur qui aura tous les arguments ayant un nom associé à une valeur via la syntaxe nom = valeur

```
def ma_fonction_avec_kwargs(**kwargs):
    print(kwargs)
    for kwarg_key, kwarg_value in kwargs.items():
        print(kwarg_key, kwarg_value)

ma_fonction_avec_kwargs(agument1="test", argument2=True, arg3=300)
```

Workshop N°6

1. Tri Personnalisé de Serveurs

- **Consignes**:
  - Vous avez une liste de tuples représentant des serveurs, où chaque tuple contient le nom du serveur et son utilisation du CPU en pourcentage (ex: [("Serveur1", 70), ("Serveur2", 20), ...]).
  - Écrivez une fonction trier_serveurs qui accepte la liste des serveurs et un argument nommé cle_tri utilisant **kwargs. Cette clé de tri doit permettre de trier soit par nom, soit par utilisation CPU.
  - Utilisez une fonction lambda à l'intérieur de trier_serveurs pour effectuer le tri en fonction de la cle_tri spécifiée.

Workshop N°6

2: Calculateur Flexible

- **Consignes:**
  - Créez une fonction calculer qui utilise *args pour accepter une liste de nombres et **kwargs pour accepter une opération mathématique spécifique (ex: somme, moyenne, produit).
  - Selon l'opération spécifiée dans **kwargs, utilisez une fonction pour calculer le résultat correspondant sur la liste de nombres.
  - Assurez-vous que votre fonction peut gérer les opérations "somme", "moyenne", et "produit".

Workshop N°6

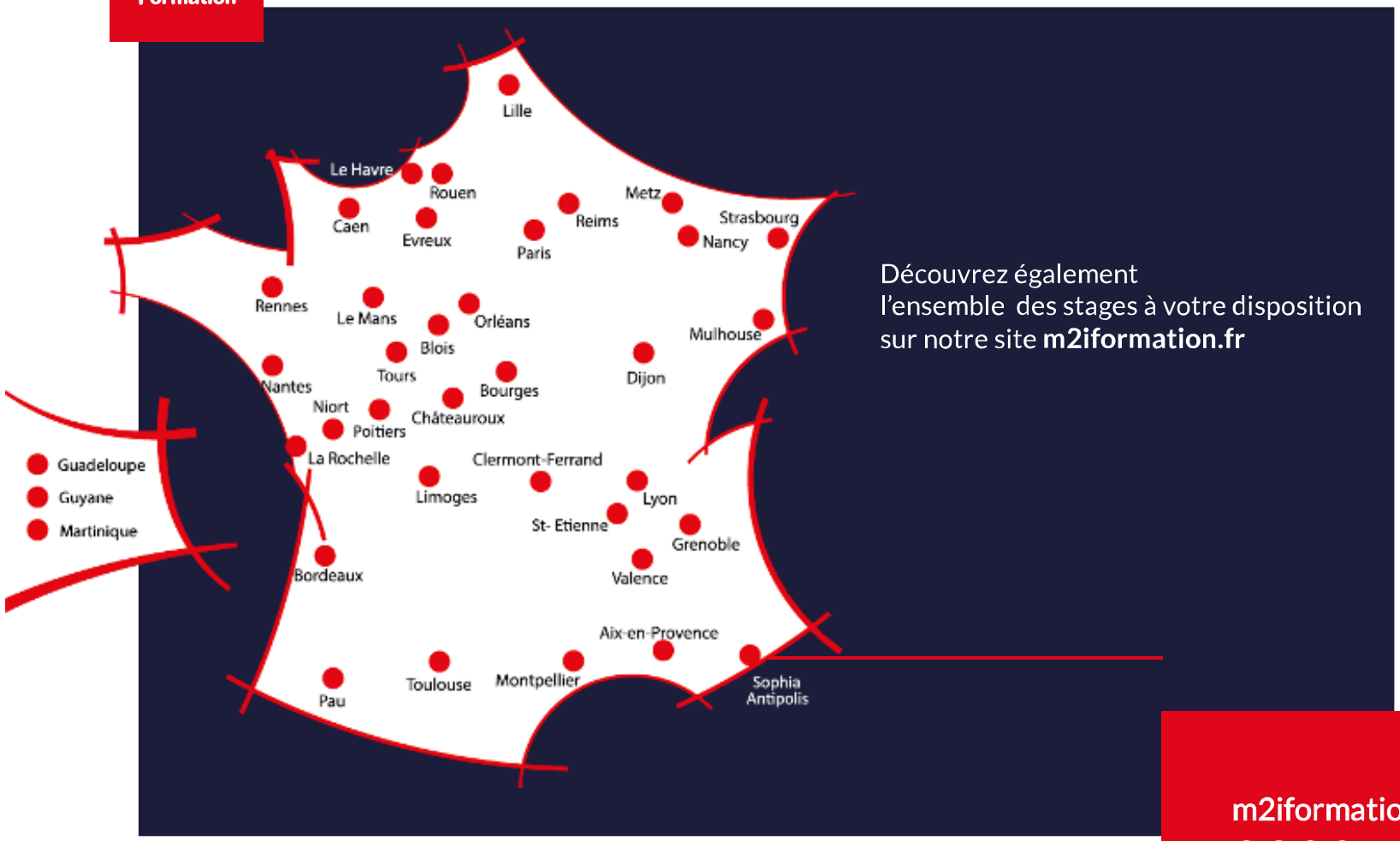
3: Filtrage Dynamique de Données

- **Consignes**:
  - Imaginez que vous avez une liste de dictionnaires représentant différents types de ressources dans un data center (ex: serveurs, disques de stockage, etc.), où chaque ressource a des attributs comme type, capacité, utilisation, etc.
  - Écrivez une fonction filtrer_ressources qui utilise *args pour accepter plusieurs critères de filtrage sous forme de tuples (attribut, valeur) et **kwargs pour spécifier une condition supplémentaire de filtrage basée sur un seuil d'utilisation.
  - Utilisez des fonctions lambda pour appliquer le filtrage et retourner une nouvelle liste de ressources filtrées.

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