10/01/2024 IC1T

Examen de janvier

Durée: 2h Version: **A**

Instructions

— Répondez sur la feuille de réponse séparée **à votre nom**. **Aucune** réponse écrite sur le présent énoncé ne sera prise en compte dans la correction
— Répondez en noircissant la case de la réponse avec un **stylo ou un feutre** (pas de crayon).
— Noircissez aussi la version (**A** ou **B**) de votre énoncé sur la feuille de réponse.
— Pour corriger une réponse, **effacez complètement** la mauvaise case avec un typex avant d'en noircir une autre.
— Vous ne devez pas remettre l'énoncé. Vous pouvez donc l'utiliser comme brouillon.
— Il n'y a pas de points négatifs.
— **Aucun** document n'est autorisé.

Question 1 (1 point)

Laquelle des propositions suivantes est fause ?

A) la mémoire est plus rapide que le stockage de masse.

B) Les stockages de masse n'ont pas besoin d'être alimentés pour conserver les données.

C) Les instructions executées par le processeur se trouvent dans la mémoire.

D) Le contenu du stockage de masse est donc organisé en une hiérarchie de répertoires

E) Python est un language interprété

F) La commande `pwd` affiche le répertoire courant

G) `C:\Users\lur` est un chemin absolu

H) Les instructions executées par le processeur sont en Python.

I) la variable `PATH` contient une liste de dossiers

J) l'interpréteur est nécessaire pour chaque exécution d'un language interprété

Question 2 (1 point)

Que vaut `a` à la fin de l'exécution du code suivant ?

```
1  a = '1'
2  a = a * 2
3  a = int(a)
4  a %= 4
```

A) 2 **B)** 4 **C)** 8 **D)** 12 **E)** 3

F) 11 **G)** 2.25 **H)** 0.25 **I)** 121 **J)** 0.5

Question 3 (2 points)

Que vaut `d` à la fin de l'exécution du code suivant ?

```
1  b = 8
2  c = b < 4 or b > 5
3  b = len(str(c)+'!!')
4  c = 2 * 2 ** 2
5  d = b + c
```

A) 9 **B)** 12 **C)** 5 **D)** 13 **E)** 20

F) 4 **G)** 8 **H)** 14 **I)** 2 **J)** 6

Question 4 (2 points)

Que va afficher le code suivant ?

```
1  def fun(li, value):
2    res = []
3    for item in li:
4      if len(item) > value:
5        res.append(item)
6    return res
7
8  print(fun(['Le', 'Python', 'est', 'trop', 'facile'], 5))
```

A) `['Le', 'Python', 'est', 'trop', 'facile']`

B) `['Le', 'Pytho', 'est', 'trop', 'facil']`

C) `['Python', 'facile']`

D) `['Le', 'est', 'trop']`

E) `[2, 6, 3, 4, 6]`

F) `['Le', 5, 'est', 'trop', 5]`

G) `[5, 'Python', 5, 5, 'facile']`

H) `[]`

I) `['L', 'P', 'e', 't', 'f']`

Question 5 (2 points)

Dans le code suivant, la fonction `fact()` devrait calculer la factorielle de son paramètre. Malheureusement, le code plante lorqu'on l'exécute.

```
1  def fact(n):
2    while n > 0:
3      res *= n
4      n -= 1
5    return res
6
7  print(fact(3))
```

Le message d'erreur est le suivant :

```
Traceback (most recent call last):
  File "program.py", line 8, in <module>
    print(fact(3))
          ^^^^^^^
  File "program.py", line 4, in fact
    res *= n
    ^^^
UnboundLocalError: cannot access local variable 'res' where it is not associated
with a value
```

Que faut-il changer pour corriger la fonction ? *(les → indiquent le niveau d'indentation)*

A) ligne 2: → `while n >= 0:`

B) entre les lignes 2 et 3: → → `res = 1`

C) ligne 1: `def fact(res):`

D) ligne 3: → → `res = n`

E) entre les lignes 4 et 5: → → `res *= n`

F) ligne 5: → → `return res`

G) ligne 4: → → `n += 1`

H) entre les lignes 1 et 2: → `res = 1`

I) ligne 7: `print(fact(res))`

J) entre les lignes 1 et 2: → `res = 0`

Programme `Pygame`

Voici le code d'un programme Pygame. Ce programme devrait permettre de cliquer dans une fenêtre et de dessiner des lignes allant d'un point cliqué à l'autre. Il manque deux instructions dans le programme.

```
1  import pygame
2  import sys
3
4  pygame.init()
5  screen = pygame.display.set_mode((800, 600))
6
7  # première instruction manquante
8
9  while True:
10   for event in pygame.event.get():
11     if event.type == pygame.QUIT:
12       sys.exit()
13     if event.type == pygame.MOUSEBUTTONDOWN:
14       # deuxième instruction manquante
15
16   pygame.draw.rect(screen,(0, 0, 0),pygame.Rect(0, 0, 800, 600))
17
18   if len(points) > 1:
19     # Dessine des segments allant d'un point à l'autre de
20     # la liste de points 'points'
21     pygame.draw.lines(screen, (255, 255, 255), False, points)
22
23   pygame.display.flip()
```

Voici des propositions pour combler les instructions manquantes *(les → indiquent le niveau d'indentation)* :

A) `points = 0`

B) → → → `points.append(event.pos)`

C) `points = [(x, y)]`

D) → → → `points = [event.pos]`

E) → → → `points = [(x, y)]`

F) `points = tuple()`

G) `points = []`

H) `points = (x, y)`

I) → → → `points.append(event.key)`

J) → → → `points.append((x, y))`

Question 6 (2 points)

Dans le programme `pygame` ci-dessus, quelle proposition choisissez-vous pour la première instruction manquante ?

Question 7 (2 points)

Dans le programme `pygame` ci-dessus, quelle proposition choisissez-vous pour la deuxième instruction manquante ?

Programme traitement de listes

Voici un programme contenant des fonctions de traitement de listes :

```
1  def f1(li):
2    res = []
3    for i, value in enumerate(li):
4      if i%2 == 0:
5        res.append(value)
6    return res
7
8  def f2(li):
9    res = []
10   for value in li:
11     res.append(str(value))
12   return res
13
14 def f3(li):
15   res = []
16   for item in li:
17     if item > value:
18       res.append(value)
19     else:
20       res.append(item)
21   return res
22
```

```
23 def f4(li):
24   res = []
25   for value in li:
26     res.append(value*2)
27   return res
28
29 value = 3
30 item = 2
31 i = 4
32 res = 1
33
34 li = [value, item, i, res, 2]
35 li = f3(li)
36 li = f4(li)
37 li = f1(li)
38 li = f2(li)
39 li = f4(li)
40
41 print(li)
```

Question 8 (2 points)

Dans le programme de traitement de listes, laquelle des variables décrites ci-dessous est globale ?

A) `value` dans `f4`

B) `res` dans `f1`

C) `i` dans `f1`

D) `value` dans `f2`

E) `item` dans `f3`

F) `res` dans `f4`

G) `value` dans `f1`

H) `value` dans `f3`

I) aucune des propositions

Question 9 (3 points)

Que va afficher le programme de traitement de listes ?

A) `[3, 3, 2]`

B) `['33', '44', '22']`

C) `['3', '2', '3', '1', '2']`

D) `[3, 2, 4, 1, 2]`

E) `[33, 22, 44, 11, 22]`

F) `['66', '44', '66', '22', '44']`

G) `['66', '66', '44']`

H) aucune des propositions

Question 10 (3 points)

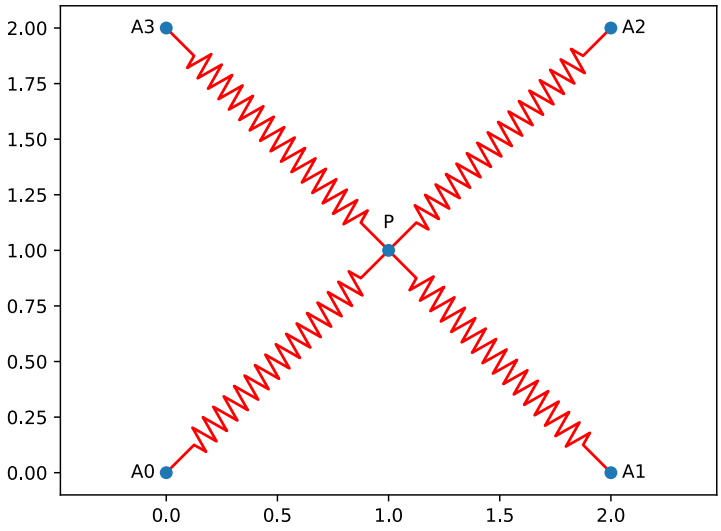
Dans le programme suivant, la fonction `net_hook_force()` calcule la résultante des forces de plusieurs ressorts accrochés à un point `P`.Pour rappel, la force d'un ressort est donnée par la loi de Hook :

$$F = -k\Delta l$$

L'autre extrémité de chaque ressort est accrochée à un point fixe. Pour chaque ressort, la fonction reçoit : la position du point fixe (dans `anchors`), la constante de raideur (dans `ks`) et la longeur au repos (dans `rest_lengths`).

```
1  import numpy as np
2
3  def net_hook_force(anchors, ks, rest_lengths, position):
4    lengths = []
5    directions = []
6    for anchor in anchors:
7      PA = np.array(position) - anchor
8      length = np.linalg.norm(PA)
9      lengths.append(length)
10     directions.append(PA/length)
11   forces = - (np.array(lengths) - rest_lengths) * ks
12   net = 0
13   for i in range(len(forces)):
14     net += forces[i]
15   return net
```

La fonction est ensuite utilisée pour une situation avec 4 ressorts disposés comme dans la figure suivantes :




```
anchors = [(0.0, 0.0), (2.0, 0.0), (2.0, 2.0), (0.0, 2.0)]
lengths = [1.0, 1.0, 1.0, 1.0]
ks = [0.5, 1.0, 1.0, 0.5]

print(net_hook_force(anchors, ks, lengths, (1.0, 1.0)))
```

Comme les deux ressorts de droite sont plus forts on devrait obtenir une résultante qui pointe vers la droite et le programme devrait afficher :

```
[0.29289322 0.0]
```

Malheureusement le programme affiche :

```
1.2426406871192852
```

Que faut-il changer pour corriger la fonction ? *(les → indiquent le niveau d'indentation)*

A) ligne 5: → `directions = np.array([])`

B) ligne 13: → `for i in range(len(anchors))`

C) ligne 7: → → `PA = anchor - position`

D) ligne 14: → → `net += directions[i] * forces[i]`

E) ligne 8: → → `length = PA @ PA`

F) ligne 4: → `lengths = np.array([])`

G) ligne 9: → → `lengths.append(PA)`

H) ligne 10: → → `directions.append(PA)`

I) ligne 11: → `forces = (lengths - rest_lengths) * ks`

J) ligne 15: → → `return forces`

Brouillon