introduction

February 19, 2023

1 Défi informatique

Ce document regroupe quelques éléments de base de la programmation informatique avec Python.

1.1 Environnement Jupyter

Vous réaliserez le défi informatique dans un environnement Jupyter. Une page Jupyter contient des cellules de texte, et des cellules de code. Vous ne devez pas modifier les cellules de texte, mais vous devrez modifier les cellules de code lorsque ce sera écrit de le faire. Pour **modifier** une cellule de code, il suffit de cliquer dessus et d'écrire à l'endroit que vous voulez. Vous devrez ensuite **exécuter** la cellule à l'aide des touches 'ctrl+enter', ou encore en utilisant le bouton 'Run' dans le haut de la page Jupyter. Si vous éprouvez des difficultés, levez votre main et un bénévole viendra vous aider.

Chaque énigme est une page Jupyter ouverte sur votre ordinateur. Ces pages sont notées de 01 à 08. La réussite des énigmes vous permettra d'obtenir des indices essentiels à la trouvaille d'un trésor sous la mer! Pour chaque énigme réussie, écrivez l'indice obtenu sur la feuille à votre table (les indices sont également numérotés de 01 à 08). Votre mission est de réussir le plus d'énigmes possible dans un temps donné.

POINTAGE: Le défi est noté sur **1200 points**. Chaque énigme réussie vaut **150 points**.

Bonne chance *hacker*!

1.2 Objets et affectation

D'abord, il existe différents **types d'objet**, dont les nombres et les chaines de caractères. Nous pouvons affecter un objet à une **variable** à l'aide de l'opérateur `=`.

[1]:
```
variable_1 = 3 # un nombre
variable_2 = "abc" # une chaîne de caractères

print(variable_1)
print(variable_2)
```

```
3
abc
```

Notez bien qu'une chaîne de caractères est délimitée par des apostrophes (`"`), mais pas une variable.

1.3 Appel de fonctions

Nous pouvons également définir des **fonctions**:

[2]:
```
def addition(a, b):
    return a + b

addition(2, 3)
```

[2]:
```
5
```

Dans l'exemple ci-dessus, la **fonction** `addition` accepte deux **arguments** `a` et `b`, et retournent la somme de ces arguemnts. Dans l'exemple, on dit que les nombres `2` et `3` sont donnés en argument à la fonction.

Dans le défi, vous n'aurez pas à coder de fonction, mais vous devez savoir comment les **appeler**. Pour appeler une fonction, il suffit de spécifier son nom (`addition` dans l'exemple précédent), les parenthèses, et les objets donnés arguments séparés d'une virgule entre les parenthèses (`(2, 3)` dans l'exemple précédent).

1.4 Découpage d'objets

Certains objets, par exemple les chaînes de caractères et les tableaux, peuvent être **découpés** à l'aide de l'opérateur `[début:fin]` où les indices `début` et `fin` délimitent le découpage. Les **indices débutent à `0`**, donc le premier élément correspond à l'indice `0`.

Dans l'exemple ci-dessous, nous définissons la chaîne de caractères `abcdef`, puis cette chaîne est découpée. Le découpage `[:3]` débute au premier caractère de la chaine (valeur par défaut lorsque l'indice `début` n'est pas spécifié) et termine au troisième caractère (indice `2`). Il est important de noter que l'élément à l'indice de fin n'est jamais inclus dans le découpage. Le découpage `[-3:]` débute au troisième caractère **à partir de la droite**, et termine au dernier caractère de la chaine (valeur par défaut lorsque l'indice `fin` n'est pas spécifié).

[3]:
```
chaine = "abcdef"

print(chaine[:3])
print(chaine[-3:])
```

```
abc
def
```

Les découpages de tableaux fonctionnent de manière similaire. On découpe d'abord sur les **lignes** du tableau, puis sur les **colonnes**.

Dans l'exemple ci-dessous, nous définissons `tableau`, puis ce tableau est découpé. Le découpage `[1, 1:]` spécifie que nous découpons la deuxième ligne (indice `1`) et de la deuxième colonne jusqu'à la dernière (indices `1:`). Notez bien que les découpages sur les ligne et les colonnes sont séparés d'une virgule (`,`) dans le découpage `[1, 1:]`.

[4]:
```
import numpy as np

tableau = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(tableau)
print()
```

```
print(tableau[1, 1:])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

[5 6]
```

1.5 Boucles

Les boucles permettent de répéter une même opération plusieurs fois. Les termes “boucle” et “itération” sont des synonymes. Une boucle `for` permet **d’itérer** à travers les éléments d’un objet. La structure d’une boucle `for` est `for <element> in <objet>:`. Les opérations **à l’intérieur** de la boucle sont exécutées pour chaque élément de l’objet. Une opération est à l’intérieur de la boucle lorsqu’elle est **indentée** par rapport à la définition de la boucle. Une **indentation** correspond à quatre (4) espaces (vous pouvez également utiliser la touche ‘tab’ du clavier pour faire les 4 espaces d’un coup).

Dans l’exemple ci-dessous, nous itérons à travers la chaîne de caractères `"abcdef"`. Les éléments `"a"`, `"b"`, `"c"`, etc. sont séquentiellement affectés à la variable `i`, puis cette variable est affichée à chaque itération à l’aide de la fonction `print`. Notez bien l’indentation de cette fonction par rapport à la boucle `for`.

[5]:
```
for i in "abcdef":
    print(i)
```

```
a
b
c
d
e
f
```