ds2 mars2024

March 15, 2024

0.0.1 Exercice 1

exemple.txt content :

abcd
mnot
ijkl
efgh

[1]:
```
""" Question 1: 2pts """
def ligne_miroir(nom_fichier):
    lignes=[]
    try:
        f=open(nom_fichier, 'r')
        lignes = f.readlines()
        f.close()
    except :
        print(f"Le fichier {nom_fichier} n'existe pas.")

    lignes_inversees = [ligne.strip()[::-1]+'\n' for ligne in lignes]
    nom_fichier_sortie = nom_fichier.split('.')[0] + '_mirror.txt'
    g=open(nom_fichier_sortie, 'w')
    g.writelines(lignes_inversees)
    g.close()

    return lignes_inversees

ligne_miroir('exemple.txt')
```

[1]:
```
['dcba\n', 'tonm\n', 'lkji\n', 'hgfe\n']
```

[2]:
```
""" Question 2: 2pts """
def inverser_lignes(nom_fichier):
    lignes=[]
    try:
        f=open(nom_fichier, 'r')
        lignes = f.readlines()
        f.close()
```

```
    except :
        print(f"Le fichier {nom_fichier} n'existe pas.")

    lignes=[lignes.strip()+'\n' for lignes in lignes]
    lignes_inversees = lignes[::-1]

    g= open(nom_fichier, 'w')
    g.writelines(lignes_inversees)
    g.close()


    return lignes_inversees

inverser_lignes('exemple.txt')
```

[2]:
```
['efgh\n', 'ijkl\n', 'mnot\n', 'abcd\n']
```

[3]:
```
""" Question 3: 2pts """
def inverser(nom_fichier):

    ligne_miroir(nom_fichier) #on crée le fichier miroir
    lignes_inv=inverser_lignes(nom_fichier.replace('.txt','_mirror.txt')) #on␣
 ↪inverse les lignes du fichier miroir

    #On copie le contenu du fichier with suffixe '_mirror' dans le fichier␣
 ↪original
    f=open(nom_fichier, 'w')
    f.writelines(lignes_inv)
    f.close()

    return lignes_inv

inverser('exemple.txt')
```

[3]:
```
['hgfe\n', 'lkji\n', 'tonm\n', 'dcba\n']
```

0.0.2 Exercice 2

[4]:
```
''' 1- fonction balise_ouverture. 2pts '''

def balise_ouverture(txt):
    if txt.isalnum() and len(txt)<10:
        return "<{}>".format(txt) #ou return f"<{txt}>" ou return "<"+txt+">"
    else:
        return "<div>"
```

```
balise_ouverture("h1"), balise_ouverture("balise_tres_longue")
```

[4]: ('<h1>', ' ')

[5]:
```
''' 2-  Fonction balise_fermeture. 1.5pts '''

def balise_fermeture(tag):
    if tag.isalnum() and len(tag)<10:
        return "</"+tag+">"
    else:
        return "</div>"

balise_fermeture("h1"),balise_fermeture("balise_tres_longue")
```

[5]: ('</h1>', ' ')

[6]:
```
''' 3- Deuxième méthode -fonction balise_fermeture- 1.5pts'''

def balise_fermeture(tag):
    b_o=balise_ouverture(tag)
    return b_o.replace("<","</")
    # ou bien
    # return b_o(tag)[:1]+"/"+b_o(tag)[1:]

balise_fermeture("h1"),balise_fermeture("balise_tres_longue")
```

[6]: ('</h1>', ' ')

[7]:
```
''' 4- Écrivez la fonction verif_ele- 2pts'''

def verif_ele(texte):
    return texte.split('>')[0][1:]==texte.split('</')[-1][:-1]

verif_ele('<h1>Ceci est une chaîne valide</h1>'),verif_ele('<h1>Ceci est une␣
 ↪chaîne invalide</h2>')
```

[7]: (True, False)

[8]:
```
''' 5- fonction html_to_list- 2pts '''

def html_to_list(ch):
    chaines=ch.replace('><','>*<').split("*")
    l= []
    for chaine in chaines:
            l.append(chaine)
    return l
```

```
# ou bien

# def html_to_list(ch):
#     chaines=ch.split("><")
#     l= []
#     for chaine in chaines:
#         if verif_ele(f'<{chaine}>'.replace('>>','>').replace('<<','<')):
#             l.append(f'<{chaine}>'.replace('>>','>').replace('<<','<'))
#     return l

ch='<h1>This is a good string</h1><h2>This is a bad string</h2><div>this␣
 ↪another string</div><<h1>This is a good string</h1><li>This is a good␣
 ↪string</li>'

html_to_list(ch)
```

[8]:
```
['<h1>This is a good string</h1>',
 '<h2>This is a bad string</h2>',
 '<div>this another string</div>',
 '<<h1>This is a good string</h1>',
 '<li>This is a good string</li>']
```

[9]:
```
''' 6- fonction récursive html_to_list_general- 3pts '''

def html_to_list_general(ch):
    L0=[]
    # if bon_texte(ch) and ch.count('> ')!=1:
    if  ch.find('><')==-1 and ch.find('> ')==-1:
    # if  ch.count('><')==0 and ch.count('> ')==0:
        L0.append(ch)
        return L0
    else:
        tag=ch.split(">")[0][1:]
        tag_fermeture=balise_fermeture(tag)
        ch_decouper=ch.split(tag_fermeture)
        element1=ch_decouper[0]+tag_fermeture
        rest_chaine=balise_fermeture(tag).join([ele.strip() for ele in␣
 ↪ch_decouper[1:]])
        L0.append(element1)
        L0.extend(html_to_list_general(rest_chaine))
        return L0

ch0='<h1>This is a good string</h1>   <h2>This is a bad string</h2><div>this␣
 ↪another string</div><h3>This is a good string</h3><img>This is a good␣
 ↪string</img>'
```

```
ch1='<h1>This is a good string</h1>   <h2>This is a bad string</
 ↪h2>\n\t<div>this another string</div><h3>This is a good string</h3><img>This␣
 ↪is a good string</img>'
html_to_list_general(ch0),html_to_list_general(ch1),
```

[9]:
```
([<h1>This is a good string</h1>',
  '<h2>This is a bad string</h2>',
  '<div>this another string</div>',
  '<h3>This is a good string</h3>',
  '<img>This is a good string</img>'],
 ['<h1>This is a good string</h1>',
  '<h2>This is a bad string</h2>',
  '<div>this another string</div>',
  '<h3>This is a good string</h3>',
  '<img>This is a good string</img>'])
```

[10]:
```
''' 7- fonction html_to_dict . 2pts'''

def html_to_dict(ch):
    l=html_to_list_general(ch)
    d={}
    for ele in l:
        tag=ele.split('>')[0][1:]
        if tag in d:d[tag].append(ele)
        else:d[tag]=[ele]
    return d

ch='<h1>Titre 1</h1><h2>Titre 2</h2><h1>Titre 3</h1>'
html_to_dict(ch)
```

[10]:
```
{'h1': ['<h1>Titre 1</h1>', '<h1>Titre 3</h1>'], 'h2': ['<h2>Titre 2</h2>']}
```