Les chaines de caractères: 1/7

- Les chaines de caractères sont des variables capables de contenir une suite de caractères. Ils sont de type <class 'str'>.
- Les caractères doivent être entourés par une quote <'> ou double qotes <">.

```
#Créer une chaine de caractères vide:
chaine = '';
#Ou:
chaine = "";

#Créer une chaine de caractères avec des caractères:
chaine = 'Bonjour';
#Ou:
chaine = "Bonjour";

#Une chaine de caractères sur plusieurs lignes:
chaine = """
Ligne 1
Ligne 2
...
"""
#C'est équivalent à:
chaine = "\nLigne 1\nLigne 2\n...\n"

# Les caractères spéciaux qu'on peut utiliser dans les chaines:
# \n : représente le retour à la ligne.
# \t : représente la tabulation.
# \r : représente le retour chariot, parfois utilisé pour les retours à la ligne.
# \caractère spéciale: si on met "\" devant un caractère spéciale comme <"> alors ce caractère
#                      sera inscrit dans le texte et non considéré comme caractère spécial.
s = "\"\\";
print(s); # Affiche <"\>
```

Les chaines de caractères: 2/7

```
#Formater une chaine pour intégrer des variables dedans:
#On place des accolades à la place de chaque variable qui sera insérée dans la chaine.
s = "La première valeur est {}, et la seconde est {}."
sFormatee = s.format("A", "B"); #La première {} sera remplacée par "A", et la seconde par "B".
print(sFormatee); #Affiche: <La première valeur est A, et la seconde est B.>

#Transformer une chaine de caractères en une liste:
#-Transformer une chaine de caractères en une liste de caractères:
s = "Hello";
l = list(s); #Renvoie la liste des caractères de la chaine <s>. Ici: l = ['H','e','l','l','o']
#-Transformer une chaine de caractères en une liste de mots:
l = s.split(sep);    #Renvoie une liste de chaînes de caractères correspondant aux différentes
                     #parties de la chaîne <s> coupée par la chaîne séparateur <sep>.
l = s.split();       #Par défaut, c'est l'espace <' '> ou la tabulation <\t> qui sont utilisés.

#Récupérer la longueur d'une chaîne de caractères:
size = len(s);  #Renvoie le nombre de caractères de la chaine <s> (la longueur de la chaîne).

#Récupérer les caractères d'une chaine:
#Les positions des caractères de la chaine sont représentées par des indices qui commencent
#par <0> et qui se terminent par <le nombre d'élèments de la chaine - 1>.
car1 = chaine[0];       #Renvoie le premier caractère de la chaine. Ici: car1 = "H"
carN = chaine[size-1];  #Renvoie le dernier caractère de la chaine. Ici: carN = "o"
#Python est un des rares langages de programmation qui permet d'accéder aux caractères d'une
#chaine à l'aide d'indices négatifs qui commencent par -1 en partant du dernier élément, en
#plus des indices positifs qui commencent par 0 en partant du premier élément.
#Ainsi, -1 est associé au dernier caractère de la chaine et -size au premier caractère de la
#chaine en utilisant les indices négatifs.
carN = chaine[-1];      #Renvoie le dernier caractère de la chaine. Ici: carN = "o"
car1 = chaine[-size];   #Renvoie le premier caractère de la chaine. Ici: car1 = "H"
```

Les chaines de caractères: 3/7

```
#Récupérer une suite de caractères d'une chaine:
#Tranches de caractères (slicing):
chaine = s[:];      # Revoie tous les caractères de la chaine.
chaine = s[i:j];    # Revoie les caractères entre l'indice i inclu et l'indice j exclu.
chaine = s[i:];     # Revoie les caractères à partir de l'indice i inclu jusqu'à la fin.
chaine = s[:j];     # Revoie les caractères à partir du début jusqu'à l'indice j exclu.
chaine = s[::k];    # Revoie tous les caractères de la chaine en avançant de k pas.
chaine = s[i:j:k];  # Revoie les caractères entre l'indice i inclu et l'indice j exclu en
                    # avançant de k pas.
chaine = s[i::k];   # Revoie les caractères à partir de l'indice i inclu jusqu'à la fin
                    # en avançant de k pas.
chaine = s[:j:k];   # Revoie les caractères à partir du début jusqu'à l'indice j exclu
                    # en avançant de k pas.
chaine = s[::-1];   # Revoie tous les caractères de la chaine en partant de la fin.

#Tester si une chaine est dans une autre chaine:
c = "o";
b = c in chaine;     #Renvoie <True> si <c> se trouve dans la chaine, et <False> sinon.
b = c not in chaine; #Renvoie <True> si <c> ne se trouve pas dans la chaine, et <False> sinon.

#Ajouter des caractères à une chaine:
#L'opéraeur d'addition <+> est un opérateur de concaténation pour les chaines car il permet de
#placer une chaine à coté d'une autre:
s1 = "abc";
s2 = "def";
s = s1 + s2;#Renvoie la concaténation de <s1> et <s2>. Ici: s = "abcdef".
s1 += s2;   #Renvoie la concaténation de <s1> et <s2> et le met dans <s1>. Ici: s1 = "abcdef".

#Modifier un caractère d'une chaine de caractères à un indice donné:
#Il n'est pas possible de modifier les caractères d'une chaine sans passer par les listes.
liste = list(chaine);
liste[i] = "a"              # Modification du caractère à l'indice i:
chaine = "".join(liste);    # Reconstruction de la chaine.
```

Les chaines de caractères: 4/7

```
#Modifier une zone de caractères d'une chaine de caractères entre deux indices donnés:
#Il n'est pas possible de modifier les caractères d'une chaine sans passer par les listes.
liste = list(chaine);
liste[i:j] = "abc"; # Modification des caractères entre l'indice i inclu et l'indice j exclu.
chaine = "".join(liste);    # Reconstruction de la chaine.

#Remplacer des caractères d'une chaine de caractères:
chaine = s.replace(chaineARemplacer, chaineDeRemplacement);
#Renvoie une chaîne en remplaçant toutes les occurrences de la chaîne passée en 1er argument
#<chaineARemplacer>, par la chaîne passée en 2ème argument <chaineDeRemplacement>.
s = "Bonjour";
chaine = s.replace("o", "R"); # Renvoie: chaine = "BRnjRur";
chaine = s.replace(chaineARemplacer, chaineDeRemplacement, nbrOccurences);
#Renvoie une chaîne en remplaçant le nombre d'occurrences à partir du début donné par le 3ème
#argument <nbrOccurences>, de la chaîne passée en 1er argument <chaineARemplacer>,
# par la chaîne passée en 2ème argument <chaineDeRemplacement>.
chaine = s.replace("o", "R", 1); # Renvoie: chaine = "BRnjour";

#Récupérer l'indice d'une suite de caractères d'une chaine de caractères:
i = s.find(chaineATrouver);  #Renvoie l'indice de la première occurrence de la chaîne cherchée
                             #<chaineATrouver>. Renvoie -1 s'il trouve pas.
i = s.index(chaineATrouver); #Renvoie l'indice de la première occurrence de la chaîne cherchée
                             #<chaineATrouver>. Lêve une exception ValueError s'il trouve pas.

#Compter le nombre d'occurrences d'une suite de caractères d'une chaine de caractères:
n = s.count(chaineATrouver);    #Renvoie le nombre d'occurrences de la chaîne cherchée
                                #<chaineATrouver> présentes dans la chaine de caractères <s>.
n = s.count(chaineATrouver [,start [,end]]);#Renvoie le nombre d'occurrences de la chaîne
                                            #cherchée <chaineATrouver> présentes dans la
                                            #chaîne de caractères <s>.
                                            #<start> et <end> sont optionnels et représentent
                                            #l'indice de début et de fin de la sous-chaine de
                                            #<s> où on effectue la recherche.
```

Les chaines de caractères: 5/7

```
#Supprimer une zone de caractères d'une chaine de caractères entre deux indices donnés:
#Il n'est pas possible de supprimer les caractères d'une chaine sans passer par les listes.
liste = list(chaine);
del(liste[i:j]); #Supprime les élèments entre l'indice i inclu à l'indice j exclu.
chaine = "".join(liste); #Reconstruction de la chaine.

#Supprimer la chaine de caractères entière:
del(chaine);

#Vider une chaine de caractères:
chaine = "";

#Fonctions utiles avec les cahines de caractères:
#L'ensemble des fonctions et méthodes de traitement sur les chaine de caractères, travaillent
#sur une copie de la chaine et ne modifie en aucun cas la chaine.
#Centrer une chaine dans un espace de longueur donnée:
s = chaine.center(n);     # Renvoie la chaîne centrée dans un espace de taille <n> caractères.
s = chaine.center(n, "-");# Renvoie la chaîne centrée dans un espace de taille <n> caractères
                          # dont le vide autour de la chaine est composé du caractère "-".

#Changer l'écriture des caractères d'une chaine de caractères:
s = chaine.lower();     # Renvoie la chaîne tout en minuscule.
s = chaine.upper();     # Renvoie la chaîne tout en majuscule.
s = chaine.capitalize();# Renvoie la chaine avec une majuscule pour la première lettre, pour
                        # le premier mot de la chaine.
s = chaine.title();     # Renvoie la chaine avec une majuscule pour la première lettre de
                        # chaque mot de la chaine.
s = chaine.strip();     # Renvoie une chaine après avoir enlevé les espaces au début et à la
                        # fin de la chaine.
```

Les chaines de caractères: 6/7

```
#Connaître la nature des caractères d'une chaine:
b = chaine.isdigit(); # Renvoie <True> si la chaine est composée de chiffres seulement
                      #(sans espace), et <False> sinon.
b = chaine.isalpha(); # Renvoie <True> si la chaine est composée de lettres seulement
                      #(sans espace), et <False> sinon.
b = chaine.isalnum(); # Renvoie <True> si la chaine est composée de caractères alphanumériques
                      # seulement (lettres ou chiffres), et <False> sinon.
b = chaine.islower(); # Renvoie <True> si la chaine est toute en minuscule, et <False> sinon.
b = chaine.isupper(); # Renvoie <True> si la chaine est toute en majuscule, et <False> sinon.
b = chaine.isspace(); # Renvoie <True> si la chaine est un espace ou un retour à la ligne, et
                      # <False> sinon.

#Trier une chaine:
def sortString(str):
    return "".join(sorted(str))
```

Les chaines de caractères: 7/7

```python
#Parcourir une chaine de caractère:
chaine = "Bonjour";
#- Parcourir une chaine avec la boucle for sans récupération d'indice:
for lettre in chaine:
    print("Lettre :", lettre);
"""
Affiche:
    Lettre : B
    Lettre : o
    Lettre : n
    Lettre : j
    Lettre : o
    Lettre : u
    Lettre : r
"""
#- Parcourir une chaine avec la boucle for avec récupération d'indice:
size = len(chaine)
for indice in range(size):
    print("Lettre à {}: {}".format(indice, chaine[indice]));
#- Parcourir une chaine avec la boucle for avec récupération d'indice avec l'enumeration:
#L'enumeration enumerate(chaine) renvoie une séquence de couples (indice,élément) de la chaine
for indice, elt in enumerate(chaine):
    print("Lettre à {}: {}".format(indice, elt));
#- Parcourir une chaine avec la boucle while:
indice = 0
while indice < len(chaine):
    print("Lettre à {}: {}".format(indice, chaine[indice]));
    indice += 1;
"""
Les 3 dernières boucles affichent:
    Lettre à 0: B
    Lettre à 1: o
    Lettre à 2: n
    Lettre à 3: j
    Lettre à 4: o
    Lettre à 5: u
    Lettre à 6: r
"""
```

Les listes: 1/7

- Les listes sont des tableaux qui peuvent contenir tout type de variable.
- Ils sont de type <class 'list'>.

```
#Créer une liste vide:
liste = list();
#Ou:
liste = [];

#Créer une liste avec des valeurs:
liste = ['Code', 2, True, 8.2];

#Créer une liste initialisée avec une même valeur:
l = [0]*10; #Crée une liste de 10 zéro. Donne [0,0,0,0,0,0,0,0,0,0]
l = [i**2 for i in range(10)]; #Crée une liste de longueur 10 initialisée par [0²,1²,2²,..,9²]
m = [[0 for i in range(3)] for j in range(3)];  #Crée une matrice de dimension 3x3 initialisée
                                                #par des 0.
m = [[0]*3]*3;  #Ne construira pas une bonne matrice avec des cases indépendantes, mais
                #produira 3 lignes pointant sur la même liste [0, 0, 0]

#Afficher une liste:
print(liste); # Affiche les élèments séparés par des virgules: ['Code', 23, True, 28.2]

#Transformer une liste en chaine de caractères:
#Les élèments de la liste doivent être des chaines de caractères.
#Si c'est pas le cas, il faut parcourir la liste et convertir tous ses éléments en chaines de
#caractères avec <str>:
l = ['A', str(2), str(True), str(8.2)];
s = "".join(l); #Construit une  chaine à partir des élèments de la liste. Ici s = "A2True8.2".
#Si on veut que les élèments de la liste soient séparés par un séparateur dans la chaine,
#alors on le précise dans la chaine sur laquelle on applique <join>:
s = "sep".join(l); #Construit une chaine à partir des élèments de la liste séparés par "sep".
s = " ".join(l); #Construit une chaine à partir des élèments de la liste séparés par un espace
                 #Ici s = "A 2 True 8.2";
s = "|".join(l); #Construit une chaine à partir des élèments de la liste séparés par <|>.
                 #Ici s = "A|2|True|8.2";
```

Les listes: 2/7

```
#Récupérer la taille d'une liste:
size = len(liste);

#Récupérer les élèments d'une liste:
#Les positions des élèments de la liste sont représentées par des indices qui commence par <0>
#et qui se terminent par <le nombre d'élèments de la liste - 1>.
elt1 = liste[0]; #Renvoie le premier élèment de la liste. Ici: elt1 = "A"
eltN = liste[size-1]; #Renvoie le dernier élèment de la liste. Ici: eltN = 8.2
#Python est un des rares langages de programmation qui permet d'accéder aux éléments d'une
#liste à l'aide d'indices négatifs qui commencent par -1 en partant du dernier élément, en
#plus des indices positifs qui commencent par 0 en partant du premier élément.
#Ainsi, -1 est associé au dernier élément de la liste et -size au premier élément de la liste
#en utilisant les indices négatifs.
eltN = liste[-1]; #Renvoie le dernier élèment de la liste. Ici: eltN = 8.2
elt1 = liste[-size]; #Renvoie le premier élèment de la liste. Ici: elt1 = "A"

#Récupérer une sous liste d'élèments d'une liste:
liste = l[:];       # Revoie tous les éléments de la liste.
liste = l[i:j];     # Revoie les éléments entre l'indice i inclu et l'indice j exclu.
liste = l[i:];      # Revoie les éléments à partir de l'indice i inclu jusqu'à la fin.
liste = l[:j];      # Revoie les éléments à partir du début jusqu'à l'indice j exclu.
liste = l[::k];     # Revoie tous les éléments de la liste en avançant de k pas.
liste = l[i:j:k];   # Revoie les éléments entre l'indice i inclu et l'indice j exclu en
                    # avançant de k pas.
liste = l[i::k];    # Revoie les éléments à partir de l'indice i inclu jusqu'à la fin en
                    # avançant de k pas.
liste = l[:j:k];    # Revoie les éléments à partir du début jusqu'à l'indice j exclu en
                    # avançant de k pas.
liste = l[::-1];    # Revoie tous les éléments de la liste en parant de la fin.
```

Les listes: 3/7

```
#Parcourir une liste:
liste = ['Code', 2, True, 8.2];
#- Parcourir une liste avec la boucle for sans récupération d'indice:
for elt in liste:
    print("Elément :", elt);
"""
Affiche:
    Elément : Code
    Elément : 2
    Elément : True
    Elément : 8.2
"""
#- Parcourir une liste avec la boucle for avec récupération d'indice:
size = len(liste)
for indice in range(size):
    print("Elément à {}: {}".format(indice, liste[indice]));
#- Parcourir une liste avec la boucle for avec récupération d'indice avec l'enumeration:
#L'enumeration enumerate(liste) renvoie une séquence de couples (indice,élément) de la liste.
for indice, elt in enumerate(liste):
    print("Elément à {}: {}".format(indice, elt));
#- Parcourir une liste avec la boucle while:
indice = 0
while indice < len(liste):
    print("Elément à {}: {}".format(indice, liste[indice]));
    indice += 1;
"""
Les 3 dernières boucles affichent:
    Elément à 0: Code
    Elément à 1: 2
    Elément à 2: True
    Elément à 3: 8.2
"""
```

Les listes: 4/7

```
#Former une liste de couples à partir de deux listes (zip):
l1 = [1, 2, 3, 3];
l2 = ['A', 'B', 'C', 'D'];
couples = list(zip(l1,l2)); # Costruit une liste de couples entre les deux listes <l1> et <l2>
print(couples);             # Affiche: [(1, 'A'), (2, 'B'), (3, 'C'), (3, 'D')]
#Parcourir la liste de couples <couples>:
for i, e in couples:
    print("Couple ({},{})".format(i, e));
"""
Affiche:
    Couple (1,A)
    Couple (2,B)
    Couple (3,C)
    Couple (3,D)
"""

#Tester si un élèment est dans une liste:
e = 2;
b = e in liste;     # Renvoie <True> si <e> se trouve dans la liste, et <False> sinon.
b = e not in liste; # Renvoie <True> si <e> ne se trouve pas dans la liste, et <False> sinon.

#Ajouter des éléments à une liste:
#-Ajouter des éléments à une liste avec l'opérateur de concaténation:
#L'opéraeur d'addition <+> est un opérateur de concaténation pour les listes car il permet de
#placer les élèments d'une liste à coté d'une autre:
l1 = [1,2];
l2 = [3,4];
l = l1 + l2;#Renvoie la concaténation de <l1> et <l2>. Ici: l = [1,2,3,4].
l1 += l2;   #Renvoie la concaténation de <l1> et <l2> et le met dans <l1>. Ici: l1 = [1,2,3,4]
#-Ajouter des éléments à une liste avec les méthodes d'extention, d'ajout, et d'insertion:
l.extend(l1);   # Place les éléments de la liste <l1> à la fin de la liste <l>.
l.append(e);    # Ajoute l'élément <e> en fin de la liste <l>.
l.insert(i, e); # Insère dans <l>, l'élément passé en 2ème paramètre <e>, à l'indice <i>.
```

Les listes: 5/7

```
#Modifier un élément d'une liste à un indice donné:
liste[i] = elt #Remplace l'élément à l'indice <i> par l'élément <elt>:

#Modifier une zone d'éléments d'une liste entre deux indices donnés:
liste[i:j] = [e1, e2, ...]; # Remplace les élèments entre l'indice <i> inclu à l'indice <j>
                            # exclu, par la liste [e1, e2, ...].

#Récupérer l'indice d'un élément d'une liste:
liste.index(elt);   # Renvoie l'indice de la première occurrence de l'élèment cherché <elt>.
                    # Lêve une exception ValueError s'il trouve pas.

#Compter le nombre d'occurrences d'un élèment dans une liste:
n = liste.count(elt);   # Renvoie le nombre d'occurrences de l'élèment <elt> présentes dans la
                        # liste <liste>.

#Supprimer un élément d'une liste égal à une valeur donnée:
liste.remove(elt); #Supprime le premier élément trouvé qui égale à <elt>.

#Supprimer un élément d'une liste à un indice donné:
del(liste[i]); #Supprime l'élèment à l'indice <i>.
#Ou:
eltSupprime = liste.pop(i); #Renvoie et supprime l'élèment à l'indice <i>.
eltSupprime = liste.pop();  #Renvoie et supprime le dernier élément de la liste.

#Supprimer une zone d'éléments d'une liste entre deux indices donnés:
del(liste[i:j]); #Supprime les élèments entre l'indice <i> inclu à l'indice <j> exclu.

#Supprimer la liste entière:
del(liste);

#Vider une liste:
liste.clear(); #Vide la liste.
#Ou:
liste = [];
```

Les listes: 6/7

```python
#Trier les éléments d'une liste
#-Trier les éléments de la liste en ordre croissant:
liste.sort();       #Trie la liste en la modifiant.
l = sorted(liste);  #Renvoie une liste triée sans modifier celle d'origine.
#-Trier les éléments de la liste en ordre décroissant:
liste.sort(reverse = True);         #Trie la liste en la modifiant.
l = sorted(liste, reverse = True);  #Renvoie une liste triée sans modifier celle d'origine.

#Inverser l'ordre des éléments d'une liste:
liste.reverse(); #Inverse l'ordre des éléments de la liste en la modifiant.
l = list(reversed(liste)); #Renvoie une liste inversée sans modifier celle d'origine.

#Enlever les doublons d'une liste:
listeSansDoublons = list(set(liste)); #<set> enlève les doublons de la liste, et <list> permet
                                      #de renvoyer une liste au lieu d'un objet.
#Appliquer <list(set(str))> sur une chaine permet de renvoyer un tableau de caractères de la
#chaine sans doublons dans le désordre.

#Calculer la somme des éléments d'une liste:
somme = sum(liste); #Renvoie la somme des éléments de la liste.

#Calculer le minimum des éléments d'une liste:
m = min(liste); #Renvoie le minimum des éléments de la liste.

#Calculer le maximum des éléments d'une liste:
M = max(liste); #Renvoie le maximum des éléments de la liste.
```

Les listes: 7/7

```
#Copier une liste:
l = liste;
#Implique que les 2 listes <l> et <liste> pointent sur la même adresse, ce qui signifie que la
#modification de l'une entrainera la modification de l'autre.
#Afin d'êviter cela, on utilise le clonage.
#-Copie profonde - Cloner une liste (deepcopy):
#Permet de créer des listes différentes avec des contenus semblables même s'il y a des sous
#listes dans les contenus qui seront copier à leurs tours aussi.
import copy
l=copy.deepcopy(liste); #Les listes <l> et <liste> sont différentes mais avec le même contenu.
#-Copie superficielle (shallow copy):
#La copie superficielle ne permet pas de clonner les sous listes de la liste qui pointerant
#toujours sur la même adresse mémoire.
l = liste.copy(); #Dans ce cas les 2 listes <l> et <liste> sont différentes mais avec le même
                  #contenu mais pas les sous-listes.
#Ou:
l = liste[:];
```

Les tuples: 1/4

- Ce sont aussi des séquences comme les listes sauf qu'on peut les créer sans pouvoir les modifier par la suite. Ils représentent une séquence de constantes.
- Ça sert à enregistrer des constantes par exemple.
- La différence par rapport aux listes, c'est que au lieu des crochets [], on utilise les parentaises ().

```
#Créer un tuple vide:
tuple0 = ();

#Créer un tuple avec une seule valeur:
tuple0 = (12,)
#Ou:
tuple0 = 12,
#Si on met juste la valeur, ça sera pas pris en tant que tuple mais en tant que type de la valeur:
tuple0 = (12); #Ici ça sera considéré comme valeur de type <int>.

#Créer un tuple avec plusieurs valeurs:
tuple0 = (12, 23, 3)

#Créer un tuple initialisé avec une même valeur:
tuple0 = (0,) * 10 #Crée un tuple de 10 zéro. Donne (0,0,0,0,0,0,0,0,0,0)

#Afficher un tuple:
print(tuple0); # Affiche: []
#S'il y a des éléments, ça affiche entre parentaises les élèments séparés par des virgules: ('Peuh !', 23, True, 28.2)

#Transformer un tuple en chaine de caractères:
chaine = "".join(tuple0); #Reconstruction de la chaine à partir des élèment du tuple.
#Si on veut que les élèments du tuple soit séparé par un espace dans la chaine alors on le précise dans la chaine
#sur laquelle on applique join:
chaine = " ".join(tuple0); #Reconstruction de la chaine à partir des élèments du tuple séparés par un espace.
chaine = "|".join(tuple0); #Reconstruction de la chaine à partir des élèments du tuple séparés par <|>.
```

Les tuples: 2/4

```
#Récupérer les élèments d'un tuple:
#On accède aux éléments du tuple, de la même manière que pour les listes:
tuple0[0] #Le premier élèment qui se trouve à l'indice 0.

#Pas de modification possible des éléments d'un tuple:

#Récupérer l'indice d'un élément d'un tuple:
tuple0.index(elt); #Renvoie l'indice de la première occurrence de l'élèment cherché <elt>.
                   #Lêve une exception ValueError s'il le trouve pas.

#Récupérer la taille d'un tuple:
size = len(tuple0)

#Tester si un élèment est dans un tuple:
e = 2;
t = ('Code', 2, True, 8.2);
b = e in t;     #Renvoie <True> si l'élèment <e> se trouve dans le tuple, et <False> sinon.
b = e not in t; #Renvoie <True> si l'élèment <e> ne se trouve pas dans le tuple <False> sinon.

#Ajouter des éléments à un tuple:
#-Ajouter des éléments à un tuple avec l'opérateur de concaténation:
#L'opéraeur d'addition <+> est un opérateur de concaténation pour les tuples car il permet de
#placer les élèments d'un tuple à coté d'un autre:
l1 = (1,2);
l2 = (3,4);
l = l1 + l2;#Renvoie la concaténation de <l1> et <l2>. Ici: l = (1,2,3,4).
l1 += l2;   #Renvoie la concaténation de <l1> et <l2> et le met dans <l1>. Ici: l1 = (1,2,3,4).
#Pas de méthode d'extention, d'insertion, ou d'ajout.

#Pas de possibilité pour supprimer des élèments d'un tuple:
```

Les tuples: 3/4

```python
#Parcourir un tuple:
#- Parcourir un tuple avec la boucle for sans récupération d'indice:
for elt in tuple0:
    print("Elément :", elt);
#- Parcourir un tuple avec la boucle for avec récupération d'indice:
size = len(tuple0)
for indice in range(size) :
    print("Elément à {}: {}".format(indice, tuple0[indice]));
#- Parcourir un tuple avec la boucle for avec récupération d'indice avec l'enumeration:
for indice, elt in enumerate(tuple0):
    print("Elément à {}: {}".format(indice, elt));
"""
Affiche:
    Elément à 0: Peuh !
    Elément à 1: 23
    Elément à 2: True
    Elément à 3: 28.2
"""
#- Parcourir un tuple avec la boucle while:
indice = 0
while indice < len(tuple0) : # 4 est le nombre d'éléments dans le tuple.
    print("Elément à {}: {}".format(indice, tuple0[indice]));
    indice += 1;

#Couper un tuple:
tup = t[:];     #Renvoie tous les éléments du tuple.
tup = t[i:j];   #Renvoie les éléments entre l'indice i inclu et l'indice j exclu.
tup = t[i:];    #Renvoie les éléments à partir de l'indice i inclu jusqu'à la fin.
tup = t[:j];    #Renvoie les éléments à partir du début jusqu'à l'indice j exclu.
tup = t[::k];   #Renvoie tous les éléments du tuple en avançant de k pas.
tup = t[i:j:k]; #Renvoie les éléments entre l'indice i inclu et l'indice j exclu en avançant de k pas.
tup = t[i::k];  #Renvoie les éléments à partir de l'indice i inclu jusqu'à la fin en avançant de k pas.
tup = t[:j:k];  #Renvoie les éléments à partir du début jusqu'à l'indice j exclu en avançant de k pas.
tup = t[::-1];  #Renvoie tous les éléments du tuple en parant de la fin.
```

Les tuples: 4/4

```
#Compter le nombre d'occurrences d'un élèment dans le tuple:
tuple0.count(elt); #Renvoie le nombre d'occurrences de l'élèment <elt> présent dans le tuple <tuple0>.

#Vider un tuple:
tuple0 = ();

#Trier les éléments d'un tuple:
#-Trier les éléments du tuple en ordre croissant:
t = sorted(tuple0);     #Renvoie un tuple triée sans modifier celui d'origine.
#-Trier les éléments du tuple en ordre décroissant:
t = sorted(tuple0, reverse = True);     #Renvoie un tuple triée sans modifier celui d'origine.

#Inverser l'ordre des éléments d'un tuple:
t = list(reversed(tuple0)); #Renvoie un tuple inversée sans modifier celui d'origine.

#Calculer la somme des éléments d'un tuple:
sum(tuple0); #Renvoie la somme des éléments du tuple.

#Calculer le minimum des éléments d'un tuple:
min(tuple0); #Renvoie le minimum des éléments du tuple.

#Calculer le maximum des éléments d'un tuple:
max(tuple0); #Renvoie le maximum des éléments du tuple.

#Enlever les doublons d'un tuple:
tupleSansDoublons = tuple(set(tuple0));
#<set> enleve les doublons du tuple, et <tuple> permet de renvoyer un tuple au lieu d'un objet.

#Copier un tuple:
tuple0 = tuple1
#Implique que les 2 tuples <tuple0> et <tuple1> sont deux tuples différents à contrario des listes.
```

TP 3

Le dessin 2D avec turtle: 1/4

- Les fonctionnalités de dessin sont rangées dans le module <turtle> qu'il faut importer pour les utiliser:

```
import turtle
```

- Ce module peut être vu comme une tortue qui porte une plume. Quand elle se déplace:
  - soit la plume est descendue, ce qui est le cas au début, et dans ce cas elle trace une ligne lors de ses déplacements;
  - soit la plume est relevée et dans ce cas, elle ne trace rien.
- Initialement la plume est orientée vers la droite.
- Changer le titre de la fenêtre:

```
turtle.title("Ma super fenêtre")
```

- Changer les dimensions de la fenêntre:

```
turtle.setup(largeur, hauteur)
```

- Changer la couleur de fond:

```
turtle.bgcolor("black")          #Un fond noir
turtle.bgcolor("#00FF00")        #Un fond vert
turtle.bgcolor((0.5, 0, 1))      #Un fond violet
```

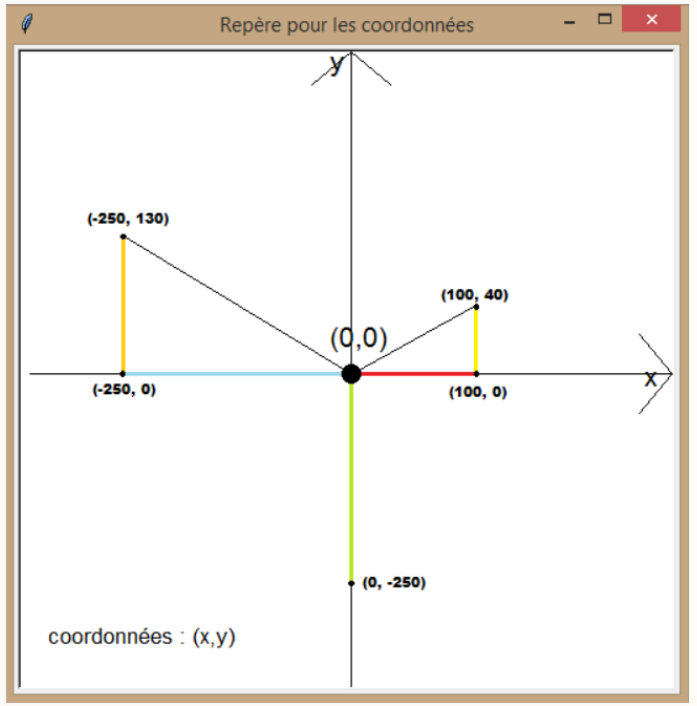
- Fermer la fenêtre via un clique sur la fenêtre:

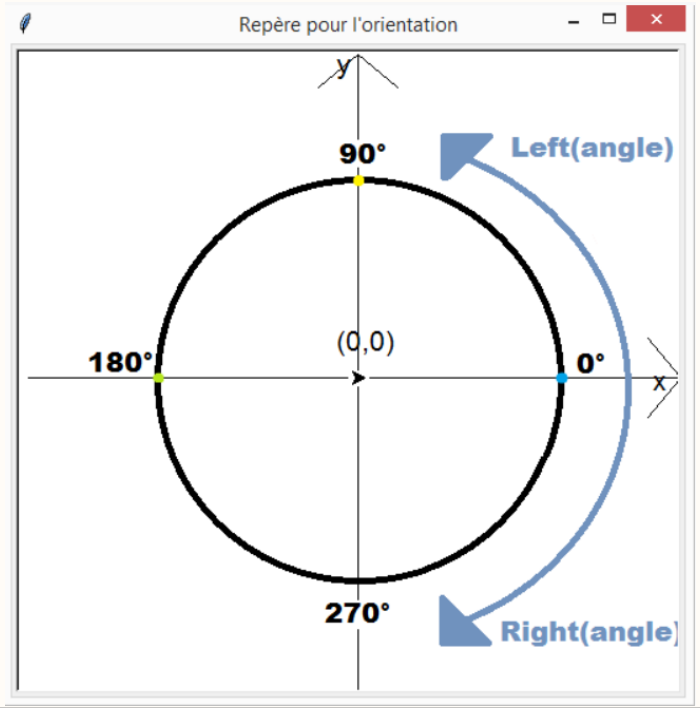
```
turtle.exitonclick();
```

Le dessin 2D avec turtle: 2/4

➢Repère, rotation et couleurs:

Couleur	Aperçu
white	
silver	
grey	
black	
red	
maroon	
lime	
green	
yellow	
olive	
blue	
navy	
fuchsia	
purple	
aqua	
teal	





Le dessin 2D avec turtle: 3/4

➢ Dessiner en 2D:

```
turtle.forward(100);          #Avancer de 100.
turtle.backward(100);         #Reculer de 100.
turtle.left(90)               #Tourner de 90° vers la gauche.
turtle.right(90);             #Tourner de 90° vers la droite.
turtle.up();                  #Relever la plume.
turtle.down();                #Remettre la plume en mode dessin.
turtle.goto(x, y);            #Aller à la position (x, y).
turtle.color("red");          #Changer la couleur de la plume.
turtle.color(c1, c2);         #Les contours seront de couleur c1 et le remplissage avec
                              #turtle.begin_fill() et turtle.end_fill() de couleur c2.
turtle.width(epaisseur);      #Choisir l'épaisseur du tracé
turtle.write(texte);          #Ecrire un texte.
turtle.circle(r);             #Trace un cercle de rayon r.
turtle.circle(r, ang);        #Trace un arc de cercle de rayon r et d'angle ang.
turtle.dot(100, 'red');       #Imprime un point rouge d'un diamètre de 100px.
turtle.dot(25)                #Imprime un point noir d'un diamètre de 25px
```

Le dessin 2D avec turtle: 4/4

➢ Remplir un dessin:

```
turtle.begin_fill();          #On commence le dessin à remplir par begin_fill().
turtle.forward(100);          #On dessine les points du dessin à remplir
turtle.right(90);
turtle.forward(100);
turtle.end_fill();            #On termine le dessin à remplir par end_fill().
```

➢ Modifier la plume:

```
turtle.hideturtle();          #Cache la flêche qui représente la plume.
turtle.shape('turtle');       #Change la forme de la plume en tortue.
                              #Par défaut la plume est une flèche.
```

➢ Nettoyer l'écran:

```
turtle.clear()                #Efface les dessins du crayon
turtle.reset()                #Fait de même et réinitialise le crayon
```

➢ Chager la vitesse de dessin:

```
turtle.speed(i);              #Change la vitesse de 1(lent) à 10(rapide).
                              #La valeur spéciale 0 est la plus rapide.
```

➢ La gestion du temps:

Les fonctionnalités de temps sont rangées dans le module <time> qu'il faut importer pour les utiliser:

```
import time
time.sleep(n);                #Met le programme en pause pendant n secondes.
```

Les fonctions : 1/4

- Les fonctions sont des bouts de code qu'on peut appeler et exécuter dans un programme avec une seule instruction, ce qui permet d'avoir un code structuré et lisible tout en évitant la répétition de certains bouts de code souvent utilisés.

```
#- Les fonctions usuelles:
#Fonctions sans retour:
print(variable);                #Affiche le contenu de la variable passée en paramètre.
help(print);                    #Affiche des informations sur la fonction ou l'objet passé en paramètre.

#Fonctions avec retour:
size = len(liste);              # Renvoie la taille de la liste passée en paramètre <liste>.
result = max(2, 5, 9, 8);       # Renvoie le max de la suite de paramètres. Donne <9>.
result = min(2, 5, 9, 8);       # Renvoie le min de la suite de paramètres. Donne <2>.
valeurEntiere = int(value);     # Convertit en integer (entier)
valeurDecimale = float(value);  # Convertit en float (nombre décimal, à virgule flottante)
b = bool(value);                # Convertit en boolean (booléen)
chaine = str(value);            # Convertit en string (chaîne de caractères)
liste = dir(x);                 # Renvoie la liste des fonctions disponibles qu'on peut utiliser
                                # avec la variable x.
```

Les fonctions : 2/4

```python
#- Définir une fonction sans paramètre et sans retour:
def nomFonction() :
    print("Bonjour!"); #Les taches à executer par la fonction.
#- Appel de la fonction sans paramètre:
nomFonction();

#- Définir une fonction avec un ou plusieurs paramètres:
def nomFonction(param1, param2) :
    print("Paramètre 1: ", param1)
    print("Paramètre 2: ", param2)
#- Appel de la fonction avec paramètres
#On passe à la fonction les valeurs que doiventt prendre ses paramètres (param1 = 2 et param2 = 3).
nomFonction(2, 3);
#On peut aussi appeler la fonction en précisant les noms de ses paramètres ce qui permet de rendre
#l'ordre des paramètres non obligatoire:
nomFonction(param1=1, param2=2) #Affiche <1> puis <2>.
#On peut aussi changer l'ordre des paramères puisqu'on précise leurs noms:
nomFonction(param2=2, param1=1) #Affiche toujours <1> puis <2>.

#- Définir une fonction avec retour:
def nomFonction(param1, param2) :
    print("Paramètre 1: ", param1)
    print("Paramètre 2: ", param2)
    return 10
#- Appel de la fonction avec retour:
result = nomFonction(2, 3); # La variable <resultat> contiendera le retour de la fonction.
```

Les fonctions : 3/4

```
#- Définir une fonction avec paramètres facultatifs:
#Les paramètres facultatifs droivent être en fin de la liste des paramètres.
def nomFonction(param1, param2 = 0) :
    print("Paramètre 1: ", param1)
    print("Paramètre 2: ", param2)
#Dans ce cas, lors de l'appel de la fonction, si on précise pas le deuxième paramètre
#alors il sera automatiquement <0>.

#- Définir une fonction avec nombre indéfini d'arguments:
def nomFonction(*arguments) :
    for element in arguments :
        print(element)
nomFonction(43, 38, "Peuh !", True)
"""
Affiche:
    43
    38
    Peuh !
    True
"""
```

Les fonctions : 4/4

```
#- Définir une fonction avec un nombre indéfini d'arguments clé valeur (dictionnaires):
def nomFonction(**arguments) :
    for cle in arguments :
        print(cle,arguments[cle])
#- Appel de la fonction avec un nombre indéfini d'arguments clé valeur:
nomFonction(arg1 = 'Peuh !', arg2 = 38) #La clé doit être sans guillemets <"">!
"""
Affiche:
    arg1 Peuh !
    arg2 38
"""

#- Définir une fonction qui renvoie un tuple (plusieurs variables):
def position():
    return 12, 23
    #Ou:
    #return (12, 23)
#- Appel de la fonction qui renvoie un tuple
x, y = position()
print("X: ",x)
print("Y: ",y)
"""
Affiche:
    X:  12
    Y:  23
"""
```

Les modules : 1/2

- **Créer un module:** C'est créer un fichier d'extension <.py>.
- **Importer un modules/des fonctions:**
- Pour les fonctions qui ne sont pas intégrées directement dans Python, on les importe en important le module où ils se trouvent avec <import> :

```
import nomModule
```

- Soit un module <nomModule.py> mit dans le même dossier du fichier <main.py> où on veut faire appel au module en question:
  - Importer une fonction <fonction0> définit dans le module <nomModule> pour l'utiliser dans le fichier <main.py>:

```
from nomModule import fonction0
```

  - Pour importer plusieurs fonctions du même module, on sépare leurs noms par des virgules:

```
from nomModule import fonction1, fonction2, fonction3,...
```

  - Importer toutes les fonctions d'un module d'un seul coup:

```
from nomModule import *
```

  - Ou importer le module tout simplement avec toutes ses fonctions:

```
import nomModule
```

    - ✓ Avec cette dernière importation, on ne pourra pas faire appel aux fonctions juste avec leurs noms, mais en précisant le nom du module suivie d'un point <nomModule.> et ensuite le nom de la fonction:

```
nomModule.fonction1(arguments)
```

Les modules : 2/2

➢ **Organiser les programmes :** Si on veut organiser les fichiers dans des dossiers, alors on précise le chemin par rapport au fichier qui fait appel au module.

➢ Mettant le module <nomModule> dans un dossier <package1>:

- Importer des fonctions définîtes dans le module <nomModule> qui se trouve dans le dossier <package1> pour les utiliser dans le fichier <main.py>:

```
from package1.nomModule import fonction
```

- Importer tout le module avec toutes ses fonctions:

```
import package1.nomModule
```

✓ Avec la dernière importation, on ne pourra pas faire appel aux fonctions juste avec leurs noms, mais en précisant le nom du dossier et du module séparés par <.>:

```
package1.nomModule.fonction(arguments)
```

- Afin d'éviter des noms longs, on redéfinit le nom du module à importer avec:

```
import package1.nomModule as md
```

✓ Dans ce cas pour appeler les fonctions du module, il suffit d'écrire:

```
md.fonction(arguments)
```

➢ **Tester le bon fonctionnement d'un module:**

On met à la fin dans le fichier du module, ce qui permet d' exécuter la partie test si on lance le module tout seul sans qu'il soit appelé depuis un autre programmes:

```
if __name__ == "__main__":          #si l'exécution s'effectue à partir de ce fichier en
                                    # tant que programme principale:
    #on peut lancer les tests des fonctions du module:
    fonction(arguments)
```

TP 4