

Exercices - Gestion des erreurs

Les codes demandés dans la suite sont à écrire dans un fichier nommé *exercicesErreurs.py*, fichier qui sera à uploader sur nsibrantly.fr .

1- TRY .. EXCEPT POUR LA LECTURE D'UN FICHIER CSV :

1^{ère} partie :

On donne ci-contre, le script python d'une fonction *lecture()* .

L'exécution dans la console de cette fonction provoque la levée de l'exception « *FileNotFoundError* » car ce fichier n'existe pas pour l'instant.

Comme cette erreur n'est pas capturée, l'exécution s'arrête :

```
def lecture(nom) :  
    """  
    nom : nom d'un fichier  
    renvoie une liste  
    dont les éléments correspondent aux lignes  
    contenues dans le fichier  
    """  
    fichier = open(nom,'r',encoding='utf-8')  
    contenuString = fichier.read()  
    l = contenuString.split("\n")  
    fichier.close()  
    return l
```

```
>>> lecture("notesNsi.txt")  
Traceback (most recent call last):  
File "<console>", line 1, in <module>  
File "C:\Users\exercicesErreurs.py", line 8, in lecture  
fichier = open(nom,'r',encoding='utf-8')  
FileNotFoundError: [Errno 2] No such file or directory: 'notesNsi.txt'
```

⇒ En écrire dans le fichier *exercicesErreurs.py* , une version modifiée qui comprend une structure *try .. except* afin de pouvoir gérer l'exception « *FileNotFoundError* ». Une fois modifié, l'appel précédent n'entraînera plus un arrêt brutal :

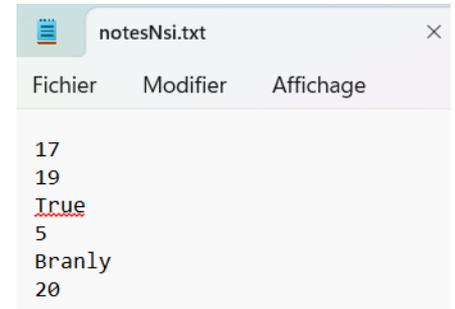
```
>>> lecture("notesNsi.txt")  
[Errno 2] No such file or directory: 'notesNsi.txt'
```

```
def lecture(nom) :  
    """  
    nom : nom d'un fichier  
    renvoie une liste  
    dont les éléments correspondent aux lignes  
    contenues dans le fichier  
    """  
    try :  
        fichier = open(nom,'r',encoding='utf-8')  
        contenuString = fichier.read()  
    except (FileExistsError,FileNotFoundError) as e :  
        print(e)  
    else :  
        l = contenuString.split("\n")  
        fichier.close()  
    return l
```

CORRIGE

2nd partie :

⇒ Sous windows, avec un clic droit dans votre répertoire de travail, créer un nouveau fichier texte que vous nommerez `notesNsi.txt`



⇒ Ecrire dans ce fichier les valeurs données ci-contre :

⇒ Exécuter alors `>>> lecture("notesNsi.txt")`
qui retournera la liste `['17', '19', 'True', '5', 'Branly', '20', '', '', '']`

3^{ème} partie :

On donne ci-contre, le script python d'une fonction `moyenne()` :

```
def moyenne(l) :  
    s = 0  
    for notes in l :  
        s = s + int(notes)  
    return f"La moyenne des notes contenues est {s/n}"
```

```
l = lecture("notesNsi.txt")  
moy = moyenne(l)  
print(moy)
```

L'exécution dans la console de cette fonction provoque la levée de l'exception « `ValueError` » car ce fichier n'existe pas pour l'instant.

Comme cette erreur n'est pas capturée, l'exécution s'arrête :

```
>>> (executing file "exercicesErreurs.py")  
Traceback (most recent call last):  
File "C:\Users\exercicesErreurs.py", line 40, in <module>  
moy = moyenne(l)  
File "C:\Users\exercicesErreurs.py", line 36, in moyenne  
s = s + int(notes)  
ValueError: invalid literal for int() with base 10: 'True'
```

⇒ Ecrire dans le fichier `exercicesErreurs.py`, une version modifiée de `moyenne()` qui comprend une structure `try .. except` afin de pouvoir gérer l'exception « `FileNotFoundError` ». Une fois modifié, l'appel précédent n'entraînera plus un arrêt brutal, mais l'affichage ci-contre dans la console :

```
>>> (executing file "exercicesErreurs.py")  
invalid literal for int() with base 10: 'True'  
invalid literal for int() with base 10: 'Branly'  
invalid literal for int() with base 10: ''  
invalid literal for int() with base 10: ''  
invalid literal for int() with base 10: ''  
La moyenne des notes contenues est 15.25
```

```
def moyenne(l) :  
    s = 0  
    n = 0  
    for notes in l :  
        try :  
            s = s + int(notes)  
            n = n + 1  
        except ValueError as e :  
            print(e)  
  
    if n == 0 : return None  
    else : return f"La moyenne des notes contenues est {s/n}"
```

CORRIGE

⇒ Modifier encore le script de cette fonction moyenne() en rajoutant un second try .. except qui puisse capturer une exception « TypeError » dans le cas où la variable mise en argument de soit pas une liste. Par exemple l'exécution suivant entraine l'arrêt du script :

```
>>> moyenne(2025)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "C:\Users\exercicesErreurs.py", line 35, in moyenne
    for notes in l :
TypeError: 'int' object is not iterable
```

Après modification, la même exécution donnera :

```
>>> moyenne(2025)
La variable mise en argument doit être une liste , 'int' object is not iterable
```

```
def moyenne(l) :
    s = 0
    n = 0
    try :
        for notes in l :
            try :
                s = s + int(notes)
                n = n + 1
            except ValueError as e :
                print(e)

        if n == 0 : return None
        else : return f"La moyenne des notes contenues est {s/n}"

    except TypeError as e :
        print(f"La variable mise en argument doit être une liste , {e}")
```

CORRIGE

4^{ième} partie :

Tester votre script avec à présent un contenu du fichier notesNsi.txt qui est quelconque. Par exemple :

⇒ L'exécution de

```
l = lecture("notesNsi.txt")
moy = moyenne(l)
print(moy)
```

Fonctionne-t-elle ?

```
notesNsi.txt
Fichier  Modifier  Affichage

je suis un grand aventurier
47
le capitaine a 10 ans
14
1
```

2- ASSERTIONS POUR TESTER UN SCRIPT :

On donne la suite de nombres suivante : 3 ⇒ 5 ⇒ 9 ⇒ 17 ⇒ 33 ⇒ 65 ⇒ 129 ⇒ ...

Une fonction nommée $u(n)$ retourne la $n^{\text{ième}}$ valeur de cette suite : $u(1) = 3$, $u(7) = 129$.

1^{ère} partie : Ecrire dans le fichier *exercicesErreurs.py* un script itératif de cette fonction $u(n)$. Y insérer une assertion qui vérifie que le paramètre n est strictement positif. Insérer un docString.

Dans la partie programme principal, écrire une assertion qui vérifie par exemple que $u(7) = 129$.

```
>>> u(-1)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "C:\Users\exercicesErreurs.py", line 70, in u
    assert n>=0, "L'argument doit être positif"
AssertionError: L'argument doit être positif
```

```
>>> help(u)
Help on function u in module __main__:

u(n)
    retourne la valeur de  $u(n) = 2u(n-1) - 1$ 
```

```
def u(n) :
    """
    |     retourne la valeur de  $u(n) = 2u(n-1) - 1$ 
    |     """
    assert n>=0 , "L'argument doit être positif"
    u = 3
    for i in range(2,n+1) :
    |     u = 2*u - 1
    return u
```

CORRIGE

2nd partie : Ecrire un script récursif de cette fonction $u(n)$. Y insérer les mêmes assertions.

```
def u(n) :
    """
    |     retourne la valeur de  $u(n) = 2u(n-1) - 1$ 
    |     """
    assert n>=0 , "L'argument doit être positif"
    if n == 1 : return 3
    else :
    |     return 2*u(n-1) - 1

assert u(7) == 129 , "Le test a échoué"
```

CORRIGE