PYTHON

PYTHON POUR LA CYBERSECURITÉ

OBJECTIFS

- A la fin de cette partie, vous devriez être en mesure de :
  - Créer et manipuler des classes et objets en Python :
    - Définir des classes avec des attributs et des méthodes.
    - Instancier des objets et interagir avec eux.
  - Appliquer l'héritage pour réutiliser et étendre des classes :
    - Créer des sous-classes à partir de classes existantes.
    - Utiliser super() pour appeler les méthodes de la classe parent.
    - Surcharger des méthodes pour personnaliser le comportement.
  - Renforcer vos compétences en programmation orientée objet (POO) :
    - Structurer votre code de manière modulaire et réutilisable.
    - Appliquer les bonnes pratiques de la POO pour des programmes clairs et efficaces.
  - Comprendre et utiliser les bibliothèques et modules Python :
    - Importer des modules existants.
    - Créer vos propres modules pour organiser votre code.

CLASSES ET OBJETS EN PYTHON

- **Classes**: Modèle (ou plan) pour créer des objets
- **Objets**: Instance d'une classe contenant des données (attributs) et des comportements (méthodes).
- **Méthodes**: Fonction définie dans une classe et appliquée à un objet.
- **Propriétés de classe**: variables déclarées en dehors des méthodes. Changer la valeur d'une propriété de classe changera sa valeur **dans toutes les instances (objets) de cette classe.**
- **Propriétés d'objet**: variables déclarées à l'intérieur des méthodes en utilisant le mot clef **self** (**self.***propriété* = *valeur*). Les propriétés d'objet sont uniques à chaque instance
- **Constructeur** (__init__(self)): Méthode évoquée lors de l'instanciation d'une classe

CLASSES ET OBJETS EN PYTHON

- Déclarer une classe:
  ```
  class NomDeLaClasse
  ```
- Déclarer un constructeur:
  ```
  def __init__(self, …)
  ```
  Toutes les méthodes d'objets prennent self comme premier paramètre
- Déclarer une méthode:
  ```
  def nom_de_la_methode(self, arg1, arg2=None, …)
  ```
- Instancier un objet:
  ```
  obj1 = NomDeLaClasse()
  ```
- Appel à une méthode:
  ```
  obj1.nom_de_la_methode(0,1)
  ```
- Notons que arg1 est obligatoire alors que arg2 est optionnel et prend None (valeur nulle) comme valeur par défaut

CLASSES ET OBJETS EN PYTHON

```python
class Voiture:
    def __init__(self, marque, modele, couleur="Blanche"):
        self.marque = marque
        self.modele = modele
        self.couleur = couleur

    def afficher_info(self):
        print(f"{self.marque} {self.modele} {self.couleur}")

# Instanciation
ma_voiture = Voiture("Toyota", "Corolla", "Noire")
ma_voiture.afficher_info()  # Affiche : Toyota Corolla Noire

# Argument Optionel
ma_voiture = Voiture("Toyota", "Corolla")
ma_voiture.afficher_info()  # Affiche : Toyota Corolla Blance

# Argument nommé
ma_voiture = Voiture("Toyota", couleur="Grise", modele="Corolla")
ma_voiture.afficher_info()  # Affiche : Toyota Corolla Grise
```

HÉRITAGE EN PYTHON

- Permet de créer une nouvelle classe (classe enfant) à partir d'une classe existante (classe parent), réutilisant ainsi ses attributs et méthodes:
  ```
  class NomDeLaClasseEnfant(NomDeLaClasseParent)
  ```
- Surcharger une méthode, y inclut le constructeur, permet de redéfinir une méthode existante dans la classe enfant.
- Pour appeler une méthode de la classe parent:
  ```
  super().nom_de_la_fonction(args…)
  ```
- Python permet le multi-héritage, la classe finale héritera de toutes les fonctions des classes parents. Si 2 classes parents partagent des méthodes avec les mêmes noms, la classe enfant héritera de celle dont le nom vient en premier. super() référencie la première classe
  ```
  class NomDeLaClasseEnfant(NomDeLaClasseParent1, NomDeLaClasseParent2, …)
  ```

HÉRITAGE EN PYTHON

```python
class VoitureElectrique(Voiture):
    def __init__(self, marque, modele, autonomie):
        super().__init__(marque, modele)  # Appelle le constructeur de la classe parent
        self.autonomie = autonomie

    def afficher_info(self):
        super().afficher_info()  # Appelle la méthode de la classe parent
        print(f"Autonomie : {self.autonomie} km")

ma_voiture_ev = VoitureElectrique("Tesla", "Model 3", 500)
ma_voiture_ev.afficher_info()
```

MULTI-HÉRITAGE EN PYTHON

```python
class GPS:
    def __init__(self, gps_marque):
        self.gps_marque = gps_marque

    def afficher_gps(self):
        print(f"GPS : {self.gps_marque}")

class Voiture:
    def __init__(self, marque, modele):
        self.marque = marque
        self.modele = modele

    def afficher_info(self):
        print(f"{self.marque} {self.modele}")

class VoitureIntelligente(Voiture, GPS):
    def __init__(self, marque, modele, gps_marque):
        Voiture.__init__(self, marque, modele)
        GPS.__init__(self, gps_marque)

    def afficher_info_complet(self):
        self.afficher_info()
        self.afficher_gps()

ma_voiture_intelligente = VoitureIntelligente("Tesla", "Model S", "Garmin")
ma_voiture_intelligente.afficher_info_complet()
```

MODULES

- **Module** : Fichier Python contenant des fonctions, classes ou variables réutilisables. **Chaque fichier Python est un module**.
- Pour importer un module, utiliser la fonction `import` .

```python
# calcul.py
def aire_cercle(rayon):
    return 3.1416 * rayon ** 2

# mon_fichier.py
import calcul
print(calcul.aire_cercle(5))  # Affiche : 78.54
```

BIBLIOTHÈQUES (LIBRARIES)

- **Bibliothèque** : Ensemble de modules prêts à l'emploi (ex : math, random, numpy).

```
mon_fichier.py
ma_bibliotheque/
├── __init__.py
├── geometrie.py
└── calculs.py

# geometrie.py
def aire_rectangle(longueur, largeur):
    return longueur * largeur

# calculs.py
def addition(a, b):
    return a + b

# mon_fichier.py
from ma_bibliotheque.geometrie import aire_rectangle
import ma_bibliotheque.calculs

print(aire_rectangle(5, 3))  # Affiche : 15
print(ma_bibliotheque.calculs.addition(10, 7))       # Affiche : 17
```

PIP - INSTALLATION

- Python Package Installer: Un outil qui permet de gérer des paquets pour Python:
  - Installer des bibliothèques Python depuis le dépôt PyPI (Python Package Index).
  - Mettre à jour et désinstaller des paquets.
  - Gérer les dépendances automatiquement.
- Vérifier si pip est installé:

```
python -m ensurepip –version
```

- Installer pip:

```
sudo apt update
sudo apt install python3-pip
```

- Vérifier l'installation de pip:

```
pip --version
```

PIP - UTILISATION

- Installer une bibliothèque:
  `pip install nom_du_paquet`
- Vérifier l'installation:
  `pip show nom_du_paquet`
- Mettre à jour un paquet:
  `pip install --upgrade nom_du_paquet`
- Désinstaller un paquet:
  `pip uninstall nom_du_paquet`
- Lister les paquets installés:
  `pip list`

EXERCICE

- Classes et Objets, Héritage, Multi-Héritage
  - Créez une classe Personne avec nom, prenom et dob (date de naissance – optionnel – None par défaut), ainsi que la méthode info() qui imprime le nom, le prenom et la date de naissance si différent de None
  - Créez une classe Etudiant qui hérite de Personne et a) ajoute la propriété promotion:int , b) surcharge la methode info() pour appeler la méthode parente at imprime sur une nouvelle ligne la promotion, et c) ajoute la méthode semestre() qui calcule le semestre en cours à partir de la date et de la promotion
  - Créez une classe Contact avec mobile et mail, ainsi que la méthode info() qui imprime le mobile et le mail
  - Modifier Etudiant pour hériter de Contact en plus de Personne et ajouter la fonction info_complete() qui fait appel à la méthode info() de la classe Etudiant et à la méthode info() de la classe Contact

EXERCICE

- Module et Bibliothèque
  - Créez l'arborescence nécessaire pour la bibliothèque Ecole qui contiendra 3 modules: personne qui contient la classe Personne, contact qui contient la class Contact, et etudiant qui contient la classe Etudiant
  - Faire les imports nécessaires pour que la classe Etudiant fonctionne
  - Créez un module main qui importe la classe Etudiant et instancie 2 étudiants (1 avec dob et l'autre sans) et fait appel aux fonctions info() et info_complete()