BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2023

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 9 pages numérotées de 1/9 à 9/9.

Le sujet est composé de trois exercices indépendants.

Chaque exercice est sur 4 points. Vous pouvez les traiter dans l'ordre que vous souhaitez.

Exercice 1 - *récursivité* 4 points

Cet exercice est consacré à l'analyse et à l'écriture de programmes récursifs.

1.

a. Expliquer en quelques mots ce qu'est une fonction récursive.

b. On considère la fonction Python suivante :

Numéro de lignes	Fonction `compte_rebours`

1
2
3
4
5
	`def compte_rebours(n):`
	`    """ n est un entier positif ou nul """`
	`    if n >= 0:`
	`        print(n)`
	`        compte_rebours(n - 1)`

L'appel `compte_rebours(3)` affiche successivement les nombres 3, 2, 1 et 0. Expliquer pourquoi le programme s'arrête après l'affichage du nombre 0.

2. En mathématiques, la factorielle d'un entier naturel *n* est le produit des nombres entiers strictement positifs inférieurs ou égaux à *n*. Par convention, la factorielle de 0 est 1. Par exemple :

- la factorielle de 1 est 1
- la factorielle de 2 est 2 x 1 =2
- la factorielle de 3 est 3 x 2 x 1 = 6
- la factorielle de 4 est 4 x 3 x 2 x 1 = 24…

Recopier et compléter sur votre copie le programme donné ci-dessous afin que la fonction récursive `fact` renvoie la factorielle de l'entier passé en paramètre de cette fonction.

Exemple : `fact(4)` renvoie 24.

Numéro de lignes	Fonction `fact`

1
2
3
4
5
6
7
	`def fact(n):`
	`    """ Renvoie le produit des nombres entiers`
	`    strictement positifs inférieurs à n """`
	`    if n == 0:`
	`        return` ***`à compléter`***
	`    else:`
	`        return` ***`à compléter`***

3. La fonction `somme_entiers_rec` ci-dessous permet de calculer la somme des entiers, de `0` à l'entier naturel `n` passé en paramètre.

Par exemple :

- Pour `n` = 0, la fonction renvoie la valeur 0.
- Pour `n` = 1, la fonction renvoie la valeur 0 + 1 = 1.

...

- Pour `n` = 4, la fonction renvoie la valeur 0 + 1 + 2 + 3 + 4 = 10.

Numéro de lignes	Fonction `somme_entiers_rec`

1
2
3
4
5
6
7
8
	`def somme_entiers_rec(n):`
	`    """ Permet de calculer la somme des entiers,`
	`    de 0 à l'entier naturel n """`
	`    if n == 0:`
	`        return 0`
	`    else:`
	`        print(n) #pour vérification`
	`        return n + somme_entiers_rec(n - 1)`

L'instruction `print(n)` de la ligne 7 dans le code précédent a été insérée afin de mettre en évidence le mécanisme en œuvre au niveau des appels récursifs.

a. Écrire ce qui sera affiché dans la console après l'exécution de la ligne suivante :

```
res = somme_entiers_rec(3)
```

b. Quelle valeur sera alors affectée à la variable `res` ?

4. Écrire en Python une fonction `somme_entiers` non récursive : cette fonction devra prendre en argument un entier naturel `n` et renvoyer la somme des entiers de 0 à `n` compris. Elle devra donc renvoyer le même résultat que la fonction `somme_entiers_rec` définie à la question 3.

Exemple : `somme_entiers(4)` renvoie 10.

Exercice 2 - *arbres binaire de recherche* 4 points

Cet exercice est consacré aux arbres binaires de recherche et à la notion d'objet.

1. Voici la définition d'une classe nommée `ArbreBinaire`, en Python :

Numéro de lignes	Classe `ArbreBinaire`
1	`class ArbreBinaire:`
2	`    """ Construit un arbre binaire """`
3	`    def __init__(self, valeur):`
4	`        """ Crée une instance correspondant`
5	`        à un état initial """`
6	`        self.valeur = valeur`
7	`        self.enfant_gauche = None`
8	`        self.enfant_droit = None`
9	`    def insert_gauche(self, valeur):`
10	`        """ Insère le paramètre valeur`
11	`        comme fils gauche """`
12	`        if self.enfant_gauche is None:`
13	`            self.enfant_gauche = ArbreBinaire(valeur)`
14	`        else:`
15	`            new_node = ArbreBinaire(valeur)`
16	`            new_node.enfant_gauche = self.enfant_gauche`
17	`            self.enfant_gauche = new_node`
18	`    def insert_droit(self, valeur):`
19	`        """ Insère le paramètre valeur`
20	`        comme fils droit """`
21	`        if self.enfant_droit is None:`
22	`            self.enfant_droit = ArbreBinaire(valeur)`
23	`        else:`
24	`            new_node = ArbreBinaire(valeur)`
25	`            new_node.enfant_droit = self.enfant_droit`
26	`            self.enfant_droit = new_node`
27	`    def get_valeur(self):`
28	`        """ Renvoie la valeur de la racine """`
29	`        return self.valeur`
30	`    def get_gauche(self):`
31	`        """ Renvoie le sous arbre gauche """`
32	`        return self.enfant_gauche`
33	`    def get_droit(self):`
34	`        """ Renvoie le sous arbre droit """`
35	`        return self.enfant_droit`

a. En utilisant la classe définie ci-dessus, donner un exemple d'attribut, puis un exemple de méthode.

b. Après avoir défini la classe `ArbreBinaire`, on exécute les instructions Python suivantes :

```
r = ArbreBinaire(15)
r.insert_gauche(6)
r.insert_droit(18)
a = r.get_valeur()
b = r.get_gauche()
c = b.get_valeur()
```

Donner les valeurs associées aux variables `a` et `c` après l'exécution de ce code.

On utilise maintenant la classe `ArbreBinaire` pour implémenter un arbre binaire de recherche.

On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel :

- on peut comparer les valeurs des nœuds : ce sont par exemple des nombres entiers, ou des lettres de l'alphabet.
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre gauche de `x`, alors il faut que `y.valeur <= x.valeur`.
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre droit de `x`, alors il faut que `y.valeur => x.valeur`.

2. On exécute le code Python suivant. Représenter graphiquement l'arbre ainsi obtenu.

```
racine_r = ArbreBinaire(15)
racine_r.insert_gauche(6)
racine r.insert droit(18)

r_6 = racine_r.get_gauche()
r_6.insert_gauche(3)
r 6.insert droit(7)

r_18 = racine_r.get_droit()
r_18.insert_gauche(17)
r 18.insert droit(20)

r_3 = r_6.get_gauche()
r_3.insert_gauche(2)
```

3. On a représenté sur la figure 1 ci-dessous un arbre. Justifier qu'il ne s'agit pas d'un arbre binaire de recherche. Redessiner cet arbre sur votre copie en conservant l'ensemble des valeurs {2,3,5,10,11,12,13} pour les nœuds afin qu'il devienne un arbre binaire de recherche.

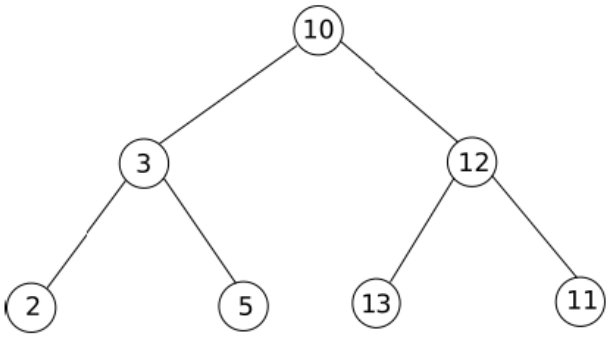


Figure 1

4. On considère qu'on a implémenté un objet `ArbreBinaire` nommé A représenté sur la figure 2.

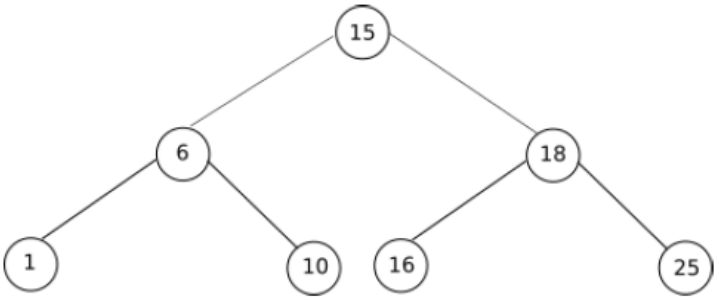


Figure 2

On définit la fonction `parcours_infixe` suivante, qui prend en paramètre un objet `ArbreBinaire T` et un second paramètre `parcours` de type liste.

Numéro de lignes	Fonction `parcours_infixe`
1	`def parcours_infixe(T, parcours):`
2	`    """ Affiche la liste des valeurs de l'arbre """`
3	`    if T is not None:`
4	`        parcours_infixe(T.get_gauche(), parcours)`
5	`        parcours.append(T.get_valeur())`
6	`        parcours_infixe(T.get_droit(), parcours)`
7	`    return parcours`

Donner la liste renvoyée par l'appel suivant : `parcours_infixe(A, [ ])`.

Exercice 3 - *programmation objet - file* 4 points

Cet exercice porte sur les structures de données (files et la programmation objet en langage python)

Un supermarché met en place un système de passage automatique en caisse.
Un client scanne les articles à l'aide d'un scanner de code-barres au fur et à mesure qu'il les ajoute dans son panier.
Les articles s'enregistrent alors dans une structure de données.

La structure de données utilisée est une file définie par la classe `Panier`, avec les primitives habituelles sur la structure de file.
Pour faciliter la lecture, le code de la classe `Panier` n'est pas écrit.

```
class Panier():
  def __init__(self):
     """Initialise la file comme une file vide."""

  def est_vide(self):
     """Renvoie True si la file est vide, False sinon."""

  def enfiler(self, e):
     """Ajoute l'élément e en dernière position de la file,
    ne renvoie rien."""

  def defiler(self):
     """Retire le premier élément de la file et le renvoie."""
```

Le panier d'un client sera représenté par une file contenant les articles scannés.
Les articles sont représentés par des tuples `(code_barre, designation, prix, horaire_scan)` où

- `code_barre` est un nombre entier identifiant l'article ;
- `designation` est une chaine de caractères qui pourra être affichée sur le ticket de caisse ;
- `prix` est un nombre décimal donnant le prix d'une unité de cet article ;
- `horaire_scan` est un nombre entier de secondes permettant de connaitre l'heure où l'article a été scanné.

1. On souhaite ajouter un article dont le tuple est le suivant `(31002, "café noir", 1.50, 50525)`.
   Ecrire le code utilisant une des quatre méthodes ci-dessus permettant d'ajouter l'article à l'objet de classe `Panier` appelé `panier1`.

2. On souhaite définir une **méthode** `remplir(panier_temp)` dans la classe `Panier` permettant de remplir la file avec tout le contenu d'un autre panier `panier_temp` qui est un objet de type `Panier`.

Recopier et compléter le code de la méthode `remplir` en remplaçant chaque `.....` par la primitive de file qui convient.

```
def remplir(self, panier_temp):
    while not panier_temp.....:
        article = panier_temp.....
        self.....(article)
```

3. Pour que le client puisse connaitre à tout moment le montant de son panier, on souhaite ajouter une **méthode** `prix_total()` à la classe `Panier` qui renvoie la somme des prix de tous les articles présents dans le panier.
Ecrire le code de la méthode `prix_total`. **Attention, après l'appel de cette méthode, le panier devra toujours contenir ses articles**.

4. Le magasin souhaite connaitre pour chaque client la durée des achats. Cette durée sera obtenue en faisant la différence entre le champ `horaire_scan` du dernier article scanné et le champ `horaire_scan` du premier article scanné dans le panier du client. Un panier vide renverra une durée égale à zéro. On pourra accepter que le panier soit vide après l'appel de cette méthode.
Ecrire une **méthode** `duree_courses` de la classe `Panier` qui renvoie cette durée.

CORRIGE

Exercice 1 - *récursivité* 4 points

1.
   a. Une fonction est dite récursive si cette fonction s'appelle elle-même
   b. Cette fonction s'arrête quand n est égal à -1. En effet, quand n = -1, n >= 0 devient False, on ne "rentre plus dans le if", les appels récursifs cessent.

2.
```
def fact(n) :
    if n == 0:
        return 1
    else :
        return n*fact(n-1)
```

3.
   a. Après l'exécution de `res = somme_entiers_rec(3)` dans la console, on obtient l'affichage suivant :
   3
   2
   1
   b.
   La valeur affectée à la variable res est 6 (3+2+1 = 6)

4.
```
def somme_entiers(n) :
    somme = 0
    while n > 0:
        somme = somme + n
        n = n - 1
    return somme
```

Exercice 2 - *arbres binaire de recherche* 4 points

1.

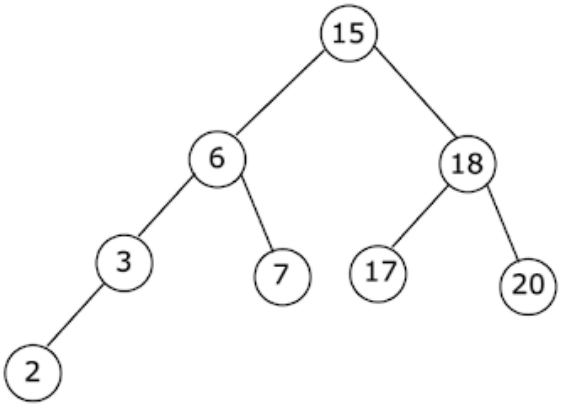
a.

Exemple d'attribut : `enfant_gauche`
Exemple de méthode : insert_gauche()

b.

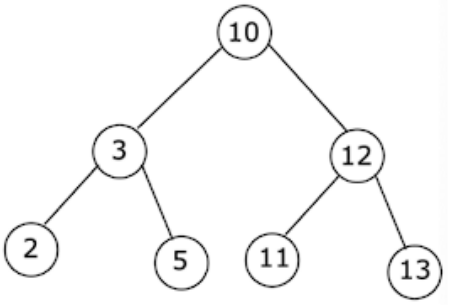
Nous avons a = 15 et c = 6

2.



3.

Nous avons 11 qui est à droite 12



4.

On obtient le tableau (liste Python) suivant : [1, 6, 10, 15, 16, 18, 25]

Exercice 3 - *programmation objet - file* 4 points

1.
```
panier1.enfiler((31002, "café noir", 1.50, 50525))
```
2.
```
def remplir(self, panier_temp):
        while not panier_temp.est_vide() :
            article = panier_temp.defiler()
            self.enfiler(article)
```
3.
```
def prix_total(self):
        p_temp = Panier()
        montant = 0
        while not self.est_vide() :
            article = self.defiler()
            montant = montant + article[2]
            p_temp.enfiler(article)
        while not p_temp.est_vide() :
            article = p_temp.defiler()
            self.enfiler(article)
        return montant
```
4.
```
def horaire_scan(self):
        if self.est_vide():
            return 0
        premier_article = self.defiler()[3]
        dernier_article = premier_article
        while not self.est_vide() :
            dernier_article = self.defiler()[3]
        return dernier_article - premier_article
```