TD semaine 4 - Modélisation et Robotique
Listes et Variables globales

1 Révisions du cours

Question A. Valeurs de retour ou print ? Selon la définition choisie parmi :

```
def double(x):
    return 2 * x

def double(x):
    print(2 * x)

def double(x):
    print(2 * x)
    return 42

def double(x):
    return 42
    print(2 * x)
```

dire ce qui sera affiché par l'instruction `print("double de trois " , double(3))`.

Correction

```
1: double de trois 6

2: 6
2: double de trois None

3: 6
3: double de trois 42

4: double de trois 42
```

Question B. Fonctions retournant une valeur. Quel est le rôle de `a` dans la définition de la fonction suivante ? Donner des exemples d'utilisation de cette fonction.

```
def avancer_toute():
    a = 0
    while not obstacle():
        av()
        a += 1
    return a
```

Correction

C'est une variable qui sert à compter le nombre de fois que l'on passe dans la boucle while. elle est ensuite utilisée comme valeur de retour. Cette fonction peut être utilisée pour mesurer la distance parcourue jusqu'à un obstacle. Par exemple :

```
distance = avancer_toute()
print("J'ai parcouru ",distance," cases")
```

2 Traitements sur les Listes

Question C. Qu'affiche le programme suivant ?

```
panier = ["banane", "mangue",1, "papaye"]
for i, e in enumerate(panier):
    print("Je mange", i+1, e)
```

Correction

```
Je mange 1 banane
Je mange 2 mangue
Je mange 3 1
Je mange 4 papaye
...
```

Question D. Prénoms Définir une fonction qui prend une liste de prénoms (par exemple ['jean','pierre','marie']) et renvoie la chaine de caractere 'jean, pierre et marie'

Correction

if len(l)==0 : return '' elif len(l)==1 : return l[0] else : s='' for e in l[:-2] : s+=e+', ' return s+l[-2]+' et '+l[-1]

```
def salutations(L):
    s = ''
    if L == []:
        return ''
    if len(L)==1:
        return L[0]
    for i in L[:-2]:
        s = s+i+', '
    return s+l[-2]+' et '+L[-1]
```

Question E. Fonction mystère On définit la fonction mystère ci-dessous :

```
def mystere(liste):
    liste2=[]
    for i, e in enumerate(liste):
        if i > e:
            liste2 += [i]
    return liste2
```

Que font les instructions suivantes :

```
print(mystere([1, 0, 516, 4, 2, 0]))
print(mystere(["42", "3", "1", "1"]))
```

Correction

l'appel à print(mystere([1, 0, 516, 4, 2, 0])) renvoie [1,4,5]

l'appel à print(mystere(["42", "3", "1", "1"])) buggue, car il teste `if 0 > "42"`

Question F. Traiter des listes de nombres Définir les fonctions suivantes :

- Une fonction qui renvoie la différence entre le maximum et le minimum dans une liste de nombres
- une fonction `filtrePos(L)` qui ne renvoie que les nombres non négatifs.
  Par ex., `filtrePos([1,-10,0,10,4])` renvoie `[1,0,10,4]`
- Une fonction `somsucc(L)` qui renvoie la somme des pairs d'éléments successifs.
  Par ex., `somsucc([1,10,4,3,3])` renvoie `[11,14,7,6]`
- Une fonction `somsucck(L,k)` qui renvoie la somme des `k` éléments successifs.
  Par ex., `somsucc([1,10,4,3,3],3)` renvoie `[15,17,10]`.
  Vous pouvez utiliser la fonction `sum` de Python qui calcule la somme des éléments d'une liste
- Une fonction `argsomsucc(L)` qui renvoie l'index de la paire d'éléments dont la somme est la plus grande.
  Par ex., `argsomsucc([1,10,4,3,3])` renvoie 1 car l'élément 1 de la liste `[11,14,7,6]` est le plus grand.

Correction :

```
def maxmin(L):
    ma=L[0]
    mi=L[0]
    for e in L:
        if e > ma:
```

```
            ma = e
        if e < mi:
            mi = e
    return ma-mi

def somsucc(L):
    R = []
    for i in range(len(L)-1):
        R += [L[i]+L[i+1]]
    return R

def somsucck_v1(L,k): # 1iere version
    R = []
    for i in range(len(L)-k+1):
        s = 0
        for j in range(k):
            s = s + L[i+j]
        R = R + [s]
    return R

def somsucck_v2(L,k): # 2eme version
    R = []
    for i in range(len(L)-k+1):
       R = R + [sum(L[i:i+k])]
    return R

def argsomsucc(L):
    i_max = 0
    m = L[0]+L[1]
    for i in range(len(L)-1):
        if L[i]+L[i+1] > m:
            m = L[i]+L[i+1]
            i_max = i
    return i_max
```

3 Deplacements et Orientation

Question G. Aller-retour. Définir une fonction `aller_retour()` qui fait que le personnage avance dans la direction actuelle jusqu'à un premier obstacle puis revient à la position initiale.

Indice : vous pouvez appeler la fonction `avancer_toute` dans votre code.

Correction

```
def aller_retour():
    distance = avancer_toute() # aller
    tg(); tg() # demi tour
    for i in range(distance) # retour
        av()
    tg(); tg() # demi tour (non précisé dans le suje
```

Question H. Orientation absolue. Au début du programme le personnage est orienté vers l'est. Le but de cet exercice est de définir trois fonctions : `tgo()` qui tourne à gauche, `tdo()` qui tourne à droite et `quelle_orientation()` qui retourne l'orientation actuelle sous la forme d'une chaîne de caractère parmi '`sud`', '`nord`', '`est`', '`ouest`'.

Pour cela nous avons commencé à écrire le code suivant qu'il ne reste plus qu'à compléter. Pour rappel, en Python, `x%y` calcule le reste de la division de `x` par `y`.

```
orientation = 0

def tgo():
    ...
    orientation = orientation + 1
    tg()

def quelle_orientation():
    if orientation % 4 == 1:
        return 'nord'
    ...
```

— Que doit-on écrire à la première ligne de `tgo()` et pourquoi ?

— Compléter le programme.

Correction

Le symbole `%` signifie ici l'opération mathématique modulo qui calcule le reste de la division d'un entier par un autre.

Comme `tgo()` doit modifier la valeur de la variable orientation, définie globalement, il est nécessaire de déclarer que c'est bien à la variable globale `orientation` que `tgo()` fait référence dans son code et non à une nouvelle variable locale de même nom. On écrit donc

```
def tgo()
    global orientation
    orientation += 1
    tg()
```

Il n'est par contre pas nécessaire de signaler que la variable `orientation` à laquelle fait référence le code de `quelle_orientation()` est bien la variable globale, car cette fonction ne modifie pas la valeur de cette variable globale elle ne fait que la lire.

Le code complet doit être le suivant

```
orientation = 0 # orientation initiale vers l'est

def tgo():
    global orientation
    orientation += 1
    tg()

def tdo():
    global orientation
    orientation += -1
    td()

def quelle_orientation():
    if orientation % 4 == 0:
        return 'est'
    if orientation % 4 == 1:
        return 'nord'
    if orientation % 4 == 2:
        return 'ouest'
    if orientation % 4 == 3:
        return 'sud'
```

Question I. Variables globales ou passage de valeur ? (exercice optionnel) Grâce à votre code, il est possible d'exécuter le programme suivant :

```
def dto():
    tgo(); tgo()

def trajet():
    av(); tgo(); av(); av(); tgo()
    av(); av(); dto()

trajet(); print(quelle_orientation())
```

Proposer une solution pour écrire le même programme sans utiliser de variable globale. Pourquoi est-ce mal adapté ici ?

Correction

Il faut que chaque fonction qui utilise et l'orientation la prenne comme paramètre en entrée et la retourne comme valeur de sortie. Pour cela on doit modifier les fonctions `tgo()` etc.

```
def tgo(o):
    tg()
    return o+1

def tdo(o):
    td()
    return o-1

def quelle_orientation(o):
    if o % 4 == 0:
        return 'est'
    if o % 4 == 1:
        return 'nord'
    if o % 4 == 2:
        return 'ouest'
```

```
    if o % 4 == 3:
        return 'sud'
```

On peut alors réécrire le code précédent comme ceci :

```
def dto(o):
    o = tgo(tgo(o))
    return o

def trajet(o):
    av()
    o = tgo(o)
    av(); av()
    o = tgo(o)
    av(); av()
    o = dto(o)
    return o

orientation = 0
orientation = trajet(orientation)
print(quelle_orientation(orientation))
```

C'est moins pratique, car il faut explicitement passer la valeur de l'orientation à toutes les fonction suceptibles de la modifier, ce qui allonge le code.

Question J. Compteur kilométrique (exercice optionnel) Sur le même modèle que l'orientation absolue, proposer une fonction `avk()` qui remplace `av()` et une fonction `kilometrage()` qui retourne le nombre de cases parcourues par le personnage depuis le début du programme.

Correction

```
km = 0

def avk():
    global km
    km += 1
    av()

def kilometrage():
    return km
```

Question K. Trouver le mur. Vous pouvez utiliser les fonctions définies jusqu'à maintenant pour les exercices qui suivent.

Votre personnage est dans une cour rectangulaire, le long d'un des murs, orienté vers l'est. L'objectif est de trouver de quel côté est le (ou les) mur(s).

— Définir une fonction `dirObstacles()` qui retourne la liste de toutes les directions dans lesquelles se trouve un obstacle dans la case adjacente au personnage. Votre fonction retournera son résultat sous la forme d'une liste de points cardinaux (nord,sud...). Cette fonction est utile pour cette question et elle le sera encore plus, plus tard.
Indice : utilisez les fonction `tgo,tdo` et `quelle_orientation`

— Définir une fonction `oriente(s)` qui prend en paramètre une direction et qui oriente le personnage vers cette direction (par exemple `oriente('nord')`).

— Définir une fonction `trouverMur()` qui oriente le personnage en direction d'un mur adjacent (s'il y en a un).

— Définir une fonction `regarde()` qui utilise la fonction `dirObstacles` pour afficher 'couloir' si le personnage est dans un couloir (c'est à dire entouré de 2 murs) et 'cul-de-sac' (c'est à dire entouré de 3 murs) s'il est dans un cul-de-sac, ou rien sinon.

Correction

```
def dirObstacles():
    resultat = []
    for i in range(4):
        tgo()
        if obstacle():
```

```
            resultat += [quelle_orientation()]
    return resultat

def oriente(s):
    for i in range(4):
        if quelle_orientation() == s:
            break
        else:
            tdo()

def trouverMur():
    oriente(dirObstacles[0])

def regarde():
    n = len(dirObstacles())
    if n == 3:
        print('cul-de-sac')
    if n == 2:
        print('couloir')
```

4 Fruits et Légumes

Question L. Ingrédients. Définir une fonction `types_elements(panier)` qui prend en entrée un liste de légumes (par exemple, la liste `['chou', 'chou', 'salade']`) et retourne une liste où chaque élément du panier apparaît une seule fois.

Correction

```
def types_elements(panier):
    uniks = []
    for element in panier:
        if element not in uniks:
            uniks = uniks + [element]
    return uniks

print(types_elements(['pomme', 'chou', 'laitue', 'chou', 'pomme', 'poire']))
```

Question M. Pas de fruits. Le panier peut contenir toute sorte de légumes mais aussi quelques fruits parmi une liste `types_de_fruits`. Écrire une fonction `legumes` qui prend en entrée le panier et la liste des types de fruits et retourne les éléments du panier qui ne sont pas des fruits.

Correction

```
def legumes(panier, types_de_fruits):
    resultat = []
    for element in panier:
        if element not in types_de_fruits:
            resultat += [element]
    return resultat

panier = ['pomme', 'chou', 'laitue', 'chou', 'pomme'
types_de_fruits = ['pomme', 'poire', 'coing']
print(legumes(panier, types_de_fruits))
```

Question N. Pas de légumes. Écrire une fonction `fruits` qui, à partir des mêmes paramètres, `panier` et `types_de_fruits`, retourne tous les fruits du panier.

Correction

```
def fruits(panier, types_de_fruits):
    resultat = []
    for element in panier:
        if element in types_de_fruits:
            resultat += [element]
    return resultat

panier = ['pomme', 'chou', 'laitue', 'chou', 'pomme'
types_de_fruits = ['pomme', 'poire', 'coing']
print(fruits(panier, types_de_fruits))
```

Question O. Les légumes puis les fruits. Toujours à partir des mêmes paramètres, définir une fonction `ranger` qui retournera une liste contenant d'abord tous les légumes du panier puis tous les fruits. *Indice :* utilisez les fonctions précédentes

Correction

```
def legumes(panier, types_de_fruits):
    resultat = []
    for element in panier:
        if element not in types_de_fruits:
            resultat += [element]
    return resultat

def fruits(panier, types_de_fruits):
    resultat = []
    for element in panier:
        if element in types_de_fruits:
            resultat += [element]
    return resultat

def ranger(panier, tf):
    a = legumes(panier, tf)
    b = fruits(panier, tf)
    return a + b

panier = ['pomme', 'chou', 'laitue', 'chou', 'pomme']
types_de_fruits = ['pomme', 'poire', 'coing']
print(ranger(panier, types_de_fruits))
```