Python OVH

m2iformation.fr

Sommaire

Modules

- Un **module** est un ensemble d'instructions provenant d'un **script** et qui peut être (ré)utilisé par d'**autres scripts**
- Intérêts : faciliter la réutilisation, la lisibilité, le débogage, le travail d'équipe
- Python vient avec un ensemble de modules natifs : scrypt, csv, datetime, math, ...
- Pour avoir la liste complète des modules fonction **help('modules')**.
- Python nous donne la possibilité de créer nos propres modules.

Modules

- Un **module** contient donc **l'ensemble des variables et des fonctions**, définies par les instructions du **script**. Il est entièrement exécuté au moment de l'instruction **import**

```
from math import pi

def circonference_cercle(rayon):
  return 2 * pi * rayon
```

- Ici le module nommé **circle** contiendra 2 éléments :
  - La variable pi
  - La fonction circonference_cercle
- On peut utiliser la fonction **dir(module)** pour en connaître le contenu.

Modules

- **L'importation** permet à un script d'utiliser le code d'un **autre module**.
- Syntaxes de l'importation :
  - import <nom_module>
  - from <nom_module> import , , ect...
  - from <nom_module> import *
- Syntaxe d'accès à un membre d'un module importé
  - .<nom_membre> avec la méthode 1
  - <nom_membre> avec la méthode 2 et 3
  - Il est possible d'ajouter un alias à un module ou un membre
    - import <nom_module> as mod_1
    - from <nom_module> import as f1

```
import circle
result = circle.circonference(10)

from circle import circonference
result = circonference(10)
```

La variable __ name __(variable Dunder)

- On retrouve souvent cette structure pour les scripts python, elle comporte beaucoup d'avantages quand on travaille avec des **imports**
- **__ name __** est une **variable** prédéfinie dans chaque module, elle contient :
  - La chaîne "__ main __" si on est dans le module principal, lancé directement depuis le script
  - Le **nom du module** quand on est dans un module importé **import**
- De ce fait, le bloc **if __ name __ == "__ main __"** n'est **exécuté** que dans le cas où l'on est dans le **module principal**. Les modules importés ne l'exécuteront pas.
- On peut voir ça comme du code verrouillé qui ne s'exécute que si on lance directement le script.

```
import math

def addition(a,b):
  return a + b

def main():
  print(addition(40, 3))

if __name__ == "__main__"
  main()
```

PEP 8

Le PEP 8, ou Python Enhancement Proposal 8, est le guide de style pour la rédaction de code Python. Il vise à améliorer la lisibilité et la cohérence du code Python à travers le monde. Voici quelques points clés et recommandations de PEP 8 pour écrire un code Python clair et maintenable :

1. **Indentation** : Utilisez 4 espaces par niveau d'indentation. L'indentation est cruciale en Python, car elle délimite les blocs de code.
2. **Longueur des lignes** : Limitez toutes les lignes à un maximum de 79 caractères pour le code et 72 caractères pour les commentaires et les docstrings. Cela garantit que le code est lisible sans avoir besoin de faire défiler horizontalement.
3. **Imports** :
   - Les imports doivent être placés en haut du fichier, juste après les commentaires et les docstrings du module, et avant les variables globales et les constantes.
   - Les imports doivent être groupés dans l'ordre suivant : bibliothèques standard, bibliothèques tierces, et enfin les imports locaux/application spécifiques. Chaque groupe doit être séparé par une ligne vide.
4. **Espaces** : Utilisez des espaces autour des opérateurs et après les virgules, mais pas directement à l'intérieur des parenthèses, des crochets ou des accolades. Par exemple, `a = f(1, 2) + g(3, 4)`.

PEP 8

5. **Commentaires** :
   - Les commentaires doivent être complets et concis.
   - Ils doivent être mis à jour lorsque le code change.
   - Les commentaires doivent expliquer le "pourquoi" plutôt que le "quoi".
6. **Conventions de nommage** :
   - **Noms de modules et de paquets** : Utilisez des noms courts et en minuscules, et si nécessaire, utilisez des underscores pour améliorer la lisibilité (ex. `mon_module`).
   - **Noms de classes** : Utilisez la convention CapWords (ou CamelCase), où la première lettre de chaque mot est en majuscule (ex. `MaClasse`).
   - **Fonctions et variables** : Utilisez des mots en minuscules séparés par des underscores (ex. `ma_fonction`).
   - **Constantes** : Utilisez des lettres majuscules avec des underscores séparant les mots (ex. `MA_CONSTANTE`).
7. **Conventions de codage** :
   - **Comparaisons** : Utilisez `is` ou `is not` pour comparer à `None` (par exemple, `if ma_var is None:`) et les opérateurs de comparaison (`<`, `>`, `!=`, etc.) pour les types numériques.
   - **Types de collections** : Préférez les constructions de type intégré (comme `dict`, `list`, `set`) aux formes de construction de type comme `dict()` ou `list()`.

Workshop Final : Système de Gestion des Incidents

Objectif: Concevoir un système pour enregistrer et suivre les incidents dans un data center. Ce système permettra d'ajouter des incidents, de mettre à jour leur statut, de supprimer des incidents résolus, et de générer des rapports sur les incidents en cours.

1. **Partie 1: Structure des Données**

- Stockez les incidents dans une liste de dictionnaires. Chaque dictionnaire doit contenir `id_incident`, `description`, `niveau_gravite` (faible, modéré, élevé), et `statut` (ouvert, en cours, résolu).

2. **Partie 2: Ajout d'Incidents**

- Créez une fonction pour ajouter un nouvel incident. Demandez à l'utilisateur de fournir une description, un niveau de gravité, et définissez automatiquement le statut de l'incident à "ouvert". Générez un ID unique pour chaque nouvel incident.

3. **Partie 3: Mise à Jour des Incidents**

- Implémentez une fonction pour mettre à jour le statut d'un incident basé sur son ID. L'utilisateur doit pouvoir changer le statut en "en cours" ou "résolu".

Workshop Final : Système de Gestion des Incidents

4. **Partie 4: Suppression des Incidents**

- Élaborez une fonction qui permet de supprimer un incident résolu du système en utilisant son ID.

5. **Partie 5: Rapports sur les Incidents**

- Développez une fonction pour afficher un rapport des incidents, permettant à l'utilisateur de voir tous les incidents ou seulement ceux filtrés par gravité ou statut.

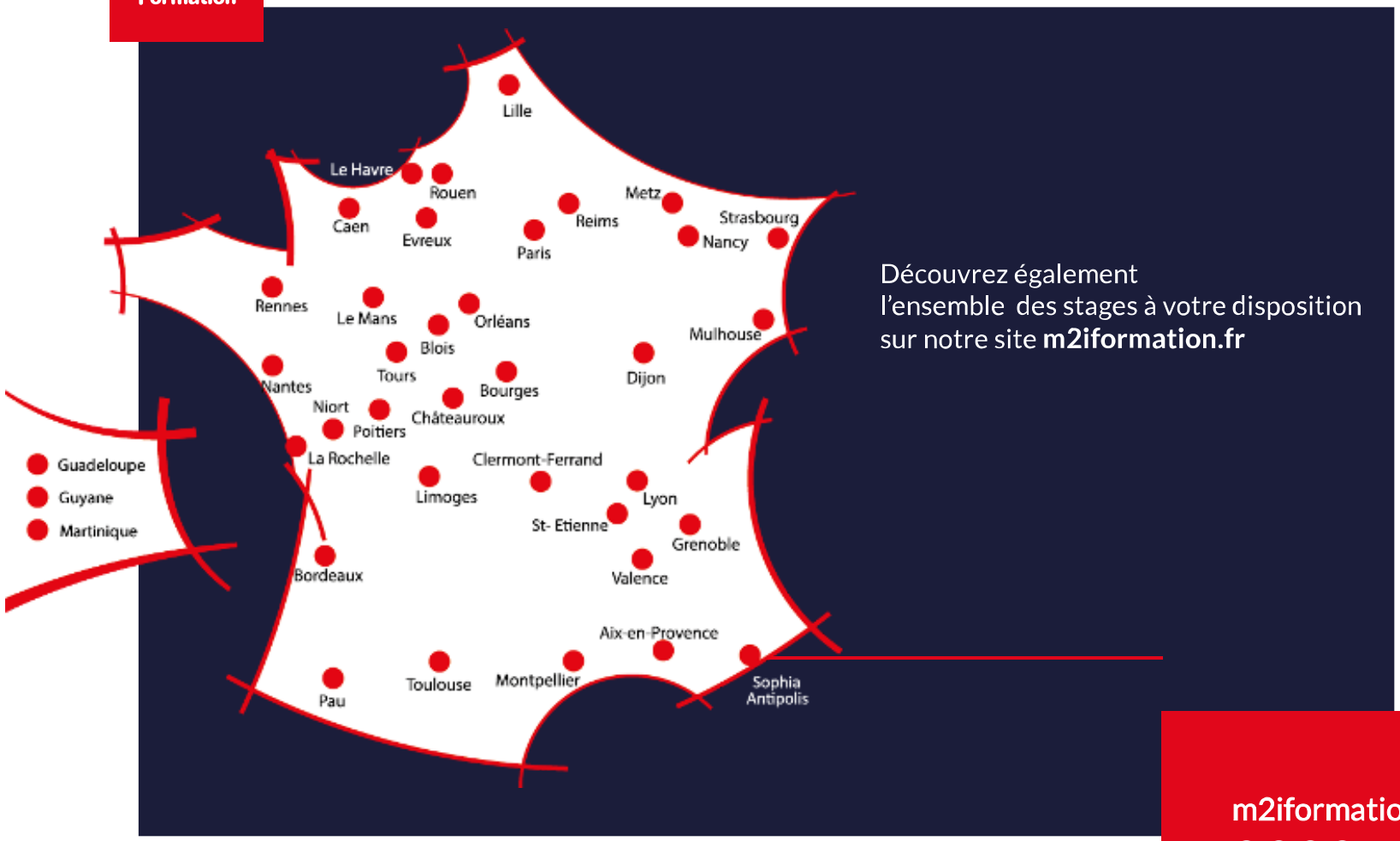
6. **Partie 6: Menu Interactif**

- Construisez un menu simple permettant à l'utilisateur de choisir des actions comme ajouter, mettre à jour, supprimer des incidents, ou afficher des rapports. Utilisez une boucle et des instructions conditionnelles pour gérer la sélection de l'utilisateur.

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