Défi Programmation

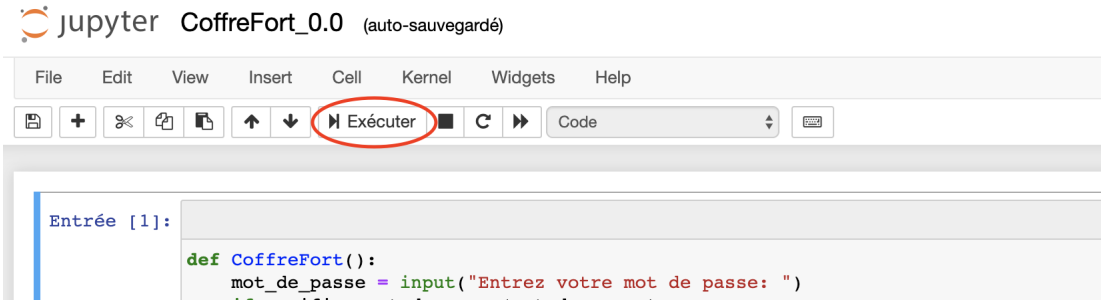
1 Mise en contexte

Bienvenue au défi de programmation ! Votre tâche consiste ici à déchiffrer le code **Python** qui vous est présenté dans chacun des fichiers pour en déceler le mot de passe. En somme :

- Vous avez **25 minutes** pour trouver **le plus de mots de passe possible**
- Chaque mot de passe **correctement** rentré vous **donne des points**

2 Fonctionnement

Sur votre écran, est présentée une liste de fichiers nommés "CoffreFort_XX.ipynb" numérotés. À chacun de ces fichiers, il y a un mot de passe à entrer afin de déverrouiller le coffre fort associé. Pour ouvrir un fichier, simplement **cliquer** dessus et il s'ouvrira dans un nouvel onglet. Dès que le fichier est ouvert, vous pouvez appuyer une fois sur le bouton 'Exécuter' pour débuter :



En lisant attentivement le code qui vous est présenté, vous devez déterminer quel est le mot de passe permettant d'ouvrir le coffre fort. Ce défi fait donc appel à votre logique, mais à quelques connaissances mathématiques. Une fois que vous pensez avoir trouvé le mot de passe, vous pouvez le rentrer dans la case prévue à cet effet :

```
Entrez votre mot de passe:
```

Afin de vous aider dans cette tâche difficile, voici quelques indications concernant le langage de programmation **Python** que vous aurez à comprendre.

3 Les fonctions

Dans chacun des fichiers, vous serez confrontés à deux fonctions, chacune début par 'def'. La première est toujours la même, et s'appelle CoffreFort(). **Ne portez pas attention à la fonction CoffreFort() !**, puisqu'elle ne sert qu'à vérifier que vous avez entré le bon mot de passe. Elle est facile à repérer puisqu'elle débute par :

```
def CoffreFort():
    ...
```

Votre objectif est de déchiffrer la fonction verifier_mot_de_passe(), car elle consiste en l'algorithme qui détermine le mot de passe.

Lorsque vous voyez une ligne comme celle ci-dessous :

```
    legume = "artichaut"
```

vous pouvez comprendre que `legume` est comme une boite où est sauvegardé le mot `"artichaud"`, `legume` est donc appelé une **variable**.
Pour décoder le mot de passe, on commence par porter attention à la **ligne de commande** suivante :

```
    def verifier_mot_de_passe(mot_de_passe):
```

`def verifier_mot_de_passe` est le nom de la **fonction** et `mot_de_passe` est une **variable** qui contient le mot de passe que vous avez inscrit.

Une **autre ligne** de commande importante est celle commençant par **`return`**

```
    return mot_de_passe == "1234"
```

Cette ligne vérifie que le mot de passe est valide avec le symbole "==", qui permet de comparer que deux variables sont les mêmes. Ici il faut donc inscrire "`1234`" comme mot de passe, qui va être dans la variable `mot_de_passe` pour réussir le coffrefort.

Mais quelle est la différence entre `1234` et `"1234"` ?
- le premier est le nombre mille deux cent trente-quatre
- le deuxième est un text `"1234"` il n'est reconnue on peut le voir grâce aux guillemet `" "`

les `" "` vont aussi être utilisé pour les mots comme : `"artichaut"` car c'est aussi une suite de caractère

Bien que `1234` et `"1234"` sont écrit de la même façon il se comporte de façon très différente :
- Nombre :

```
    dix = 10
    huit = 8
    dix + huit == 18
    dix - huit == 2
    huit * dix == 80
    huit / dix == 0.8
```

- Suite de caractères :

```
    dix = "10"
    huit = "8"
    dix + huit == "108"
    dix - huit == IMPOSSIBLE ERREUR
    huit * dix == IMPOSSIBLE ERREUR
    huit / dix == IMPOSSIBLE ERREUR
    dix * 3 == "101010"
    huit * 5 == "88888"
```

- La même chose est possible avec des lettres comme caractère

```
    "pizza" + "viande" + "miam" == "pizzaviandemiam"
    "pizza" * 4 == "pizzapizzapizzapizza"
```

4 Autres fonctions

Au travers des fichiers, vous retrouverez d'autres fonctions déjà incluses dans le langage **Python**. Les voici :

1. `len()`

```
salutation = "Bonjour"
len(salutation) == 7
```

La fonction `len()` retourne le nombre de caractères de la variable choisie (dans ce cas-ci `salutation` contient le mot `"Bonjour"`); puisque `"Bonjour"` contient 7 lettres, le résultat sera simplement 7.

2. `str()` et `int()`

```
username = "darkshadow" + str(34)
username == "darkshadow34"
```

Pour pouvoir fusionner du text et des chiffres il faut transformer les chiffres en text. C'est possible avec la fonction `str(chiffre)`.

```
calculatrice = 15 + int("10")
calculatrice == 25
```

Il est aussi possible de transformer des chiffres en text en utilisant `int()`. Ça permet de faire des additions de nombres sans avoir d'erreur.

```
Erreur = 15+"10"
#!?!#!? ERREUR #!#!?#!
```

3. `element[n]`

```
big = "BRAIN"
big[2] == "A"
```

Cette fonction sert à aller chercher le caratère à la position inscrit entre `[]` dans le text.

```
go = "SCIENCES"
go[0] == "S"
```

Attention ! Le premier caractère du text est en fait à la position 0 !

```
big = "BRAIN"
big[-3] == "A"
big[-1] == "N"
```

Les nombres négatifs permettent de partir de la fin du text en commençant par -1.

4. `element[n:m]`

```
go = "science"
go[1:4] == "cie"
```

Cette fonction permet de chercher plusieurs caractères de suite. En commençant par la lettre à la position `[ici:]` jusqu'à la lettre avant la position `[:ici]`.

```
go = "science"
go[0:-1] == "scienc"
```

On peut utiliser des nombres négatif

5. `range(n, m)`

```
range(0, 5) == [0,1,2,3,4]
```

Cette fonction est très utile pour les boucles `For i in range(0,5):`

5 Symboles mathématique python

1. ***opération :*** `//`

```
13 // 4 == 3
12 // 4 == 3
11 // 4 == 2
10 // 4 == 2
...
```

Ce symbole représente la division entière, elle ne garde rien après la virgule.

2. ***opération :*** `%`

```
13 \% 4 == 1
12 \% 4 == 0
11 \% 4 == 3
10 \% 4 == 2
...
```

Ce symbole représente l'opération modulo, elle garde le reste de la division.

3. ***opération :***

```
2 ** 1 == 2
2 ** 2 == 4
2 ** 3 == 8
...
```

Ce symbole est utilisé pour représenter un exposant.