Modélisation et robotique

Exercices 4

1 Révisions du cours

Question A. Valeurs de retour ou effet de bord ? Selon la définition choisie parmi : 0.5 pt 3 min

1.
```
def double(x):
    return 2 * x
```
2.
```
def double(x):
    print(2 * x)
```
3.
```
def double(x):
    print(2 * x)
    return 42
```

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

1.0.1 Correction CORRECTION

```
1: double de trois 6

2: 6
2: double de trois None

3: 6
3: 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. 0.5 pt 3 min

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

1.0.2 Correction 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 %s cases" % distance)
```

Question C. 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. 1 pt 6 min

1.0.3 Correction 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 sujet)
```

Question D. Orientation absolue. Au début du programme le personnage est orienté vers l'ouest. 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`'.

2 pt
12 min

Pour cela nous avons commencé à écrire le code suivant qu'il ne reste plus qu'à compléter.

```
orientation = 0

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

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

— Quel est le sens du symbole `%` dans ce code ?
— Que doit-on écrire à la première ligne de `tgo()` et pourquoi ?
— Compléter le programme.

1.0.4 Correction 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 E. Effet de bord ou passage de valeur ? 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 ?

2 pt
12 min

1.0.5 Correction CORRECTION

Il faut que chaque fonction qui utilise et modifie 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):
    orientation += 1
    tg()
    return o

def tdo(o):
    orientation += -1
    td()
    return o

def quelle_orientation(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'
```

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

o = 0
o = trajet(o); print(quelle_orientation(o))
```

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.

2 Petits programmes

Question F. Orientation initiale. Ajouter une fonction `orienter(point_cardinal)` qui prend en entrée un point cardinal ('`sud`', '`est`', etc.) et oriente le personnage dans cette direction.

1 pt
6 min

2.0.1 Correction CORRECTION

```
def orienter(pc):
    ntg = orientation % 4 # nombre de quarts de tour à droite à faire pour aller à l'est
    if pc == 'nord':
        ntg += 3
    if pc == 'ouest':
        ntg += 2
    if pc == 'sud':
        ntg += 1
    for i in range(ntg):
        tdo()
```

Question G. Compteur kilométrique. 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.

2 pt
12 min

2.0.2 Correction CORRECTION

```
compteur_casometrique = 0
```

```
def avk():
    global compteur_casometrique
    compteur_casometrique += 1
    av()

def kilometrage():
    return compteur_casometrique
```

Question H. 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.

2 pt
12 min

2.0.3 Correction CORRECTION

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

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

Question I. Pas de fruits. Le panier peut contenir toute sorte de légumes mais aussi quelques fruits parmi une liste connues : `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.

1 pt
6 min

2.0.4 Correction 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', 'poire']
types_de_fruits = ['pomme', 'poire', 'coing']
print(legumes(panier, types_de_fruits))
```

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

1 pt
6 min

2.0.5 Correction 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', 'poire']
types_de_fruits = ['pomme', 'poire', 'coing']
print(fruits(panier, types_de_fruits))
```

Question K. 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.

1 pt
6 min

2.0.6 Correction 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', 'poire']
types_de_fruits = ['pomme', 'poire', 'coing']
print(ranger(panier, types_de_fruits))
```

3 Problèmes

Vous pouvez utiliser les fonctions définies jusqu'à maintenant.

Question L. Trouver le mur. Votre personnage est dans une cour rectangulaire, le long d'un des murs, orienté vers l'est. Trouvez de quel côté est le mur.

1.5 pt
9 min

Recommandation : définir et utiliser une fonction `obstacles()` qui retourne 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. Cette fonction est utile pour cette question et elle le sera encore plus, plus tard.

3.0.1 Correction CORRECTION

```
def obstacles():
    resultat = []
    for i in range(4):
        tgo()
        if obstacle():
            resultat += [quelle_orientation()]
    return resultat

def trouver_mur():
    obs = obstacles()
    return obs[0]
```

Question M. Revenir au point de départ. Choisissez une direction et faites le tour de la cour en revenant exactement au point de départ du personnage et en respectant son orientation initiale.

1.5 pt
9 min

3.0.2 Correction CORRECTION

```
def tour_de_cours():
    o = quelle_orientation()
    orienter(trouver_mur())
    tgo()
    d = avancer_toute()
    tgo()
    for i in range(3):
        avancer_toute()
        tgo()
    for i in range(d):
        av()
    orienter(o)
```

Question N. Retenir la sortie. On se donne un système de coordonnées cartésiennes pour repérer le personnage sur la grille de la simulation. On utilise pour cela des paires d'entiers (x, y) où la coordonnée `x` augmente de `1` à chaque déplacement vers l'est et la coordonnée `y` augmente de `1` à chaque déplacement vers le nord (ces coordonnées respectives diminuent dans les directions opposées).

Une porte de sortie vient de s'ouvrir dans l'un des murs de la cour (elle n'est pas près d'un coin). En supposant que votre personnage est initialement aux coordonnées (40, 67), faites lui trouver les coordonnées de la sortie et revenir à sa position initiale. Votre fonction retournera la sortie sous la forme d'une paire de coordonnées.

3 pt
18 min

3.0.3 Correction CORRECTION

La sortie est trouvée lorsque le personnage n'est entouré d'aucun obstacle. à ce moment on enregistre sa position.

```
xcoord, ycoord = 40, 67
def avxy():
    global xcoord
    global ycoord
    if av():
        o = quelle_orientation()
        if o == 'est':
            xcoord += 1
        if o == 'ouest':
            xcoord += -1
        if o == 'nord':
            ycoord += 1
        if o == 'sud':
            ycoord += -1


def trouver_sortie():
    o = quelle_orientation()
    d = 0
    orienter(trouver_mur())
    tgo()
    # avancer au mur suivant en comptant les pas
    while not obstacle():
        avxy()
        d += 1
        if obstacles() == []: # on a trouvé la sortie
                tdo(); avxy()
                sortie = xcoord, ycoord
                dto(); avxy(); tdo()
    tgo()
    for i in range(3):
        while not obstacle():
            avxy()
            if obstacles() == []: # on a trouvé la sortie
                tdo(); avxy()
                sortie = xcoord, ycoord
                dto(); avxy(); tdo()
        tgo()
    for i in range(d):
        av()
```

```
    orienter(o)
    return sortie
```

Question O. Généraliser (bonus). Généraliser votre fonction au cas où il y a plusieurs sorties (mais aucune dans un coin) en retournant la liste des sorties. Cette fois-ci on veut les coordonnées de la sortie mais aussi la direction qu'il faut suivre pour l'emprunter. Par exemple, si on a une sortie dans le mur nord de coordonnées (45, 70) on retiendra dans la liste le triplet `(45, 70, 'nord')`.

Le personnage a maintenant la liste des sorties et il est retourné à son point de départ. Il veut emprunter la sortie qui nécessitera le moins de changements de case de sa part. Aidez-le !

Même question si le personnage pouvait se déplacer en diagonale à l'aide de fonctions `avancer_gauche()` et `avancer_droite()`.

3 pt
18 min

3.0.4 Correction CORRECTION

```
# -*- coding: utf-8 -*-

from gardenworld import *
init("info2_1")

# Placement du personnage
tg()
while not obstacle():
    av()
td()
av(); av();


orientation = 0 # orientation initiale vers l'est

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

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

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

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'

def orienter(pc):
    ntg = orientation % 4 # nombre de quarts de tour à droite à faire pour aller à l'est
    if pc == 'nord':
        ntg += 3
    if pc == 'ouest':
        ntg += 2
    if pc == 'sud':
        ntg += 1
    for i in range(ntg % 4):
        tdo()

def obstacles():
    resultat = []
    for i in range(4):
        tgo()
        if obstacle():
            resultat += [quelle_orientation()]
    return resultat

def trouver_mur():
    obs = obstacles()
    return obs[0]

xcoord, ycoord = 40, 67
def avxy():
    global xcoord
    global ycoord
    if av():
        o = quelle_orientation()
        if o == 'est':
            xcoord += 1
        if o == 'ouest':
            xcoord += -1
        if o == 'nord':
            ycoord += 1
        if o == 'sud':
            ycoord += -1

def trouver_sorties():
    o = quelle_orientation()
    d = 0
    sorties = []
    orienter(trouver_mur())
    tgo()
    # avancer au mur suivant en comptant les pas
```

```
    while not obstacle():
        avxy()
        d += 1
        if obstacles() == []: # on a trouvé la sortie
                tdo(); avxy()
                sorties += [(xcoord, ycoord, quelle_orientation())]
                dto(); avxy(); tdo()

    tgo()
    for i in range(3):
        while not obstacle():
            avxy()
            if obstacles() == []: # on a trouvé la sortie
                tdo(); avxy()
                sorties += [(xcoord, ycoord, quelle_orientation())]
                dto(); avxy(); tdo()

        tgo()
    for i in range(d):
        av()
    orienter(o)
    return sorties

def distance(u, v):
    x0, y0 = u
    x1, y1 = v
    return abs(x1 - x0) + abs(y1 - y0) # distance de Manhattan

def distanceamoi(u):
    x0, y0, o0 = u
    return distance((x0, y0), (xcoord, ycoord))

sorties = trouver_sorties()
x, y, o = min(sorties, key = distanceamoi)

orienter('est')
for i in range(x - xcoord):
    avxy()
orienter('nord')
for i in range(y - ycoord):
    avxy()
orienter('ouest')
for i in range(xcoord - x):
    avxy()
orienter('sud')
for i in range(ycoord - y):
    avxy()
orienter(o)
avxy()
```

```
if (x, y) == (xcoord, ycoord): # cas où l'on a d'abord heurté le mur de la sortie
    avxy()
```