p2

Pierre Boudes

December 13, 2015

Contents

1 QCM

1.1 liste1

```
xs = [3, 8]
xs += [5]
print xs
```

Ce code affiche :

1 `[3, 8, 5]`

-0.25 `[3, 5, 8]`

-0.25 `[3, 8]`

-0.25 `[5]`

-0.25 `[8, 13]`

1.2 liste2

```
xs = [12, 18]
xs.append(1)
print xs
```

Ce code affiche :

1 `[12, 18, 1]`

-0.25 `[12, 1, 18]`

-0.25 `[12, 18]`

-0.25 `[12]`

-0.25 `[13, 19]`

1.3 liste3

```
xs = range(5) + range(3)
print xs[2:]
```

Ce code affiche :

1 `[2, 3, 4, 0, 1, 2]`

-0.25 `[0, 1, 2, 3, 4, 0, 1, 2]`

-0.25 `[0, 1]`

-0.25 `[5, 3]`

-0.25 `[0, 1, 2, 3, 4, 5, 0, 1, 2, 3]`

1.4 liste4

```
xs = range(5) + range(3)
print xs[4:8]
```

Ce code affiche :

1 `[4, 0, 1, 2]`

-0.25 `[0, 1, 2, 3]`

-0.25 `[0, 1]`

-0.25 `[4, 5, 6, 7]`

1.5 liste5soupes

```
xs = ["choux", "carottes", "poireaux"]
soupes = []
for i in xs:
    for j in xs:
        if i < j:
            soupes.append(i + "-" + j)
print soupes
```

Ce code affiche :

1 `['choux-carottes', 'choux-poireaux', 'carottes-choux', 'carottes-poireaux', 'poireaux-choux', 'poireaux-carottes']`

-0.25 `['choux-choux', 'choux-carottes', 'choux-poireaux', 'carottes-choux', 'carottes-carottes', 'carottes-poireaux', 'poireaux-choux', 'poireaux-carottes', 'poireaux-poireaux']`

-0.25 `['choux-carottes', 'poireaux-choux', 'poireaux-carottes']`

-0.25 `['choux-poireaux', 'carottes-choux', 'carottes-poireaux']`

1.6 type1

Quel est le type de `3 + 5 * 10` ?

1 int

-0.25 float

-0.25 list

-0.25 set

-0.25 tuple

-0.25 str

-0.25 dict

-0.25 aucun, il y a une erreur

1.7 type2

Quel est le type de `3 + 5 * 10.0` ?

1 float

-0.25 int

-0.25 list

-0.25 set

-0.25 tuple

-0.25 str

-0.25 dict

-0.25 aucun, il y a une erreur

1.8 type3

Quel est le type de `{"chou", "chou", "carotte"}` ?

1 set

-0.25 float

-0.25 list

-0.25 int

-0.25 tuple

-0.25 str

-0.25 dict

-0.25 aucun, il y a une erreur

1.9 type4

Quel est le type de `{"i": 0, "delai": 10}` ?

1 dict

-0.25 float

-0.25 list

-0.25 int

-0.25 tuple

-0.25 dict

-0.25 str

-0.25 aucun, il y a une erreur

1.10 type5

Quel est le type de `{}` ?

1 dict

-0.25 float

-0.25 list

-0.25 int

-0.25 tuple

-0.25 set

-0.25 str

-0.25 aucun, il y a une erreur

1.11 type6

Quel est le type de `"bonjour !" * 5` ?

1 str

-0.25 int

-0.25 float

-0.25 list

-0.25 set

-0.25 dict

-0.25 tuple

-0.25 aucun, il y a une erreur

1.12 type7

Quel est le type de `"bonjour !" + 5` ?

1 aucun, il y a une erreur

-0.25 str

-0.25 int

-0.25 float

-0.25 list

-0.25 set

-0.25 dict

-0.25 tuple

1.13 type8

Quel est le type de `[1, 'a', 3] + [1, 2]` ?

1 list

-0.25 str

-0.25 int

-0.25 float

-0.25 aucun, il y a une erreur

-0.25 set

-0.25 dict

-0.25 tuple

1.14 type9

Quel est le type de `'chou'+'fleur'` ?

1 str

-0.25 list

-0.25 int

-0.25 float

-0.25 aucun, il y a une erreur

-0.25 set

-0.25 dict

-0.25 tuple

1.15 type10

Quel est le type de `'chou'-'fleur'` ?

1 aucun, il y a une erreur

-0.25 str

-0.25 int

-0.25 float

-0.25 list

-0.25 set

-0.25 dict

-0.25 tuple

1.16 type11

Quel est le type de `(1,'chou','fleur')` ?

1 tuple

-0.25 str

-0.25 aucun, il y a une erreur

-0.25 int

-0.25 float

-0.25 list

-0.25 set

-0.25 dict

1.17 set1

```
s1 = set(["gauche", "haut", "haut", "droite"])
s2 = (s1 | {"bas", "droite"}) - {"haut"}
print(s2)
```

Ce code affiche :

1 `set(['droite', 'gauche', 'bas'])`

-0.25 `set(['droite', 'bas'])`

-0.25 `set(['droite', 'haut', 'gauche', 'bas'])`

-0.25 `set(['droite'])`

-0.25 `set(['droite', 'haut', 'haut', 'gauche', 'bas'])`

1.18 set2

```
s1 = set(["gauche", "haut", "haut"])
s2 = set(["gauche", "droite"])
s3 = ?? # intersection
```

Pour trouver l'intersection de s1 et de s2 on écrira :

1 `s3 = s1 & s2`

-0.25 `s3 = s1 | s2`

-0.25 `s3 = s1 - s2`

-0.25 `s3 = s1 + s2`

-0.25 `s3 = s1 / s2`

-0.25 `s3 = s1 ^ s2`

1.19 numpy

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 6, 2])
print ((2 * a + b) / 2)
```

Ce code affiche :

1 `[3 5 4]`

-0.25 `[1, 2, 3, 1, 2, 3, 4, 6, 2]`

-0.25 `[1, 3, 1, 3, 4, 6]`

-0.25 `[2, 4, 6, 4, 6, 2]`

-0.25 `[1, 2, 3, 2, 3, 1]`

1.20 split1

```
menu = "choux, carottes, poireaux, etc."
ingredients = menu.split(',')
print(ingredients)
```

Ce code affiche :

1 `["choux", " carottes", " poireaux", " etc."]`

-0.25 `["choux", "carottes", "poireaux", "etc."]`

-0.25 `["choux" "carottes" "poireaux"]`

-0.25 `["choux", "carottes", "poireaux"]`

1.21 join1

```
xs = ["choux", "carottes", "poireaux"]
print('et '.join(xs))
```

Ce code affiche :

1 `chouxet carotteset poireaux`

-0.25 `choux et carottes et poireaux et`

-0.25 `choux et carottes et poireaux`

-0.25 `etchouxetcarottesetpoireaux`

-0.25 `et choux et carottes et poireaux et`

1.22 fonction1

Les 7 nains veulent une fonction `partageable()` qui prend en entrée un nombre et retourne True si il est divisible par 7 et False sinon. Quel code choisir ?

```
def partageable_happy():
    if a % 7 == 0:
        return True
    return False

def partageable_sneezy(pizza):
    return 0 == pizza % 7

def partageable_sleepy(a):
    if a > 7:
        return True
    else:
        return False

def partageable_grumpy(a):
    if a % 7 == 0:
        print True
    else:
        print False
```

1 la fonction de `sneezy`

-0.25 la fonction de `sleepy`

-0.25 la fonction de `grumpy`

-0.25 la fonction de `happy`

1.23 foonone

```
def foo():
    print "foo"
def bar():
    return "bar"
a = [foo(), bar()]
```

Que vaut a ?

1 `[None, "bar"]`

-0.25 `["foo", "bar"]`

-0.25 `[foo(), bar()]`

-0.25 `[foo(), bar()]`

1.24 trianglelogo

```
from robosim import *
init()
angle = 60
longueur = 10
for i in range(3):
    tg(angle)
    av(longueur)
```

Il y a une erreur dans ce code. On voulait faire parcourir un triangle équilatéral de côté 10 au robot, mais ce n'est pas ce qu'il fait.

1 L'erreur est à la troisième ligne

-0.25 L'erreur est à la quatrième ligne

-0.25 L'erreur est à la cinquième ligne

-0.25 L'erreur est à la sixième ligne

-0.25 L'erreur est à la septième ligne

1.25 carrelogo

```
from robosim import *
init()
angle = 90
longueur = 10
for i in range(4):
    tg(angle)
av(longueur)
```

Il y a une erreur dans ce code. On voulait faire parcourir un carré de côté 10 au robot, mais ce n'est pas ce qu'il fait.

1 L'erreur est à la troisième ligne

-0.25 L'erreur est à la quatrième ligne

-0.25 L'erreur est à la cinquième ligne

-0.25 L'erreur est à la sixième ligne

-0.25 L'erreur est à la septième ligne

1.26 zip1

```
x = zip(*[range(1,4), ['TS', 'PN', 'JC']])
print(x)
```

Ce code affiche :

1 `[(1, 'TS'), (2, 'PN'), (3, 'JC')]`

-0.25 `[1, 2, 3, 'TS', 'PN', 'JC']`

-0.25 `[[1, 2, 3], ['TS', 'PN', 'JC']]`

-0.25 Rien, il y a une erreur

1.27 boucle1

```
for i in range(1000):
    for j in range(1000):
        print('bonjour')
    for k in range(1000):
        print('bonjour')
print(x)
```

Ce code affiche :

1 Deux millions de 'bonjour'

-0.25 un milliard de 'bonjour'

-0.25 deux 'bonjour'

-0.25 un million de 'bonjour'

-0.25 Rien, il y a une erreur

1.28 boucle2

```
for i in range(1000):
    for j in range(1000):
        for k in range(1000):
            print('bonjour')
```

Ce code affiche :

1 un milliard de 'bonjour'

-0.25 Deux millions de 'bonjour'

-0.25 deux 'bonjour'

-0.25 un million de 'bonjour'

-0.25 Rien, il y a une erreur

1.29 gene1

```
def legumes():
    yield "chou"
    yield "fraise"
    yield "patate"

for i in legumes():
    print(i)
```

Ce code affiche :

1 `chou`, `fraise` et `patate`

-0.25 `chou`

-0.25 `fraise`

-0.25 `patate`

-0.25 Rien, il y a une erreur

1.30 gene2

```
def legumes():
    yield "chou"
    yield "fraise"
    yield "patate"

gen = legumes()
gen.next()
print gen.next()
```

Ce code affiche :

1 `chou`

-0.25 `chou`, `fraise` et `patate`

-0.25 `fraise`

-0.25 `patate`

-0.25 Rien, il y a une erreur

1.31 fauxfact

```
def factorielle(n)
    return sum(range(n + 1))

print factorielle(5)
```

Ce code affiche :

1 Rien, il y a une erreur

-0.25 120

-0.25 0

-0.25 15

1.32 sqrt

```
from math import *
print (sqrt(4), abs(-3), 2**3)
```

Ce code affiche :

1 `(2.0, 3, 8)`

-0.25 Rien, il y a une erreur

-0.25 `(4.0, 3, 9)`

-0.25 `(8.0, 3, 8)`

-0.25 `(2.0, -3, 9)`

-0.25 `(8.0, 3, 9)`

1.33 inset

```
x = {1, 4, 5}
y = {"a": 1, "b": 2, 5: 4}
for e in x:
    if e in y:
        print e
```

Ce code affiche :

1 `5`

-0.25 Il y a une erreur

-0.25 `1` puis `5`

-0.25 `1` puis `4`

-0.25 Rien

1.34 atan2

```
from math import *
print degrees(atan2(0, -1))
```

Ce code affiche :

1 `180`

-0.25 Rien, il y a une erreur

-0.25 `90`

-0.25 `-90`

-0.25 `0`

1.35 optim

La méthode 1+1ES est un algorithme évolutionnaire d'optimisation d'un fonction avec paramètres :

1 dans lequel on fait muter les paramètres alétoirement en ajoutant un bruit gaussien à la meilleur solution actuelle et on teste si la nouvelle solution est encore meilleure, itérativement.

-0.25 dans lequel on cherche la meilleure solution en déplaçant le robot au hasard.

-0.25 dans lequel on obtient les bons paramètres en prenant la moyenne entre les anciens paramètres et des nouveaux paramètres tirés aléatoirement.

1.36 PID

En robotique un contrôleur PID signifie :

1 correcteur proportionnel, intégrale, dérivée

-0.25 pilote initial et dérivé

-0.25 pilote intégral différentiel

-0.25 pas-à-pas interne de dérapage

1.37 gardenworld1

```
from gardenworld import *
init('monde_sans_obstacle')
# personnage en position initiale
for i in range(100):
    td()
```

```
    av()
    for j in range(2):
        tg()
# position finale
```

Dans un monde sans obstacle, avec le personnage oritenté vers l'est, on exécute le code suivant. Où sera le personnage par rapport à sa position initiale ?

1 à la position initiale orienté vers l'est

-0.25 une case plus au sud orienté vers l'ouest

-0.25 une case plus au sud et une case plus à l'est orienté vers l'ouest

-0.25 une case plus à l'est orienté vers l'est

-0.25 une case plus à l'est orienté vers l'ouest

1.38 alphacap

```
angle = 0; x, y = 460, 460; alpha = 0.9
for i in range(100):
    angle = alpha * angle + (1 - alpha) * cap(x, y)
    oriente(angle) # oriente le robot
    av(5)
```

Pour que le robot prenne le cap du point (x, y) plus rapidement, que faut-il faire ?

1 diminuer alpha

-0.25 augmenter alpha

-0.25 augmenter angle

-0.25 diminuer angle

1.39 trunc

```
maxi = 2
for i in [1, 9, 8, 4]:
    print min(i, maxi)
```

Ce code affiche :

1 1 2 2 2

-0.25 1 9 8 4

-0.25 2 9 8 4

-0.25 min min min min

1.40 distance

```
def distance(a, b):
    (x0, y0) = a
    (x, y) = b
    return (x - x0)**2 + (y - y0)**2

distance(position(), (100, 100))
```

La fonction `distance` doit calculer la distance euclidienne entre deux points du plan :

1 elle sera exécutée mais elle n'aura pas le bon résultat

-0.25 son exécution provoquera une erreur

-0.25 il manque un `print` car il faut toujours afficher les valeurs calculées

1.41 dist2

```
def f(x, y, alpha, beta):
    for i in range(200):
        avancer_vers(x, y, alpha, beta)
        d =  distance((x, y), position())
        if d < 10:
            break
    return d
```

On veut optimiser les paramètres alpha et beta de la fonction `avancer_vers` de telle sorte qu'elle permette d'approcher au plus près du but (x, y) en un minimum d'étapes. Faut-il :

1 retourner d + i puis trouver alpha et beta tels que f prenne la plus petite valeur possible ?

-0.25 retourner d + i et trouver alpha et beta tels que f prenne la plus grande valeur possible ?

-0.25 retourner d - i et trouver alpha et beta tels que f prenne la plus grande valeur possible ?

-0.25 retourner d - i et trouver alpha et beta tels que f prenne la plus petite valeur possible ?

1.42 state

```
state = {"etat": "depart"}
state["i"] = 0

def progression(evt):
    state["i"] += 1
    if evt == "avancer":
        state["etat"] = "devant"
    if evt == "reculer":
        state["etat"] = "depart"

for e in ["avancer", "reculer", "avancer"]:
    progression(e)

print state
```

Ce code affiche :

1 {'etat': 'devant', 'i': 3}

-0.25 {'etat': 'depart', 'i': 3}

-0.25 {'etat': 'avancer', 'i': 3}

-0.25 {'etat': 'reculer', 'i': 3}

2 Problèmes

2.1 Orientation (exercice de TD)

Dans `gardenworld`, on supposera que l'on a une fonction `orientation()` sans paramètre qui retourne l'orientation actuelle du robot, parmi les points cardinaux `Est`, `Ñord`, `Ouest`, `Sud`. On ne présente plus les fonctions `av(x)`,

`tg()`, `td()` et `obstacle()`. Définir une fonction `obstacles()` qui retourne la liste des directions (sous forme de points cardinaux) dans lesquelles se trouve un obstacle dans la case à côté du robot.

2.2 Split avec plusieurs séparateurs (exercice de TD)

Écrire une fonction `split(chaine, seps)` qui prend en entrée une chaine de caractères et une liste de séparateurs et retourne la liste des mots de la chaîne de départ en les ayants séparés selon les séparateurs. Ainsi la fonction split usuelle correspondra au cas particulier ou `seps` contient un seul séparateur. Exemple :

```
>>> split('hello,jo.tu vas bien', [',','.',' '])
['hello', 'jo', 'tu', 'vas', 'bien']
```

2.3 Progression

Définir une fonction de progression pour le Thymio de façon à que le Thymio change de couleur alternativement entre bleu et rouge (`become.blue`, `become.red`) à chaque fois qu'on appuie sur le bouton 0 (le bouton pointant vers l'arrière). Vous ferait particulièrement attention à éviter qu'une seule pression sur le bouton fasse changer le Thymio plusieurs fois de couleur. Vous insérerez votre fonction dans le code suivant (dites simplement où).

```
import pythymio as pt
custom = pt.customEvents('colors')
state = {}

with pt.thymio(['buttons'], custom) as Thym:

    Thym.loop(progression)
```

Question A. Orientation (exercice de TD) Dans `gardenworld`, on supposera que l'on a une fonction `orientation()` sans paramètre qui retourne l'orientation actuelle du robot, parmi les points cardinaux `Est`, `Ñord`, `Ouest`, `Sud`. On ne présente plus les fonctions `av(x)`, `tg()`, `td()` et `obstacle()`. Définir une fonction `obstacles()` qui retourne la liste des directions (sous forme de points cardinaux) dans lesquelles se trouve un obstacle dans la case à côté du robot.

Question B. Split avec plusieurs séparateurs (exercice de TD) Écrire une fonction `split(chaine, seps)` qui prend en entrée une chaine de caractères et une liste de séparateurs et retourne la liste des mots de la chaîne de départ en les ayants séparés selon les séparateurs. Ainsi la fonction split usuelle correspondra au cas particulier ou `seps` contient un seul séparateur. Exemple : splitjoin »> split('hello,jo.tu vas bien', [',','.',' ']) ['hello', 'jo', 'tu', 'vas', 'bien']

Progression Définir une fonction de progression pour le Thymio de façon à que le Thymio change de couleur alternativement entre bleu et rouge (`become.blue`, `become.red`) à chaque fois qu'on appuie sur le bouton 0 (le bouton pointant vers l'arrière). Vous ferait particulièrement attention à éviter qu'une seule pression sur le bouton fasse changer le Thymio plusieurs fois de couleur. Vous insérerez votre fonction dans le code suivant (dites simplement où). prog import pythymio as pt custom = pt.customEvents('colors') state =

with pt.thymio(['buttons'], custom) as Thym:

Thym.loop(progression)

2

QCM de Modélisation Robotique **Partiel 2 décembre 2015**

⟵ codez votre numéro d'étudiant ci-contre, et écrivez votre nom et prénom ci-dessous.

Nom et prénom :

. . . . . .

etu8

---

orgamc orgamc

```
(md5 "salut")
```