TD2: TYPES DE DONNÉES.

Rappel sur les chaînes de caractères

Les chaînes de caractères sont données entre guillemets. Par exemple : `s = "bonjour"`.

Comme déjà mentionné, l'accès aux éléments et aux sous-chaînes d'une chaîne `s` s'effectue via les instructions `s[n]`, `s[-n]`, `s[m:n]`, `s[m:]` et `s[:n]`.

Les chaînes de caractères sont des structures de données non modifiables. Cela signifie que nous ne pouvons pas procéder à des réaffectation de leurs éléments :

```
>>> s =  "abba"
>>> s[2] = "z"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Le parcours d'une chaîne de caractères s'effectue via les instructions:

```
for variable in chaine:
  instructions...
```

Il existe diverses méthodes sur les chaînes de caractères, nous en aborderons quelques unes.

TD 2

Rappel sur les tests conditionnels `if... else...`

En Python, un tel test conditionnel `if... else...` est obtenu via la syntaxe suivante (les blocs `elif` et `else` sont facultatifs) :

```
if condition:
  instructions...
elif:
  instructions...
...
else:
  instructions...
```

TD 2

Exercice 1 (chaînes de caractères).

1. Initialisez une variables `c` contenant la chaîne de caractères contenant des chiffres et des lettres, du type `"X44bf38j23jdjgfjh737nei47"`.
2. Écrivez un programme qui construit deux chaînes de caractères `c_alpha` et `c_num` telles que `c_alpha` représente la suite des lettres de `c` et `c_num` représente la suite des chiffres de `c`. Effectuez un parcours de votre chaîne et utilisez un test conditionnel ainsi que les méthodes `str.isalpha()` et `str.digit()` qui permettent de déterminer si la chaîne `str` est composée de caractères alphabétiques ou numériques.
3. Écrivez un programme qui détermine si la sous-chaîne `j23` apparaît dans votre chaîne, et, si c'est le cas, qui la remplace par `j24`. La chaîne de départ `c` devra donc être modifiée dans ce cas. Utilisez un test conditionnel ainsi que les méthodes `str.find()` et `str.replace()` qui permettent de retrouver et remplacer des sous-chaînes dans la chaîne `str`.
4. Écrivez un programme qui détermine si la sous-chaîne `f37` apparaît dans votre chaîne, mais pas forcément de manière consécutive. En utilisant la méthode `str.find()`, recherchez le premier indice d'apparition du caractère `f`, puis celui de `3`, puis celui de `7` et vérifiez que ces indices forment bien une suite croissante.

TD 2

Exercice 2 (chaînes de caractères).

1. Initialisez une variables `texte` contenant la chaîne de caractères suivante :

```
"We introduce here the Python language"
```

   a) Créez un script qui compte le nombre de caractères de cette chaîne. Vous initialiserez une variable “compteur” à 0, puis, à chaque caractère parcouru, incrémenterez cette variable. Vérifiez que votre résultat corresponde bien à celui de l’instruction `len(texte)`.

   b) Créez un autre script qui compte le nombre de caractères de cette chaîne qui ne sont pas des espaces. Pensez à utiliser un test conditionnel dont la syntaxe est décrite ci-dessus.

   c) Sachant que dans cette chaîne de caractères, les mots sont séparés par des espaces, créez un script qui compte le nombre de mots de cette chaîne. Pensez encore une fois à utiliser un test conditionnel.

2. Initialisez une variables `texte2` contenant le texte suivant :

```
"We introduce here the Python language. To learn more about the language,
 consider going through the excellent tutorial https://docs.python.org/
 tutorial. Dedicated books are also available, such as
                        http://www.  diveintopython.net/."
```

Dans cette chaîne de caractères, les mots ne sont plus uniquement séparés par des espaces, mais également par des symboles de ponctuation. Testez votre script qui compte le nombre de mots sur cette chaîne de caractères. Votre script est-il toujours valable ? Si oui, pourquoi ? Sinon, comment devez-vous le modifier ? On aimerait que chaque adresse web ne soit comptée que comme un seul mot.

TD 2

Rappel sur les listes:

Les listes sont données entre crochets, les éléments étant séparés par des virgules. Par exemple : `ma_liste = [1, 2, 3]`

L'accès aux éléments et aux sous-listes d'une liste `l` s'effectue par la même syntaxe que pour les chaînes : on utilise les instructions `l[n]`, `l[-n]`, `l[m:n]`, `l[m:]` et `l[:n]`.

Contrairement aux chaînes, les listes sont des structures de données modifiables. Cela signifie que nous pouvons procéder à des réaffectation de leurs éléments :

```
>>> l =  [2, "abba", 3.7, True]
>>> l[2] = 101
>>> print l
[2, 'abba', 101, True]
```

Le parcours d'une liste s'effectue via les instructions :

```
for variable in liste:
  instructions...
```

Il existe diverses méthodes sur les listes, nous en aborderons quelques unes.

TD 2

Exercice 3 (Listes).

1. Créez un programme qui demande à l'utilisateur d'entrer trois mots à la suite, puis renvoie les trois mots triés par ordre alphabétique. Utilisez une liste pour stocker les trois mots. Construisez et triez la liste grâce aux méthodes `append()` et `sort()`, respectivement.
2. Modifiez votre programme de manière à ce que l'utilisateur puisse entrer autant de mots qu'il le souhaite. Le processus de saisie s'arrête lorsque l'utilisateur entre le mot "FIN". Utilisez une boucle `while`.

TD 2

Exercice 4 (Listes).

On considère les deux listes suivantes :

```
Couleurs = ["Pique","Trefle","Carreau","Coeur"]
valeurs = [2, 3, 4, 5, 6, 7, 8, 9, 10,"valet","dame","roi","as"]
```

1. À partir de ces deux listes, générer une liste `cartes` contenant toutes les 52 cartes sous forme de chaînes de caractères. Utilisez un double parcours de ces listes, c'est-à-dire une double boucle for, ainsi que la méthode `str.append()`.
2. Importez la fonction `shuffle` de la librairie `random` grâce à l'instructions `from random import shuffle`. Cette fonction `shuffle()` permet de mélanger une liste. Mélangez alors la liste de vos cartes.
3. Créez ensuite quatre listes `joueur1`, `joueur2`, `joueur3` et `joueur4` qui correspondent à la distribution votre jeu de cartes mélangé à quatre joueurs différents. Les cartes doivent être distribuées une par une et à tour de rôle. Utilisez un compteur qui compte modulo 4 pour simuler la distribution aux joueurs à tour de rôle. Utilisez un test conditionnel avec des conditions "elif".

TD 2

Exercice 5 (Listes, fichiers et listes de listes).

Le fichier `diamonds.csv` contient des données d'environ 54000 diamants. Nous allons récolter ces données sous forme de listes afin de pouvoir les manipuler.

1. Copiez le fichier `diamonds.csv` dans le répertoire dans lequel se trouve votre script actuel. En Python, il est possible de lire ce fichier et de créer une liste des lignes de ce fichier. Pour cela, utilisez instructions suivantes :

```
1 with open("diamonds.csv", "r") as f:
2     diamants = f.readlines()
```

La liste créée s'appelle `diamants`. Les éléments de cette liste sont des chaînes de caractères qui correspondent aux lignes du fichier. Quelle est la longueur de cette liste ? Afficher les premier, deuxième et troisième éléments de cette liste. Remarquer que le premier élément de la liste correspond aux variables mesurées sur les diamants. Les autres lignes correspondent aux données proprement dite.

TD 2

2. On aimerait maintenant que chaque élément de la liste soit une liste plutôt qu'une chaîne de caractères. Pour cela, on utilise la méthode `split()` qui permet de couper une chaîne en une liste d'éléments. Cette méthode est illustrée ci-dessous :

```
>>> diamants[2]                  # chaine de caracteres
'0.21,"Premium","E",59.8,326,3.89,3.84,2.31\n'
>>> diamants[2].split(",")       # chaine transformee en liste
['0.21', '"Premium"', '"E"', '59.8', '326', '3.89', '3.84', '2.31\n']
```

Testez cette commande. Écrivez un programme qui applique la méthode `split()` à tous les éléments de votre liste `diamants`. Utilisez un parcours de la liste `diamants` et un compteur qui représente le numéro de ligne courant.

3. Créez une liste `diamants_100` qui contient les 100 éléments de la liste `diamants` qui succèdent au tout premier élément. Affichez les 20 premiers éléments de `diamants_100`.

4. Créez une liste `diamants_prix` qui contient les prix des 54000 diamants convertis en nombres réels (type `float`). Utilisez un parcours de la liste `diamants_prix`.

Affichez les 20 premiers éléments de `diamants_100`.

TD 2

Rappel sur les tuples

Les listes sont données entre parenthèses, les éléments étant séparés par des virgules.
Par exemple : `ma_tuple = (1, 2, 3)`

L'accès aux éléments et aux sous-tuples d'un tuple `t` s'effectue par la même syntaxe que pour les listes : on utilise les instructions `t[n]`, `t[-n]`, `t[m:n]`, `t[m:]` et `t[:n]`.

Contrairement aux listes, les tuples sont des structures de données non modifiables. Cela signifie que nous ne pouvons pas procéder à des réaffectation de leurs éléments, ni modifiez leur longueur, etc. :

```
>>> t = (1, 2, 3, "good", True)
>>> t[3] = "bad"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> t.append(28.45)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

Le parcours d'un tuple s'effectue via les instructions :

```
for variable in tuple:
  instructions...
```

Exercice 6 (Listes de tuples).

1. Créez un programme qui demande à l'utilisateur d'entrer le prénom, le nom et le numŕo de matricule d'un étudiant, puis stocke ces informations dans un tuple à trois éléments. Affichez ce tuple à l'écran. L'intérêt d'utiliser un tuple dans ce cas réside dans le fait de ne pas pouvoir modifier les informations d'un étudiant, ce qui est plus sécurisé.
2. Modifiez votre programme de manière à ce que l'utilisateur puisse entrer autant de saisies qu'il le souhaite. Le processus de saisie s'arrête lorsque l'utilisateur entre le mot "FIN". La liste des étudiants saisis sera stockée dans une liste de tuples. Utilisez une boucle `while`.
3. Affichez de manière conviviale tous les étudiants de votre liste construite au point précédent. Utilisez un parcours de liste.

TD 2

Rappel sur les dictionnaires:

Les dictionnaires sont des couples d'éléments "clé-valeur" donnés entre accolades, les clés sont séparées de leurs valeurs correspondantes par des double points et les couples "clé- valeur" sont séparés entre eux par des virgules.

Par exemple :

```
mon_dico = {"FR" : 643801, "DE" : 357168, "GB" : 229848}
```

L'accès à la valeur correspondante à la clé `key` d'un dictionnaire `d` s'effectue via l'instruction `d[key]`. L'affectation d'une valeur `value` à une clé `key` d'un dictionnaire `d` s'effectue via l'instruction `d[key] = value`.

Tout comme les listes, les dictionnaires sont des structures de données modifiables. Cela signifie que nous pouvons procéder à des réaffectation de leurs éléments :

```
>>> mon_dico = {"FR" : 643801, "DE" : 357168, "GB" : 229848}
>>> mon_dico["DE"] = 257200
>>> mon_dico
{'DE': 257200, 'GB': 229848, 'FR': 643801}
>>> mon_dico["BE"] = 30528
>>> mon_dico
{'DE': 257200, 'GB': 229848, 'BE': 30528, 'FR': 643801}
>>> del mon_dico["FR"]
>>> mon_dico
{'DE': 257200, 'GB': 229848, 'BE': 30528}
```

TD 2

Le parcours des clés, des valeurs, ou des couples “clé-valeur” d’un dictionnaire s’effectue via les quatre types d’instructions suivantes :

```
# par defaut, le parcours d’un dico correspond parcours de ses cles
# dans le cas ci-dessous, "variable" prend les valeurs de cles du dico
for variable in dico:
  instructions...
# la syntaxe suivante est plus precise mais equivalente
for variable in dico.keys():
  instructions...
# pour le parcours des valeurs d’un dico
for variable in dico.values():
  instructions...
# pour le parcours des couples "cle-valeur" d’un dico
for variable_cle, variable_valeur in dico.items():
  instructions...
```

Il existe diverses méthodes sur les dictionnaires, nous en aborderons quelques unes.

TD 2

Exercice 7 (Dictionnaires).

Cet exercice correspond à une généralisation de l'exercice 6 dans le cas des dictionnaires.

1. Créez un dictionnaire qui contient comme clé un certain nombres de mots en français et comme valeurs la traduction de ces mots en anglais. Ce dictionnaire fait donc office de traducteur français-anglais.
2. Ajoutez à votre dictionnaire le couple clé-valeur `"cerveau" : "brain"`.
3. Recherchez si votre dictionnaire contient la traduction du mot "cerveau" et si tel est le cas affichez sa traduction anglaise. Effectuez un parcours par clés de votre dictionnaire.
4. Créez un nouveau dictionnaire dont les clés et valeurs correspondent aux valeurs et aux clés de votre dictionnaire précédent. Ainsi, votre nouveau dictionnaire fera office de traducteur anglais-français au lieu de français-anglais. Pour construire ce nouveau dictionnaire, utilisez un parcours clé-valeur de votre dictionnaire initial.
5. Recherchez si votre dictionnaire contient la traduction du mot "brain".
6. Recherchez si votre dictionnaire contient la valeur "cerveau" et si tel est le cas, afficher sa clé correspondante. Effectuez un parcours clés-valeurs de votre dictionnaire.
7. Modifiez votre dictionnaire de départ de sorte que les valeurs ne soient plus des simples mots anglais, mais des listes de mots correspondant à autant de traductions possibles de vos clés.
8. Ajoutez à votre nouveaux dictionnaire le couple clé-valeur `"chemin" : ["path", "way"]`.
9. Recherchez la deuxième traduction du mot "chemin".
10. Effacez la clé "chemin" et sa valeur correspondante de votre dictionnaire (fonction `del`).

TD 2

Exercice 8 (Dictionnaire de tuples).

Cet exercice correspond à une généralisation de l'exercice 6 dans le cas des dictionnaires.

1. Créez un programme qui demande à l'utilisateur d'entrer le prénom, le nom et le numŕo de matricule d'un étudiant, puis stocke ces informations dans un dictionnaire. Les clés du dictionnaires correspondront aux noms des étudiants et ses valeurs seront des tuples à trois éléments (prénom, nom, matricule). Affichez ce dictionnaire à l'écran. L'intérêt d'utiliser un dictionnaire dans ce cas réside dans le fait de pouvoir accéder aux informations des étudiants à leur noms.
2. Modifiez votre programme de manière à ce que l'utilisateur puisse entrer autant de saisies qu'il le souhaite. Le processus de saisie s'arrête lorsque l'utilisateur entre le mot "FIN". Le dictionnaire des étudiants saisis sera stockée dans une liste de tuples. Utilisez une boucle `while`.
3. Affichez de manière conviviale tous les étudiants de votre dictionnaire construit au point précédent. Utilisez un parcours par clés de votre dictionnaire.
4. En utilisant la syntaxe `in` ou la méthode `has_key` [1], déterminer si un étudiant du nom de "Obama" appartient à votre dictionnaire, et, si tel est le cas, renvoyez les informations cet létudiant.
5. Créez un script qui détermine si le numéro de matricule 12345678 existe dans votre diction- naire, et si tel est le cas, renvoyez les informations de létudiant correspondant. Utilisez un parcours "clé-valeur" de votre dictionnaire.
6. Question subsidiaire. Dans l'état actuel de votre programme, il est impossible d'enregistrer deux étudiants qui portent le même nom. Pourquoi cela ? Modifier votre programme du point 2 de manière à pouvoir enregistrer des étudiants portant le même nom. Pour cela, les clés de votre dico ne devront plus correspondre uniquement au nom des étudiants : mais attention, les clés ne peuvent pas être des listes, tuples, etc.