Initiation à la programmation

SL25Y031

Cours/TD – Chapitre 6

Université Paris Cité

Objectifs :

- Savoir détecter les erreurs dans du code et le débugguer.
- Savoir implémenter un certain nombre d'opérations classiques sur les listes.

1 Implémentation d'opérations classiques sur les listes : debugguage

Exercice 1 (Présence dans une liste, $\star$)
On cherche à implémenter une fonction `is_present` *prenant en argument une valeur quelconque* `x` *et une liste* `l`*, et renvoyant* `True` *si l'un des éléments de* `l` *vaut* `x` *et* `False` *sinon. Pour chacune des propositions suivantes,*

- *expliquer pourquoi celle-ci est fausse (il peut y avoir plusieurs problèmes) ;*
- *donner un exemple de valeurs de* `x` *et* `l` *pour lesquelles la fonction ne se comporte pas comme souhaité (indiquer le résultat erroné) ;*
- *effectuer une correction minimale de cette proposition.*

1.

```
# x: any; l: list
def is_present(x, l):
  for y in l:
    if(y == x):
      return True
    else:
      return False

  return False
```

2.

```
# x: any; l: list
def is_present(x, l):
  presence = True
  for y in l:
    if(y == x):
      presence = True
    else:
      presence = False

  return presence
```

3.
```python
# x: any; l: list
def is_present(x, l):
  presence = False
  i = 0
  while((i < len(l)) and presence):
    if(l[i] == x):
      presence = True

  return presence
```

4.
```python
# x: any; l: list
def is_present(x, l):
  presence = False
  i = 0
  while((i < len(l)) and (not presence)):
    presence = (presence and (l[i] == x))

  return presence
```

□

Exercice 2 (Première position dans une liste, ⋆)

On cherche à implémenter une fonction `first_pos` *prenant en argument une valeur quelconque* `x` *et une liste* `l`*, et renvoyant le premier indice dans* `l` *d'un élément valant* `x` *si un tel indice existe et* `None` *sinon. Pour chacune des propositions suivantes,*

- *expliquer pourquoi celle-ci est fausse (il peut y avoir plusieurs problèmes) ;*
- *donner un exemple de valeurs de* `x` *et* `l` *pour lesquelles la fonction ne se comporte pas comme souhaité (indiquer le résultat eronné) ;*
- *effectuer une correction minimale de cette proposition.*

1.
```python
# x: any; l: list
def first_pos(x, l):
  for i in range(len(l)):
    if(x == l[i]):
      return i
    else:
      return None

  return None
```

2.
```python
# x: any; l: list
def first_pos(x, l):
  p = 0
  for i in range(len(l)):
    if(x == l[i]):
      p = i
    else:
      p = None

  return p
```

3.
```python
# x: any; l: list
def first_pos(x, l):
```

```
  p = None
  i = 0
  while(i < len(l)):
    if(x == l[i]):
      p = i
      i += 1

  return p
```

4.

```
# x: any; l: list
def first_pos(x, l):
  p = None
  i = 0
  while((i < len(l)) and (p != None)):
    if(x == l[i]):
      p = i
    i += 1

  return p
```

□

Exercice 3 (Compter dans une liste, ⋆)

On cherche à implémenter une fonction `count` *prenant en argument une valeur quelconque* `x` *et une liste* `l`*, et renvoyant le nombre d'occurrences de* `x` *dans* `l`*. Pour chacune des propositions suivantes,*

- *expliquer pourquoi celle-ci est fausse (il peut y avoir plusieurs problèmes) ;*
- *donner un exemple de valeurs de* `x` *et* `l` *pour lesquelles la fonction ne se comporte pas comme souhaité (indiquer le résultat eronné) ;*
- *effectuer une correction minimale de cette proposition.*

1.

```
# x: any; l: list
def count(x, l):
  k = 0
  for y in l:
    if(x == y):
      k += 1
      return k

  return k
```

2.

```
# x: any; l: list
def count(x, l):
  k = len(l)
  for y in l:
    if(x == y):
      k += 1
    else :
      k -= 1

  return k
```

□

Exercice 4 (Construire la liste des positions, ⋆)
On cherche à implémenter une fonction `pos` *prenant en argument une valeur quelconque* `x` *et une liste* `l`*, et renvoyant la liste des indices dans* `l` *où* `x` *apparaît.*

Contrat :

`x, l=2, [3, 4, 5, 8]`	→	*retourne :* `[]`
`x, l=3, [3, 3, 4, 3, 4, 5, 6, 4, 3]`	→	*retourne :* `[0, 1, 3, 8]`
`x, l=3, [3, 3, 4, 3, 4, 5, 6, 4, 3]`	→	*retourne :* `[2, 4, 7]`

Pour chacune des propositions suivantes,

- *expliquer pourquoi celle-ci est fausse (il peut y avoir plusieurs problèmes) ;*
- *donner un exemple de valeurs de* `x` *et* `l` *pour lesquelles la fonction ne se comporte pas comme souhaité (indiquer le résultat erroné) ;*
- *effectuer une correction minimale de cette proposition.*

1.
```python
# x: any; l: list
def pos(x, l):
  res = []
  for i in range(len(l)):
    if(l[i] == x):
      res = [i]

  return res
```

2.
```python
# x: any; l: list
def pos(x, l):
  return [i for i in range(l) if(x == i)]
```

3.
```python
# x: any; l: list
def pos(x, l):
  return [l[i] for (i, y) in enumerate(l) if(x == y)]
```

□

Exercice 5 (Échanger deux éléments dans une liste, ⋆)
On cherche à implémenter une fonction `swap` *prenant en argument deux entiers* `i` *et* `j` *et une liste* `l`*, et échangeant les éléments aux indices* `i` *et* `j` *dans* `l`*. Considérer la proposition suivante.*

- *Expliquer pourquoi celle-ci est fausse (il peut y avoir plusieurs problèmes).*
- *Donner un exemple de valeurs de* `i`*,* `j` *et* `l` *pour lesquelles la fonction ne se comporte pas comme souhaité (indiquer le résultat éronné).*
- *Effectuer une correction minimale de cette proposition.*

```python
# i, j: int; l: list
def swap(i, j, l):
  l[i] = l[j]
  l[j] = l[i]
```

□

Exercice 6 (Inverser une liste, ⋆⋆)
On cherche à implémenter une fonction `my_reverse` *prenant en argument une liste* `l` *et inversant* `l` *comme le ferait* `l.reverse`*. Par exemple, si* `l=['a', 'b', 'c', 'd']`*, juste après l'exécution de* `my_reverse(l)`*,* `l` *vaudra* `['d', 'c', 'b', 'a']`*. Considérer la proposition suivante.*

- *Expliquer pourquoi celle-ci est fausse (il peut y avoir plusieurs problèmes).*
- *Donner un exemple de valeur de* `l` *pour laquelle la fonction ne se comporte pas comme souhaité (indiquer le résultat éronné).*

— *Effectuer une correction minimale de cette proposition.*

```
# l: list
def reverse(l):
  for i in range(len(l)):
    l[i] = l[len(li) - 1 - i]
```

□

Exercice 7 (Rotation avant dans une liste, ⋆⋆⋆)
On cherche à implémenter une fonction `rotate_r` *prenant en argument une liste* `l` *et décalant tous les éléments de* `l` *vers l'indice supérieur (ou au début de la liste pour le dernier élément). Par exemple, si* `l=['a', 'b', 'c', 'd']`*, juste après l'exécution de* `rotate_r(l)`*,* `l` *vaudra* `['d', 'a', 'b', 'c']`*. Considérer la proposition suivante.*

— *Expliquer pourquoi celle-ci est fausse (il peut y avoir plusieurs problèmes).*
— *Donner un exemple de valeur de* `l` *pour laquelle la fonction ne se comporte pas comme souhaité (indiquer le résultat éronné).*
— *Effectuer une correction minimale de cette proposition.*

```
# l: list
def rotate_r(l):
  if(len(l) < 1): return # Equivalent à "return None".

  tmp = l[-1]
  for i in range(len(l), 0, -1): l[i] = l[i-1]
  l[0] = tmp
```

□

Exercice 8 (Rotation arrière dans une liste, ⋆⋆⋆)
On cherche à implémenter une fonction `rotate_l` *prenant en argument une liste* `l` *et décalant tous les éléments de* `l` *vers l'indice inférieur (ou à la fin de la liste pour le premier élément). Par exemple, si* `l=['a', 'b', 'c', 'd']`*, juste après l'exécution de* `rotate_l(l)`*,* `l` *vaudra* `['b', 'c', 'd', 'a']`*. Considérer la proposition suivante.*

— *Expliquer pourquoi celle-ci est fausse (il peut y avoir plusieurs problèmes).*
— *Donner un exemple de valeur de* `l` *pour laquelle la fonction ne se comporte pas comme souhaité (indiquer le résultat éronné).*
— *Effectuer une correction minimale de cette proposition.*

```
# l: list
def rotate_l(l):
  if(len(l) < 1): return # Equivalent à "return None".

  tmp = l[0]
  for i, x in enumerate(l): l[i-1] = x
  l[-1] = tmp
```

□

2 À faire chez soi

Exercice 9 (Addition, ⋆⋆⋆)
Dans cet exercice, les nombres entiers sont représentés par des listes de chiffres. Plus précisément, un entier est représenté par une liste de valeurs de type `int` *toutes entre 0 et 9 (inclus) et se lisant de la droite vers la gauche, c.-à-d. que le premier élément de la liste est le nombre d'unités, le second élément est le nombre de dizaines, le troisième élément est le nombre de centaines, etc. Par exemple,* 843 *est représenté par* `[3, 4, 8]` *et* 29 *par* `[9, 2]`.

Écrire une fonction `add` *prenant en argument deux listes* `l1` *et* `l2`*, et retournant la liste représentant la somme des deux nombres représentés par* `l1` *et* `l2`.

Par exemple, si les entrées sont les listes `l1=[3, 4, 8]` *et* `l2 = [9, 2]`*, la valeur à retourner est la liste* `[2, 7, 8]`*, qui représente le nombre* 872*. Le premier élément,* `2`*, s'obtient en additionnant* `3` *et* `9` *et en notant la retenue. Le second élément,* `7`*, s'obtient en additionnant* `4`*,* `2` *et la retenue précédente. Le troisième élément,* `8`*, s'obtient directement à partir du* `8`*, le nombre de centaines du premier nombre. Le calcul s'arrête là car il n'y a plus de chiffre à additionner ni de retenue.*

Contrat :

`l1, l2 = [5, 3, 4, 2], [6, 3, 4]`	→	*retour :* `[1, 7, 8, 2]`
`l1, l2 = [2, 1], [3, 1]`	→	*retour :* `[5, 2]`
`l1, l2 = [1], [9, 9]`	→	*retour :* `[0, 0, 1]`

□