Programmation 2

Slides modified from https://github.com/UGE-IGM/courspython

I Manipulation de sous-listes : les *slices* (tranches)

Accès à une slice

- Syntaxe : `lst[i, j]` construit une nouvelle liste contenant les éléments d'indices `i` à `j-1` de `lst`
- Attention, l'élément d'indice `i` est **inclus** mais celui d'indice `j` est **exclu** !

In [1]:
```
lst = [3, 'toto', 4.5]
      print(lst[1:len(lst)])
print(lst[0:1])
```

```
['toto', 4.5]
[3]
```

- Si `i` est omis, il prend la valeur par défaut `0`
- Si `j` est omis, il prend la valeur par défaut `len(lst)`

In [2]:
```
lst = [3, 'toto', 4.5]
      print(lst[:len(lst)-1])
print(lst[:])  # cette instruction crée une copie de lst !
```

```
[3, 'toto']
[3, 'toto', 4.5]
```

Affectation de slice

On peut aussi utiliser la syntaxe des tranches pour modifier en une seule fois une partie de la liste

```
In [3]: lst = [3, 'toto', 4.5, 5]
        lst[1:3] = ["riri", "fifi", "loulou"]
print(lst)
```

```
[3, 'riri', 'fifi', 'loulou', 5]
```

La liste `["riri", "fifi", "loulou"]` est insérée à la place de la slice `['toto', 4.5]`

Tranches avec intervalle

Il est possible de compléter la notation des tranches en spécifiant un "pas" `k` : `lst[i:j:k]` . Dans ce cas, on sélectionne uniquement les éléments d'indices `i` , `i+k` , `i+2*k` , etc. en s'arrêtant à l'indice `j` (exclu).

In [4]:
```
lst = [0, 1, 2, 3, 4, 5]
print(lst[1:6:2])
```

```
[1, 3, 5]
```

Le pas peut être négatif pour parcourir la liste depuis la fin vers le début :

In [5]:
```
lst = [0, 1, 2, 3, 4, 5]
print(lst[6:1:-1])
```

```
[5, 4, 3, 2]
```

II Indices négatifs

Il est possible d'utiliser des indices négatifs pour accéder aux éléments d'une liste. Dans ce cas, les éléments sont numérotés à partir de la droite, en commençant par l'indice `-1` et jusqu'à l'indice `-len(lst)` :

In [6]:
```
lst = [3, 'toto', 4.5]
      print(lst[-1])
print(lst[-3])
```

```
4.5
3
```

On voit que `lst[-1]` est une autre façon de désigner l'élément `lst[len(lst)-1]`, et `lst[-len(lst)]` désigne `lst[0]`. Tenter d'accéder à un indice plus petit provoque une erreur :

In [7]:
```
lst = [3, 'toto', 4.5]
      print(lst[-4])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [7], in <cell line: 2>()
      1 lst = [3, 'toto', 4.5]
----> 2 print(lst[-4])

IndexError: list index out of range
```

Ces indices négatifs sont utiles comme raccourcis d'écriture dans certains cas particuliers, par exemple pour construire la copie d'une liste privée de son dernier élément :

In [ ]:
```
lst = [3, 'toto', 4.5]
print(lst[:-1])
```

In [ ]:
```

```

Exercice 1 : Écrire une fonction qui affiche tous les éléments d'une liste (un par ligne)

In [ ]:
```
def affiche_1_elements_sur_2(lst):
    """Affiche un élément sur deux d'une liste passée en argument. L'affichage se fait un élément par lign

    :param lst: List

    >>> lst = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    >>> affiche_1_elements_sur_2(lst)
    1
    3
    5
    7
    9
    """
    for pos in range(len(lst)):
        if pos % 2 == 0:
            print(lst[pos])

lst = [3, 'toto', 4.5, None, "Cunégonde"]
affiche_1_elements_sur_2(lst)
```

Exercice 2 : Écrire une fonction recevant une liste et une valeur, et renvoyant `True` si la valeur apparaît dans la liste (`False` sinon)

In [8]:
```
def appartient(lst, val):
        """Fonction déterminant si une valeur est bien un élément d'une liste

    :param lst: List
    :param val: whatever
    :return value: boolean

    >>> appartient([], 'Hildegarde')
    False
    >>> appartient(['Hildegarde', 'Cunégonde', 'Médor'], 'Cunégonde')
    True
    >>> appartient(['Hildegarde', 'Cunégonde', 'Médor'], 'Ursule')
    False
    """
    for pos in range(len(lst)):
        if lst[pos] == val:
            return True
    return False

lst = ['Hildegarde', 'Cunégonde', 'Médor']

print(appartient(lst, 'Cunégonde'))

if appartient(lst, 'Médor'):
    print('Bon chien !')
```

```
True
Bon chien !
```

Exercice 3 : Écrire une fonction recevant une liste **non vide d'éléments comparables entre eux** et renvoyant la valeur du plus petit élément qui apparaît dans la liste

In [9]:
```
def minimum(lst):
    """Fonction déterminant la valeur minimale d'une liste NON VIDE.
    Les éléments de la liste doivent être comparables, même s'ils ne sont pas
    forcément tous du même type.

    :param lst: List
    :return value: valeur minimale de la liste

    >>> minimum([4, 6.6, 2, -7, 13.0, -6, 0])
    -7
    >>> minimum([])
    Traceback (most recent call last):
    ...
    IndexError: list index out of range
    """
    res = lst[0]  # plante si lst == []
    for pos in range(len(lst)):
        if lst[pos] < res:
            res = lst[pos]
    return res

lst_1 = [4, 6.6, 2, -7, 13.0, -6, 0]
lst_2 = ['Hildegarde', 'Cunégonde', 'Médor']
lst_3 = ['Hildegarde', 'Cunégonde', 3, 2]
print(minimum(lst_1))
print(minimum(lst_2))
print(minimum(lst_3))
```

```
-7
Cunégonde

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [9], in <cell line: 27>()
     25 print(minimum(lst_1))
     26 print(minimum(lst_2))
---> 27 print(minimum(lst_3))

Input In [9], in minimum(lst)
     16 res = lst[0]  # plante si lst == []
     17 for pos in range(len(lst)):
---> 18     if lst[pos] < res:
     19         res = lst[pos]
     20 return res

TypeError: '<' not supported between instances of 'int' and 'str'
```

Même question pour le plus grand élément

In [10]:
```
def maximum(lst):
    """Fonction déterminant la valeur maximale d'une liste NON VIDE.
    Les éléments de la liste doivent être comparables, même s'ils ne sont pas
    forcément tous du même type.

    :param lst: List
    :return value: valeur maximale de la liste

    >>> maximum([4, 6.6, 2, -7, 13.0, -6, 0])
    13.0
    >>> maximum([])
    Traceback (most recent call last):
    ...
    IndexError: list index out of range
    """
    res = lst[0]
    for pos in range(len(lst)):
        if lst[pos] > res:
            res = lst[pos]
    return res

lst_1 = lst = [4, 6.6, 2, -7, 13.0, -6, 0]
lst_2 = ['Hildegarde', 'Cunégonde', 'Médor']
lst_3 = ['Hildegarde', 'Cunégonde', 3, 2]
print(maximum(lst_1))
print(maximum(lst_2))
print(maximum(lst_3))
```

```
13.0
Médor

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [10], in <cell line: 27>()
     25 print(maximum(lst_1))
     26 print(maximum(lst_2))
---> 27 print(maximum(lst_3))

Input In [10], in maximum(lst)
     16 res = lst[0]
     17 for pos in range(len(lst)):
---> 18     if lst[pos] > res:
     19         res = lst[pos]
     20 return res

TypeError: '>' not supported between instances of 'int' and 'str'
```

Exercice 4 : Écrire une fonction `somme(lst)` qui renvoie la somme des éléments d'une liste dont tous les éléments sont **des nombres**.

In [11]:
```
def somme(lst):
    """Fonction calculant la somme des éléments d'une liste d'entiers ou de flottant.

    :param lst: Liste d'entiers
    :return value: int ou float

    >>> somme([])
    0
    >>> somme([1, 2, 3])
    6
    """
    somme = 0
    for pos in range(len(lst)):
        somme += lst[pos]
    return somme

print(somme([1, 2, 3]))
```

```
6
```

Exercice 5 : Écrire une fonction `renverser` renversant l'ordre des éléments d'une liste `lst` en la modifiant.

In [12]:
```
def renverser(lst):
    """Fonction modifiant la liste passée en paramètre en lui renversant l'ordre de ses éléments

    :param lst: List

    >>> lst = [3, 'toto', 4.5, False, None, 4.5]
    >>> renverser(lst)
    >>> lst
    [4.5, None, False, 4.5, 'toto', 3]
    """
    i = 0
    j = len(lst) - 1
    while i < j:
        lst[i], lst[j] = lst[j], lst[i]
        i += 1
        j -= 1

lst = [3, 'toto', 4.5, False, None, 4.5]
print(lst)
renverser(lst)
print(lst)
```

```
[3, 'toto', 4.5, False, None, 4.5]
[4.5, None, False, 4.5, 'toto', 3]
```

Exercice 6 : Écrire une fonction `etend_liste(une_liste, autre_liste)` recevant deux listes et ajoutant tous les éléments de la seconde à la fin de la première

In [13]:
```
def etend_liste(une_liste, autre_liste):
    """Fonction ajoutant tous les éléments de la seconde liste passée en paramètres à la fin de la premiè

    :param une_liste: List
    :param autre_liste: List

    >>> lst_un = [3, 'toto', 4.5]
    >>> lst_deux = [False, None]
    >>> etend_liste(lst_un, lst_deux)
    >>> lst_un
    [3, 'toto', 4.5, False, None]
    >>> lst_deux
    [False, None]
    """
    for i in range(len(autre_liste)):
        une_liste.append(autre_liste[i])

lst_un = [3, 'toto', 4.5]
lst_deux = [False, None]

print("lst_un =", lst_un)
print("lst_deux =", lst_deux)
etend_liste(lst_un, lst_deux)
print("lst_un =", lst_un)
print("lst_deux =", lst_deux)
```

```
lst_un = [3, 'toto', 4.5]
lst_deux = [False, None]
lst_un = [3, 'toto', 4.5, False, None]
lst_deux = [False, None]
```

Exercice 7 : Écrire une fonction renvoyant le plus petit indice où apparaît un élément `val` dans une liste `lst` (on renverra `None` si `val` n'apparaît pas dans la liste)

In [14]:
```
def chercher(lst, val):
    """ Fonction renvoyant le plus petit indice où apparaît l'élément val dans la liste lst

    :param lst: List
    :param val: whatever
    :return value: int ou None

    >>> chercher([3, 'toto', 4.5, False, None, 4.5], 4.5)
    2
    >>> chercher([3, 'toto', 4.5, False, None, 4.5], 'AP1')
    """
    for i in range(len(lst)):
        if lst[i] == val:
            return i
    return None

lst = [3, 'toto', 4.5, False, None, 4.5]
indice = chercher(lst, 4.5)
print(indice)
indiceNone = chercher(lst, 'AP1')
print(indiceNone)
```

```
2
None
```

Exercice 8 : Écrire une fonction renvoyant le nombre de fois où apparaît un élément `x` dans une liste `lst`

In [15]:
```
def compter(lst, x):
        """ Fonction renvoyant le nombre de fois où l'élément x dans la liste lst

    :param lst: List
    :param val: whatever
    :return value: int

    >>> compter([3, 'toto', 4.5, False, None, 4.5], 4.5)
    2
    >>> compter([3, 'toto', 4.5, False, None, 4.5], 'AP1')
    0
    """
    cpt = 0
    i = 0
    for i in range(len(lst)):
        if lst[i] == x:
            cpt += 1
    return cpt

lst = [3, 'toto', 4.5, False, None, 4.5]
print(compter(lst, 4.5))
```

```
2
```

Exercice 9 : Écrire une fonction retirant la première occurrence d'un élément `x` dans une liste `lst` (on ne fera rien si la liste ne contient pas `x`)

In [16]:
```
def chercher(lst, val):
    """ Fonction renvoyant le plus petit indice où apparaît l'élément val dans la liste lst

    :param lst: List
    :param val: whatever
    :return value: int ou None

    >>> chercher([3, 'toto', 4.5, False, None, 4.5], 4.5)
    2
    >>> chercher([3, 'toto', 4.5, False, None, 4.5], 'AP1')
    """
    for i in range(len(lst)):
        if lst[i] == val:
            return i
    return None

def retirer(lst, x):
    """ Fonction retirant la première occurence de l'élément x dans la liste lst

    :param lst: List
    :param x: whatever
    :return value: int

    >>> lst = [3, 'toto', 4.5, False, None, 4.5]
    >>> retirer(lst, 4.5)
    >>> lst
    [3, 'toto', False, None, 4.5]
    >>> retirer(lst, 4.5)
    >>> lst
    [3, 'toto', False, None]
    >>> retirer(lst, 4.5)
    >>> lst
    [3, 'toto', False, None]
    """
    i = chercher(lst, x)
    if i != None:
        while i < len(lst)-1:
            lst[i] = lst[i+1]
            i += 1
        lst.pop()

def retirer2(lst, x): # version utilisant pop(i)
    """ Fonction retirant la première occurence de l'élément x dans la liste lst

    :param lst: List
    :param x: whatever
    :return value: int

    >>> lst = [3, 'toto', 4.5, False, None, 4.5]
    >>> retirer(lst, 4.5)
    >>> lst
    [3, 'toto', False, None, 4.5]
    >>> retirer(lst, 4.5)
    >>> lst
    [3, 'toto', False, None]
    >>> retirer(lst, 4.5)
    >>> lst
    [3, 'toto', False, None]
    """
    i = chercher(lst, x)
    if i != None:
        lst.pop(i)

lst = [3, 'toto', 4.5, False, None, 4.5]
print("lst =", lst)
retirer(lst, 4.5)
print("lst =", lst)
print("---------------")
lst2 = [3, 'toto', 4.5, False, None, 4.5]
print("lst2 =", lst2)
retirer2(lst2, 4.5)
print("lst2 =", lst2)
retirer2(lst2, 5) # On essaie de retirer un élément qui n'est pas dans la liste
print("lst2 =", lst2)
```

```
lst = [3, 'toto', 4.5, False, None, 4.5]
lst = [3, 'toto', False, None, 4.5]
---------------
lst2 = [3, 'toto', 4.5, False, None, 4.5]
lst2 = [3, 'toto', False, None, 4.5]
lst2 = [3, 'toto', False, None, 4.5]
```

Exercice 10 : Écrire une fonction qui insère un élément `x` à la position `i` de la liste `lst` (si `i` est trop grand, la fonction insère `x` à la fin de `lst` ; si `i` est trop petit, elle insère `x` au début de `lst`)

In [17]:
```
def ajouter(lst, i, x):
    """ Fonction qui insère la valeur `x` à la position `i` de la liste `lst`

    :param lst: List
    :param i: int
    :param x: whatever
    :return value: int

    >>> lst = [3, 'toto', 4.5, False, None, 4.5]
    >>> ajouter(lst, 3, 79)
    >>> lst
    [3, 'toto', 4.5, 79, False, None, 4.5]
    >>> ajouter(lst, -6, 'petit')
    >>> lst
    ['petit', 3, 'toto', 4.5, 79, False, None, 4.5]
    >>> ajouter(lst, 16, 'grand')
    >>> lst
    ['petit', 3, 'toto', 4.5, 79, False, None, 4.5, 'grand']
    """
    # 1. Détermine une valeur correcte pour i
    # Si i < 0: i devient 0
    # Si i > len(lst): i devient len(lst)
    i = max(0, min(i, len(lst)))
    # 2. On ajoute une case vide à la fin
    lst.append(None)
    # 3. Décalage des cases sur la droite à partir de i
    j = len(lst) - 1
    while j > i:
        lst[j] = lst[j-1]
        j -= 1
    # 4. On insère x à l'indice i
    lst[i] = x

lst = [3, 'toto', 4.5, False, None, 4.5]
print(lst)
ajouter(lst, 3, 79) # indice ok
print(lst)
ajouter(lst, -6, 'petit') # indice trop petit
print(lst)
ajouter(lst, 16, 'grand') # indice trop grand
print(lst)
```

```
[3, 'toto', 4.5, False, None, 4.5]
[3, 'toto', 4.5, 79, False, None, 4.5]
['petit', 3, 'toto', 4.5, 79, False, None, 4.5]
['petit', 3, 'toto', 4.5, 79, False, None, 4.5, 'grand']
```

In [ ]: