#5 CODAGE DE TEXTE

> **Objectifs:**
>
> - Découvrir différents encodages de caractères
> - Encoder et décoder des chaînes de caractères
> - Manipuler des chaînes de caractères

Caractère et Système d'encodage

Un caractère peut représenter une lettre latine, un chiffre, un signe de ponctuation, une lettre arabe, un emoji, un hiérogliphe égyptien, ..., mais aussi un retour à la ligne, une tabulation, un caractère de contrôle, ...

Une chaîne de caractère (string) est une suite ordonnée de caractères.

Un système d'encodage associe à chaque caractère un code binaire et permet:

- l'encodage: du caractère au code binaire
- le décodage: du code binaire au caractère

Les principaux systèmes d'encodage:

- ASCII (7 bits): le latin de base (pour l'anglais)
- ISO-8859-1 (8 bits): ASCII + accents (latin 1- supplement)
- UTF-8 (de 1 à 4 octets) tous les caractères du standart unicode

https://www.unicode.org/charts/

Mission 5.1.

> Trouvez dans la table ASCII le code de la lettre 'A' et celui de la lettre 'a'
> Trouvez le code d'une lettre en arabe.
> Trouvez le code d'un emoji.

```
# Votre réponse ici
```

En Python:

En Python, le type pour un caractère ou une chaîne de caractère est `string`.

Une `string` est une sorte de liste Python (`list`) de caractères, donc on peut utiliser pratiquement tout ce qu'on a vu sur les listes Python (tout sauf l'affectation d'un élément de la liste):

- méthode pour la longueur du texte: `len( texte )`
- itération sur la longueur du texte: `for i in range (len( texte)):`
- itération sur les caractères du texte: `for a in  texte:`
- récupération d'un caractère avec l'indice: `texte[ i ]`
- opération de slicing: `texte[ i:j ]` , `texte[ :j ]` , `texte[ :j ]`

Pour encoder et décoder un caractère, on utilise les fonctions natives `ord()` et `chr( )`:

- `ord( c )` : prend en argument un caractère `c` et renvoie l'entier correpondant (en décimal)
- `chr( n )` : prend en argument un entier `n` (en décimal) et renvoie le caractère correspondant

Pour représenter un nombre entier dans une base autre que décimale:

- `0x41` : chaine de caractère avec le code `0x` pour hexadécimal suivi du nombre codé en hexadécimal
- `0b101001` : chaine de caractère avec le code `0b` pour binaire suivi du nombre codé en binaire
- `hex()` : base 10 à base 16
- `bin()` : base 10 à base 2
- `int( 0x... )` : hexadécimal à base 10
- `int( 0b... )` : binaire à base 10
- `int( '...' , n )` : base n à base 10

Pour encoder et décoder une chaîne de caractère, on utilise les méthodes natives `encode()` et `decode( )`:

- `str.encode( codage, error)` : prend en arguments optionnels le codage choisi ('UTF-8' par défaut), ainsi que la méthode pour gérer les erreurs de codage (soulever une erreur, remplacer le caractère,..) et renvoie la chaîne `str` codée en `byte` avec le système d'encodage choisi
- `byte.decode()` : prend en arguments optionnels le codage choisi ('UTF-8' par défaut), ainsi que la méthode pour gérer les erreurs de codage (soulever une erreur, remplacer le caractère,..) et renvoie le code `byte` décodé en `str` avec le système d'encodage choisi

https://www.w3schools.com/python/ref_string_encode.asp

Mission 5.2.

> Programmez une fonction qui affiche chaque caractère de la table ASCII sur une ligne comportant:
>
> > le caractère ; le code en hexadécimal ; le code en décimal ;le code en binaire

```
# Programmer ici

def printASCIITable( ):



```

Mission 5.3.

> Programmez une fonction qui convertit un texte en majuscule :
>
> > paramètre: texte à convertir (`string`) uniquement en caractères ASCII
> > retour: texte en majuscule (`string`)

```python
# Programmer ici

def myUpper ( texte ):




assert myUpper('abcd') == 'ABCD'
assert myUpper('aBcD') == 'ABCD'
assert myUpper('z1y;.\n') == 'Z1Y;.\n'
```

Mission 5.4.

> Programmez une fonction qui convertit un texte en ASCII en remplaçant les caractères non ASCII par un caractère donné en paramètre :
>
> > paramètres: texte à convertir (`string`), caractère de remplacement.
> > retour: texte convertit (`string`)

```python
# Programmer ici

def replaceNonASCII( texte  , c ):







assert replaceNonASCII('abcd', '?') == 'abcd'
assert replaceNonASCII('abcdé', '?') == 'abcd?'
assert replaceNonASCII('abΩcdé', '@') == 'ab@cd@'
```

Mission 5.5.

> Programmez une fonction qui vérifie si un texte est un palindrome:
>
> > paramètre: texte (`string`)
> > retour: True si le texte est un palindrom, False sinon (`boolean`)

```python
# Programmer ici

def isPalindrome( texte ):




assert isPalindrome( 'LOL') and isPalindrome( 'rever') and isPalindrome( 'kayak')
assert not isPalindrome( 'LOABOL')
# assert isPalindrome('si bene te tua laus taxat sua laute tenebis')
```

Mission 5.6.

> Programmez une fonction qui enlève les espaces éventuels à gauche et à droite d'un texte:
>
> > paramètre: texte (`string`)
> > retour: texte sans espace à gauche et à droite (`string`)

```
# Programmer ici

def myStrip( texte ):




assert myStrip( 'abc' ) == 'abc'
assert myStrip( ' abc' ) == 'abc'
assert myStrip( 'abc ' ) == 'abc'
assert myStrip( '   abc  ' ) == 'abc'
assert myStrip( '   a b   c  ' ) == 'a b   c'
```

Mission 5.7.

> Programmez une fonction qui sépare un texte selon un caractère spécifié et renvoie une liste dont les éléments sont les textes séparés:
>
> > paramètres: texte (string) , caractère de séparation (string)
> > retour: liste des textes séparés (list)

En option (++):

> même chose mais le séparateur est un texte (plusieurs caractères)

```
# Programmer ici

def mySplit( texte , c ):




assert mySplit( 'a;b;c', ';' ) == ['a' , 'b' , 'c' ]
assert mySplit( ' ab : c:d :' , ':' ) == [' ab ' , ' c' , 'd ', '' ]
assert mySplit( 'abcbbaa' , 'b' ) == ['a' , 'c' , '', 'aa' ]
```