23/05/2024 IC1T

# Examen de programmation

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 fausse ?

**A)** Le stockage de masse est plus lent que la mémoire.

**B)** Un programme est une suite d'instructions visant à la résolution d'un problème.

**C)** `.\Document\labo.txt` est un chemin absolu

**D)** La mémoire est volatile.

**E)** Un SSD est un stockage de masse

**F)** Python est un language interprété

**G)** La commande `cd` permet de changer de répertoire courant

**H)** Les instructions executées par le processeur sont en langage machine.

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

**J)** `numpy` est un module qui doit être installé pour être utilisé.

## Question 2 (1 point)

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

```
1  import numpy as np
2
3  x = np.array([2, 4, 1])
4  x *= np.array([3, 2, 5])
5  x -= 1
```

**A)** `18`

**B)** `[17 -8 -9]`

**C)** `239`

**D)** `37`

**E)** `[1 4 1 3 2 5]`

**F)** `[5 8 5]`

**G)** `[17 -7 -8]`

**H)** `[5 7 4]`

**I)** `-1`

**J)** Message d'erreur

## 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  if c:
4    b = 2 + 3 * 4 - b
5  else:
6    b = b * 2 - 2 * 4
7  d = b + len(str(c))
```

**A)** `9` **B)** `12` **C)** `1` **D)** `13` **E)** `20`

**F)** `3` **G)** `8` **H)** `10` **I)** `2` **J)** `16`

## Question 4 (2 points)

Que va afficher le code suivant ?

```
1  def fun(L):
2    if len(L) == 0:
3        return 0
4    if L[0] % 2 == 0:
5        return L[0] / 2 + fun(L[1:])
6    else:
7        return fun(L[1:])
8
9  print(fun([4, 6, 7, 3, 12, -2]))
```

**A)** `0`

**B)** `[4, 6, 7, 3, 12, -2]`

**C)** `[4, 6, 12, -2]`

**D)** `5.0`

**E)** `[7, 3]`

**F)** `[2, 3, 6, -1]`

**G)** `[3.5, 1.5]`

**H)** `10.0`

**I)** `[]`

## Question 5 (2 points)

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

```
1  x = 1
2  L = []
3  while len(L) < 5:
4    div = 0
5    d = 1
6    while d <= x:
7      if x % d == 0:
8        div += 1
9      d += 1
10   if div == 2:
11     L.append(x)
12   x += 1
```

**A)** `[]`

**B)** `[1, 1, 1, 1, 1]`

**C)** `[1, 1, 2, 3, 5]`

**D)** `[2, 4, 6, 8, 10]`

**E)** `[2, 2, 2, 2, 2]`

**F)** `[2, 3, 5, 7, 11]`

**G)** `[2, 3, 5, 8, 13]`

**H)** Le programme ne s'arrête pas (boucle infinie)

**I)** Le programme plante (message d'erreur)

## Question 6 (2 points)

Dans le code suivant, la fonction `fun()` renvoyer des paires d'éléments consécutifs de la liste d'entrée *(`[[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]` pour l'appel de la ligne 8)*. Malheureusement, le code plante lorqu'on l'exécute.

```
1  def fun(L):
2    res = []
3    if len(L) > 2:
4      for i in range(len(L)):
5        res.append([L[i], L[i + 1]])
6    return res
7
8  print(fun([1, 2, 3, 4, 5, 6]))
```

Le message d'erreur est le suivant :

```
Traceback (most recent call last):
  File "/Users/lur/test.py", line 8, in <module>
    print(fun([1, 2, 3, 4, 5, 6]))
          ^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/lur/test.py", line 5, in fun
    res.append([L[i], L[i + 1]])
                      ~^^^^^^^
IndexError: list index out of range
```

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

**A)** ligne 2: → `res = 0`

**B)** ligne 3: → `if len(L) < 2:`

**C)** ligne 4: → → `for i in range(len(L)-1):`

**D)** entre les lignes 4 et 5: → → → `i = i - 1`

**E)** ligne 5: → → → `res.append([L[i], L[i - 1]])`

**F)** ligne 6: → `return res[:-1]`

**G)** ligne 8: `print(fun([1, 2, 3, 4, 5]))`

**H)** Les propositions A et D

**I)** Les propositions B et E

**J)** Aucune proposition n'est correcte

## Programme triangle de Sierpiński

Voici le code d'un programme dessinant le triangle de Sierpiński:

```
1  import turtle
2  from math import cos, pi, sin
3
4  def mid(p1, p2):
5    x = p1[0] + p2[0] / 2
6    y = p1[1] + p2[1] / 2
7    return (x, y)
8
9  def triangle(p1, p2, p3):
10   turtle.up()
11   turtle.goto(p1)
12   turtle.down()
13   turtle.goto(p2)
14   turtle.goto(p3)
15   turtle.goto(p1)
16
17 def sierpinski(p1, p2, p3, depth):
18   if depth == 0:
19     triangle(p1, p2, p3)
20   else:
21     sierpinski(p1, mid(p1, p2), mid(p3, p1), depth - 1)
22     sierpinski(mid(p1, p2), p2, mid(p2, p3), depth - 1)
23     sierpinski(mid(p3, p1), mid(p2, p3), p3, depth - 1)
24
25 def polar(radius, angle):
26   x = radius * cos(angle)
27   y = radius * sin(angle)
28   return (x, y)
29
30 depth = 5
31 radius = 300
32 sierpinski(
33   polar(radius, pi / 2),
34   polar(radius, pi / 2 + 2 * pi / 3),
35   polar(radius, pi / 2 - 2 * pi / 3),
36   depth,
37 )
38 turtle.done()
```

Ce programme utilise le module `turtle` dont voici les principales fonctions :

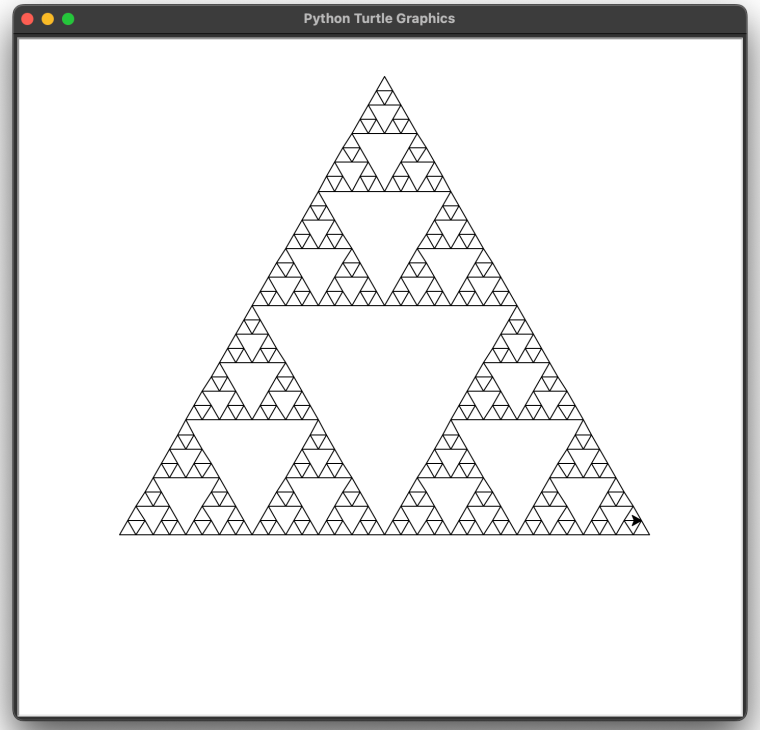
— `turtle.up()`: Relever le crayon

— `turtle.down()`: Abaisser le crayon

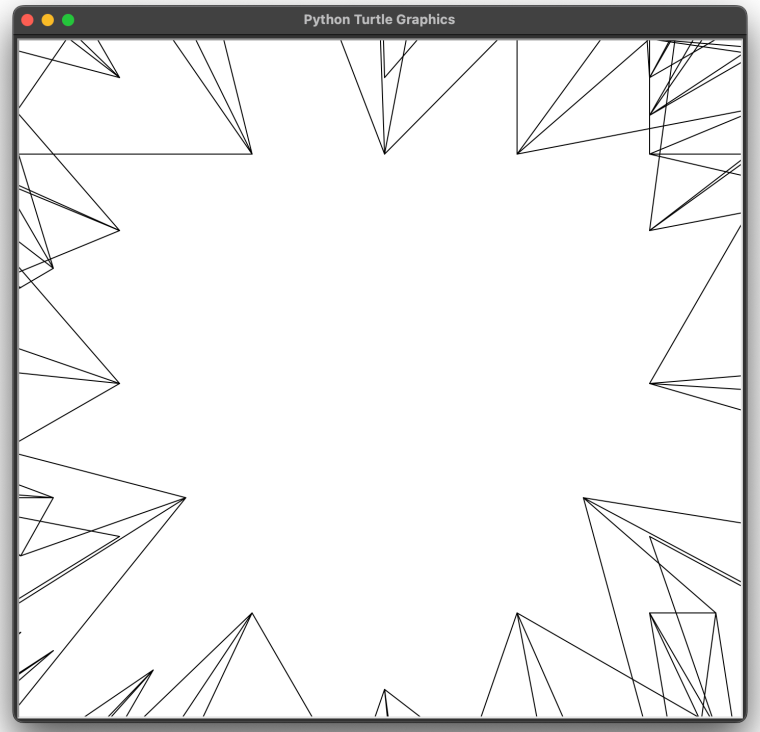
— `turtle.goto(p)`: Aller au point de coordonnées `p = (x, y)`

— `turtle.done()`: Permet que la fenêtre de résultat reste ouverte une fois le dessin terminé.

Le résultat de programme devrait être le suivant:




Cepandant une erreur s'est glissée dans le programme et on obtient ceci :



## Question 7 (3 points)

Dans le programme du triangle de Sierpiński, où se situe l'erreur ?

**A)** Dans l'appel à la fonction `sierpinski()` fait à la ligne 32

**B)** Dans la définition de la fonction `triangle()`

**C)** La valeur initiale de `depth` est incorrecte (ligne 30)

**D)** Dans les `import`

**E)** Dans le corp du `if` de la définition de la fonction `sierpinski()`

**F)** Dans le corp du `else` de la définition de la fonction `sierpinski()`

**G)** Dans la définition de la fonction `mid()`

**H)** Dans la définition de la fonction `polar()`

**I)** À plusieurs endroits en même temps

## Question 8 (2 points)

Dans le programme du triangle de Sierpiński, laquelle des variables décrites ci-dessous est globale ?

**A)** `p1` dans `mid`

**B)** `p2` dans `triangle`

**C)** `depth` dans `sierpinksi`

**D)** `radius` dans `polar`

**E)** `x` dans `mid`

**F)** `p3` dans `sierpinski`

**G)** `y` dans `mid`

**H)** `angle` dans `polar`

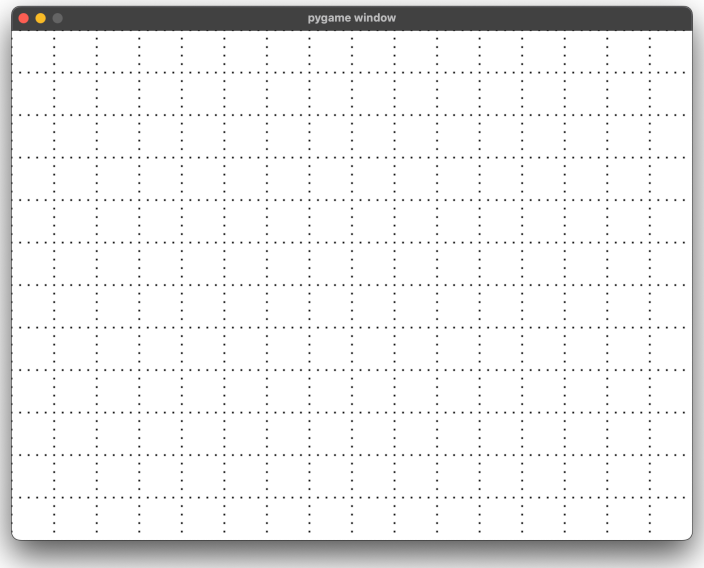
**I)** aucune des propositions

## Programme `pygame`

Voici le code d'un programme `pygame` où il manque deux instructions :

```python
import pygame

pygame.init()
size = (800, 600)
width, height = size
screen = pygame.display.set_mode(size)

running = True

while running:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      # Première instruction manquante

  pygame.draw.rect(screen, (255, 255, 255), pygame.Rect(0, 0, width, height))

  minor = 10  # Espace entre les points (en pixels)
  major = 5  # Espace entre les "lignes" (en points)
  for i in range(0, width, minor):
    for j in range(0, height, minor):
      # Deuxième instruction manquante
        pygame.draw.circle(
          surface=screen,
          color=(0, 0, 0),
          center=(i, j),
          radius=1,
        )

  pygame.display.flip()
```

Ce programme devrait afficher une grille pointillée :



## Question 9 (2 points)

Dans le programme `pygame` ci-dessus, quelle proposition choisissez-vous pour la première instruction manquante ? *(les → indiquent le niveau d'indentation)*

**A)** → → → `running = False`

**B)** → → → `points = []`

**C)** → → → `running = True`

**D)** → → → `pygame.QUIT`

**E)** → → → `points = [(x, y)]`

**F)** → → → `points = tuple()`

**G)** → → → `event.exit()`

**H)** → → → `break`

**I)** → → → `points.append(event.pos)`

**J)** → → → `print('.')`

## Question 10 (3 points)

Dans le programme `pygame` ci-dessus, quelle proposition choisissez-vous pour la deuxième instruction manquante ? *(les → indiquent le niveau d'indentation)*

**A)** → → →
```
if i % (major * minor) == 0 and j % (major * minor) == 0:
```

**B)** → → →
```
if i % minor == 0 and j % minor == 0:
```

**C)** → → →
```
if i % major == 0 or j % major == 0:
```

**D)** → → →
```
if i % (major * minor) == 0 or j % (major * minor) == 0:
```

**E)** → → →
```
if i % major == 0 and j % minor == 0:
```

**F)** → → →
```
if i % minor == 0 or j % major == 0:
```

**G)** → → →
```
if i % (major / minor) == 0 or j % (major / minor) == 0:
```

**H)** → → →
```
if i % minor == 0 or j % minor == 0:
```

**I)** → → →
```
if i % major == 0 and j % major == 0:
```

**J)** → → →
```
if i % (major / minor) == 0 and j % (major / minor) == 0:
```

**Brouillon**