PARCOURS D'UNE CHAINE DE CARACTERES ET RECHERCHE D'UN CARACTERE DANS LA CHAINE

Niveau : Première NSI

Durée : 1h

Objectifs :

- Recherche d'un élément dans une donnée indexée (chaîne de caractères)
- Parcours séquentiel d'une chaîne de caractères
- Etude du coût

Prerequis :

- Représentation d'un caractère
- Constructions élémentaires de Python : Affectation, séquences conditionnelles, boucles, fonctions
- Notions de compléxité

1 - Chaînes de caractères

On appelle **caractère** tout symbole qui peut être écrit :

- les lettres de l'alphabet latin : abcd...xyzABCD...XYZ
- les chiffres décimaux : 0123456789
- les symboles de ponctuation (y compris l'espace) : .,;:!?
- les symboles de parathésage : (){}[]
- et bien d'autres caractères comme les lettres accentuées àéùèÀÈ... et les lettres d'autres alphabets ou caratères spéciaux 千ʒ¢©§

Un **chaîne de caractère** (*string* en anglais) est une séquence de caractères, c'est-à-dire des caractères qui se suivent les uns derrière les autres. Une **chaîne vide** est une chaîne qui ne contient aucun caractère.

En python, les chaînes de caratères doivent être placées entre **quotes** :

- apostrophes simples (')
- guillemets doubles (")
- les triples apostrophes(''')
- ou des triples guillemets (""")

In [17]:

```
chaine1 = 'Bonjour'
chaine1
```

Out[17]:

```
'Bonjour'
```

In [18]:

```
chaine2 = "Aujourd'hui il fait chaud"
chaine2
```

Out[18]:

```
"Aujourd'hui il fait chaud"
```

Il existe des caractères particuliers qui n'entrainent aucun affichage. On trouve entre autres :

- **\n** permet d'insérer des **sauts à la ligne**

In [19]:

```
print("Une chaîne de caractères \nsur plusieurs \nlignes")
```

```
Une chaîne de caractères
sur plusieurs
lignes
```

- **\t** permet d'insérer des marques de **tabulation** dans une chaîne

In [20]:

```
print ("Une chaîne\t avec \ttabulations")
```

```
Une chaîne	 avec	 tabulations
```

Pour connaître la taille d'une chaîne de caractères, il est possible d'utiliser, sous Python, la **fonction len()**.

In [21]:

```
print(len(''))
print(len('a'))
chaine = "Bienvenue en NSI !"
print(len(chaine))
```

```
0
1
18
```

2 - Parcourir une chaîne de caractères

2.1 - Notation indicielle

Dans Python, une chaîne est manipulée comme une séquence indexée de caractères. Ainsi, **chaque caractère est accessible directement par son index ou indice**.

Le premier caractère (le plus à gauche) d'une chaîne de **longueur** n a pour **indice 0** et le dernier l'**indice** $n-1$.

Question : Justifier les résultats obtenus pour les lignes de code suivantes :

In [22]:

```
chaine = "Bienvenue en NSI !"
print(chaine[0])
print(chaine[5])
print(chaine[17])
print(chaine[20])
```

```
B
e
!

---------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
 last)
<ipython-input-22-6b0eeb60b840> in <module>
      3 print(chaine[5])
      4 print(chaine[17])
----> 5 print(chaine[20])

IndexError: string index out of range
```

En plus de cet accès unitaire aux caractères, il est possible d'accéder à des sous-chaînes en précisant la tranche souhaitée par l'indice de son premier caractère et l'indice du caractère suivant le dernier caractère de cette tranche.

Question : Justifier les résultats obtenus pour les lignes de code suivantes :

In [23]:

```
chaine = "Bienvenue en NSI !"
print(chaine[0:8])
print(chaine[:8])
print(chaine[13:])
```

```
Bienvenu
Bienvenu
NSI !
```

2.2 - Parcourir une chaîne de caractères

Il est très souvent nécessaire de parcourir une chaîne de caractères afin d'obtenir la réponse à un problème donné. Par exemple, si on veut compter le nombre d'espaces, le nombre d'occurences de tel ou tel caractère.

Question : Expliquer le rôle de la fonction suivante

In [24]:

```
def parcours1(chaine,caractere) :
    for i in range (0,len(chaine)) :
        if chaine[i] == caractere :
            return True
    return False
```

Question : Tester la fonction pour les arguments suivants. Conclure sur le rôle de la fonction **parcours1**.

In [26]:

```
parcours1('Bonjour','n')
```

Out[26]:

```
True
```

In [27]:

```
parcours1('Bonjour',' ')
```

Out[27]:

```
False
```

In [28]:

```
parcours1('Bienvenue en NSI !',' ')
```

Out[28]:

```
True
```

Question : Modifier le programme précédent afin de retourner l'indice de la première occurence d'un caractère dans la chaîne. Si le caractère n'est pas dans la chaîne il faudra retourner -1.

In [32]:

```
def parcours2(chaine,caractere) :
    for i in range (0,len(chaine)) :
        if chaine[i] == caractere :
            return i
    return -1
```

Question : Tester la fonction pour les cas suivants.

In [33]:

```
parcours2('Bienvenue en Numérique et Sciences Informatiques !','z')
```

Out[33]:

```
-1
```

In [34]:

```
parcours2('Bienvenue en Numérique et Sciences Informatiques !','B')
```

Out[34]:

```
0
```

In [35]:

```
parcours2('Bienvenue en Numérique et Sciences Informatiques !','n')
```

Out[35]:

```
3
```

Question : Quel cas (cas le pire), le nombre d'opérations élémentaires réalisées est maximal ? Donner la valeur maximale de ce nombre d'opérations pour les cas précédents

Réponse : Le cas entrainant un nombre maximal d'opérations est le cas où le caractère n'est pas présent dans la chaîne de caractères. Pour la 'Bienvenue en Numérique et Sciences Informatiques !' ce nombre est de 50 comme le nombre de caractère contenu dans la chaîne

Question : Donner l'ordre de grandeur de la compléxité de l'algorithme **"parcours2"**.

Réponse : L'ordre de grandeur de la compléxité de la fonction parcours2 est $O(n)$.

Question : Modifier la fonction **"parcours2"** afin de comptabiliser le nombre de boucles exécutées par la fonction.

In [36]:

```
def parcours3(chaine,caractere) :
    nb_op = 0
    for i in range (0,len(chaine)) :
        nb_op = nb_op + 1
        if chaine[i] == caractere :
            return i,nb_op
    return -1,nb_op
```

Question : Tester la fonction pour un pire des cas et justifier l'ordre de la compléxité donné plus haut.

In [37]:

```
parcours3('Bienvenue en Numérique et Sciences Informatiques !','z')
```

Out[37]:

```
(-1, 50)
```

Réponse : Pour $n = 50$, le nombre de boucles parcourues est de 50, donc n

3 - Recherche de l'occurence d'un caractère dans une chaîne de caractères

La fonction **occurence(chaine,caractere)** permet de retourner le nombre d'occurence du caractère dans la chaîne passés en paramètre.

Question : Compléter, ci-dessous, la fonction "occurence".

In [38]:

```
def occurence(chaine,caractere) :
    nb_occurence = 0
    for i in range (0,len(chaine)) :
        if chaine[i] == caractere :
            nb_occurence += 1
    return nb_occurence
```

Question : Tester la fonction pour différents cas.

In [39]:

```
occurence('Bienvenue en Numérique et Sciences Informatiques !','z')
```

Out[39]:

```
0
```

In [40]:

```
occurence('Bienvenue en Numérique et Sciences Informatiques !','n')
```

Out[40]:

```
5
```

In [41]:

```
occurence('Bienvenue en Numérique et Sciences Informatiques !','i')
```

Out[41]:

```
4
```

Question : Donner l'ordre de grandeur de la compléxité dans le pire des cas.

Réponse : Le cas le pire est lorsque la chaine de grandeur n est constituée de n fois le caractère passé en paramètre. Dans ce cas le nombre d'opération est : 1 affectation + n boucles + n affectations c'est-à-dire $2 * n + 1$ opérations. L'ordre de grandeur est donc $O(n)$

4 - Itérabilité des chaînes de caractères

Sous Python, une chaîne de caractère est un **objet itérable** c'est-à-dire qu'il est possible de la **parcourir directement à l'aide d'une boucle for**.

Le programme de recherche de caractère dans une chaîne, **"parcours1"**, peut alors s'écrire sous la forme :

In [43]:

```
def parcours1_V2(chaine,caractere) :
    for lettre in chaine :
        if lettre == caractere :
            return True
    return False
```

Question : Tester la fonction pour différents arguments

In [45]:

```
parcours1_V2('Bonjour','n')
```

Out[45]:

```
True
```

In [46]:

```
parcours1_V2('Bonjour',' ')
```

Out[46]:

```
False
```

In [47]:

```
parcours1_V2('Bienvenue en NSI !',' ')
```

Out[47]:

```
True
```

Question : Sur le même principe modifier le programme **"parcours2"** qui permet de retourner l'indice de la première occurence.

In [48]:

```
def parcours2_V2(chaine,caractere) :
    for i,lettre in enumerate(chaine) :
        if lettre == caractere :
            return i
    return -1
```

In [49]:

```
parcours2_V2('Bonjour','n')
```

Out[49]:

```
2
```

In [50]:

```
parcours2_V2('Bienvenue en NSI !',' ')
```

Out[50]:

```
9
```

Question : Toujours sur le même principe modifier le programme "occurence".

In [51]:

```
def occurence_V2(chaine,caractere) :
    nb_occurence = 0
    for lettre in chaine :
        if lettre == caractere :
            nb_occurence += 1
    return nb_occurence
```

Question : Tester la fonction pour différents cas.

In [52]:

```
occurence_V2('Bienvenue en Numérique et Sciences Informatiques !','B')
```

Out[52]:

```
1
```

In [53]:

```
occurence_V2('Bienvenue en Numérique et Sciences Informatiques !','N')
```

Out[53]:

```
1
```

In [54]:

```
occurence_V2('Bienvenue en Numérique et Sciences Informatiques !','e')
```

Out[54]:

```
9
```

5 - Pour aller plus loin

Question : Modifier le programme **"parcours2_V2"** qui permet de retourner les indices de chacune des occurences du caractère dans la chaîne. Tester le programme.

In [56]:

```
def parcours2_V3(chaine,caractere) :
    indices = []
    for i,lettre in enumerate(chaine) :
        if lettre == caractere :
            indices.append(i)
    return indices
```

In [57]:

```
recherche2_V3('Bienvenue en NSI !',' ')
```

Out[57]:

```
[9, 12, 16]
```

In [58]:

```
recherche2_V3('Bienvenue en NSI !','e')
```

Out[58]:

```
[2, 5, 8, 10]
```

Question : Modifier le programme "occurence_V2" qui permet de retourner les occurences de chacun des caractères présent dans la chaîne. Tester le programme.

In [59]:

```
def occurence_V3(chaine) :
    occurence={}
    for lettre in chaine :
        occurence[lettre] = occurence.get(lettre,0) + 1
    return occurence
```

In [60]:

```
occurence_V3('Bienvenue en NSI !')
```

Out[60]:

```
{'B': 1,
 'i': 1,
 'e': 4,
 'n': 3,
 'v': 1,
 'u': 1,
 ' ': 3,
 'N': 1,
 'S': 1,
 'I': 1,
 '!': 1}
```