**Python**

[**https://pushover.net/**](https://pushover.net/) **:Pushover permet d’envoyer des notifications via email quand il est connecté à un programme**

* **Exceptions**
* Le mot cle **finally**  permet d’exécuter code après un bloc try, quel que soit le résultat de l'exécution dudit bloc**.**
* **Les Assertions :** sont un moyen simple de s'assurer, avant de continuer, qu'une condition est respectée. En général, on les utilise dans des blocs try … except.
* **Lever une exception :** consister a généré une erreur provenant de la classe **BaseException**. Le bloc **try….except** permet ainsi capturer et de traiter cette exception.
* **Liste et Tuple**
* **chaine.split(…) :** permet de séparer une chaine de caractère en **List**
* **‘’…’’.join(liste) :** permet de transformer une liste en chaine de caractère.
* **Jour 17**
* **Programme de QUIZ**

1. Créer un objet **Question()** qui va recevoir nos questions et nos réponses :

class Question:  
 def \_\_init\_\_(*self*, text:str, answer: str):  
 *self*.text = text  
 *self*.answer = answer

1. Créer une banque de question dans un fichier .py , cette banque est en réalité une **liste de dictionnaire(question, réponse)**
2. Dans la fichier main.py, défini une liste qui va recevoir toutes nos questions, et cela va être faite à l’aide de la boucle **for.** Mais tout au début on doit importer notre fichier de banque de question et pour notre objet Question :

question\_bank = []  
for dict\_question in var:  
 new\_question = Question(dict\_question.get("question"), dict\_question.get("correct\_answer"))  
 question\_bank.append(new\_question)

# a5-

quiz = QuizBrain(question\_bank)

# a-6  
while quiz.still\_has\_question():  
 quiz.next\_question()

**a5-** Ensuite nous venons créer une Objet QuizBrain qui prendra comme argument **question\_bank**

**a6-** Ensuite dans notre main.py on fait une boucle qui aura pour condition la fonction **still\_has\_question()** et appellera la fonction **next\_question()** (conf. 5-)

1. Nous allons créer une nouvelle classe **QuizBrain** dans un nouveau fichier .py , ensuite écrit le constructeur qui prendra nos listes de questions en paramètre et initialisera **le numero de la question, la liste des question et le score.**

def \_\_init\_\_(*self*, question\_list):  
 *self*.question\_number = 0  
 *self*.question\_list = question\_list  
 *self*.score = 0

1. Ecrire une methode pour permettre à afficher une nouvelle question. Cette question est la propriété liste de notre classe QuizBrain en fonction du numero de la question comme indice.

def next\_question(*self*):  
 cur\_question = *self*.question\_list[*self*.question\_number]  
 *self*.question\_number += 1  
 user\_answer = input(f"Q.{*self*.question\_number }: {cur\_question.text} (True/False): ")

# a7-  
 *self*.check\_answer(user\_answer, cur\_question.answer)

**a7-**  on appelle la methode **check\_answer()** dans la methode next\_question()

1. Ecrire une fonction qui va permettre de savoir si on finit avec toutes nos question :

def still\_has\_question(*self*):  
 return *self*.question\_number < len(*self*.question\_list)

1. Ecrire une methode qui va vérifier les réponses de l’utilisateurs et faire un feedback, cette methode prendra **la réponse de l’utilisateur et la vrai réponse.**  En fonction de la réponse de l’utilisateur il va incrémenter son score.

def check\_answer(*self*, user\_answer, correct\_answer):  
 if user\_answer.lower() == correct\_answer.lower():  
 print("You go it right !")  
 *self*.score += 1  
 else:  
 print("That is wrong.")  
 print(f"The correct answer was: {correct\_answer}")  
 print(f"Your current score is: {*self*.score}/{*self*.question\_number}")

1. A la fin on affiche ce message :

print("You have completed the Quiz")  
print(f"You final score was: {quiz.score}/{quiz.question\_number}")

* **Variable de classe :**

C’est une variable qui est commune a tous les objets

**Exple :**

**class Personne :**

VAR\_CLASSE= value

**def \_\_init\_\_(seft) :**

var\_instance = value2

**Personne.VAR\_CLASSE = ‘’Paul’’** On accède de cette façon a la variable de classe.

* **Héritage : class Fille(Parent)**

on se sert de **super().methode\_Parent**  pour appeler une methode de la classe mère dans une classe fille.

* **Projet : Amazon Price tracking**
* **1- Avoir les entêtes du navigateur :**
* Vous pouvez voir les en-têtes de votre navigateur en vous rendant sur ce site : <http://myhttpheader.com/>
* **2- demander la page html du produit en utilisant la bibliothèque requête :**
* Utilisant le header de son navigateur venant du site ci-dessus

header = { **"User-Agent":** "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36",

**"Accept-Language":** "en-GB,en-US;q=0.9,en;q=0.8"

}

* Utiliser maintenant la methode requête :

response = requests.get(**url**, **headers**=header)

* Utiliser **lxml** comme parser a la place du **html.parser**
* **3- Se servir de BeautifulSoup** pour avoir le prix de l’objet
* **Day48 :** **Selenium**
* Télécharger chromeDriver en fonction de la version de **google chrome** (<https://chromedriver.chromium.org/downloads> )
* Déplacer le dossier dézipper dans un endroit(dossier)
* Copier le chemin de ce dossier et coller le dans votre projet.
* Ensuite installer **sélénium** qui est une bibliothèques qui contient des paquet qui vont nous aider à : **interagir avec le navigateur.**
  + Pour que Selenium travaille avec le navigateur(chrome), nous allons avoir **besoin d’un pont qui relie le code de sélénium avec le navigateur** . Ce pont est fourni par le **chromeDriver**
* La nouvelle version de sélénium nous permet de directement installer le package et d’utiliser pour cela on devra importer certaines bibliothèques :
* Pour Faire un clic sur une page on se sert de **click()**
* Pour écrire quelques chose dans le un **input** et appuyer sur entrer

Faut importer

from selenium.webdriver.common.keys import Keys