2022, Polynésie

BACCALAURÉAT GÉNÉRAL, NSI, 2022, Polynésie

Durée de l'épreuve : 3 heures 30
L'usage de la calculatrice avec mode examen actif est autorisé.
L'usage de la calculatrice sans mémoire, « type collège » est autorisé.

Le candidat traite au choix 3 exercices parmi les 5 exercices proposés.

Sujet non officiel

Ce n'est pas le sujet officiel, quelques modifications mineures ont été apportées.

Le sujet d'origine est disponible en téléchargement.

EXERCICE 1 (4 points)

> Cet exercice traite du thème « programmation », et principalement de la récursivité.

On rappelle qu'une chaine de caractères peut être représentée en Python par un texte entre guillemets `""` et que :

- la fonction `len` renvoie la longueur de la chaine de caractères passée en paramètre ;
- si une variable `ch` désigne une chaine de caractères, alors `ch[0]` renvoie son premier caractère, `ch[1]` le deuxième, etc ;
- l'opérateur `+` permet de concaténer deux chaines de caractères.

Exemples

Console Python

```
>>> texte = "bricot"
>>> len(texte)
6
>>> texte[0]
"b"
>>> texte[1]
"r"
>>> "a" + texte
"abricot"
```

On s'intéresse dans cet exercice à la construction de chaines de caractères suivant certaines règles de construction.

Règle A

Une chaine est construite suivant la règle A dans les deux cas suivants :

- soit elle est égale à `"a"` ;
- soit elle est de la forme `"a" + chaine + "a"`, où `chaine` est une chaine de caractères construite suivant la règle A.

Règle B

Une chaine est construite suivant la règle B dans les deux cas suivants :

- soit elle est de la forme `"b" + chaine + "b"`, où `chaine` est une chaine de caractères construite suivant la règle A ;
- soit elle est de la forme `"b" + chaine + "b"`, où `chaine` est une chaine de caractères construite suivant la règle B.

Fonction `choice` du module `random`

On a reproduit ci-dessous l'aide de la fonction `choice` du module `random`.

Console Python

```
>>> from random import choice
>>> help(choice)
Help on method choice in module random:
choice(seq) method of random.Random instance
Choose a random element from a non-empty sequence.
```

La fonction `A` ci-dessous renvoie une chaine de caractères construite suivant la règle A, en choisissant aléatoirement entre les deux cas de figure de cette règle.

Script Python

```
def A():
    if choice([True, False]):
        return "a"
    else:
        return "a" + A() + "a"
```

1.a) Cette fonction est-elle récursive ? Justifier.

Réponse

La fonction `A` s'appelle elle-même, donc `A` est une fonction récursive.

1.b) La fonction `choice([True, False])` peut renvoyer `False` un très grand nombre de fois consécutives. Expliquer pourquoi ce cas de figure amènerait à une erreur d'exécution.

Réponse

Si `choice([True, False])` renvoie `False` consécutivement un nombre de fois supérieur à la limite de profondeur de récursion autorisée (1000 par défaut avec Python), dans ce cas une erreur d'exécution se produit.

Pour aller plus loin

On pourrait modifier cette limite à 10^6 avec le code suivant

Script Python

```
import sys
sys.setrecursionlimit(10**6)
```

Dans la suite, on considère une deuxième version de la fonction `A`. À présent, la fonction prend en paramètre un entier `n` tel que,

- si la valeur de `n` est négative ou nulle, la fonction renvoie `"a"` ;
- si la valeur de `n` est strictement positive, elle renvoie une chaine de caractères construite suivant la règle A avec un `n` décrémenté de 1, en choisissant aléatoirement entre les deux cas de figure de cette règle.

Script Python

```
def A(n):
    if ... or choice([True, False]) :
        return "a"
    else:
        return "a" + ... + "a"
```

2.a) Recopier sur la copie et compléter aux emplacements des points de suspension `...` le code de cette nouvelle fonction `A`.

Réponse

Script Python

```
def A(n):
    if n <= 0 or choice([True, False]) :
        return "a"
    else:
        return "a" + A(n - 1) + "a"
```

2.b) Justifier le fait qu'un appel de la forme `A(n)` avec `n` un nombre entier positif inférieur à 50, termine toujours.

Réponse

Pour $n > 0$, l'appel à `A(n)` provoque ou bien un arrêt de la fonction, ou bien un appel récursif avec le paramètre `n - 1`.

Un appel à `A(50)` pourrait provoquer dans le pire des cas 50 appels récursifs pour arriver à `A(0)` qui termine, ou alors terminer avant !

On donne ci-après le code de la fonction récursive B qui prend en paramètre un entier `n` et qui renvoie une chaine de caractères construite suivant la règle B.

Script Python

```python
def B(n):
    if n <= 0 or choice([True, False]):
        return "b" + A(n - 1) + "b"
    else:
        return "b" + B(n - 1) + "b"
```

On admet que :

- les appels `A(-1)` et `A(0)` renvoient la chaine `"a"` ;
- l'appel `A(1)` renvoie la chaine `"a"` ou la chaine `"aaa"` ;
- l'appel `A(2)` renvoie la chaine `"a"`, la chaine `"aaa"` ou la chaine `"aaaaa"`.

3. Donner toutes les chaines possibles renvoyées par les appels `B(0)`, `B(1)` et `B(2)`.

Réponse

- `B(0)` renvoie `"bab"`
- `B(1)` renvoie `"bab"` ou `"bbabb"`.
- `B(2)` renvoie `"bab"`, `"baaab"`, `"bbabb"` ou `"bbbabbb"`.

On suppose maintenant qu'on dispose d'une fonction `raccourcir` qui prend comme paramètre une chaine de caractères de longueur supérieure ou égale à 2, et renvoie la chaine de caractères obtenue à partir de la chaine initiale en lui ôtant le premier et le dernier caractère.

Par exemple :

Console Python

```
>>> raccourcir("abricot")
"brico"
>>> raccourcir("ab")
""
```

4.a) Recopier sur la copie et compléter les points de suspension `...` du code de la fonction `regle_A` ci-dessous pour qu'elle renvoie `True` si la chaine passée en paramètre est construite suivant la règle A, et `False` sinon.

Script Python

```python
def regle_A(chaine):
    n = len(chaine)
    if n >= 2:
        return chaine[0] == "a" and chaine[n - 1] == "a" and regle_A(...)
    else:
        return chaine == ...
```

Réponse

Script Python

```python
def regle_A(chaine):
    n = len(chaine)
    if n >= 2:
        return chaine[0] == "a" and chaine[n - 1] == "a" and regle_A(raccourcir(chaine))
    else:
        return chaine == "a"
```

4.b) Écrire le code d'une fonction `regle_B`, prenant en paramètre une chaine de caractères et renvoyant `True` si la chaine est construite suivant la règle B, et `False` sinon.

Réponse

Script Python

```
def regle_B(chaine):
    n = len(chaine)
    if n >= 2:
        return chaine[0] == "b" and chaine[n - 1] == "b" and (
            regle_A(raccourcir(chaine)) or regle_B(raccourcir(chaine))
        )
    else:
        return False
```

EXERCICE 2 (4 points)

> Cet exercice traite du thème « architecture matérielle », et principalement d'ordonnancement et d'expressions booléennes.

Un système est composé de 4 périphériques, numérotés de 0 à 3, et d'une mémoire, reliés entre eux par un bus auquel est également connecté un dispositif ordonnanceur. À l'aide d'un signal spécifique envoyé sur le bus, l'ordonnanceur sollicite à tour de rôle les périphériques pour qu'ils indiquent le type d'opération (lecture ou écriture) qu'ils souhaitent effectuer, et l'adresse mémoire concernée.

Un tour a lieu quand les 4 périphériques ont été sollicités. **Au début d'un nouveau tour, on considère que toutes les adresses sont disponibles en lecture et écriture.**

Si un périphérique demande l'écriture à une adresse mémoire **à laquelle on n'a pas encore accédé** pendant le tour, l'ordonnanceur répond `"OK"` et l'écriture a lieu. Si on a déjà demandé la lecture ou l'écriture à cette adresse, l'ordonnanceur répond `"ATT"` et l'opération n'a pas lieu.

Si un périphérique demande la lecture à une adresse à laquelle on n'a pas encore accédé en écriture pendant le tour, l'ordonnanceur répond `"OK"` et la lecture a lieu. Plusieurs lectures peuvent avoir donc lieu pendant le même tour à la même adresse.

Si un périphérique demande la lecture à une adresse à laquelle on a déjà accédé en écriture, l'ordonnanceur répond `"ATT"` et la lecture n'a pas lieu.

Ainsi, pendant un tour, une adresse peut être utilisée soit une seule fois en écriture, soit autant de fois qu'on veut en lecture, soit pas utilisée.

Si un périphérique ne peut pas effectuer une opération à une adresse, il demande la même opération à la même adresse au tour suivant.

1. Le tableau donné en annexe 1 indique, sur chaque ligne, le périphérique sélectionné, l'adresse à laquelle il souhaite accéder et l'opération à effectuer sur cette adresse. Compléter dans la dernière colonne de cette annexe, à rendre avec la copie, la réponse donnée par l'ordonnanceur pour chaque opération.

Annexe 1

N° périphérique	Adresse	Opération	Réponse de l'ordonnanceur
0	`10`	écriture	`"OK"`
1	`11`	lecture	`"OK"`
2	`10`	lecture	`"ATT"`
3	`10`	écriture	`"ATT"`
0	`12`	lecture	
1	`10`	lecture	
2	`10`	lecture	
3	`10`	écriture	

Réponse

N° périphérique	Adresse	Opération	Réponse de l'ordonnanceur
0	`10`	écriture	`"OK"`
1	`11`	lecture	`"OK"`
2	`10`	lecture	`"ATT"`
3	`10`	écriture	`"ATT"`
0	`12`	lecture	`"OK"`
1	`10`	lecture	`"OK"`
2	`10`	lecture	`"OK"`
3	`10`	écriture	`"ATT"`

Il s'agit d'un nouveau tour, les lectures sont possibles, la première écriture ne l'est pas, on a déjà accéder en lecture pendant le tour à l'adresse demandée.

On suppose dans toute la suite que :

- le périphérique 0 écrit systématiquement à l'adresse `10` ;
- le périphérique 1 lit systématiquement à l'adresse `10` ;
- le périphérique 2 écrit alternativement aux adresses `11` et `12` ;
- le périphérique 3 lit alternativement aux adresses `11` et `12` ;

Pour les périphériques 2 et 3, le changement d'adresse n'est effectif que lorsque l'opération est réalisée.

2. On suppose que les périphériques sont sélectionnés à chaque tour dans l'ordre 0 ; 1 ; 2 ; 3. Expliquer ce qu'il se passe pour le périphérique 1.

Réponse

- À chaque début de tour, le périphérique 0 demande à écrire à l'adresse `10` ; c'est accepté.
- Juste après, le périphérique 1 demande à lire à l'adresse `10` ; c'est refusé.

Le périphérique 1 ne pourra jamais lire l'adresse `10`.

Les périphériques sont sollicités de la manière suivante lors de quatre tours successifs :

- au premier tour, ils sont sollicités dans l'ordre 0 ; 1 ; 2 ; 3 ;
- au deuxième tour, dans l'ordre 1 ; 2 ; 3 ; 0 ;
- au troisième tour, 2 ; 3 ; 0 ; 1 ;
- puis 3 ; 0 ; 1 ; 2 au dernier tour.
- Et on recommence...

3.a) Préciser pour chacun de ces tours si le périphérique 0 peut écrire et si le périphérique 1 peut lire.

Réponse

- **Tour 1** : 0 ; 1 ; 2 ; 3
  - 0 peut écrire, puis
  - 1 ne peut pas lire
- **Tour 2** : 1 ; 2 ; 3 ; 0
  - 1 peut lire, puis
  - 0 ne peut pas écrire
- **Tour 3** : 2 ; 3 ; 0 ; 1
  - 0 peut écrire, puis
  - 1 ne peut pas lire
- **Tour 4** : 3 ; 0 ; 1 ; 2
  - 0 peut écrire, puis
  - 1 ne peut pas lire

3.b) En déduire la proportion des valeurs écrites par le périphérique 0 qui sont effectivement lues par le périphérique 1.

Réponse

- Au tour 1, la valeur écrite par le périphérique 0 sera lue par le périphérique 1 au tour suivant.
- Au tour 2, rien n'est écrit par le périphérique 0.
- Au tour 3, la valeur écrite par le périphérique 0 **ne sera jamais** lue par le périphérique 1 ; en effet, une autre écriture intervient avant la prochaine lecture.
- Au tour 4, la valeur écrite par le périphérique 0 **ne sera jamais** lue par le périphérique 1 ; en effet, une autre écriture intervient avant la prochaine lecture.

Ainsi, une seule valeur sur trois sera effectivement lue. La proportion est $\frac{1}{3}$.

On change la méthode d'ordonnancement : on détermine l'ordre des périphériques au cours d'un tour à l'aide de deux listes

d'attente `ATT_L` et `ATT_E` établies au tour précédent.

Au cours d'un tour, on place dans la liste `ATT_L` toutes les opérations de lecture mises en attente, et dans la liste d'attente `ATT_E` toutes les opérations d'écriture mises en attente.

Au début du tour suivant, on établit l'ordre d'interrogation des périphériques en procédant ainsi :

- on interroge ceux présents dans la liste `ATT_L`, par ordre croissant d'adresse,
- on interroge ensuite ceux présents dans la liste `ATT_E`, par ordre croissant d'adresse,
- puis on interroge les périphériques restants, par ordre croissant d'adresse.

4. Compléter et rendre avec la copie le tableau fourni en annexe 2, en utilisant l'ordonnancement décrit ci-dessus, sur 3 tours.

Annexe 2

Tour	N° périphérique	Adresse	Opération	Réponse ordonnanceur	`ATT_L`	`ATT_E`
1	0	`10`	écriture	`"OK"`	vide	vide
1	1	`10`	lecture	`"ATT"`	`(1, 10)`	vide
1	2	`11`	écriture			
1	3	`11`	lecture			
2	1	`10`	lecture			vide
2						
2						
2						
3	0	`10`	écriture		vide	vide
3	1	`10`	lecture			vide
3	2	`11`	écriture	`"OK"`	`(1, 10)`	vide
3	3	`12`	lecture			

Réponse

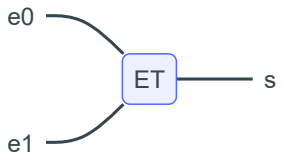
Tour	N° périphérique	Adresse	Opération	Réponse ordonnanceur	`ATT_L`	`ATT_E`
1	0	`10`	écriture	`"OK"`	vide	vide
1	1	`10`	lecture	`"ATT"`	`(1, 10)`	vide
1	2	`11`	écriture	`"OK"`	`(1, 10)`	vide
1	3	`11`	lecture	`"ATT"`	`(1, 10), (3, 11)`	vide
2	1	`10`	lecture	`"OK"`	`(3, 11)`	vide
2	3	`11`	lecture	`"OK"`	vide	vide
2	0	`10`	écriture	`"ATT"`	vide	`(0, 10)`
2	2	`12`	écriture	`"OK"`	vide	`(0, 10)`
3	0	`10`	écriture	`"OK"`	vide	vide
3	1	`10`	lecture	`"ATT"`	`(1, 10)`	vide
3	2	`11`	écriture	`"OK"`	`(1, 10)`	vide
3	3	`12`	lecture	`"OK"`	`(1, 10)`	vide

Les colonnes **e0** et **e1** du tableau suivant recensent les deux chiffres de l'écriture binaire de l'entier **n** de la première colonne.

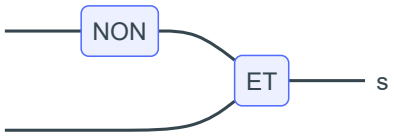
nombre n	écriture binaire de n sur deux bits	e1	e0
0	00	0	0
1	01	0	1
2	10	1	0
3	11	1	1

L'ordonnanceur attribue à deux signaux sur le bus de données les valeurs de **e0** et **e1** associées au numéro du circuit qu'il veut sélectionner. On souhaite construire à l'aide des portes ET, OU et NON un circuit pour chaque périphérique.
Chacun des quatre circuits à construire prend en entrée deux signaux **e0** et **e1**, le signal de sortie **s** valant 1 uniquement lorsque les niveaux de **e0** et **e1** correspondent aux bits de l'écriture en binaire du numéro du périphérique correspondant.

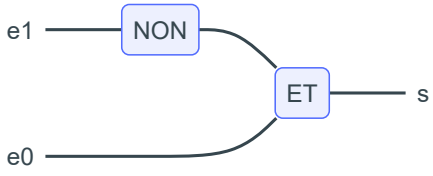
Par exemple, le circuit ci-dessous réalise la sélection du périphérique 3. En effet, le signal **s** vaut 1 si et seulement si **e0** et **e1** valent tous les deux 1.



5.a) Recopier sur la copie et indiquer dans le circuit ci-dessous les entrées **e0** et **e1** de façon que ce circuit sélectionne le périphérique 1.

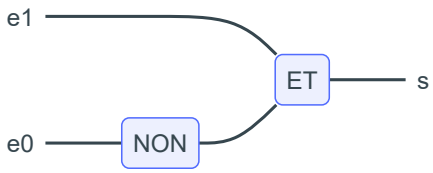


Réponse



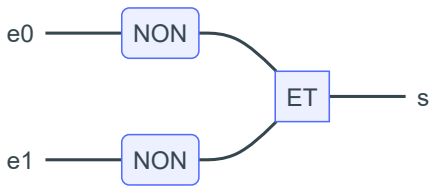
5.b) Dessiner un circuit constitué d'une porte ET et d'une porte NON, qui sélectionne le périphérique 2.

Réponse



5.c) Dessiner un circuit permettant de sélectionner le périphérique 0.

Réponse



EXERCICE 3 (4 points)

> Cet exercice traite du thème « base de données », et principalement du modèle relationnel et du langage SQL.

L'énoncé de cet exercice peut utiliser les mots du langage SQL suivants :

```
CREATE TABLE, SELECT, FROM, WHERE, JOIN ON, INSERT INTO, VALUES, UPDATE, SET, DELETE, COUNT, DISTINCT, AND, OR, AS,
```

`ORDER BY, ASC, DESC`

Un site web recueille des données de navigation dans une base de données afin d'étudier les profils de ses visiteurs. Chaque requête d'interrogation d'une page de ce site est enregistrée dans une première table dénommée **Visites** sous la forme d'un 5-uplet : `(identifiant, adresse IP, date et heure de visite, nom de la page, navigateur)`.

Le chargement de la page `index.html` par `192.168.1.91` le 12 juillet 1998 à 22 h 48 aura par exemple été enregistré de la façon suivante :

`(1534, "192.168.1.91", "1998-07-12 22:48:00", "index.html", "Internet explorer 4.1")`.

La commande SQL ayant permis de créer cette table est la suivante :

SQL

```sql
CREATE TABLE Visites (
    identifiant INTEGER NOT NULL UNIQUE,
    ip VARCHAR(15),
    dateheure DATETIME,
    nompage TEXT,
    navigateur TEXT
);
```

1.a) Donner une commande d'interrogation en langage SQL permettant d'obtenir l'ensemble des 2-uplets `(adresse IP, nom de la page)` de cette table.

Réponse

SQL

```sql
SELECT ip, nompage FROM Visites;
```

1.b) Donner une commande en langage SQL permettant d'obtenir l'ensemble des adresses IP ayant interrogé le site, sans doublon.

Réponse

SQL

```sql
SELECT DISTINCT ip FROM Visites;
```

1.c) Donner une commande en langage SQL permettant d'obtenir la liste des noms des pages visitées par l'adresse IP `192.168.1.91`

Réponse

SQL

```sql
SELECT nompage FROM Visites WHERE ip = '192.168.1.91';
```

Ce site web met en place, sur chacune de ses pages, un programme en JavaScript qui envoie au serveur, à intervalle régulier de 15 secondes, le temps en secondes de présence sur la page. Ces envois contiennent tous la valeur de `identifiant` correspondant au chargement initial de la page.
Par exemple, si le visiteur du 12 juillet 1998 est resté 65 secondes sur la page, celle-ci a envoyé au serveur les 4 doublets `(1534, 15)`, `(1534, 30)`, `(1534, 45)` et `(1534, 60)`.

Ces données sont enregistrées dans une table nommée `Pings` créée avec la commande ci-dessous :

SQL

```sql
CREATE TABLE Pings (
    identifiant INTEGER,
    duree INTEGER
);
```

En plus de l'inscription d'une ligne dans la table `Visites`, chaque chargement d'une nouvelle page provoque l'insertion d'une ligne dans la table `Pings` comprenant l'identifiant de ce chargement et une durée de 0.

Les attributs `identifiant` des tables `Visites` et `Pings` partagent les mêmes valeurs.

2.a) De quelle table l'attribut `identifiant` est-il la clé primaire ?

Réponse

L'attribut `identifiant` est la clé primaire de la table `Visites`.

La table `Pings` a une clé primaire composite constituée de `identifiant` et de `duree`.

2.b) De quelle table l'attribut `identifiant` est-il une clé étrangère ?

Réponse

L'attribut `identifiant` est clé étrangère dans la table `Pings`.

2.c) Par conséquent, quelles vérifications sont automatiquement effectuées par le système de gestion de base de données ?

Réponse

Lors d'un enregistrement dans la table `Pings`, le système de gestion de base de données va vérifier :

- la contrainte d'unicité : si la clé `identifiant` est effectivement présente en tant que clé primaire dans la table `Visites` ;
- la contrainte d'inclusion : les valeurs de l'attribut de la clé étrangère doivent être incluses dans le domaine des valeurs de la clé primaire.

3. Le serveur reçoit le doublet `(identifiant, duree)` suivant : `(1534, 105)`. Écrire la commande SQL d'insertion qui permet d'ajouter cet enregistrement à la table `Pings`.

Réponse

SQL

```sql
INSERT INTO Pings VALUES (1534, 105);
```

On envisage ensuite d'optimiser la table en se contentant d'une seule ligne par identifiant dans la table `Pings` : les valeurs de l'attribut `duree` devraient alors être mises à jour à chaque réception d'un nouveau doublet `(identifiant, duree)`.

4.a) Écrire la requête de mise à jour permettant de fixer à 120 la valeur de l'attribut `duree` associée à l'identifiant 1534 dans la table `Pings`.

Réponse

SQL

```sql
UPDATE Pings SET duree = 120 WHERE identifiant = 1534;
```

Remarque : Dans le cadre de cette optimisation, l'attribut `identifiant` est clé primaire.

4.b) Expliquer pourquoi on ne peut pas être certain que les données envoyées par une page web, depuis le navigateur d'un client, via plusieurs requêtes formulées en JavaScript, arrivent au serveur dans l'ordre dans lequel elles ont été émises.

Réponse

Les paquets IP sont routés indépendamment et peuvent donc éventuellement suivre des chemins différents s'il existe plusieurs itinéraires disponibles. Les paquets émis dans un certain ordre peuvent ainsi être reçus dans un autre ordre par le serveur.

Ainsi, on ne peut pas être certain que les données envoyées par une page web, depuis le navigateur d'un client, via plusieurs requêtes formulées en JavaScript, arrivent au serveur dans l'ordre dans lequel elles ont été émises,

4.c) En déduire qu'il est préférable d'utiliser une requête d'insertion plutôt qu'une requête de mise à jour pour ajouter des données à la table `Pings`.

Réponse

Il est préférable d'utiliser une requête d'insertion plutôt qu'une requête de mise à jour pour ajouter des données à la table `Pings` afin de s'assurer de conserver dans la table `Ping` un enregistrement représentatif du temps passé par un utilisateur sur une page donnée (dans ce cas, lors d'une requête sur la table `Pings`, il faudrait alors rechercher la valeur maximale relative à un identifiant de connexion donné).

5. Écrire une requête SQL utilisant le mot-clé `JOIN` et une clause `WHERE`, permettant de trouver les noms de toutes les pages qui ont été consultées plus d'une minute par au moins un utilisateur.

Réponse

SQL

```sql
SELECT Visites.nompage FROM Visites JOIN Pings ON Visites.identifiant = Pings.identifiant WHERE Pings.duree > 60 ;
```

Remarque : Ici la table `Pings` a à nouveau une clé primaire composite constituée de `identifiant` et de `duree`.

EXERCICE 4 (4 points)

> Cet exercice traite du thème « structures de données », et principalement des piles.

La classe `Pile` utilisée dans cet exercice est implémentée en utilisant des listes Python et propose quatre éléments d'interface :

- Un constructeur qui permet de créer une pile vide, représentée par `[]` ;
- La méthode `est_vide()` qui renvoie `True` si l'objet est une pile ne contenant aucun élément, et `False` sinon ;
- La méthode `empiler` qui prend un objet quelconque en paramètre et ajoute cet objet au sommet de la pile. Dans la représentation de la pile dans la console, cet objet apparait à droite des autres éléments de la pile ;
- La méthode `depiler` qui renvoie l'objet présent au sommet de la pile et le retire de la pile.

Exemples :

Console Python

```
>>> ma_pile = Pile()
>>> ma_pile.empiler(2)
>>> ma_pile
[2]
>>> ma_pile.empiler(3)
>>> ma_pile.empiler(50)
>>> ma_pile
[2, 3, 50]
>>> ma_pile.depiler()
50
>>> ma_pile
[2, 3]
```

La méthode `est_triee` ci-dessous renvoie `True` si, en dépilant tous les éléments, ils sont traités dans l'ordre croissant, et `False` sinon.

Script Python

```
1  def est_triee(self):
2      if not self.est_vide():
3          e1 = self.depiler()
4          while not self.est_vide():
5              e2 = self.depiler()
6              if e1 ... e2 :
7                  return False
8              e1 = ...
9      return True
```

1. Recopier sur la copie les lignes 6 et 8 en complétant les points de suspension.

Réponse

Script Python

```
1  def est_triee(self):
2      if not self.est_vide():
3          e1 = self.depiler()
4          while not self.est_vide():
5              e2 = self.depiler()
6              if e1 > e2 :
7                  return False
8              e1 = e2
9      return True
```

On crée dans la console la pile `A` représentée par `[1, 2, 3, 4]`.

2.a) Donner la valeur renvoyée par l'appel `A.est_triee()`.

Réponse

La valeur 4 est d'abord dépilée, puis 3. L'ordre n'est pas croissant, ainsi `A.est_triee()` renvoie `False`.

2.b) Donner le contenu de la pile `A` après l'exécution de cette instruction.

Réponse

`A` sera représenté par `[1, 2]`.

On souhaite maintenant écrire le code d'une méthode `depile_max` d'une pile non vide ne contenant que des nombres entiers et renvoyant le plus grand élément de cette pile en le retirant de la pile.

Après l'exécution de `p.depile_max()`, le nombre d'éléments de la pile `p` diminue donc de 1.

Script Python

```
1  def depile_max(self):
2      assert not self.est_vide(), "Pile vide"
3      q = Pile()
4      maxi = self.depiler()
5      while not self.est_vide():
6          elt = self.depiler()
7          if maxi < elt:
8              q.empiler(maxi)
9              maxi = ...
10         else :
11             ...
12     while not q.est_vide():
13         self.empiler(q.depiler())
14     return maxi
```

3. Recopier sur la copie les lignes 9 et 11 en complétant les points de suspension.

Réponse

Script Python

```
1  def depile_max(self):
2      assert not self.est_vide(), "Pile vide"
3      q = Pile()
4      maxi = self.depiler()
5      while not self.est_vide():
6          elt = self.depiler()
7          if maxi < elt:
8              q.empiler(maxi)
9              maxi = elt
10         else :
11             q.empiler(elt)
12     while not q.est_vide():
13         self.empiler(q.depiler())
14     return maxi
```

On crée la pile `B` représentée par `[9, -7, 8, 12, 4]` et on effectue l'appel `B.depile_max()`.

4.a) Donner le contenu des piles `B` et `q` à la fin de chaque itération de la boucle `while` de la ligne 5.

Réponse

Initialisation

- `B` contient `[9, -7, 8, 12]` ;
- `q` est vide ;
- `maxi` est égal à `4`.

Juste avant le premier tour de boucle

Fin du tour 1

- `B` contient `[9, -7, 8]` ;
- `q` contient `[4]` ;
- `maxi` est égal à `12`.

Fin du tour 2

- `B` contient `[9, -7]` ;
- `q` contient `[4, 8]` ;
- `maxi` est égal à `12`.

Fin du tour 3

- `B` contient `[9]` ;
- `q` contient `[4, 8, -7]` ;
- `maxi` est égal à `12`.

Fin du tour 4

- `B` est vide ;
- `q` contient `[4, 8, -7, 9]` ;
- `maxi` est égal à `12`.

4.b) Donner le contenu des piles `B` et `q` avant l'exécution de la ligne 14.

Réponse

La dernière boucle renverse la pile `q` dans la pile `B`, ainsi, à la ligne 14 :

- `q` est vide ;
- `B` contient `[9, -7, 8, 4]`.

4.c) Donner un exemple de pile qui montre que l'ordre des éléments restants n'est pas préservé après l'exécution de `depile_max`.

Réponse

Avec une pile `B` qui contient `[3, 1, 2]`

Initialisation

- `B` contient `[3, 1]` ;
- `q` est vide ;
- `maxi` est égal à `2` .

Juste avant le premier tour de boucle

Fin du tour 1

- `B` contient `[3]` ;
- `q` contient `[1]` ;
- `maxi` est égal à `2` .

Fin du tour 2

- `B` est vide ;
- `q` contient `[1, 2]` ;
- `maxi` est égal à `3` .

La dernière boucle renverse la pile `q` dans la pile `B` , ainsi, à la ligne 14 :

- `q` est vide ;
- `B` contient `[2, 1]` .

Sans `3` dans la pile `B` initiale, on a dans l'ordre `[1, 2]` ce qui est différent de `[2, 1]` obtenu ici avec `depile_max` .

On a ainsi un exemple où l'ordre des éléments restants n'est pas préservé après l'exécution de `depile_max` .

On donne le code de la fonction `traiter` :

Script Python

```
1   def traiter(self):
2       q = Pile()
3       while not self.est_vide():
4           q.empiler(self.depile_max())
5       while not q.est_vide():
6           self.empiler(q.depiler())
```

5.a) Donner les contenus successifs des piles `B` et `q`

- avant la ligne 3,
- avant la ligne 5,
- à la fin de l'exécution de la fonction `traiter` lorsque la fonction `traiter` est appelée avec la pile `B` contenant `[1, 6, 4, 3, 7, 2]` .

Réponse

Avec `B = [1, 6, 4, 3, 7, 2]`, un appel `B.traiter()` conduit successivement à :

- Avant la ligne 3,
  - `B` contient `[1, 6, 4, 3, 7, 2]` ;
  - `q` est vide.
- Avant la ligne 5,
  - `B` est vide ;
  - `q` contient `[7, 6, 4, 3, 2, 1]`
- À la fin,
  - `B` contient `[1, 2, 3, 4, 6, 7]`
  - `q` est vide.

5.b) Expliquer le traitement effectué par cette méthode.

Réponse

Ce traitement est un tri de la pile. On construit d'abord `q` comme la pile des éléments de `self` dans l'ordre décroissant. On renverse ensuite la pile, qui se retrouve comme si on avait empilé les éléments de `self` dans l'ordre croissant.

EXERCICE 5 (4 points)

> Cet exercice traite du thème « algorithmique », et principalement des algorithmes sur les arbres binaires.

On manipule ici les arbres binaires avec trois fonctions :

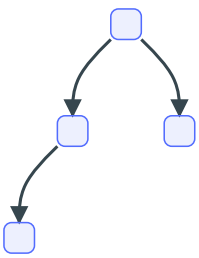
- `est_vide(A)` renvoie `True` si l'arbre binaire `A` est vide, `False` s'il ne l'est pas ;
- `sous_arbre_gauche(A)` renvoie le sous-arbre à gauche de l'arbre binaire `A` ;
- `sous_arbre_droite(A)` renvoie le sous-arbre à droite de l'arbre binaire `A`.

L'arbre binaire renvoyé par les fonctions `sous_arbre_gauche` et `sous_arbre_droite` peut éventuellement être l'arbre vide.

On définit la hauteur d'un arbre binaire non vide de la façon suivante :

- si ses sous-arbres à gauche et à droite sont vides, sa hauteur est 0 ;
- si l'un des deux au moins est non vide, alors sa hauteur est égale à $1 + M$, où M est la plus grande des hauteurs de ses sous-arbres (à gauche et à droite) non vides.

1.a) Donner la hauteur de l'arbre ci-dessous.



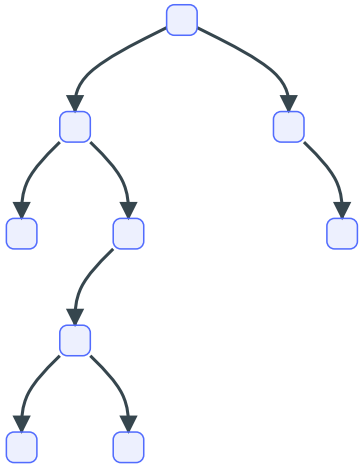
Réponse

Avec cette définition, la hauteur de cet arbre binaire est 2.

1.b) Dessiner sur la copie un arbre binaire de hauteur 4.

Réponse

Avec cette définition, voici un arbre binaire de hauteur 4.



La hauteur d'un arbre est calculée par l'algorithme récursif suivant :

Pseudo Code

```
1  Algorithme hauteur(A) :
2      test d'assertion : A est supposé non vide
3      si sous_arbre_gauche(A) vide et sous_arbre_droite(A) vide :
4          renvoyer 0
5      sinon, si sous_arbre_gauche(A) vide :
6          renvoyer 1 + hauteur(sous_arbre_droite(A))
7      sinon, si ... :
8          renvoyer ...
9      sinon:
10         renvoyer 1 + max(
11             hauteur(sous_arbre_gauche(A)),
12             hauteur(sous_arbre_droite(A))
13         )
```

2. Recopier sur la copie les lignes 7 et 8 en complétant les points de suspension.

Réponse

Pseudo Code

```
Algorithme hauteur(A) :
    test d'assertion : A est supposé non vide
    si sous_arbre_gauche(A) vide et sous_arbre_droite(A) vide :
        renvoyer 0
    sinon, si sous_arbre_gauche(A) vide :
        renvoyer 1 + hauteur(sous_arbre_droite(A))
    sinon, si sous_arbre_droite(A) vide :
        renvoyer 1 + hauteur(sous_arbre_gauche(A))
    sinon:
        renvoyer 1 + max(
            hauteur(sous_arbre_gauche(A)),
            hauteur(sous_arbre_droite(A))
        )
```

Pour aller plus loin

Si on définit la hauteur d'un arbre binaire vide égale à 0, (celle d'un arbre réduit à un seul nœud égal à 1), on a un algorithme bien plus simple, et qui traite tous les cas :

Pseudo Code

```
Algorithme hauteur(A) :
    si A vide :
        renvoyer 0
    sinon:
        renvoyer 1 + max(
            hauteur(sous_arbre_gauche(A)),
            hauteur(sous_arbre_droite(A))
        )
```

On considère un arbre binaire `R` dont on note `G` le sous-arbre à gauche et `D` le sous-arbre à droite. On suppose que `R` est de hauteur 4 et `G` de hauteur 2.

3.a) Justifier le fait que `D` n'est pas l'arbre vide et déterminer sa hauteur.

Réponse

Si `D` est égal à l'arbre vide, alors la hauteur de `R` est égale à `1 + hauteur(G)` qui est égal à $1 + 2 = 3$, or `R` est de hauteur 4. Contradiction.

Ainsi `D` n'est pas l'arbre vide.

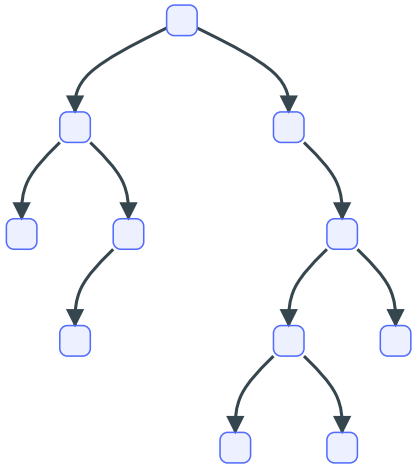
Dans ce cas `1 + max(hauteur(G), hauteur(D))` est égal à 4. D'où

- `1 + max(2, hauteur(D))` est égal à 4.
- `max(2, hauteur(D))` est égal à 3.
- `hauteur(D)` est égal à 3.

3.b) Illustrer cette situation par un dessin.

Réponse

- Cet arbre est de hauteur 4,
- son sous arbre à gauche est de hauteur 2,
- son sous arbre à droite est de hauteur 3.



Soit un arbre binaire non vide de hauteur `h`. On note `n` le nombre de nœuds de cet arbre. On admet que $h+1 \leqslant n \leqslant 2^{h+1}-1$.

4.a) Vérifier ces inégalités sur l'arbre binaire de la question **1.a)**.

Réponse

Dans la question **1.a)**, l'arbre binaire possède $n=4$ nœuds et a une hauteur $h=2$.

On a bien $2+1 \leqslant 4 \leqslant 2^{2+1}-1$ qui s'écrit aussi $3 \leqslant 4 \leqslant 7$

4.b) Expliquer comment construire un arbre binaire de hauteur `h` quelconque ayant `h+1` nœuds.

Réponse

Il **suffit**, par exemple, de construire un arbre binaire où pour chaque nœud, soit le sous arbre à gauche est vide, soit celui à droite.

- Cela peut être toujours celui à gauche qui est vide, on parle alors d'arbre peigne à droite.
- Cela peut être toujours celui à droite qui est vide, on parle alors d'arbre peigne à gauche.

4.c) Expliquer comment construire un arbre binaire de hauteur `h` quelconque ayant $2^{h+1}-1$ nœuds.

> *Indication* : $2^{h+1}-1 = 1+2+4+\ldots+2^h$.

Réponse

Il **faut**, dans ce cas, construire un arbre binaire complet ; les sous-arbres vides sont tous à la même profondeur.

Pour aller plus loin

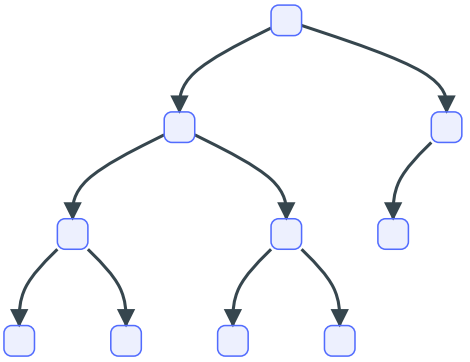
Avec l'autre définition de la hauteur, l'inégalité s'écrit $h \leqslant n \leqslant 2^h-1$, et elle est encore valable pour l'arbre binaire vide. C'est plus simple et général.

L'objectif de la fin de l'exercice est d'écrire le code d'une fonction `fabrique(h, n)` qui prend comme paramètres deux nombres entiers positifs `h` et `n` tels que $h+1 < n < 2^{h+1} - 1$, et qui renvoie un arbre binaire de hauteur `h` à `n` nœuds.

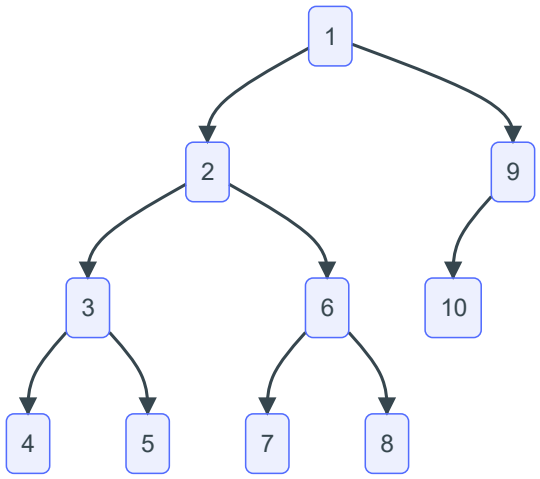
Pour cela, on utilise les deux fonctions suivantes :

- `arbre_vide()` , qui renvoie un arbre vide ;
- `arbre(gauche, droite)` qui renvoie l'arbre de fils à gauche `gauche` et de fils à droite `droite` .

5. Recopier sur la copie l'arbre binaire ci-dessous et numéroter ses nœuds de 1 en 1 en commençant à 1, en effectuant un parcours en profondeur préfixe.



Réponse



La fonction `fabrique` ci-dessous a pour but de répondre au problème posé. Pour cela, la fonction `annexe` utilise la valeur de `n` , qu'elle peut modifier et renvoie un arbre binaire de hauteur `hauteur_max` dont le nombre de nœuds est égal à la valeur de `n` au moment de son appel.

Script Python

```
1  def fabrique(h, n):
2      def annexe(hauteur_max):
3          if n == 0:
4              return arbre_vide()
5          elif hauteur_max == 0:
6              n = n - 1
7              return ...
8          else:
9              n = n - 1
10             gauche = annexe(hauteur_max - 1)
11             droite = ...
12         return arbre(gauche, droite)
13     return annexe(h)
```

6. Recopier sur la copie les lignes 7 et 11 en complétant les points de suspension.

Réponse

Script Python

```python
def arbre_vide():
    return []

def arbre(gauche, droite):
    return [gauche, droite]

def fabrique(h, n):
    def annexe(hauteur_max):
        nonlocal n  # danger (1)
        if n == 0:
            return arbre_vide()
        elif hauteur_max == 0:
            n = n - 1
            return arbre(arbre_vide(), arbre_vide())
        else:
            n = n - 1
            gauche = annexe(hauteur_max - 1)
            droite = annexe(hauteur_max - 1)
        return arbre(gauche, droite)
    return annexe(h)

def taille(arbre):
    if arbre == []:
        return 0
    else:
        gauche, droite = arbre
        return 1 + taille(gauche) + taille(droite)

def hauteur(arbre):
    if arbre == []:
        return -1  # ceci est une hérésie
    else:
        gauche, droite = arbre
        return 1 + max(hauteur(gauche), hauteur(droite))


for h in range(2, 5):
    for n in range(h + 1, 2**(h + 1)):
        n_sav = n
        arbre_hn = fabrique(h, n)
        print(arbre_hn)
        assert (h, n_sav) == (hauteur(arbre_hn), taille(arbre_hn))
```

1. Il faut ajouter cette ligne pour autoriser `annexe` à modifier la variable `n`.
   Cette instruction `nonlocal n` semble hors programme.

On peut tester le code et obtenir

Console Python

```
>>> %Run fabrique_arbres.py
[[[[], []], [], []]
[[[[], []], [[], []]], []]
[[[[], []], [[], []]], [[], []]]
[[[[], []], [[], []]], [[[], []], []]]
[[[[], []], [[], []]], [[[], []], [[], []]]]
[[[[[], []], [], []], []]
[[[[[], []], [[], []]], [], []]
[[[[[], []], [[], []]], [[], []]], []]
[[[[[], []], [[], []]], [[[], []], []]], []]
[[[[[], []], [[], []]], [[[], []], [[], []]]], []]
...
```

On peut proposer une autre version, récursive et bien plus simple pour cette question

Script Python

```python
def fabrique(h, n):
    if n == 0:
        return arbre_vide()
    else:
```

```
        reste = n - 1                    # 1 pour la racine du sous-arbre
        n_gauche = min(reste, 2**h - 1)  # le plus possible à gauche
        n_droite = reste - n_gauche      # la suite à droite
        return arbre(fabrique(h-1, n_gauche), fabrique(h-1, n_droite))
```