INTERROS des LYCÉES
Collection dirigée par Éric MAURETTE

numérique et sciences informatiques

Term

Stéphane PASQUET
Mikaël LEOPOLDOFF

Nathan

Remerciements

Les auteurs et l'équipe de Prepamath Édition tiennent à remercier Jean-Côme Charpentier, David Guénée et Bruno Mermet pour l'aide précieuse qu'ils ont apportée à cet ouvrage.

Illustrations : Prisca Baverey, Stéphane Pasquet

Coordination éditoriale : François Déliac

Conception couverture : Simon Geliot

Votre avis nous intéresse : `contact@prepamath.fr`

Retrouvez l'actualité des Interros des Lycées sur Facebook `facebook.com/interrosdeslycees/` et Instagram `@interrosdeslycees`.

Avant-propos

En classe de terminale générale, le programme de l'enseignement de spécialité de numérique et sciences informatiques (NSI) est conçu autour de quatre thèmes principaux :

- les données et leurs représentations,
- les bases de données,
- la programmation, les langages et les algorithmes,
- les architectures matérielles et réseaux.

À ces thèmes s'ajoutent des notions transverses telles que l'histoire de l'informatique ou les interfaces qui permettent la commande des systèmes.

Comme pour notre ouvrage de première NSI, nous avons mis l'accent sur la programmation et pour cela, proposé de télécharger la plupart des programmes mentionnés dans ce livre.

À partir de votre smartphone ou de votre tablette, les QRcodes vous permettront très facilement de télécharger les fichiers présentés dans l'ouvrage et de les envoyer par mail vers un ordinateur personnel. Gagnant ainsi un temps précieux, vous pourrez tester ces programmes chez vous et/ou les modifier pour exercer votre créativité et assimiler parfaitement leur fonctionnement.

Dans ce livre, les méthodes les plus utiles de Python 3 vous sont présentées. La liste exhaustive se trouve sur le site suivant :

`https://docs.python.org/fr/3/`

Des vidéos vous sont aussi proposées, exposant des explications détaillées sur le fonctionnement de certains algorithmes (voir la liste complète en fin d'ouvrage).

Même si l'informatique s'appuie sur une base théorique, nous insistons sur l'importance fondamentale de la pratique : chaque élève doit être capable de prendre des initiatives pour construire ses propres programmes, se poser des questions et ainsi les faire évoluer à sa guise.

Nombreux et variés, les exercices proposés dans cet ouvrage formeront un bon tremplin pour atteindre cet objectif.

Nous vous souhaitons à présent une bonne lecture et un bon apprentissage de cette matière passionnante, en constante évolution et pleine d'avenir.

Les auteurs

Table des matières

Méthodes de travail

Les conseils qui suivent ont été mis en pratique par un grand nombre de majors (sortis premiers) de Polytechnique ou de l'ÉNA, par des professionnels de l'organisation et sont également recommandés par de nombreux professeurs. Pour approfondir votre méthode de travail et obtenir les meilleurs résultats pendant vos études, nous vous recommandons la lecture du livre « Comment travailler plus efficacement », par F. Déliac, U. Hadrien, E. Matrullo et E. Maurette.

Faire des « feed-back »

Le « ***feed-back*** » est le conseil le plus important et le plus utilisé par ceux qui réussissent brillamment leurs études. Il consiste à contrôler systématiquement, sans s'aider de notes, ce que l'on vient d'apprendre (exercices et cours). Ce contrôle peut se faire mentalement, oralement ou par écrit.

- Dans les transports, essayez de vous rappeler mentalement, et sans vous aider de vos notes, le cours et les exercices vus le matin en classe (feed-back mental).
- Après avoir relu votre cours le soir, essayez de retrouver par écrit les principaux paragraphes et démonstrations sans regarder votre leçon (feed-back écrit).
- Après avoir résolu un problème, prenez 5 minutes pour contrôler par écrit que vous vous rappelez clairement l'énoncé ainsi que la démarche de résolution (feed-back écrit).
- Expliquez à des amis la leçon que vous venez d'apprendre ou l'exercice que vous venez de résoudre : c'est un excellent feed-back oral. Choisissez le type de « feed-back » qui vous convient le mieux et faites-en le plus régulièrement possible (après chaque cours et chaque série d'exercices). Pour être efficace, un « feed-back » doit se faire sans l'aide de vos notes. Ainsi, faire des fiches de résumés de cours à partir de vos cahiers ouverts ne constitue nullement un « feed-back ».

Miser sur la qualité

De nombreux témoignages démontrent que pour obtenir de bons résultats, il est préférable de faire un nombre limité d'exercices de manière approfondie plutôt que d'en survoler un grand nombre. Une tendance très répandue consiste à abattre une grande quantité d'exercices, à la chaîne, mais superficiellement, en espérant que le jour du contrôle, on aura déjà vu ce type de problème et que l'on saura s'en souvenir. Cette méthode est absolument inefficace car la seule manière de se souvenir d'un exercice de mathématiques ou de physique, c'est de l'avoir parfaitement compris et assimilé. Ainsi :

- À la fin d'un problème, prenez 5 à 10 minutes pour essayer de trouver un moyen de le généraliser ou de le compliquer (c'est ce que font souvent les professeurs pour concevoir leurs contrôles écrits) ; trouvez ce que cela pourrait changer dans la solution.

- Prenez également l'habitude, après chaque exercice, de faire un « feed-back » en faisant ressortir la démarche générale et en tissant des liens avec le cours. Bref, il ne faut pas vous contenter de résoudre l'exercice, mais il vous faut lui apporter de la valeur ajoutée et vous interroger sur son contenu.
- *Idem* pour le cours. Ne vous contentez pas de le parcourir de manière passive. Il vous faut avoir la rigueur d'effacer toutes les zones d'ombre. Pour chaque théorème, il faut vous demander quels types d'exercices son utilisation permettra de résoudre.

Travailler par « couches successives »

Cette méthode, très utile pour les étudiants préparant des examens ou des révisions, peut également être utilisée dès le lycée.

On observe que pour apprendre un gros volume de cours, rien n'est plus inefficace que de l'attaquer de front, de manière linéaire. La bonne manière consiste à d'abord survoler l'ensemble, en ne retenant que la structure, c'est-à-dire les grands titres, ainsi que les noms des paragraphes (première couche, étape devant durer 5 minutes). Dans l'étape suivante (deuxième couche, d'une durée de 10 minutes), on reprend son cours du début en retenant cette fois également les théorèmes et résultats importants. Après cette deuxième couche, on a déjà une idée claire de la structure de l'ensemble du cours. On peut alors aborder la dernière étape (troisième couche) : on reprend son cours au début pour, cette fois-ci, l'étudier en profondeur en apprenant le détail des démonstrations.

Il est à noter que cette méthode peut être également appliquée avec succès à des matières littéraires, ainsi qu'aux révisions du bac de français. Par exemple :

- Pour la préparation d'un contrôle, on commencera par passer en revue rapidement l'ensemble du cours et des exercices du chapitre précédemment étudié, avant de les réviser en détail. Ainsi aura-t-on développé une compréhension synthétique et claire.
- De même, avant d'aborder un problème volumineux (tel qu'un contrôle écrit), il est préférable d'en survoler l'ensemble avant de l'attaquer.

Travailler sa rapidité

Pour acquérir de la rapidité, trois voies sont possibles :

- Prenez l'habitude, en travaillant chez vous, de vous concentrer sur une seule chose à la fois, c'est-à-dire ne pas attaquer un problème ou une dissertation, en rêvassant à ce que vous pourriez trouver à manger dans le réfrigérateur ou en écoutant de la musique.
- Prenez l'habitude de travailler chez vous dans les mêmes conditions qu'en devoirs surveillés. Le minutage de chacun des exercices de ce livre est fait en ce sens. Cependant, cela ne devrait pas, une fois la résolution faite, vous empêcher d'y réfléchir plus calmement afin de vérifier la bonne assimilation du problème. Si les seuls moments où vous vous pressez sont les contrôles écrits, vous ne deviendrez jamais rapide.

- Essayez de contenir tout votre travail à la maison dans une plage horaire serrée. Engagez-vous, par exemple, à travailler chez vous tous les jours entre 18 h et 20 h et efforcez-vous de ne jamais déborder (quelle que soit votre charge de travail). En effet, si l'on ne se donne pas de limite de temps pour accomplir un travail, on a naturellement tendance à le laisser traîner en longueur et à rêvasser. L'étroitesse de la plage horaire vous obligera à ne pas vous endormir et à devenir efficace.

Spécial Baccalauréat

L'épreuve terminale obligatoire de spécialité est composée de deux parties : une partie écrite et une partie pratique, chacune notée sur 20. La note de la partie écrite a un coefficient de 0,75 et celle de la partie pratique a un coefficient de 0,25. La note globale de l'épreuve est donnée sur 20 points. Elle est à traiter en 3h30.

- Les exercices peuvent porter sur plusieurs notions (par exemple, bases de données relationnelles et programmation Python).
  Chaque exercice commence par une mention « Cet exercice porte sur ... ».
  À l'aide de cette mention, vous pouvez choisir de commencer à traiter l'exercice qui porte sur vos points forts.
- L'épreuve écrite est composée de 3 exercices obligatoires : deux sur 6 points et un sur 8 points.
- Si vous êtes habitués à utiliser la méthode du feed-back exposée dans les premières pages de cet ouvrage, il est bon d'envisager un court feed-back mental sur les parties du cours concernées par l'épreuve du jour. Si vous êtes nerveux, cela vous aidera à vous mettre en confiance et à « démystifier » l'épreuve, que vous pourrez aborder avec la même sérénité qu'un devoir surveillé classique.
- Il faut absolument soigner la présentation. Au cours de l'année scolaire, votre professeur s'est habitué à votre style de rédaction, à votre écriture. Il sait ce que vous valez. Ce n'est pas le cas du correcteur qui lira votre copie. Il n'aura ni l'indulgence que pourrait avoir votre professeur, ni la patience d'essayer de vous déchiffrer si votre copie est présentée comme un brouillon. Par conséquent, il vous faut faire un réel effort de présentation le jour du bac, plus encore que pendant l'année scolaire. Vous devez notamment veiller à indiquer clairement le numéro des questions auxquelles vous répondez de façon à faciliter la correction de votre copie.
- Enfin, relisez-vous. Réservez-vous pour cela le temps nécessaire. Si vous vous laissez 60 minutes pour la résolution d'un exercice, il vous restera 10 minutes pour sa relecture. Celle-ci doit être très attentive (ce n'est pas toujours facile à l'issue de plus de trois heures d'épreuve). Enfin, souvenez-vous, si vous manquez de temps pour tout relire, qu'il vaut mieux une relecture partielle très attentive qu'une relecture « express » en diagonale, totalement inefficace pour détecter les erreurs résiduelles.


ECE
ÉCOLE D'INGÉNIEURS
ENGINEERING SCHOOL
Tu prévois quoi après ton Bac ?
Rejoins l'ECE : l'école d'ingénieurs qui rend l'Intelligence Artificielle indispensable
6 Campus Paris - Lyon - Bordeaux Rennes - Marseille - Toulouse
3 Programmes Bachelor - Ingénieur - MSc
Retrouvez-nous sur
Inscris-toi à nos prochains événements

Chapitre 1

Programmation orientée objet, programmation dynamique

Plan du chapitre

1 Programmation orientée objet (POO)

Exercice type 1 — Lycée Henri Alain-Fournier, Bourges

On considère une classe `Personnage` représentant un personnage de jeu.
Le plateau du jeu est représenté par un repère orthonormé à trois axes. La position du joueur dans le plateau est repérée par ses attributs x, y et z.
Écrire les méthodes `avance`, `droite` et `saute` permettant respectivement de faire avancer, aller à droite et sauter le personnage, c'est-à-dire d'augmenter de 1 respectivement x, y et z.
Implémenter de plus une autre méthode `coord` renvoyant les coordonnées sous forme d'un triplet.

Voir corrigé page 6

PROBLÉMATIQUE

Comment modéliser des éléments du monde réel sous forme d'objets informatiques afin de les manipuler de manière simple ?

1.1 Historique et définitions

La *programmation orientée objet* (en abrégé : POO) est un paradigme de programmation qui consiste à considérer un programme comme un ensemble d'objets en interaction.

Les *objets* peuvent représenter un concept du monde réel (comme un meuble, une personne ou encore un livre par exemple).

Pour faire de la POO, on peut utiliser un *langage à classes*, comme Python.

La POO trouve son origine en Norvège au début des années 1960. Elle fut élaborée par les informaticiens Ole-Johan Dahl et Kristen Nygaard.

Dans les années 1970, elle fut reprise par l'informaticien américain Alan Kay qui contribua à son essor. Cependant, il fallu attendre les années 1990 pour que ses principes soient clairement formalisés.

Retrouvez une présentation originale de la programmation orientée objet en vidéo.

1.2 Les classes sous Python

1.2.1 - Introduction

Nous allons considérer un objet `Individu` qui représente une personne.

Cette personne a plusieurs caractéristiques. Par exemple :

- sa date de naissance ;
- son genre ;
- sa taille ;
- sa masse ;
- etc.

qui vont représenter les *variables* de la personne.

De plus, cette personne peut faire plusieurs choses :

- parler ;
- manger ;
- marcher ;
- etc.

qui vont correspondre à des *fonctions* propres à elle-même.

Pour définir cette personne ainsi que les fonctions qui lui sont associées, on peut définir une *classe*.

1.2.2 - Définitions et vocabulaire

Définition 1

Une *classe* est un type qui définit en son sein :

- des variables d'instances (appelées aussi *attributs*) ;
- des méthodes (les fonctionnalités associées à la classe).

Exemple : voici un exemple en Python où l'on définit une classe `Personne` à l'aide de trois caractéristiques (année de naissance, taille en cm, masse en kg) et où l'on définit une fonction propre à la classe (grandir) qui admet un paramètre numérique (exprimé en cm).

```python
class Personne:
    def __init__(self, birthyear, tall, weight):
        self.naissance = birthyear
        self.taille = tall
        self.poids = weight

    def grandir(self,h):
        self.taille += h

luc = Personne(2002,178,69)
init_tall = luc.taille
luc.grandir(8)
new_tall = luc.taille
print("Luc pèse", luc.poids, "kg.")
print("Luc mesure à présent", new_tall, "cm après avoir
    grandi de 8 cm en 1 an. Avant ça, il mesurait:",
    init_tall, "cm.")
```

Ce programme affiche :

```
Luc pèse 69 kg.
Luc mesure à présent 186 cm après avoir grandi de 8 cm en
1 an. Avant ça, il mesurait: 178 cm.
```

Définitions 2

Dans l'exemple précédent,

- `naissance`, `taille` et `poids` sont les *attributs* de la classe `Personne` ;
- `grandir` est une *méthode* de la classe `Personne`.

COURS

On accède aux attributs d'une classe à l'aide de la syntaxe :

`<nom de la classe>.<nom de l'attribut>`

On accède aux méthodes d'une classe à l'aide de la syntaxe :

`<nom classe>.<nom méthode>(<arguments potentiels>)`

1.2.3 - Constructeur d'une classe

Définition 3

Le *constructeur* d'une classe est une fonction :

- dont le nom est nécessairement `__init__` ;
- admettant au moins un paramètre nommé `self` placé obligatoirement en premier s'il y en a plusieurs.

Le paramètre `self` représente l'objet cible : c'est une variable qui contient une référence vers l'objet qui est en cours de création. Grâce à ce dernier, on va pouvoir accéder aux attributs et fonctionnalités de l'objet cible.

1.2.4 - Instanciation et variables d'instance

Une classe est une sorte de moule qui sert à fabriquer des objets.

L'*instanciation* est l'opération qui consiste à créer un objet. L'objet ainsi créé est alors appelé une *instance* de la classe.

Exemple : si on reprend la classe `Personne` de l'exemple précédent, en définissant :

```
luc = Personne(2002,178,69)
```

nous avons défini une instance de la classe `Personne`.

On peut ainsi définir plusieurs instances d'une même classe, chacune ayant des attributs qui lui sont propres.

Si on demande d'afficher la variable `luc`, on voit apparaître quelque chose de la forme :

```
<__main__.Personne object at 0x02CDDC50>
```

Cela signifie que `luc` est une référence vers l'objet créé, c'est-à-dire une indication sur son emplacement en mémoire. Ici, par exemple, l'objet `Personne` se trouve en mémoire à la position 0x02CDDC50.

Un attribut de la classe est représenté par une variable appelée *variable d'instance*, accessible à l'aide du paramètre `self`. L'idée, dans notre exemple, est que le constructeur initialise ces variables d'instances, qui seront alors stockées dans l'objet et en mémoire pour toute la durée de vie de l'objet.

Il est important de faire la différence entre les deux types de variables qui se trouvent dans le code de ce constructeur :

- la variable `self.taille` (par exemple) représente la variable d'instance, c'est-à-dire celle associée à l'objet, qui existe à partir de la création de l'objet jusque sa destruction ;
- la variable `taille` représente le paramètre reçu par le constructeur et n'existe que dans le corps de ce dernier.

Le paramètre `self` permet donc d'accéder aux variables d'instance, c'est-à-dire aux attributs de l'objet cible, depuis le constructeur.

1.2.5 - Méthodes d'une classe

Définition 4

> Une *méthode* d'une classe est une fonction définie au sein de la classe et admettant au moins le paramètre `self` (obligatoirement en premier s'il y a plusieurs paramètres).

Le paramètre `self` représente l'objet cible sur lequel la méthode est appelée. Il permet notamment d'avoir accès aux variables d'instance de l'objet.

Dans l'exemple précédent, `grandir(self,h)` est une méthode admettant deux paramètres, dont le second est un nombre. Cette méthode redéfinit la variable d'instance `self.taille` en lui ajoutant la valeur passée en second paramètre.

Une méthode est appelée en spécifiant l'objet cible (ici, `luc`) suivi de la méthode en utilisant l'opérateur d'appel : le point. Cela donne ici :

luc.grandir(8)

1.2.6 - Redéfinition des opérations (hors-programme)

Maintenant que l'on sait définir un objet à l'aide d'une classe, on peut s'intéresser aux opérations que l'on peut faire sur ces objets.

Par exemple, si on souhaite définir un objet `vector` représentant un vecteur du plan, il serait utile de pouvoir définir la somme de deux de ces vecteurs, qui est aussi un objet de classe `vector` (car la somme de deux vecteurs du plan est aussi un vecteur du plan).

Python permet de définir une telle opération de la manière suivante :

```python
class Vector:
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def __add__(self,other):
        return Vector(self.x + other.x, self.y + other.y)
```

COURS

En mode console, cela donne par exemple :

```
>>> u , v = vector(-2,8) , vector(3,-5)
>>> w = u + v
>>> print("w({},{})".format(w.x,w.y))
w(1,3)
```

Cette capacité du langage Python s'appelle la *surcharge d'opérateur*.

1.2.7 - Encapsulation

La classe `vector` définie précédemment aurait pu être définie autrement :

```
class vector:
    def __init__(self,x,y):
        self.coord = (x, y)

    def __add__(self,other):
        return Vector(self.coord[0] + other.coord[0], self.
    coord[1] + other.coord[1])
```

On est passé de deux variables d'instance à une seule, mais cette différence ne doit pas être perçue par l'utilisateur : la manière dont on gère les valeurs passées en paramètres au sein de la classe ne doit pas être vue « de l'extérieur ». Ce principe fondamental de la POO est appelé *encapsulation*. Cela peut être réalisé grâce à la définition de méthodes adéquates.

Solution de l'exercice type 1 — **Lycée Henri Alain-Fournier, Bourges**

Une définition possible de la classe `Personnage` est la suivante :

```
class Personnnage:
    def __init__(self,x,y,z):
       self.x, self.y, self.z = x, y, z
    def avance(self):
        self.x += 1
    def droite(self):
        self.y += 1
    def saute(self):
        self.z += 1
    def coord(self):
        return (self.x, self.y, self.z)
```

...

Solution de l'exercice type 1 (suite) — Lycée Henri Alain-Fournier, Bourges

En utilisant le code, voici ce que donne un exemple d'utilisation :

```
>>> Lara = Personnnage(0,0,0)
>>> Lara.avance()
>>> Lara.saute()
>>> Lara.coord()
(1,0,1)
```

Voir énoncé page 1

2 Programmation dynamique

Exercice type 2 — Lycée André Theuriet, Civray

Étant donné un système de monnaie S comportant des pièces de valeurs différentes, écrire un programme qui affiche le nombre *minimal* de pièces à utiliser pour rendre une somme donnée, ainsi que leur énumération.

Voir corrigé page 12

2.1 Introduction

Il s'agit d'un problème d'optimisation, puisqu'on veut utiliser un nombre de pièces et de billets minimal. A travers l'étude de ce problème d'algorithmique classique, nous allons voir qu'il existe de multiples possibilités d'y répondre. La programmation dynamique est la plus efficace.

2.2 Approche gloutonne

C'est l'approche que font bon nombre de commerçants quand il rendent la monnaie à leurs clients.

Elle consiste à prendre le billet ou la pièce de valeur maximale en dessous de la somme à restituer, puis à recommencer ce geste par rapport à la différence. Cela donne l'algorithme suivant :

```
V ← valeur à atteindre
Tant que V n'est pas nulle:
   Choisir le billet ou la pièce de plus grande valeur vi
   telle que vi ⩽ V
   V ← V - vi
Fin du Tant que
```

COURS

Exemple : comment rendre la somme de 14 € dans un système monétaire européen simplifié constitué uniquement de pièces de 1, 2, 5 et 10 € ?

On a alors $S = \{1; 2; 5; 10\}$.

Pour rendre 14 €, la plus grande valeur possible à déduire est 10 €, ce qui donne $14 - 10 = 4$.
La plus grande valeur possible à déduire de 4 € est 2×2 €, ce qui donne $4 - 2 \times 2 = 0$.
La solution renvoyée par notre algorithme est donc le quadruplet $(0; 2; 0; 1)$, soit un total de 3 pièces. Les autres possibilités sont :

- $(0; 2; 2; 0)$, soit 4 pièces,
- $(0; 7; 0; 0)$, soit 7 pièces,
- $(14; 0; 0; 0)$, soit 14 pièces.

L'algorithme glouton a ici donné la solution optimale. Mais cela ne serait pas forcément le cas pour d'autres systèmes.
Imaginons en effet un système composé uniquement de pièces de 1, 7 et 9 €, c'est-à-dire que $S = \{1; 7; 9\}$.
L'algorithme glouton nous amène à la solution $(5; 0; 1)$, soit un rendu de 6 pièces. Alors que la solution optimale est $(0; 2; 0)$, soit seulement 2 pièces.

Conclusion : l'algorithme glouton a le mérite d'être simple, mais il ne rend pas toujours la solution optimale. Il ne convient donc pas à la résolution de notre problème.

2.3 Approche récursive « diviser pour régner »

Notons à présent $S = \{v_1; v_2; \ldots; v_n\}$, v_i étant les valeurs des n pièces ou billets composant notre système monétaire S, et V la somme à rendre.
Nous allons calculer le nombre minimal de pièces à rendre que nous noterons `minimum`. Il servira ensuite à déterminer le n-uplet optimal recherché. La méthode précédente suggère de prendre une approche récursive.

- Si $V = 0$, on retourne « 0 »... Logique !
- Sinon, on fixe la valeur de la variable `minimum` (qui représentera au final le plus petit nombre de pièces) délibérément à une valeur assez grande (de toute évidence, le nombre de pièces est inférieur à la somme totale, donc fixer `minimum` à `V+1` nous assure que les valeurs de `minimum` qui seront calculées par la suite seront inférieures).
- On parcourt le système monétaire `S` et pour chaque valeur `S[i]`, on teste si elle est inférieure ou égale à `V` ; si tel est le cas, on prend en compte la valeur `S[i]` et on ajoute le nombre minimum de pièces ou billets nécessaires pour atteindre la valeur `V - S[i]`.

Si la valeur obtenue est plus petite que `minimum` alors cela signifie que l'on a trouvé une combinaison plus optimale que les précédentes et on change la valeur de la variable `minimum` pour la valeur de `n`.

- Une fois que l'on a parcouru le système monétaire, on retourne la valeur de `minimum`, qui correspond au plus petit nombre possible de pièces et de billets.

Le script ci-dessous est de type « diviser pour régner », puisque nous avons ramené le calcul du nombre recherché pour le montant V au même calcul pour des montants inférieurs ($V-$piece1, $V-$piece2, etc.). Nous avons divisé le problème initial, puis appliqué un traitement récursif. Notons cependant que pour cela, de nombreux calculs redondants ont été effectués. Cet algorithme n'est donc pas optimisé.

```python
def Nombre(V, S = [1,7,9]):
    if V == 0:
        return 0
    else:
        minimum = V + 1
        for piece in S:
            if piece <= V:
                n = 1 + Nombre(V - piece)
                if n < minimum:
                    minimum = n
        return minimum

print(Nombre(14))
```

2.4 Approche dynamique Top Down

Nous allons améliorer l'algorithme précédent en mémorisant dans une liste nommée `L` les calculs effectués à chaque étape. `L` est une liste unidimensionnelle de $(V+1)$ éléments qui va contenir le nombre de pièces minimal `L[v]` que l'on doit utiliser pour rendre la monnaie, pour chaque montant `v` compris dans $[0;\ V]$. Cette liste est construite de manière récursive à partir du montant initial V. De cette manière, les redondances de l'algorithme précédent disparaissent et l'algorithme est optimisé.

C'est ce qu'on appelle l'approche dynamique *Top Down* (de haut en bas).

Définition 5 : Top Down

> La forme récursive *Top Down* est une technique d'optimisation ayant pour but de diminuer le temps d'exécution d'une fonction récursive en mémorisant (dans une liste par exemple) les valeurs retournées par cette fonction après avoir vérifié qu'elles n'existaient pas encore dans la liste.

COURS

Cette technique est aussi appelée *mémoïsation*.

```
def construct_liste(V):
    L = [0]*(V+1)
    return nombre_top_down(V, L)

def nombre_top_down(value, L, S = [1,7,9]):
    if V == 0:
        return 0
    elif L[V] > 0:
        return L[V]
    else:
        minimum = V + 1
        for piece in S:
            if piece <= V:
                n = 1 + nombre_top_down(V - piece, L)
                if n < minimum:
                    minimum = n
                    L[V] = minimum
        return minimum

print(ConstructListe(14))
```

2.5 Approche dynamique Bottom Up

Dans cette approche, nous conservons notre liste `L` qui va être, cette fois, construite de manière itérative, en partant de 0 pour aller jusqu'à V. Il n'y a plus d'appel récursif comme c'était le cas pour la fonction `NombreTopDown`.

Cette technique de programmation dynamique est appelée *Bottom Up* (de bas en haut).

Définition 6 : Bottom Up

La forme itérative *Bottom Up* est une technique consistant à résoudre des problèmes en partant de celui qui a la taille la plus petite et en allant vers celui qui a la taille la plus grande, tout en stockant les résultats de chacun d'eux dans une liste (par exemple).

Les techniques d'optimisation « Bottom Up » et « Top Down » sont à la base de la programmation dynamique, ce qui nous amène à la définition page ci-contre.

Définition 7 : programmation dynamique

La *programmation dynamique* est une méthode de résolution de problèmes d'optimisation en scindant le problème en plusieurs sous-problèmes et en utilisant l'une des méthodes « Top Down » ou « Bottom Up ».

Ainsi, la programmation dynamique est un paradigme de programmation complémentaire à « diviser pour régner », ce dernier se révélant inefficace dans certains cas.

Voici le fonctionnement du programme final.

- On commence par initialiser deux listes `L` et `valeurs`, toutes les deux étant de dimension `V+1`, `V` étant la valeur à décomposer, les indices variant de 0 à `V`.
  `L` est une liste où `L[x]` contiendra le nombre minimum de pièces et billets qui compose la valeur `x` (comprise entre 0 et `V`) tandis que `valeurs[x]` contiendra l'indice `i` pour lequel `S[i]` $\leqslant$ `x` et pour lequel `1+L[x-S[i]]` est plus petit ou égal au nombre minimum de pièces et billets nécessaires pour la décomposition de la somme souhaitée.
- On remplit donc `L` avec la boucle : `for x in range(1,V+1)`.
- Dans cette boucle, on commence par initialiser le minimum à `V+1` ; nous sommes ainsi assuré.e.s que le minimum est plus petit que cette valeur.
- On parcourt ensuite tous les indices `i` de 0 à `len(S)`, et on effectue le test `if (S[i]<=x) and (1+L[x-S[i]]<minimum)` pour déterminer la valeur du nouveau minimum car si ces conditions sont remplies, le minimum devient `1+L[x-S[i]]`. Dans ce cas, on précise que l'indice à garder est celui sur lequel on est : on le stocke dans la variable `v` pour plus tard.
- Une fois à l'extérieur du test conditionnel, pour la valeur d'argent `x` sur laquelle on est, nous stockons le minimum obtenu dans `L[x]` ainsi que l'indice `v` obtenu dans `valeurs[x]`.
- À la sortie de la boucle itérative sur `x`, les listes `L` et `valeurs` sont ainsi remplies respectivement du minimum de pièces et billets qui composent chaque indice jusqu'à `V`.
  On initialise alors la variable `x` à `V` et une liste vide `R`, qui contiendra le rendu de monnaie, donc la décomposition.
- Pour remplir la liste `R`, il suffit de soustraire à `x` la valeur que l'on y insère, et cette valeur correspond à `S[valeurs[x]]`.
- On finit par retourner `L[V]`, à savoir le nombre minimum de pièces et billets nécessaires pour décomposer `V`, ainsi que la liste `R` nouvellement obtenue.

Le programme qui suit affiche `(2, [7, 7])`, qui représente bien la solution à notre problème :

- la solution optimale est composée de 2 pièces ;
- l'énumération des pièces à utiliser pour un montant de 14 € est : (7 €, 7 €).

COURS

Solution de l'exercice type 2

Lycée André Theuriet, Civray

```
def NombreBottomUp(V):
    S = [1,7,9]
    L = [0]*(V+1)
    valeurs = [0]*(V+1)
    for x in range(1,V+1):
        minimum = V + 1
        for i in range(len(S)):
            if (S[i] <= x) and (1+L[x-S[i]] < minimum):
                minimum = 1 + L[ x - S[i] ]
                v = i
        L[x] = minimum
        valeurs[x] = v
    x, R = V, [ ]
    while x > 0:
        R.append(S[ valeurs[x] ])
        x -= S[ valeurs[x] ]

    return L[V], R

print(NombreBottomUp(14))
```

Voir énoncé page 7

1 QCM Vérification de connaissances

10 min Corrigé p. 30

Pour chacune des questions suivantes, plusieurs propositions de réponses sont faites dont une seule est correcte. Laquelle ?

1 On définit la classe `identite` de la manière suivante :

```
class identite:
    def __init__(self, nom, prenom, an):
        self.nom = nom
        self.prenom = prenom
        self.an = an

    def age(self, a):
       return a - self.an
```

Si on tape :

```
>>> hugo = identite('Hugo' , 'Dupont' , 1999)
>>> print( hugo.age(2020) )
```

que s'affiche-t-il ?

a 'Hugo' **b** 'Dupont' **c** 1999 **d** 21

2 On définit la classe `voiture` de la manière suivante :

```
class voiture:
    def __init__(self, nom, couleur, puissance):
        self.nom = nom
        self.couleur = couleur
        self.puissance = puissance

    def nc(self):
        return nom + couleur
```

Si on tape :

```
>>> F430 = voiture('Ferrari' , 'Rouge' , 43)
>>> print(F430.nc())
```

que s'affiche-t-il ?

a 'Ferrari Rouge' **b** Une erreur

3 Qu'affiche le programme suivant ?

```
class toto:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def is_square(self):
        if self.a**2 + self.b**2 == self.c**2:
            return True
        else:
            return False

t = toto(3,4,5)

print(t.is_square())
```

a Une erreur **b** True **c** False

2 V/F Vérification des connaissances

10 min Corrigé p. 30

Les affirmations suivantes sont-elles vraies ou fausses ? Justifier la réponse.

1 *La programmation dynamique.*

(a) La mémoïsation est une forme itérative.

(b) La programmation dynamique consiste à résoudre des problèmes, de celui qui a la taille la plus petite vers celui qui a la taille la plus grande.

(c) Un problème d'optimisation où la variable peut être réelle peut être résolu en utilisant la programmation dynamique.

(d) Le problème du rendu de monnaie est un problème classique qui ne peut être résolu que par programmation dynamique.

2 *La Programmation Orientée Objet (POO).*

(a) Les variables d'une classe sont appelées des *méthodes*.

(b) Dans une classe, la définition d'une méthode `__init__` n'est pas obligatoire.

Classes

3 La classe « Piece »

★ 5 min Corrigé p. 31

Lycée Camille Jullian, Bordeaux

Voici un programme en Python :

```
import random

class Piece :
    def alea(self) :
        return random.randint(0,1)

    def moyenne(self, n):
        tirage = [ ]
        for i in range (n) :
            tirage.append(self.alea())

        return sum(tirage) / n

p = Piece()
print(p.moyenne(100))
```

Expliquez en détail ce qu'il permet d'afficher.

4 Écrire une classe

★ 10 min Corrigé p. 31

Lycée Sophie Germain, Paris

Pour chacune des questions suivantes, un constructeur initialisant les mesures devra être présent dans chacune des classes.

1 Écrire une classe nommée `Carre` admettant la mesure des côtés d'un carré en attribut, avec deux méthodes :

- une méthode `perimetre` permettant de retourner le périmètre du carré ;
- une méthode `aire` permettant de retourner son aire.

2 Écrire une classe `Triangle` représentant un triangle et admettant la mesure des trois côtés comme attributs, avec deux méthodes :

- une méthode `aire` renvoyant l'aire du triangle à l'aide de la formule de Héron :
$$\mathcal{A} = \sqrt{p(p-a)(p-b)(p-c)},$$
où a, b et c sont les longueurs des trois côtés et où $p = \dfrac{a+b+c}{2}$;
- une méthode `rect` qui renvoie `True` si le triangle est rectangle, et `False` sinon.

3 Définir alors un carré de côté 5, puis afficher son aire et son périmètre. Définir ensuite un triangle de côtés 5.4, 9 et 7.2 pour afficher son aire et regarder s'il est rectangle.

5 Devinette et remplissage

★ 10 min Corrigé p. 33

Lycée François Villon, Les Mureaux

On considère la classe `mystere` définie de la manière suivante :

```
class mystere:
    def __init__(self):
        self.Liste = [ ]

    def add(self, x):
        if len(self.Liste) == 0 or x >= self.Liste[ len
    (self.Liste)-1 ]:
            self.Liste.append(x)
        elif x < self.Liste[0]:
            self.Liste = [x] + self.Liste
        else:
            for i in range(len(self.Liste)):
                if x > self.Liste[i] and x <= self.
    Liste[i+1]:
                    self.Liste = self.Liste[:i+1] + [x]
      + self.Liste[i+1:]
                    break
```

1 Que fait le programme suivant ?

```
L = mystere()

for i in [78,89,10,50,7]:
    L.add(i)

print(L.Liste)
```

2 Ajoutez à cette classe une méthode `ecc` qui crée et renvoie une liste dont les éléments sont les cumuls de ceux de l'attribut `Liste` de la classe.

Par exemple, si `Liste = [1,2,4,5]`, `ecc` doit retourner :

`[1, 3 ,7, 12].`

6 Une classe « Livre »

★ 10 min Corrigé p. 34

Lycée François Mauriac, Bordeaux

On définit une classe `Livre` de la manière suivante :

```python
class Livre:
    def __init__(self, titre, auteur):
        self.titre = titre
        self.auteur = auteur

    def afficher_infos(self):
        print(f"Titre : {self.titre}, Auteur : {self.auteur}")

    def meme_auteur(self, autre_livre):
        return self.auteur == autre_livre.auteur

# Exemple d'utilisation
livre1 = Livre("1984", "George Orwell")
livre2 = Livre("La Ferme des animaux", "George Orwell")
livre3 = Livre("Le théorème du Perroquet", "Denis Guedj")

print(livre1.meme_auteur(livre3))
```

1 Dans le programme ci-dessus,
- **(a)** Combien a-t-on défini d'instances ? Quelles sont-elles ?
- **(b)** Combien la classe `Livre` a-t-elle d'attributs ? Citez-les.
- **(c)** Combien la classe `Livre` a-t-elle de méthodes ? Citez-les en précisant leur rôle.

2 Qu'affiche le programme ?

7 Compte bancaire

★ 20 min Corrigé p. 35

Lycée Richelieu, Versailles

On souhaite implémenter une classe `Compte_bancaire` telle que les instances de `Compte_bancaire` possèdent un seul attribut `solde` (pouvant être positif ou négatif) ainsi que les méthodes suivantes :

- une méthode `transferer` pour transférer un montant X positif vers une instance B de `Compte_bancaire` ;
- une méthode `prelever` pour prélever un montant X positif depuis une instance B de `Compte_bancaire` ;
- une méthode `etat_solde` pour renvoyer « Positif », « Négatif » ou « Nul » lorsqu'on lui demande l'état du compte.

Dans ce qui suit, on considère que le code sera écrit et indenté à l'intérieur d'un bloc commençant par `class Compte_bancaire`.

1. Écrire la méthode `__init__` en précisant si nécessaire le ou les arguments à utiliser.
2. Écrire la méthode `etat_solde` en précisant si nécessaire le ou les arguments à utiliser.
3. Soit A une variable recevant une instance de `Compte_bancaire` : `A = Compte_bancaire(1000)`.
   Prévoir le résultat de l'instruction suivante : `A.etat_solde()`.
4. **(a)** Écrire la méthode `transferer`.
   (b) Écrire la méthode `prelever`.
   (c) Écrire les instructions permettant de modéliser toutes les étapes de la situation suivante :
   - un compte bancaire A contient 1 000 € ;
   - un autre compte bancaire B contient 5 000 € ;
   - le compte bancaire B prélève 1 100 € le compte A ;
   - le solde du compte A est affiché (avec un `print`).

8 La classe « Temps » ★★ 25 min Corrigé p. 36

Lycée Sainte-Clotilde, Strasbourg

En Python, écrire une classe `Temps` qui permet de définir un horaire au format hh : mm : ss et qui admet les méthodes suivantes :

- `affiche`, qui affiche l'horaire au format « 12 h 37 min 45 s » ;
- `add`, qui ajoute deux horaires de la classe `Temps` ;
- `sub`, qui calcule la différence entre deux horaires de la classe `Temps`.

9 Classes « Mot » et « Phrase » ★★ 30 min Corrigé p. 36

Lycée Raoul Follereau, Nevers

On donne page ci-contre la classe `Mot`.

1. Indiquer ce que fait la fonction `fctA()` définie par :

```
def fctA():
  m = Mot("Socrate")
  m.doReverse()
  print(m.value())
```

```
from random import shuffle

class Mot:
  def __init__(self,m):
    self.m = m

  def do_reverse(self):
        # renverse la chaîne 'm'
    self.m = self.m[::-1]

  def do_shuffle(self):
    L = list(self.m)
    shuffle(L)
    self.m = ''.join(L)

  def value(self):
    return self.m
```

2 Indiquer ce que fait la fonction `fctB()` définie par :

```
def fctB():
  m = Mot("Socrate")
  m.doShuffle()
  print(m.value())
```

On souhaite écrire une classe `Phrase`. Toutes les questions suivantes porteront sur cette classe.

3 Écrire un constructeur qui définit une liste `self.mots` remplie de tous les mots de la phrase passée en paramètre, chacun des mots devant être de classe `Mot`.

4 Écrire deux méthodes `doReverse` et `value` telles que la fonction `fctC()` suivante :

```
def fctC():
  p = Phrase("Tous les hommes sont mortels")
  p.doReverse()
  print(p.value())
```

affiche :

« mortels sont hommes les Tous »

5 Écrire une méthode `doShuffle` afin que la fonction suivante :

```
def fctD():
  p = Phrase("Tous les hommes sont mortels")
  p.doShuffle()
  print(p.value())
```

affiche les mots de la phrase « Tous les hommes sont mortels » dans un ordre aléatoire.

6 On définit la méthode `motAt` de la manière suivante :

```
def motAt(self,pos):
  return self.mots[pos]
```

Que fait la fonction `fctE()` suivante ?

```
def fctE():
  p = Phrase("Tous les hommes sont mortels")
  m = p.motAt(3)
  m.doReverse()
  print(p.value())
```

7 On définit la méthode `insert` de la manière suivante :

```
def insert(self,pos,chaine):
  self.mots.insert(pos,Mot(chaine))
```

Que fait la fonction `fctF()` suivante ?

```
def fctF():
  p = Phrase("Tous les hommes sont mortels")
  p.insert(3,"ne")
  p.insert(5,"pas")
  print(p.value())
```

8 On définit la méthode `remove` de la manière suivante :

```
def remove(self,pos):
    self.mots.pop(pos)
```

Que fait la fonction `fctG()` suivante ?

```
def fctG():
  p = Phrase("Tous les hommes sont mortels")
  m = p.motAt(4)
  m.doShuffle()
  p.remove(2)
  p.insert(2,m.value())
  print(p.value())
```

10 Personnage d'un jeu vidéo

★★★ 30 min Corrigé p. 39

Lycée André Theuriet, Civray

Vous êtes en train de développer un jeu vidéo et vous avez besoin de créer une classe pour représenter les personnages du jeu. Chaque personnage a un nom, un niveau, et des points de vie. Vous devez pouvoir créer des personnages, les faire gagner de l'expérience, et vérifier leur état de santé.

Instructions :

1 Créez une classe `Personnage` avec les attributs suivants :

- `nom` : une chaîne de caractères représentant le nom du personnage.
- `niveau` : un entier représentant le niveau du personnage.
- `points_de_vie` : un entier représentant les points de vie du personnage.

Ajoutez un constructeur `__init__` pour initialiser ces attributs.

2 Implémentez les méthodes suivantes.

(a) `gagner_experience(self, experience)` : une méthode qui prend un nombre d'expérience en paramètre et augmente le niveau du personnage en fonction de l'expérience gagnée. Supposons que chaque tranche de 100 points d'expérience augmente le niveau de 1.

(b) `est_vivant(self)` : une méthode qui retourne `True` si le personnage a des points de vie supérieurs à 0, sinon `False`.

(c) `afficher_etat(self)` : une méthode qui affiche les informations du personnage sous la forme « Nom : [nom], Niveau : [niveau], Points de vie : [points_de_vie] ».

INTERROS

Programmation dynamique

11 Découpe d'une chaîne

★ 15 min Corrigé p. 39

Lycée Sainte-Anne, Brest

Complétez le programme suivant :

```
def decouper_chaine(chaine, dictionnaire, memo={}):
    if chaine in memo:
        return ...

    if chaine == "":
        return ...

    resultats = []
    for i in range(1, len(chaine) + 1):
        prefixe = chaine[:i]
        if prefixe in dictionnaire:
            for suffixe_decoupe in decouper_chaine(
    chaine[i:], dictionnaire, memo):
                resultats.append([prefixe] +
    suffixe_decoupe)

    memo[chaine] = resultats
    return resultats
```

Le but de ce programme est de proposer une liste de plusieurs « découpes » d'une chaîne de caractère selon un dictionnaire de mots fourni.

```
>>> dictionnaire = {
"chat", "chapeau", "au", "lait", "laitue", "eau", "tue", "lai"
}
>>> chaine = "chapeaulaitueau"
>>> decoupes = decouper_chaine(chaine, dictionnaire)
>>> print("Découpes possibles :", decoupes)
Découpes possibles :
[['chapeau', 'lai', 'tue', 'au'],
['chapeau', 'laitue', 'au']]
```

12 Modélisation d'une suite numérique ★★ 20 min

Corrigé p. 40

Lycée François Mauriac, Bordeaux

On considère la suite numérique (u_n) définie par ses deux premiers termes $u_0 = 1$, $u_1 = 2$ et par la relation de récurrence suivante :

$$\forall\, n \in \mathbb{N}, \quad u_{n+2} = \frac{u_n}{u_{n+1}}.$$

Écrire un programme Python permettant de calculer u_n pour un entier n donné en utilisant la programmation dynamique (version Bottom Up et Top Down).

13 Somme dans une suite numérique ★★★ 30 min

Corrigé p. 41

Lycée polyvalent Émile Zola, Aix-en-Provence

On définit une suite (u_n) par son premier terme $u_0 = 1$ et par la relation de récurrence :

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \frac{1}{n+1}\sum_{k=0}^{n}(k+1)u_k.$$

1. Calculer u_1, u_2 et u_3.
2. Proposer deux programmes permettant de calculer u_n pour tout entier naturel n en utilisant le paradigme de la programmation dynamique (version Bottom Up et Top Down).

14 Le retour du problème du sac à dos ★★★ 30 min

Corrigé p. 42

Lycée Jean-Joseph Fourier, Auxerre

Nous avons vu dans l'exercice **??** page **??** ce en quoi consiste le problème du sac à dos. Nous en avons d'ailleurs donné une solution récursive de type « diviser pour régner ».

1. Proposer une approche dynamique de type Bottom Up et Top Down de ce problème qui affiche uniquement la valeur maximale des objets.

   Vérifier les programmes avec l'exemple où le poids maximal est $P = 10$ et où la collection d'objets est $E = \{(5,7); (3,1); (3,4); (3,2)\}$.
2. Compléter l'approche Bottom Up (par exemple) afin d'afficher la combinaison des objets retenus.
3. Que dire de la complexité de l'approche Bottom Up par rapport à celle de l'approche récursive initiale ?

15 La civilisation Maya

★★★ 45 min

Corrigé p. 45

Extrait du Bac 2024, Métropole, septembre

Cet exercice porte sur les bases de numération, la structure de données PILE et la POO.

La civilisation Maya est une ancienne civilisation de Mésoamérique principalement connue pour ses avancées dans les domaines de l'écriture, de l'art, de l'architecture, de l'agriculture, des mathématiques et de l'astronomie.

La numération Maya est une numération positionnelle de base 20 (dite vigésimale) utilisant trois symboles pour former les « chiffres » :

- une coquille pour le zéro ,
- un point pour l'unité ,
- un trait pour la valeur 5 .

Les « chiffres » sont les suivants. Ils utilisent une numération additive :

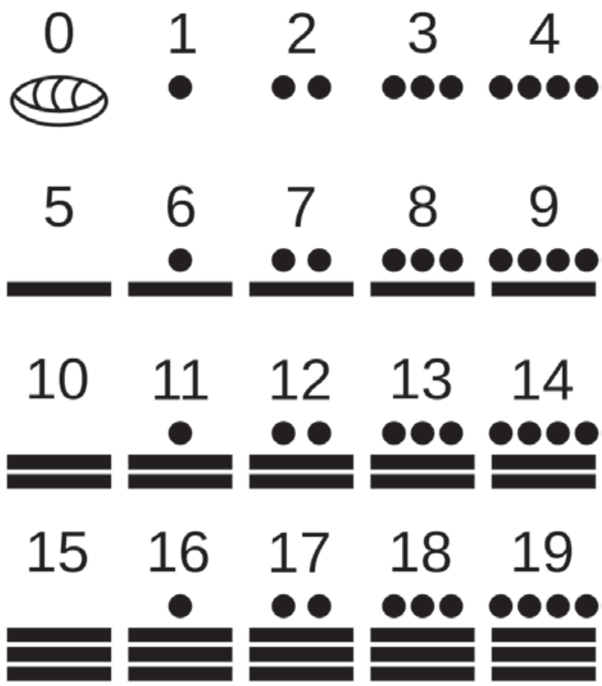
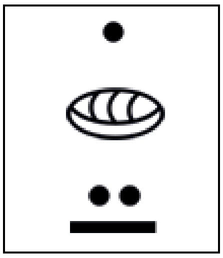


Figure 1.1 – Table des « chiffres » et valeurs correspondantes (source : Wikipédia)

Dans une version simplifiée de ce système, l'écriture d'un nombre se fait par empilement de « chiffres ». Chaque étage correspond à un chiffre de poids 20 fois supérieur au poids du chiffre de l'étage inférieur.

Ainsi la valeur du chiffre de l'étage le plus bas est multipliée par 20^0 soit 1, du second étage par 20^1, du troisième étage par 20^2, et ainsi de suite.

Exemple : la représentation Maya de l'entier 407 est la suivante.



1 Compléter le tableau donné en annexe à rendre avec la copie.

2 Justifier que l'écriture Maya de l'entier 3 435 est :

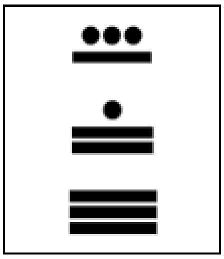


Figure 1.2 – Écriture Maya de l'entier 3 435

On modélise l'écriture d'un entier dans sa représentation Maya par une pile formée de listes. Chacune de ces listes est composée de trois entiers et modélise le « chiffre » d'un étage :

- le premier entier vaut 0 ou 1 suivant s'il y a ou non une coquille ;
- le deuxième représente le nombre de points ;
- le troisième représente le nombre de traits.

Ainsi, la modélisation Maya de l'entier 3 435 est :

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

et celle de l'entier 407 est :

```
[[0, 2, 1], [1, 0, 0], [0, 1, 0]].
```

INTERROS

Le sommet de la pile se situe en fin de liste. On dispose de la classe suivante :

```
class Maya:
    def __init__(self):
        self.nombre = []
    def ajouter(self, chiffre):
        """ chiffre est une liste de longueur 3.
        La méthode empile le chiffre au sommet de la
    pile """
        self.nombre.append(chiffre)
    def retirer(self):
        """ depile et renvoie le chiffre qui etait au
    sommet de
            la pile """
        if not self.estVide():
            return self.nombre.pop()
    def estVide(self):
        return self.nombre == []
    def nbEtages(self):
        """ renvoie le nombre de chiffres de la pile ""
    "
        ...
    def MayaToDec(self):
        """ renvoie le nombre entier correspondant a la
        modelisation Maya de l'instance courante """
        coeff = 20**...
        ch_Dec = 0
        while ... :
            ch_Maya = ...
            ch_Dec = ch_Dec + (valeurChiffre(ch_Maya))
    * coeff
            coeff = ...
        return ch_Dec
    def multiplie(self):
        """ renvoie le resultat de la multiplication
    par 20 d'un
        nombre en modelisation Maya. """
        ...
    def somme(self, maya2):
        """ ajoute maya2 à l'instance courante et
    renvoie le
        resultat en modelisation Maya """
        if self.nbEtages() == maya2.nbEtages()
            ...
```

3 Écrire une suite d'instructions permettant de créer une instance, nommée M, de la classe Maya qui modélise le nombre entier 3 435.

4 Écrire la méthode `nbEtages` de la classe `Maya`. Celle-ci renvoie le nombre de « chiffres » utilisés pour écrire le nombre correspondant en écriture Maya.

De l'écriture Maya à l'écriture décimale

5 Écrire une fonction `valeurChiffre` ayant pour paramètre une liste `L`. Celle-ci renvoie la valeur de l'entier associé à la liste `L = [c,p,t]` où `c` (de valeur 0 ou 1) indique la présence d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant un « chiffre » Maya.

Exemple :

```
>>> valeurChiffre([0, 2, 3])
>>> 17
>>> valeurChiffre([1, 0, 0])
>>> 0
```

6 Recopier et compléter les lignes 2, 4, 5 et 7 de la méthode `MayaToDec` suivante de la classe `Maya`. Cette méthode renvoie la valeur de l'entier associé à l'objet `Maya`. On pourra utiliser les méthodes `estVide`, `nbEtages` et `retirer`.

```
1  def MayaToDec(self):
2      coeff = 20**(self.nbEtages() - 1)
3      ch_Dec = 0
4      while not self.estVide() :
5          ch_Maya = self.retirer()
6          ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)
       ) * coeff
7          coeff = coeff // 20
8      return ch_Dec
```

De l'écriture décimale vers sa modélisation Maya

On considère que la fonction `DecToVige` est déjà écrite. Celle-ci prend en paramètre un entier n et renvoie la décomposition en base 20 de celui-ci sous la forme d'une liste $[a_0,a_1,\ldots,a_p]$ telle que :

$$n = a_0 \times 20^0 + a_1 \times 20^1 + a_2 \times 20^2 + \cdots + a_p \times 20^p.$$

INTERROS

Exemple :

```
>>> DecToVige(3435)
[15, 11, 8]
>>> DecToVige(407)
[7, 0, 1]
```

7 Écrire la fonction `decompChiffre` qui prend en paramètre un entier n compris entre 0 et 19 et renvoie la liste `[c, p, t]` où `c` vaut 0 ou 1 et indique la présence ou non d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant le « chiffre » Maya correspondant.

Exemple :

```
>>> decompChiffre(17)
[0, 2, 3]
>>> decompChiffre(0)
[1, 0, 0]
```

8 Écrire la fonction `DecToMaya` qui prend en paramètre un entier n et renvoie la modélisationMaya d'un objet `M` de la classe `Maya` correspondant.

Exemple :

```
>>> DecToMaya(3435).nombre
[[0, 0, 3], [0, 1, 2], [0, 3, 1]]
>>> DecToMaya(407).nombre
[[0, 2, 1], [1, 0, 0], [0, 1, 0]]
```

Opérations sur les nombres en modélisation Maya

On souhaite additionner des nombres directement à partir de leur modélisation Maya.

9 Écrire la méthode `multiplie` de la classe `Maya` qui renvoie le résultat de la multiplication par 20 d'un nombre en modélisation Maya.

Exemple :

```
>>> M = Maya()
>>> M.ajouter([0, 0, 3])
>>> M.ajouter([0, 1, 2])
>>> M.multiplie().nombre
[[1, 0, 0], [0, 0, 3], [0, 1, 2]]
```

On donne la fonction `mystere` suivante :

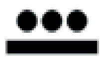
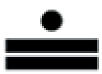
```
def mystere(m1, m2, ret):
    c = 0
    p = (m1[1] + m2[1] + ret) % 5
    if m1[1] + m2[1] + ret >= 5:
        ret = 1
    else:
        ret = 0
    t = (m1[2] + m2[2] + ret) % 4
    if m1[2] + m2[2] + ret < 4:
        ret = 0
    else:
        ret = 1
    if (m1[0] == 1 and m2[0] == 1) or (p + t == 0 and
    ret == 1):
        c = 1
    return ([c, p, t], ret)
```

10 Donner les résultats renvoyés par les deux appels suivants :

- `mystere([0, 1, 1], [0, 3, 1], 0)`
- `mystere([0, 1, 1], [0, 4, 2], 0)`

11 Écrire une méthode `somme` de la classe `Maya` permettant d'ajouter à l'instance courante un autre nombre `maya2` de même taille en modélisation Maya.

ANNEXE À RENDRE AVEC LA COPIE

Étage	Écriture Maya	Valeur du « chiffre » de l'étage	Valeur dans la conversion
3		$1 \times 5 + 3 \times 1 = 8$	$8 \times 20^2 = 3\,200$
2			
1			

CORRIGÉS

1 QCM Vérification de connaissances

Énoncé p. 13

1 Réponse $\boxed{\text{d}}$. En effet, la méthode calcule la différence entre la valeur de l'argument `a` et la valeur de `self.an`, qui correspond à l'année (sans doute de naissance) de Hugo. Il s'agit donc ici de $2020 - 1999 = 21$.

2 Réponse $\boxed{\text{b}}$. En effet, dans la déclaration de la méthode `nc`, les variables `nom` et `couleur` ne sont pas définies. Il aurait fallu écrire `self.nom + self.couleur` pour que cela soit correct.

Rappelons que la déclaration d'une méthode peut faire appel :

- à des variables locales (qui doivent être initialisées avant utilisation et qui sont « détruites » à la fin de la méthode) ;
- à des attributs de l'objet lui-même qui « appartiennent » à l'objet, qui sont partagés entre toutes les méthodes, et qui subsistent en mémoire jusqu'à ce que l'objet lui-même soit détruit. Dans ce cas, il faut y faire référence en utilisant le paramètre `self`, qui est le premier argument de toute méthode.

3 Réponse $\boxed{\text{b}}$. Ici, tout est correctement défini. Ainsi, la méthode `is_square` regarde si $a^2 + b^2$ est bien égal à c^2, ce qui est le cas pour notre exemple. « True » est donc affiché.

2 V/F Vérification des connaissances

Énoncé p. 14

1 *La programmation dynamique.*

(a) *Faux*. La mémoïsation, ou Top Down, est une forme récursive.

(b) *Faux*. La technique décrite dans la question est celle du Bottom Up. La programmation dynamique, quant à elle, consiste à découper un problème en plusieurs sous-problèmes de tailles inférieures et en utilisant l'une des techniques Bottom Up ou Top Down pour les résoudre.

(c) *Faux*. La programmation dynamique ne sert que lorsque la variable est entière.

(d) *Faux*. Nous avons vu dans le cours qu'il existait d'autres façons que la programmation dynamique pour résoudre le problème du rendu de monnaie, comme l'approche gloutonne ou diviser pour régner. Cependant, la programmation dynamique a pour objectif d'optimiser le code. C'est donc cette technique qui est préférable aux autres car elle nécessite moins d'opérations de calcul et de traitement que les autres.

2 *La Programmation Orientée Objet (POO).*

(a) *Faux*. Une *méthode* est une *fonction* d'une classe. Les *attributs* sont les variables de la classe.

(b) *Vrai*. On peut tout à fait définir une classe sans la méthode `__init__`. En voici un exemple :

```python
class toto:
    b = 5
    def fct(self,a): return a * self.b
d = toto()
print(d.fct(10))
```

qui renvoie « 50 ».

Le constructeur `__init__` est rendu obligatoire dès lors qu'un paramètre est passé lors de l'appel de la classe (par exemple, `d = toto(7)`) ou qu'il est nécessaire d'initialiser certains attributs dès la création de l'objet.

3 La classe « Piece »

Énoncé p. 15

Lycée Camille Jullian, Bordeaux

La classe `Piece` est composée de deux méthodes.

- La méthode `alea` retourne simplement un nombre aléatoire de l'ensemble {0; 1}.
- La méthode `moyenne`, qui admet un attribut `n`, commence par définir une liste vide nommée `tirage`. Ensuite, une boucle itérative est créée, pour 100 itérations. Au cours de chacune d'elles, on insère dans la liste `tirage` le retour de la méthode `alea` (on insère donc « 0 » ou « 1 » n fois).
  Une fois la boucle finie, on retourne la valeur de « `sum(tirage) / n` », c'est-à-dire la somme des nombres contenus dans la liste `tirage` divisée par n : il s'agit ici du calcul du nombre moyen de « 1 » obtenus lors des n choix aléatoires.

Ainsi, dans ce contexte, si on décide d'associer « Pile » à « 1 » lors du lancer d'une pièce, le programme calcule le nombre moyen de « Pile » sur 100 lancers.

4 Écrire une classe

Énoncé p. 15

Lycée Sophie Germain, Paris

1

```python
class Carre:
    def __init__(self,a):
        self.cote = a
    def perimetre(self):
        return self.cote*4
    def aire(self):
        return self.cote**2
```

COURS
INTERROS
CORRIGÉS

CORRIGÉS

2 Une classe possible est la suivante :

```
class Triangle:
    def __init__(self,a,b,c):
        self.a, self.b, self.c = a, b, c

    def aire(self):
        p = (self.a + self.b + self.c)/2
        return (p*(p-self.a)*(p-self.b)*(p-self.c))
    **0.5

    def rect(self):
        L = [self.a, self.b, self.c]
        L.sort()

        return L[2]**2 == L[0]**2 + L[1]**2
```

3 Définissons dans une console le carré et le triangle demandés.

```
>>> C = Carre(5)
>>> C.aire() , C.perimetre()
(25, 20)
>>> T = Triangle(5.4 , 9 , 7.2)
>>> (T.aire() , T.rect())
(19.440000000000001, True)
```

On voit ainsi que le triangle défini est rectangle et que son aire vaut 19,44.

Remarque : on pourrait améliorer légèrement l'affichage dans le cas où le triangle est rectangle.

En effet, dans ce cas précis, pour calculer l'aire du triangle, on peut utiliser la formule $\dfrac{b \times h}{2}$, où b et h sont respectivement une base du triangle et la hauteur relative à cette base.
Ainsi, dans la méthode `aire`, on pourrait écrire le code page ci-contre.

À RETENIR

Quel que soit le contexte d'un programme, il ne faut pas s'éloigner de ses règles. Si le programme concerne des mathématiques alors il faut utiliser les propriétés mathématiques pour optimiser le code.

```
class Triangle:
    def __init__(self,a,b,c):
        self.cotes = [a, b, c]
        self.cotes.sort()

    def rect(self):
        return self.cotes[2]**2 == self.cotes[0]**2
     + self.cotes[1]**2

    def aire(self):
        if self.rect():
            return self.cotes[0] * self.cotes[1] /
    2
        else:
            p = (self.cotes[0] + self.cotes[1] +
    self.cotes[2])/2
            return (p*(p-self.cotes[0])*(p-self.
    cotes[1])*(p-self.cotes[2]))**0.5
```

5 Devinette et remplissage

Énoncé p. 16

Lycée François Villon, Les Mureaux

1 `L.Liste` est *a priori* une liste (d'après la ligne : `self.Liste = [ ]`). Ainsi, l'affichage sera une liste. Regardons maintenant de plus près la définition de la fonction `add` :

- avant tout, si l'argument `x` est supérieur ou égal au dernier élément de la liste ou si la liste est vide, on ajoute la valeur de `x` à la liste ;
- sinon, si `x` contient une valeur strictement plus petite que le premier élément de la liste, on place la valeur de `x` en tête de la liste ;
- dans les autres cas, on parcourt la liste et on teste si la valeur contenue dans `x` est comprise entre deux valeurs de la liste. Si tel est le cas, on insère cette valeur entre les deux autres valeurs.

On peut alors dire que la fonction `add` ajoute la valeur de l'argument tout en faisant en sorte que la liste soit rangée dans l'ordre croissant.

Ainsi, le programme proposé affichera :

```
[7, 10, 50, 78, 89]
```

2 Voici une proposition pour la méthode `ecc` :

```
def ecc(self):
  E, prev = [], 0
  for nb in self.Liste:
    prev = prev + nb
    E.append(prev)
  return E
```

Ici, on commence par créer une liste `E` dont le seul élément est le premier de la liste `self.Liste`, puis on parcourt `self.Liste` à partir du deuxième élément : pour chaque nombre d'index `i`, on lui ajoute `E[i-1]` et on stocke le résultat à la fin de la liste `E`, donc dans `E[i]`.
La boucle terminée, on retourne `E`, qui est la liste de tous les cumuls.

Remarque : en statistiques, on appelle ces nombres les *effectifs cumulés croissants*.

6 Une classe « Livre »

Énoncé p. 16

Lycée François Mauriac, Bordeaux

1 **(a)** Dans le programme, il y a trois instances :

```
livre1 = Livre("1984", "George Orwell")
livre2 = Livre("La Ferme des animaux",
                "George Orwell")
livre3 = Livre("Le théorème du Perroquet",
                "Denis Guedj")
```

(b) La classe `Livre` a deux attributs : `titre` et `auteur`.

(c) La classe `Livre` a deux méthodes :

- `afficher_infos(self)` : une méthode qui affiche les informations du livre sous la forme « Titre : [titre], Auteur : [auteur] » ;
- `meme_auteur(self, autre_livre)` : une méthode qui prend un autre objet de type `Livre` en paramètre et retourne `True` si les deux livres ont le même auteur, sinon `False`.

2 Le programme affiche « False » car `livre1` et `livre3` n'ont pas le même auteur.

7 Compte bancaire

Énoncé p. 17

Lycée Richelieu, Versailles

1 Le constructeur de la classe peut être le suivant :

```
class Compte_bancaire:
    def __init__(self, solde_initial):
        self.solde = solde_initial
```

2 La méthode `etat_solde` peut être définie comme suit :

```
def etat_solde(self):
    if self.solde > 0:
        return "Positif"
    elif self.solde < 0:
        return "Négatif"
    else:
        return "Nul"
```

3 Le résultat est le suivant :

```
>>> A = Compte_bancaire(1000)
>>> A.etat_solde()
"Positif"
```

4 **(a)** La méthode `transférer` peut être définie ainsi :

```
def transferer(self, montant, destination):
    destination.solde += montant
    self.solde -= montant
```

(b) La méthode `prelever` peut être définie ainsi :

```
def prelever(self, montant, origine):
    origine.solde -= montant
    self.solde += montant
```

(c) La suite d'instructions peut être la suivante :

```
A = Compte_bancaire(1000)
B = Compte_bancaire(5000)
B.prelever(1100, A)
print(A.solde)
```

COURS INTERROS CORRIGÉS

CORRIGÉS

8 La classe « Temps »

Énoncé p. 18

Lycée Sainte-Clotilde, Strasbourg

Une définition possible de la classe Temps est la suivante :

```
class Temps:
    def __init__(self,h,m,s):
        self.h, self.m, self.s = h, m, s

    def affiche(self):
        return '{} h {} min {} s'.format(str(self.h),
    str(self.m), str(self.s))

    def add(self,t):
        sec = self.s + t.s
        rs = sec % 60
        minute = self.m + t.m + sec//60
        rm = minute % 60
        rh = self.h + t.h + minute//60
        return Temps(rh,rm,rs)

    def sub(self,t):
        a = self.h*3600 + self.m*60 + self.s
        b = t.h*3600 + t.m*60 + t.s
        diff = abs(b-a)
        h = diff // 3600
        m = (diff-3600*h)//60
        s = diff-3600*h-60*m
        return Temps(h,m,s)
```

Avec cette définition, voici ce que l'on peut obtenir :

```
>>> t1, t2 = Temps(8,45,37), Temps(15,31,18)
>>> (t1+t2).affiche(), (t1-t2).affiche()
'24 h 16 min 55 s' , '6 h 45 min 41 s'
```

9 Classes « Mot » et « Phrase »

Énoncé p. 18

Lycée Raoul Follereau, Nevers

1 L'instruction « `m = Mot("Socrate")` » définit dans un premier temps un objet `Mot` comme étant la chaîne de caractères « Socrate ».

Ensuite, « `m.doReverse()` » fait appel à la méthode `doReverse` de la classe `Mot` qui, comme son nom l'indique, transforme la chaîne de caractères en inversant les lettres.

Enfin, « `print(m.value())` » affiche ce qui est contenu dans `self.m` de la classe `Mot`, donc affiche ici : « etarcoS » (le mot écrit à l'envers).

2 Comme dans la fonction précédente, on commence par définir un objet de classe « Mot », puis on fait appel à la méthode `doShuffle` qui retourne une anagramme du mot.

La fonction `fctB()` affiche donc une anagramme de « Socrate ».

3 Un constructeur possible est le suivant :

```
class Phrase:
  def __init__(self,phrase):
    L = phrase.split()
    self.mots = []
    for m in L:
      self.mots.append(Mot(m))
```

On commence par définir une liste `L` composée de tous les mots de la phrase passée en paramètre avec la méthode `split` : à ce stade, les mots se sont que des chaînes de caractères. Or, nous souhaitons que chaque mot soit de classe `Mot`. On parcourt alors cette liste et, à partir de chaque item, on définit un objet de classe `Mot` que l'on insère dans une liste `self.mots` (« self » est indispensable pour lier cette nouvelle liste à l'objet de classe `Phrase` ainsi défini).

4 Une proposition de méthodes est la suivante :

```
def doReverse(self):
  self.mots = self.mots[::-1]

def value(self):
  result  = ''
  for m in self.mots:
    result += m.value() + ' '
  return result
```

Dans la méthode `doReverse`, nous n'avons fait qu'inverser l'ordre des items de la liste `self.mots` avec l'opération `[::-1]`, et dans la méthode `value`, nous avons initialisé une chaîne de caractères `result` comme étant vide, puis parcouru chaque item de la liste `self.mots` pour concaténer à `result` la chaîne de caractères correspondant à l'item (`m.value()`) en y ajoutant un espace afin qu'au final, `result` représente la phrase.

CORRIGÉS

5 Ici, nous pouvons utiliser à nouveau la fonction `shuffle` du module `random`, ce qui donne :

```
def doShuffle(self):
  shuffle(self.mots)
```

6 Dans la fonction `fctE()`, on commence par définir une phrase, puis un mot `m` comme étant l'objet `Mot` en position 3 dans la phrase définie. Ici, il s'agit donc de « sont » (attention : le premier mot est à la position 0, comme dans toute liste). Ensuite, on applique la méthode `doReverse` de la classe `Mot` au mot ainsi trouvé, ce qui a pour effet de mettre ses lettres dans le sens inverse.

On affiche ensuite la phrase ainsi obtenue, ce qui donne :

Tous les hommes tnos mortels

7 Dans la fonction `fctE()`, on commence par définir une phrase, puis on fait appel à la méthode `insert` qui insère le mot « ne » dans la phrase en position 3, c'est-à-dire après le 3e mot de la phrase. On fait de même en position 5, c'est-à-dire après le 5e mot : on y insère le mot « pas ». Et enfin, on affiche la phrase ainsi obtenue, ce qui donne :

Tous les hommes ne sont pas mortels

8 On commence par prendre le mot en position 4 dans la phrase, donc « mortels ». On mélange ensuite les lettres dans ce mot en faisant appel à la méthode `doShuffle` de la classe `Mot`. Ensuite, on fait appel à la méthode `remove` de la classe `Phrase` en passant en paramètre le nombre 2 : on supprime alors le mot « hommes ». Enfin, on insère en position 2 le contenu de la variable `m`, c'est-à-dire l'anagramme de « mortels » obtenu précédemment. On affiche alors le résultat, ce qui donne par exemple :

Tous les elmsort sont elmsort

Remarque : le dernier mot n'est plus « mortels » ; en effet, nous avons définit `m` comme l'objet en position 4 dans l'objet `p`. Dès lors que l'on écrit `m.doShuffle`, cet objet est modifié dans l'objet `p`. Ainsi, à ce stade, l'objet « mortels » n'existe plus mais il est remplacé par une anagramme de la chaîne de caractères qu'il représente.

Télécharger le programme complet à l'aide du QR code.

10 Personnage d'un jeu vidéo

Énoncé p. 21

Lycée André Theuriet, Civray

1 La classe `Personnage` peut être définie avec son constructeur de la manière suivante :

```
class Personnage:
    def __init__(self, nom, niveau, points_de_vie):
        self.nom = nom
        self.niveau = niveau
        self.points_de_vie = points_de_vie
```

2 (a)

```
def gagner_experience(self, experience):
        self.niveau += experience // 100
```

(b)

```
def est_vivant(self):
        return self.points_de_vie > 0
```

(c)

```
def afficher_etat(self):
        print(f"Nom : {self.nom}, Niveau : {self
    .niveau}, Points de vie : {self.
    points_de_vie}")
```

11 Découpe d'une chaîne

Énoncé p. 22

Lycée Sainte-Anne, Brest

Voici le programme dument complété :

```
def decouper_chaine(chaine, dictionnaire, memo={}):
    if chaine in memo:
        return memo[chaine]
    if chaine == "":
        return [[]]
    resultats = []
    for i in range(1, len(chaine) + 1):
        prefixe = chaine[:i]
        if prefixe in dictionnaire:
            for suffixe_decoupe in decouper_chaine(
    chaine[i:], dictionnaire, memo):
                resultats.append([prefixe] +
    suffixe_decoupe)
    memo[chaine] = resultats
    return resultats
```

CORRIGÉS

12 Modélisation d'une suite numérique

Énoncé p. 23

Lycée François Mauriac, Bordeaux

Une version Bottom Up du programme dynamique est la suivante :

Bottom Up

```
def u(n):
    U = [ 0 ] * (n+1)
    for i in range(n+1):
        if i == 0: U[i] = 1
        elif i == 1: U[i] = 2
        else:
            U[i] = U[i-2] / U[i-1]
    return U[n]
```

Une version optimisée (en espace mémoire) est la suivante :

Bottom Up (optimisée en espace mémoire)

```
def u(n):
    u,v = 1,2
    for i in range(2,n+1):
        if n == 0: return u
        elif n == 1: return v
        else:
            w = u / v
            u = v
            v = w
    return w
```

Une version Top Down est la suivante :

Top Down

```
def u(n, U = None):
    if n == 0: return 1
    elif n == 1: return 2
    else:
        U = [None] * (n+1)
        U[0], U[1] = 1, 2
        return u_cache(U,n)

def u_cache(U,n):
    if U[n] is None:
        U[n] = u(n-2, U) / u(n-1, U)
    return U[n]
```

Une manière plus simple peut aussi être envisagée :

Top Down

```
def u(n, U = None):
  if U is None or U == []:
    U = [1,2] + [None] * (n-1)
  if U[n] is None:
    U[n] = u(n-2, U) / u(n-1, U)
  return U[n]
```

13 Somme dans une suite numérique

Énoncé p. 23

Lycée polyvalent Émile Zola, Aix-en-Provence

1 Bien que mathématique, et non informatique, cette question permet de comprendre le mode de génération des termes de la suite définie.

- $u_1 = u_{0+1} = \dfrac{1}{0+1}\displaystyle\sum_{k=0}^{0}(k+1)u_k = u_0 = 1\,;$
- $u_2 = u_{1+1} = \dfrac{1}{1+1}\displaystyle\sum_{k=0}^{1}(k+1)u_k = \dfrac{1}{2}(1u_0 + 2u_1) = \dfrac{3}{2}\,;$
- $u_3 = u_{2+1}$

$$= \frac{1}{2+1}\sum_{k=0}^{2}(k+1)u_k$$

$$= \frac{1}{3}(1u_0 + 2u_1 + 3u_2)$$

$$= \frac{1}{3} \times \left(3 + \frac{9}{2}\right) = \frac{5}{2}.$$

2 La difficulté de ce genre de calcul réside dans la sommation. Il faut penser à la stocker au même titre que les termes de la suite.

Bottom Up

```
def u(n):
    U = [None] * (n+1)
    U[0] = (1,0)
    for k in range(1,n+1):
        S = U[k-1][1] + k * U[k-1][0]
        U[k] = (S/k, S)
    return U[n][0]

for n in range(20):
    print (u(n))
```

CORRIGÉS

L'idée ici est donc de stocker non pas un terme mais un couple constitué du terme u_k et de la somme $u_0 + 2u_1 + \cdots + (k+1)u_k$ pour pouvoir calculer par la suite u_{k+1}.

Top Down

```
def u(n, U = None):
    if U is None or U == []:
        U = [(1,0)] + [(None,None)] * n
    if U[n][0] is None:
        val = u(n-1, U)[1] + n * u(n-1, U)[0]
        U[n] = (val / n, val)
    return U[n]

for n in range(20):
    print(u(n)[0])
```

14 Le retour du problème du sac à dos

Énoncé p. 23

Lycée Jean-Joseph Fourier, Auxerre

1 Rappelons avant tout la fonction récursive initiale du problème :

```
def sac(E,P,i):
    if i == 0:
        return 0
    if E[i-1][0] > P:
        return sac(E, P, i-1)
    else:
        return max(sac(E, P, i-1), E[i-1][1] + sac(
    E, P-E[i-1][0], i-1))

def initSac(E,P):
    return sac(E, P, len(E))
```

- *Approche dynamique de type « Top Down ».*

  Pour améliorer cette fonction récursive, il nous faut mémoriser les solutions aux sous-problèmes (mémoïsation). Nous allons donc stocker les valeurs de `V[i][p]`, avec $0 \leqslant i \leqslant n$ et $0 \leqslant p \leqslant P$, où P est le poids maximal autorisé, dans une liste nommée par exemple `temp`.
  La solution est très proche de l'initiale.

```
def sac(E,P,i,temp):
    if temp[i][P]>0: return temp[i][P]
    if i == 0: return 0
    if E[i-1][0] > P:
        temp[i][P] = sac(E, P, i-1,temp)
        return temp[i][P]
    else:
        temp[i][P] = max(sac(E, P, i-1, temp), E[i
    -1][1] + sac(E, P - E[i-1][0], i-1, temp))
        return temp[i][P]

def initSac(E,P):
    temp = [ [0] * (P+1) for i in range(len(E)+1)
    ]
    return sac(E, P, len(E), temp)
```

Ce qui donne pour notre exemple :

```
>>> E, P = [ (5,7) , (3,1) , (3,4) , (3,2) ] , 10
>>> initSac(E,P)
11
```

Cela signifie que la valeur maximale des objets que l'on peut mettre dans le sac est égale à 11 ($7+4$, avec un poids de $5+3=8$).

- *Approche dynamique de type « Bottom Up ».*

```
def sac(E,P):
    temp = [ [0] * (P+1) for i in range(len(E)+1)]
    for i in range(1,len(E)+1):
        for p in range(P+1):
            if E[i-1][0] > p:
                temp[i][p] = temp[i-1][p]
            else:
                temp[i][p] = max(temp[i-1][p], E[i
    -1][1] + temp[i-1][ p - E[i-1][0] ])
    return temp[len(E)][P]
```

Avec cette approche, il n'y a plus de récursivité mais une itération. On garde à peu près la même structure pour la fonction `sac`.

On obtient bien entendu le même résultat, à savoir « 11 ».

COURS
INTERROS
CORRIGÉS

CORRIGÉS

2 Comme nous pouvons le voir ici, ces solutions n'affichent que la valeur optimale. Le problème du sac à dos étant d'obtenir la combinaison des objets retenus, il nous faut compléter ces solutions.

Si on complète l'approche Bottom up, cela donne le programme suivant :

```
def sac(E,P):
    temp = [ [0] * (P+1) for i in range(len(E)+1)]
    garde  = [ [False] * (P+1) for i in range(len(E)
    +1) ]
    for i in range(1,len(E)+1):
        for p in range(P+1):
            if E[i-1][0] > p:
                temp[i][p] = temp[i-1][p]
            else:
                if E[i-1][1] + temp[i-1][p-E[i
    -1][0]] > temp[i-1][p]:
                    temp[i][p] =  E[i-1][1] + temp[
    i-1][ p - E[i-1][0] ]
                    garde[i][p] = True
                else:
                    temp[i][p] = temp[i-1][p]
    p,L = P,[]
    for i in range(len(E),0,-1):
        if garde[i][p]:
            L.append(i)
            p -= E[i-1][0]

    return temp[len(E)][P], L
```

Ici, on ajoute une liste `garde` qui stocke le numéro des objets que l'on met dans le sac.

3 Dans l'approche récursive initiale, la complexité est en $O\left(2^n\right)$. En effet, deux appels récursifs sont fait pour n objets.

Dans l'approche Bottom Up, elle est en $O(n \times P)$ car il y a deux boucles imbriquées portant sur n et P, et qu'il y a le même nombre d'opérations élémentaires dans chacune d'elles.

ATTENTION

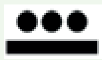
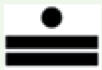
Il ne faut pas conclure que la complexité est linéaire par rapport à P. En effet, la variable P est de taille $\log_2(P)$ et on peut alors écrire que le poids maximal est $P = 2^{\log_2(\mathrm{P})}$. La complexité du programme dynamique est alors exponentielle par rapport à la taille de P. Cependant, elle reste meilleure que celle du programme initial.

15 La civilisation Maya

Énoncé p. 24

Extrait du Bac 2024, Métropole, septembre

1 Le tableau complété est le suivant :

Étage	Écriture Maya	Valeur du « chiffre » de l'étage	Valeur dans la conversion
3		$1 \times 5 + 3 \times 1 = 8$	$8 \times 20^2 = 3\,200$
2		$2 \times 5 + 1 \times 1 = 11$	$11 \times 20^1 = 220$
1		$3 \times 5 = 15$	$15^2 0^0 = 15$

2 En additionnant les nombres trouvés précédemment, on a :

$$3\,200 + 220 + 15 = 3\,435.$$

3 Les instructions pour créer une instance de la classe `Maya` modélisant le nombre 3 435 sont :

```
M = Maya()
M.ajouter([0, 0, 3])
M.ajouter([0, 1, 2])
M.ajouter([0, 3, 1])
```

4 La méthode `nbEtages` de la classe `Maya` est :

```
def nbEtages(self):
    return len(self.nombre)
```

5 La fonction `valeurChiffre` est :

```
def valeurChiffre(L):
    return L[1] + 5 * L[2]
```

6 La méthode `MayaToDec` complétée est :

```
def MayaToDec(self):
    coeff = 20**...
    ch_Dec = 0
    while ... :
        ch_Maya = ...
        ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) *
 coeff
        coeff = ...
    return ch_Dec
```

COURS
INTERROS
CORRIGÉS

7 La fonction `decompChiffre` est :

```
def decompChiffre(n):
    if n == 0:
        return [1, 0, 0]
    return [0, n % 5, n // 5]
```

8 La fonction `DecToMaya` est :

```
def DecToMaya(n):
    M = Maya()
    for coeff in DecToVige(n):
        M.ajouter(decompChiffre(coeff))
    return M
```

9 La méthode `multiplie` de la classe `Maya` est :

```
def multiplie(self):
    result = Maya()
    result.nombre = [[1,0,0]] + self.nombre
    return result
```

10 Les résultats renvoyés par les appels de la fonction `mystere` sont :

- `mystere([0, 1, 1], [0, 3, 1], 0)` renvoie `([0, 4, 2], 0)`
- `mystere([0, 1, 1], [0, 4, 2], 0)` renvoie `([1, 0, 0], 1)`

Cette fonction renvoie en effet la somme de deux « chiffres » Mayas.

11 La méthode `somme` de la classe `Maya` est :

```
def somme(self, maya2):
    if self.nbEtages() != maya2.nbEtages():
        return None
    result = Maya()
    ret = 0
    while not self.estVide():
        ch1 = self.retirer()
        ch2 = maya2.retirer()
        ch_somme, ret = mystere(ch1, ch2, ret)
        result.ajouter(ch_somme)
    return result
```

Téléchargez la classe complète.

Annexe

Liste des vidéos

Chapitre **Titre** **Page**