COURS PI

L'école sur-mesure

de la Maternelle au Bac, Établissement d'enseignement privé à distance, déclaré auprès du Rectorat de Paris

Terminale - Module 3 - Programmation et algorithmique approfondies

Numérique et Sciences Informatiques

v.5.1

- ***Guide de méthodologie*** *pour appréhender notre pédagogie*
- ***Leçons détaillées*** *pour apprendre les notions en jeu*
- ***Exemples et illustrations*** *pour comprendre par soi-même*
- ***Prolongement numérique*** *pour être acteur et aller + loin*
- ***Exercices d'application*** *pour s'entraîner encore et encore*
- ***Corrigés des exercices*** *pour vérifier ses acquis*

www.cours-pi.com

Paris & Montpellier

EN ROUTE VERS LE BACCALAURÉAT

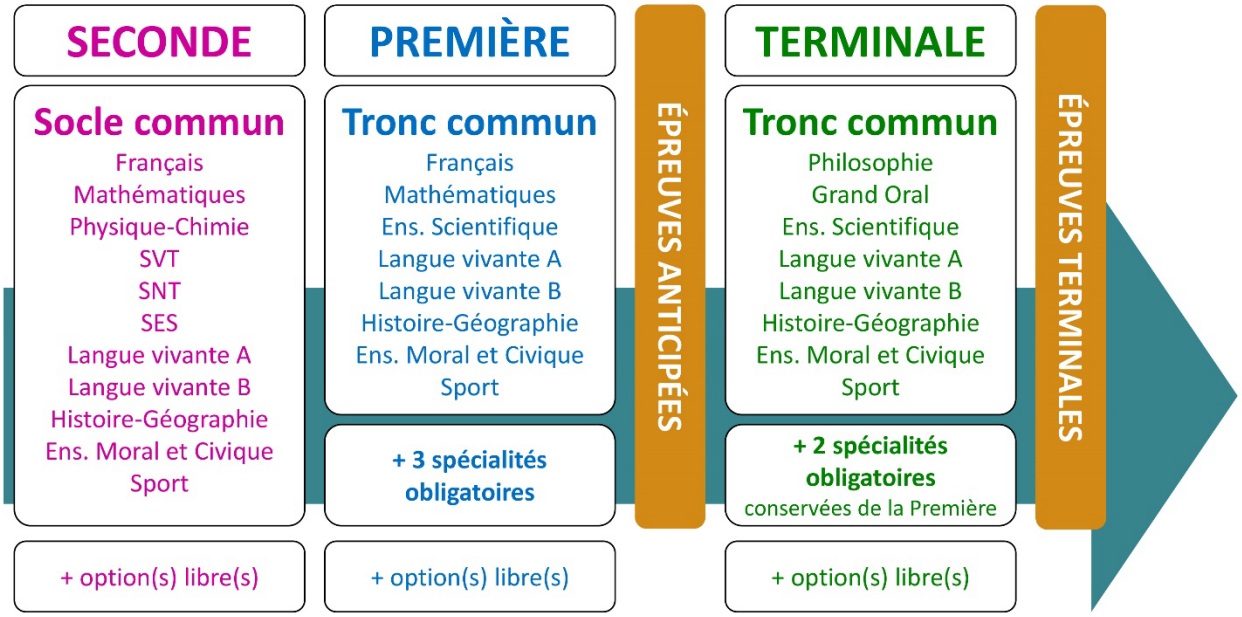
Comme vous le savez, **la réforme du Baccalauréat** est entrée en vigueur progressivement jusqu'à l'année 2021, date de délivrance des premiers diplômes de la nouvelle formule.

Dans le cadre de ce nouveau Baccalauréat, **notre Etablissement**, toujours attentif aux conséquences des réformes pour les élèves, s'est emparé de la question avec force **énergie** et **conviction** pendant plusieurs mois, animé par le souci constant de la réussite de nos lycéens dans leurs apprentissages d'une part, et par la **pérennité** de leur parcours d'autre part. Notre Etablissement a questionné la réforme, mobilisé l'ensemble de son atelier pédagogique, et déployé tout **son savoir-faire** afin de vous proposer un enseignement tourné continuellement vers l'**excellence**, ainsi qu'une scolarité tournée vers la **réussite**.

- Les **Cours Pi** s'engagent pour faire du parcours de chacun de ses élèves un **tremplin vers l'avenir**.
- Les **Cours Pi** s'engagent pour ne pas faire de ce nouveau Bac un diplôme au rabais.
- Les **Cours Pi** vous offrent **écoute** et **conseil** pour coconstruire une **scolarité sur-mesure**.

LE BAC DANS LES GRANDES LIGNES

Ce nouveau Lycée, c'est un enseignement à la carte organisé à partir d'un large tronc commun en classe de Seconde et évoluant vers un parcours des plus spécialisés année après année.



CE QUI A CHANGÉ

- Il n'y a plus de séries à proprement parler.
- Les élèves choisissent des spécialités : trois disciplines en classe de Première ; puis n'en conservent que deux en Terminale.
- Une nouvelle épreuve en fin de Terminale : le Grand Oral.
- Pour les lycéens en présentiel l'examen est un mix de contrôle continu et d'examen final laissant envisager un diplôme à plusieurs vitesses.
- Pour nos élèves, qui passeront les épreuves sur table, le Baccalauréat conserve sa valeur.

CE QUI N'A PAS CHANGÉ

- Le Bac reste un examen accessible aux candidats libres avec examen final.
- Le système actuel de mentions est maintenu.
- Les épreuves anticipées de français, écrit et oral, tout comme celle de spécialité abandonnée se dérouleront comme aujourd'hui en fin de Première.

A l'occasion de la réforme du Lycée, nos manuels ont été retravaillés dans notre atelier pédagogique pour un accompagnement optimal à la compréhension. Sur la base des programmes officiels, nous avons choisi de créer de nombreuses rubriques :

- **Observe, word bank et l'essentiel** pour souligner les points de cours à mémoriser au cours de l'année
- **À vous de jouer** pour mettre en pratique le raisonnement vu dans le cours et s'accaparer les ressorts de l'analyse, de la logique, de l'argumentation, et de la justification
- **Pour aller plus loin** pour visionner des sites ou des documentaires ludiques de qualité
- Et enfin ... la rubrique **Les Clés du Bac by Cours Pi** qui vise à vous donner, et ce dès la seconde, toutes les cartes pour réussir votre examen : notions essentielles, méthodologie pas à pas, exercices types et fiches étape de résolution !

NUMÉRIQUE ET SCIENCES INFORMATIQUES TERMINALE

Module 3 – Programmation et algorithmique approfondies

L'AUTEUR

Adrien SAURAT

« L'enseignement, c'est favoriser l'autonomie et l'enrichissement des élèves, avec en autres objectifs, apprendre un métier. » Professeur et formateur en informatique avec plus de douze ans d'expérience en développement web et dans l'animation du réseau Canopé, il se passionne aussi pour le théâtre et l'écriture de nouvelles.
Des passions qui l'ont déjà conduit sur les planches du Festival d'Avignon.

PRÉSENTATION

Ce **cours** est divisé en chapitres, chacun comprenant :

- Le **cours**, conforme aux programmes de l'Education Nationale
- Des **applications** dont les **corrigés** se trouvent en **fin de chapitre**
- Des **exercices d'entraînement** et leurs **corrigés** en **fin de fascicule**
- Des **devoirs** soumis à correction (et ***se trouvant hors manuel***). Votre professeur vous renverra le corrigé-type de chaque devoir après correction de ce dernier.

Pour une manipulation plus facile, les corrigés-types des exercices d'application et d'entraînement sont regroupés en fin de manuel.

CONSEILS A L'ÉLÈVE

Vous disposez d'un support decours complet : **prenez le temps** de bien le lire, de le comprendre mais surtout de l'**assimiler**. Vous disposez pour cela d'exemples donnés dans le cours et d'exercices types corrigés.
Vous pouvez rester un peu plus longtemps sur une unité mais travaillez régulièrement.

LES DEVOIRS

Les devoirs constituent le moyen d'évaluer l'acquisition de vos *savoirs* (« Ai-je assimilé les notions correspondantes ? ») et de vos *savoir-faire* (« Est-ce que je sais expliquer, justifier, conclure ? »).

Placés à des endroits clés des apprentissages, ils permettent la vérification de la bonne assimilation des enseignements.

Aux *Cours Pi*, vous serez accompagnés par un professeur selon chaque matière tout au long de votre année d'étude. Référez-vous à votre « Carnet de Route » pour l'identifier et découvrir son parcours.
Avant de vous lancer dans un devoir, assurez-vous d'avoir **bien compris les consignes**.
Si vous repérez des difficultés lors de sa réalisation, n'hésitez pas à le mettre de côté et à revenir sur les leçons posant problème. **Le devoir n'est pas un examen**, il a pour objectif de s'assurer que, même quelques jours ou semaines après son étude, une notion est toujours comprise.
Aux *Cours Pi*, chaque élève travaille à son rythme, parce que chaque élève est différent et que ce mode d'enseignement permet le « sur-mesure ».
Nous vous engageons à respecter le moment indiqué pour faire les devoirs. Vous les identifierez par le bandeau suivant :

Vous pouvez maintenant
faire et envoyer le devoir n°1

Il est **important de tenir compte des remarques, appréciations et conseils du professeur-correcteur**. Pour cela, il est **très important d'envoyer les devoirs au fur et à mesure** et non groupés. **C'est ainsi que vous progresserez !**

Donc, dès qu'un devoir est rédigé, envoyez-le aux *Cours Pi* par le biais que vous avez choisi :

1) Par **soumission en ligne** via votre espace personnel sur **PoulPi**, pour un envoi **gratuit**, **sécurisé** et plus **rapide**.
2) Par **voie postale** à *Cours Pi*, 9 rue Rebuffy, 34 000 Montpellier
   *Vous prendrez alors soin de joindre une **grande enveloppe libellée à vos nom et adresse**, et **affranchie au tarif en vigueur** pour qu'il vous soit retourné par votre professeur*

***N.B. :** quel que soit le mode d'envoi choisi, vous veillerez à **toujours joindre l'énoncé du devoir ;** plusieurs énoncés étant disponibles pour le même devoir.*
***N.B. :** si vous avez opté pour un envoi par voie postale et que vous avez à disposition un scanner, nous vous engageons à conserver une copie numérique du devoir envoyé. Les pertes de courrier par la Poste française sont très rares, mais sont toujours source de grand mécontentement pour l'élève voulant constater les fruits de son travail.*

SOUTIEN ET DISPONIBILITÉ

VOTRE RESPONSABLE PÉDAGOGIQUE

Professeur des écoles, professeur de français, professeur de maths, professeur de langues : notre Direction Pédagogique est constituée de spécialistes capables de dissiper toute incompréhension.

Au-delà de cet accompagnement ponctuel, notre Etablissement a positionné ses Responsables pédagogiques comme des « super profs » capables de co-construire avec vous une scolarité sur-mesure.
En somme, le Responsable pédagogique est votre premier point de contact identifié, à même de vous guider et de répondre à vos différents questionnements.

Votre Responsable pédagogique est la personne en charge du suivi de la scolarité des élèves.
Il est tout naturellement votre premier référent : une question, un doute, une incompréhension ? Votre Responsable pédagogique est là pour vous écouter et vous orienter. Autant que nécessaire et sans aucun surcoût.

QUAND PUIS-JE LE JOINDRE ?

Du lundi au vendredi : horaires disponibles sur votre carnet de route et sur PoulPi.

QUEL EST SON RÔLE ?

Orienter les parents et les élèves.
Proposer la mise en place d'un accompagnement individualisé de l'élève.
Faire évoluer les outils pédagogiques.
Encadrer et coordonner les différents professeurs.

VOS PROFESSEURS CORRECTEURS

Notre Etablissement a choisi de s'entourer de professeurs diplômés et expérimentés, parce qu'eux seuls ont une parfaite connaissance de ce qu'est un élève et parce qu'eux seuls maîtrisent les attendus de leur discipline. En lien direct avec votre Responsable pédagogique, ils prendront en compte les spécificités de l'élève dans leur correction. Volontairement bienveillants, leur correction sera néanmoins juste, pour mieux progresser.

QUAND PUIS-JE LE JOINDRE ?

Une question sur sa correction ?

- faites un mail ou téléphonez à votre correcteur et demandez-lui d'être recontacté en lui laissant un message avec votre nom, celui de votre enfant et votre numéro.
- autrement pour une réponse en temps réel, appelez votre Responsable pédagogique.

LE BUREAU DE LA SCOLARITÉ

Placé sous la direction d'Elena COZZANI, le Bureau de la Scolarité vous orientera et vous guidera dans vos démarches administratives. En connaissance parfaite du fonctionnement de l'Etablissement, ces référents administratifs sauront solutionner vos problématiques et, au besoin, vous rediriger vers le bon interlocuteur.

QUAND PUIS-JE LE JOINDRE ?

Du lundi au vendredi : horaires disponibles sur votre carnet de route et sur PoulPi.
04.67.34.03.00
scolarite@cours-pi.com

LE SOMMAIRE

CHAPITRE 1. Modèles et paradigmes de programmation 1

OBJECTIFS

- Comprendre que tout programme est aussi une donnée.
- Comprendre que la calculabilité ne dépend pas du langage de programmation utilisé.
- Montrer, sans formalisme théorique, que le problème de l'arrêt est indécidable
- Écrire un programme récursif.
- Analyser le fonctionnement d'un programme récursif.
- Utiliser des API (Application Programming Interface) ou des bibliothèques. Exploiter leur documentation.
- Créer des modules simples et les documenter.
- Distinguer sur des exemples les paradigmes impératif, fonctionnel et objet.
- Choisir le paradigme de programmation selon le champ d'application d'un programme.

COMPÉTENCES VISÉES

- La nature d'un programme.
- Le concept de récursivité.
- La modularité en programmation (grâce aux API ou bibliothèques).
- Différents paradigmes de programmation.

Première approche 2
1. Qu'est-ce qu'un programme ? 4
2. Récursivité 6
3. Modularité 9
4. Paradigmes 14
Le temps du bilan 17
Les Clés du Bac 20

CHAPITRE 2. Algorithmes avancés 29

OBJECTIFS

- Calculer la taille et la hauteur d'un arbre.
- Parcourir un arbre de différentes façons.
- Rechercher/insérer une clé dans un arbre de recherche.
- Parcourir un graphe en profondeur d'abord, en largeur d'abord.
- Repérer la présence d'un cycle dans un graphe.
- Chercher un chemin dans un graphe.
- Écrire un algorithme utilisant la méthode « diviser pour régner ».
- Utiliser la programmation dynamique pour écrire un algorithme.
- Étudier l'algorithme de Boyer-Moore pour la recherche d'un motif dans un texte.

COMPÉTENCES VISÉES

- Le concept et l'utilisation d'un arbre binaire.
- L'exploration de graphes.
- La méthode « Diviser pour régner ».
- Comment utiliser la programmation dynamique.
- La recherche d'un motif dans un texte.

Première approche 30
1. Arbres binaires 34
2. Graphes 43
3. La méthode « diviser pour régner » 47
4. Programmation dynamique 48
5. Recherche textuelle 50
Le temps du bilan 54
Les Clés du Bac 56

CHAPITRE 3. Mise au point des programmes et gestion des bugs 69

OBJECTIFS

Dans la pratique de la programmation, savoir répondre aux causes typiques de bugs comme :

- les problèmes liés au typage ;
- les effets de bord non désirés ;
- les débordements dans les tableaux ;
- l'instruction conditionnelle non exhaustive ;
- le choix des inégalités ;
- les comparaisons et calculs entre flottants ou mauvais nommage des variables.

COMPÉTENCES VISÉES

- Savoir mettre au point un programme.
- Connaître des méthodes de gestion des bugs.

Première approche 70
1. Mise au point des programmes 74
2. Anticiper les erreurs 79
Le temps du bilan 82
Les Clés du Bac 83

CORRIGÉS à vous de jouer et exercices 89

SUGGESTIONS CULTURELLES

OUVRAGES

- **Python pour les Nuls,** *John Paul MUELLER*
- **Apprendre à programmer avec Python**, *Gérard Swinnen*

FILMS ET SERIES

- **Office Space**, *Mike Judge*
- **Jobs**,*Joshua Michael Stern*
- **Snowden**, *Oliver Stone*
- **The Imitation Games**, *Morten Tyldum*

DOCUMENTAIRES AUDIOVISUELS

- **Citizenfour**, *Laura Poitras*
- **Les ordinateurs du passé**, *Le Vortex*
- **1ère et Terminale NSI - Mise au point de programmes et gestion des bugs en Python,** *Cédric GERLAND*

SITES ET OUTILS EN LIGNE

- **www.deepdreamgenerator.com**
- **www.code.org**
- **www.openclassrooms.com**

PRÉSENTATION DE L'ÉPREUVE

Le bac approche, voilà de quoi vous y préparer !

Dans le cadre d'une scolarisation à la maison, une partie de l'épreuve (dite « épreuve pratique ») est annulée. Vous n'aurez donc qu'à travailler l'épreuve dite « écrit ». En quoi consiste-t-elle ?

Le texte officiel annonce ceci :

La partie écrite (d'une durée de 3 heures 30) consiste en **la résolution de trois exercices permettant d'évaluer les connaissances et les capacités attendues conformément aux programmes de première et de terminale de la spécialité**. Chaque exercice est noté sur 4 points.

Le sujet propose cinq exercices, parmi lesquels le candidat choisit les trois qu'il traitera. Ces cinq exercices permettent d'aborder les différentes rubriques du programme, sans obligation d'exhaustivité. Le sujet comprend obligatoirement au moins un exercice relatif à chacune des trois rubriques suivantes :

- traitement de données en tables et bases de données ;
- architectures matérielles, systèmes d'exploitation et réseaux ;
- algorithmique, langages et programmation.

Note importante : les trois exercices mentionnés totalisent 12 points, car dans le cadre d'une épreuve complète, la partie pratique apporte les 8 points supplémentaires. **Dans notre cas, cette note de 12 points sera rapportée à 20 points pour représenter l'intégralité de votre résultat.**

Que peut-on conclure de ce texte ?

Trois exercices à choisir parmi cinq, qui couvrent tout le programme NSI (première et terminale), sans oublier le fait que le programme SNT de seconde pose des bases qu'il est bon de connaître (portes logiques, binaire, etc.). Ces cinq exercices proposés couvrent des domaines variés, ce qui vous permettra de trouver au moins un sujet ou deux sur lesquels vous serez peut-être plus à l'aise. **Mais bien sûr, il n'est pas prudent de négliger certaines parties du programme, car elles sont souvent liées !**


Le Cours

CHAPITRE 1
MODÈLES ET PARADIGMES DE PROGRAMMATION

Les programmes informatiques, outre le fait qu'ils sont codés dans différents langages (assembleur, C, C++, rubis, perle, JavaScript, PHP, python, go, s'écharpe, etc.), sont aussi écrits suivant différents paradigmes et organisations. Ainsi, suivant l'application que l'on souhaite écrire, on utilisera une programmation fonctionnelle ou objet, pour ne citer qu'un choix possible parmi d'autres.

Souvent, on sera aussi amené à faire appel à des interfaces ou bibliothèques externes permettant de tirer parti de fonctions que l'on n'a pas soi-même codées. **Cette modularité permet un gain de temps considérable dans l'écriture de certains programmes**, puisque ce sont autant de lignes de code sur lequel nous n'aurons pas à passer trop de temps.

Dans ce chapitre, nous aborderons ces aspects et nous compléterons ce tour d'horizon en abordant des notions comme la récursivité (lorsqu'une fonction s'appelle elle-même) ou la calculabilité d'un programme.

OBJECTIFS

- Comprendre que tout programme est aussi une donnée.
- Comprendre que la calculabilité ne dépend pas du langage de programmation utilisé.
- Montrer, sans formalisme théorique, que le problème de l'arrêt est indécidable.
- Écrire un programme récursif.
- Analyser le fonctionnement d'un programme récursif.
- Utiliser des API (Application Programming Interface) ou des bibliothèques. Exploiter leur documentation.
- Créer des modules simples et les documenter.
- Distinguer sur des exemples les paradigmes impératif, fonctionnel et objet.
- Choisir le paradigme de programmation selon le champ d'application d'un programme.

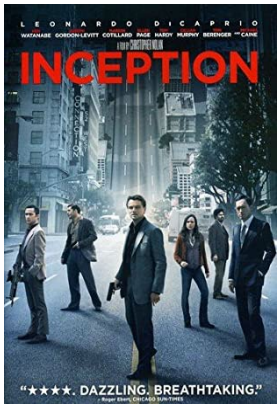
COMPÉTENCES VISÉES

- La nature d'un programme.
- Le concept de récursivité.
- La modularité en programmation (grâce aux API ou bibliothèques).
- Différents paradigmes de programmation.

MATÉRIEL NÉCESSAIRE

- Un ordinateur pouvant exécuter Python.

Première approche



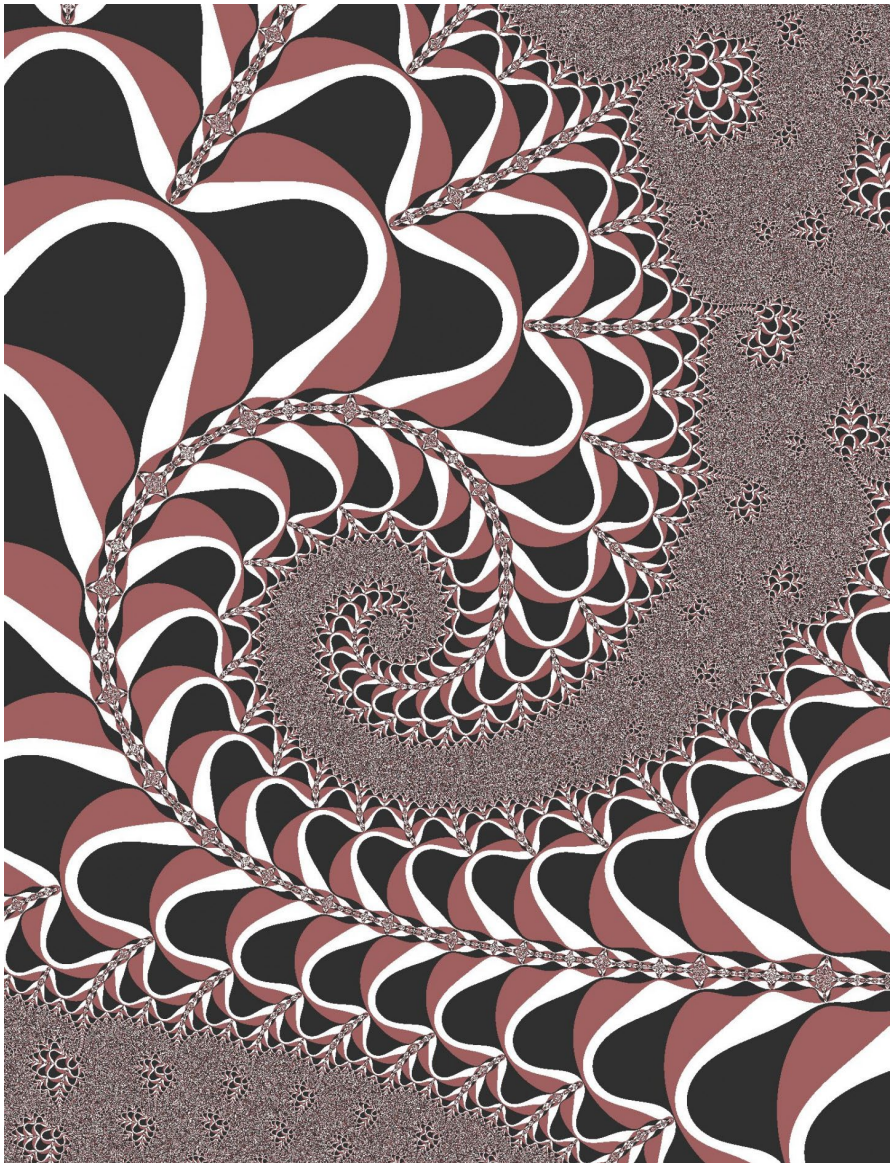
Avez-vous vu le film Inception, sorti en 2010 et réalisé par Christopher Nolan ?

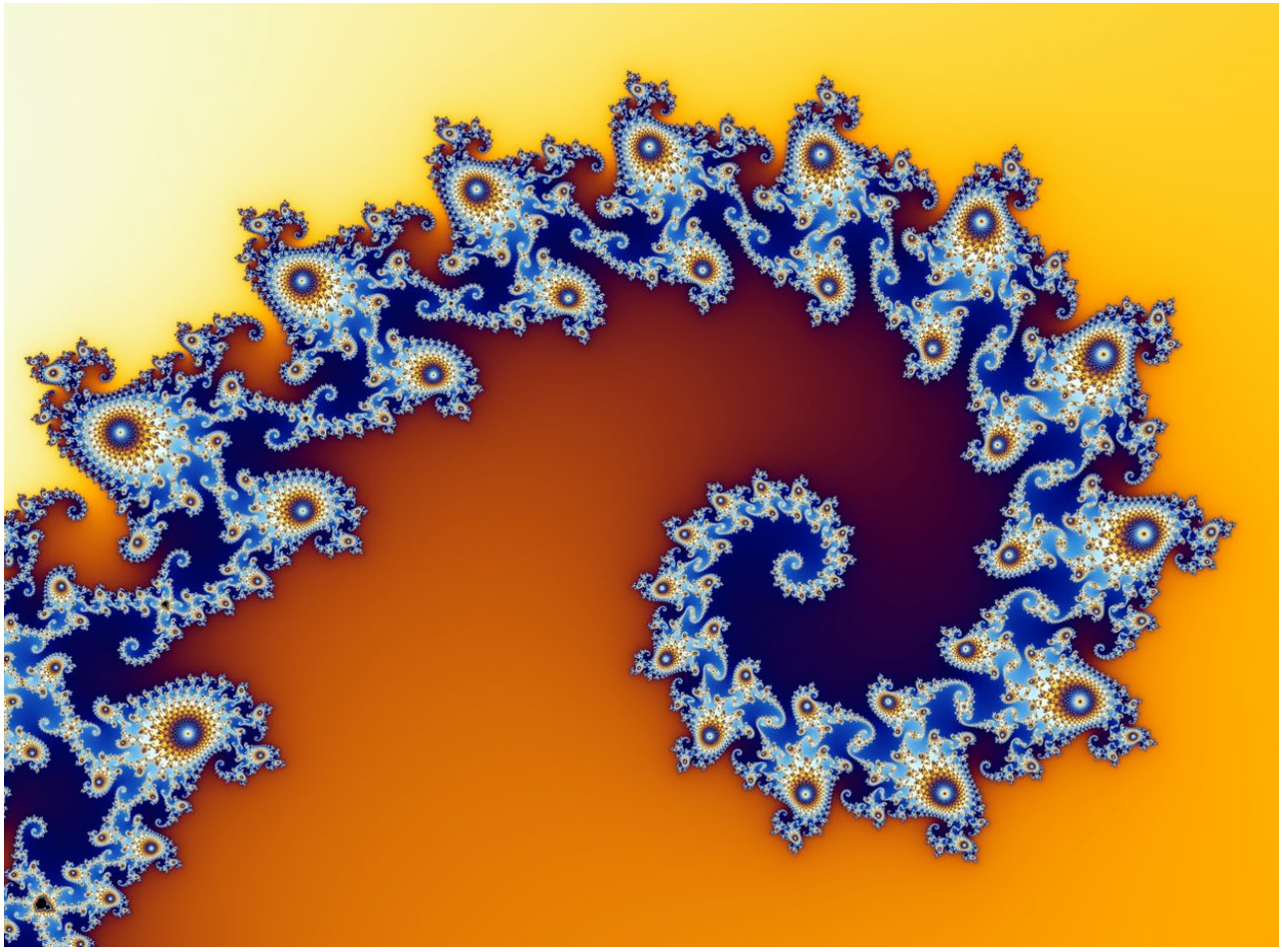
On y retrouve des couches d'événements durant lesquels les protagonistes entrent dans des sous-histoires à répétition, puis les quittent en suivant un ordre logique (ils ferment la dernière histoire ouverte, comme on résout des parenthèses en mathématiques).

Gardez en tête cet aspect du scénario, car nous étudierons dans ce chapitre une notion appelée récursivité et qui partage quelques caractéristiques avec cette architecture !

Mais à vrai dire, nous pouvons déjà l'anticiper un peu plus, en jetant un œil du côté d'une autre curiosité toujours liée aux mathématiques : les fractales ! En avez-vous déjà entendu parler ? Une fractale présente une structure tout à fait particulière.

En voici deux exemples :





Voyez-vous ce qui caractérise une fractale ?

Peut-être arriverez-vous à le discerner grâce à cette image fixe, mais si ce n'est pas le cas n'hésitez pas à en chercher une vidéo sur internet, ce qui pourrait vous permettre de mieux visualiser leur caractère unique.

CORRECTION :

Une fractale présente une structure similaire, quelle que soit l'échelle à laquelle on la considère ! Si l'on « zoome » ou « dézoome » en l'observant, on retrouve toujours inévitablement les mêmes formes.
Pour voir de plus près comment est constituée une fractale simple, à même de mieux permettre de saisir ces subtilités, n'hésitez pas à chercher des informations quant au « flocon de Koch » ! Il est facile d'en trouver des versions animées, qui permettent de mieux se rendre compte de la construction répétitive de ce genre de structures.

Le principe de récursivité, que nous examinerons dans ce chapitre, repose sur une mécanique similaire.

MODÈLES ET PARADIGMES DE PROGRAMMATION

Qu'est-ce qu'un programme ?

Qu'est-ce qu'un programme ?

Nous avons déjà vu dans un autre module que les processeurs de nos machines étaient faits pour interpréter des suites binaires. En effet les commandes de base sont constituées de zéros et de uns. **Évidemment il n'est pas possible pour des humains de coder directement dans un tel langage, raison pour laquelle ont été inventés des langages de niveau supérieur.**

L'assembleur fut l'un des premiers. Ce qu'on appelle toutefois un langage de bas niveau, très proche des spécificités du processeur qu'il est supposé faire fonctionner. D'autres langages de niveau supérieur ont donc vu le jour, ce qui a permis de développer du code de plus en plus indépendant de la machine ciblée.

En effet, si l'on code par exemple en python, en dehors de quelques ajustements on pourra programmer de la même façon pour une machine tournant sous Windows, Mac ou Linux. Ceci est permis par l'utilisation d'un interpréteur ou d'un compilateur. Dans le cas de python, il s'agit d'un interpréteur, c'est-à-dire que le code est lu et exécuté au moment où on demande à python de faire. Pour d'autres langages, comme le C++, on a recours à un compilateur qui va transformer notre programme en langage machine exécutable. Dans les deux cas, il s'agit bien de transformer notre code prévu pour être lu par des humains En langage machine prévu pour être exécuté par un processeur. En ce sens, on peut dire que tout programme est une donnée, puisque nos programmes alimentent ses compilateurs et interpréteurs.

Même nos systèmes d'exploitation sont aussi des programmes dont le but est d'exécuter d'autres programmes. On peut aussi faire référence au fait que pour télécharger un logiciel sur Internet, on utilise un outil de téléchargement (comme Filezilla) qui lui-même est un programme.

Calculabilité et décidabilité

Tous les problèmes ne peuvent pas forcément être résolus avec un algorithme. C'est ce qu'ont démontré en 1937 Alonzo Church et Alan Turing (que vous avez peut-être déjà rencontrés au détour du cours de SNT consacré à la machine Enigma, dont il a aidé à déchiffrer les secrets).

Considérons un problème correspondant à une question fermée, dans le sens où la réponse à ce problème serait de type oui ou non (True ou False). Si une solution à base d'algorithmes existe, le problème est dit **décidable**. Si par contre il n'y a pas d'algorithme pour résoudre ce problème, ce dernier peut être déclaré **indécidable**.

Considérons maintenant un problème correspondant à une question plus ouverte, dans le sens où on attend comme résultat une valeur, produite à partir d'un calcul. Là encore deux cas de figure se présentent : si un algorithme est capable de résoudre ce problème, il s'agit d'un problème **calculable**, mais si c'est impossible on parlera de problème **non-calculable**.

Un problème indécidable ou non calculable peut très bien avoir une solution, ces termes signifient simplement qu'il n'existe pas d'algorithme capable de résoudre ce problème.

Problème de l'arrêt

Vous avez probablement connu ce cas en apprenant à réaliser des boucles (for, while…) en Python : il suffit d'une erreur d'inattention pour exécuter un programme qui part dans une boucle infinie et qu'il faut stopper avec un arrêt brutal (control+C, voire l'appel au gestionnaire de tâches) !

C'est un problème ancien qui est courant en informatique, auquel on est souvent confronté en débutant mais qui peut surprendre de temps en temps certaines personnes expérimentées. C'est l'un des problèmes qui a intéressé Turing dans les années 1930.

En l'occurrence, il serait très pratique de disposer d'algorithmes permettant de s'assurer qu'un programme va s'arrêter. Il existe d'ailleurs des logiciels conçus pour détecter certains types d'erreurs dans le code source d'autres programmes. Nous pouvons toutefois prouver que le problème de l'arrêt est indécidable, c'est-à-dire qu'il est impossible d'écrire un algorithme capable de s'assurer qu'un programme donné va s'arrêter ou non.

Considérons donc un algorithme hypothétique que nous allons nommer AlgoArrêt et qui attend en paramètres d'entrée un programme P et les données x qui l'accompagnent (il s'agit des données utilisées par le programme P).

Le but de l'algorithme AlgoArret est :

- de répondre oui (ou Vrai, ou True) S'il considère que le programme P s'arrête ;
- de répondre non (ou Faux, ou False) s'il considère que le programme P ne s'arrête jamais.

Pour la suite de la démonstration, nous allons utiliser un raisonnement par l'absurde, comme vous l'avez sans doute déjà vu en mathématiques. Ces raisonnements se basent sur une hypothèse que l'on va tâcher d'amener à son terme, et son issue impossible prouve que l'hypothèse était fausse dès le départ.

- Si P s'arrête, alors AlgoArrêt renvoie OUI.
- Si P ne s'arrête pas, AlgoArrêt renvoie NON.

Mais prenons un deuxième programme appelé « calcul ». Dans son code, il utilise l'algorithme AlgoArrêt.

On pourrait l'écrire ainsi :

```
fonction calcul(x):
  si AlgoArrêt(x,x) == OUI
        boucle infinie
  sinon
        sortie
```

Pourquoi AlgoArrêt(x,x) ? N'oubliez pas qu'il attend deux paramètres, un programme et ses données. Or un programme peut être une donnée et c'est ce qu'on s'apprête à faire là : nous allons faire passer un programme en paramètre.

Si on appelle donc calcul(calcul), c'est à dire qu'on appelle le programme calcul en le passant lui-même en paramètre, on obtient ceci :

```
fonction calcul(calcul):
  si AlgoArrêt(calcul,calcul) == OUI
        boucle infinie
  sinon
        sortie
```

Qu'avons-nous-là ?
Un programme qui dit que si AlgoArrêt(calcul, calcul) est égal à OUI (ce qui signifie qu'il s'arrête), nous l'envoyons vers une boucle infinie… Donc il ne s'arrête finalement pas ?

De même si AlgoArrêt(calcul, calcul) est égal à NON (ce qui signifie qu'il ne s'arrête pas), on va vers la sortie. Et donc il s'est arrêté !

Dans les deux cas, c'est impossible. Nous venons donc de prouver par l'absurde qu'AlgoArrêt ne peut pas exister.

Le problème de l'arrêt est indécidable !

EN PASSANT, SUR LE WEB…

La démonstration en vidéo

Si le fait de bien comprendre ce raisonnement à l'écrit se révèle compliqué, peut-être qu'une illustration en vidéo pourrait vous aider ?

Sur la chaîne YouTube du « Centre Henri Lebesgue », la vidéo de Vincent Guirardel intitulée « Le problème de l'arrêt d'une machine de Turing » propose une autre démonstration de l'indécidabilité du problème de l'arrêt.

Vous pourrez aussi visionner « La machine de Turing : tout ne se calcule pas » de la chaîne « Le blob, l'extra-média » !

02 MODÈLES ET PARADIGMES DE PROGRAMMATION

Récursivité

Factorielle : rappels

Abordons une notion un peu originale ! En effet, **la récursivité désigne des bouts de codes qui s'appellent eux-mêmes !** On peut ainsi définir une fonction Calcul_recursif(x) qui pour une valeur x initiale va être appelée, mais dont le code va contenir en interne un nouvel appel à cette même fonction !

Il est un peu difficile d'imaginer de manière abstraite comment cela fonctionne concrètement et à quoi cela peut bien servir, ne tardons donc pas à plonger dans un exemple. **Dans ce domaine, une application concrète et accessible peut être trouvée dans une notion mathématique : le calcul d'une factorielle.**

Un petit rappel préalable s'impose donc. Voici comment on calcule une factorielle :

pour tout nombre entier n (supérieur à 1), la factorielle de n est égale à
n * (n-1) * (n-2) * … * 1

On note ceci « n! » (mais cet aspect ne sera pas important dans notre application algorithmique, c'est le calcul qui va nous intéresser). Bien sûr, les points de suspension sont ici remplacés par autant de parenthèses que nécessaire, où n est réduit jusqu'à arriver à 1.

Exemple :
5! = 5*4*3*2*1 = 120

Factorielle : implémentation en Python

Voici une façon d'implémenter ce que nous venons d'évoquer en Python :

```
def factorielle(n):
    print("Calcul de factorielle pour n = " + str(n))
    if n == 1:
        return 1
    else:
        resultat = n * factorielle(n-1)
        tmp = "Résultat temp " + str(n) + " * factorielle("  + str(n-1) + "): "
        print(tmp + str(resultat))
        return resultat

print(factorielle(5))
```

Bien sûr, la dernière ligne ne fait pas partie de la fonction et n'est là que pour tester l'ensemble. Exécutez tout ça, quel en est le résultat ?

Vous devriez voir ceci :

```
>>>
Calcul de factorielle pour n = 5
Calcul de factorielle pour n = 4
Calcul de factorielle pour n = 3
Calcul de factorielle pour n = 2
Calcul de factorielle pour n = 1
Résultat temp 2 * factorielle(1): 2
Résultat temp 3 * factorielle(2): 6
Résultat temp 4 * factorielle(3): 24
Résultat temp 5 * factorielle(4): 120
120
>>>
```

EXERCICE 01 — Interprétation du calcul de factorielle

- **Pouvez-vous décrire le fonctionnement du traitement en vous basant sur l'ensemble des lignes affichées dans la console ?**

Note : *la correction de cet exercice représente une partie importante du cours, elle n'est pas à négliger mais n'allez la consulter qu'après avoir analysé de votre côté le déroulement de l'algorithme.*

03 MODÈLES ET PARADIGMES DE PROGRAMMATION

Modularité

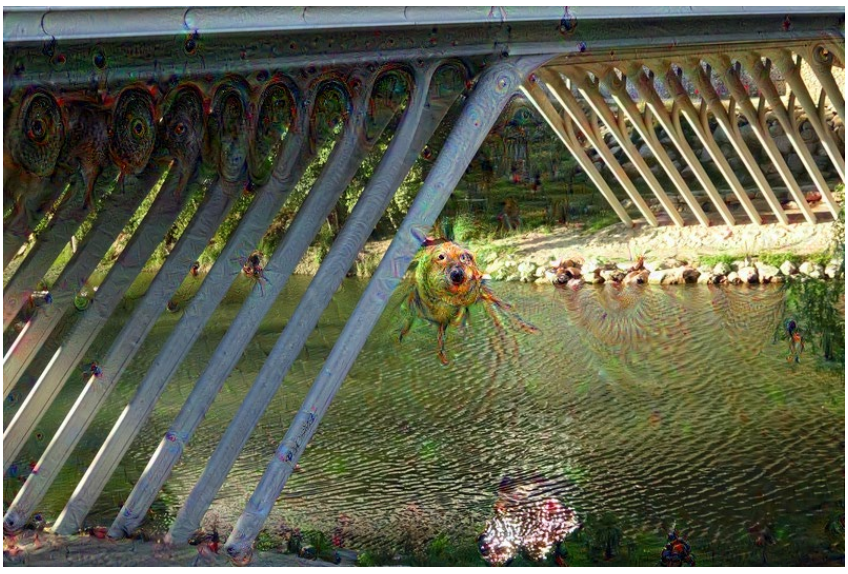
Qu'est-ce qu'une API ?

On appelle API (acronyme pour « Application Programming Interface ») une interface permettant de se connecter à un ensemble de fonctions permettant d'utiliser un package de services spécifiques.

Une API peut permettre d'utiliser les fonctions d'un logiciel installé sur notre machine, mais aussi (et de plus en plus souvent) d'appeler des fonctions proposées publiquement par des services web. C'est le cas notamment de l'API de Google permettant de se connecter à son IA « artistique » DeepDream**. Il est possible grâce à elle d'envoyer (via Python ou autre) une image et de recevoir en retour cette image modifiée à la manière… d'un « rêve » informatique !** Si vous ne connaissez pas DeepDream, voici un exemple de ce que peut donner une image avant et après ce traitement :

Image source :

Image traitée via l'API DreepDream



Voyez-vous la différence ? Au milieu de l'image, une espère de… poisson… ? Deepdream ajoute souvent des figures plus ou moins animales dans le décor.

Quel code a pu permettre cela ?

Avec Python, cela peut ressembler à ceci :

```
1  import requests
2  import urllib
3
4  r = requests.post(
5      "https://api.deepai.org/api/deepdream",
6      files={
7          'image': open('img/nomimage.jpg', 'rb'),
8      },
9      headers={'api-key': '**votre clé API**'}
10 )
11 print(r.json())
12
13 resultat = r.json()
14 nom_img = resultat['output_url']
15 print(nom_img)
16
17 urllib.request.urlretrieve(nom_img,"nomsauvegarde.jpg")
18
```

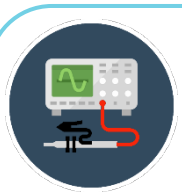
La ligne 9 contient la clé API personnelle que l'on est supposé demander individuellement à Google (c'est gratuit aujourd'hui). De nombreux fournisseurs d'API fonctionnent ainsi, ces clés permettant de compter le nombre d'appels effectués par chaque personne, et de limiter éventuellement l'usage du service. On peut par exemple être limité à 100 appels par heure. Pour Google, il n'y a pas de telle limite, mais par contre (pour les mêmes raisons d'économie de puissance côté serveur) les images renvoyées sont de taille réduite.

Vous pouvez consulter cet exemple ainsi que d'autres à l'adresse suivante :

https://deepai.org/machine-learning-model/deepdream

Vous y trouverez aussi une clé API de test (limitée à quelques récupérations d'images).

Essayez par vous-même ! N'hésitez pas à partager votre expérience dans la salle d'étude « Informatique et Technologies » sur le forum de Poulpi !



POUR ALLER PLUS LOIN

Des répertoires d'APIs publiques

Il est possible de trouver sur internet des listes d'APIs disponibles gratuitement et qui permettent de s'entraîner à l'utilisation de telles fonctions.

Dans certains cas, les sites proposant de telles interfaces proposent aussi des exemples de codes, y compris en Python.

Exemples de sites proposant des listes de services :

https://public-apis.xyz/
https://public-apis.io/
https://github.com/public-apis/public-apis

Il existe aussi des listes françaises, bien sûr. N'hésitez pas à en chercher ! Voici l'une d'entre elles, proposées par l'État français :

https://api.gouv.fr/

(On y trouve des indications sur les sources de ces données, en opendata)

Bibliothèques

Les API représentent un vaste sujet, aux possibilités nombreuses, mais qui peut se révéler un peu compliqué à approfondir dans le cadre de ce cours. Voyons plus en détail le concept de bibliothèques, assez proche, et dont vous avez pu déjà expérimenter quelques aspects durant les chapitres de SNT et NSI.

En effet, dans vos scripts Python, vous avez parfois utilisé la commande « import » pour utiliser de nouvelles fonctionnalités.

Ainsi, le module Random ajoute des fonctions liées aux données pseudo-aléatoires (l'aléatoire total n'existe pas vraiment en informatique), ou le module PIL ajoute des fonctionnalités concernant le traitement des images. Il en existe une multitude, pour des buts très divers.

Créer son propre module Python

Il est très facile de fabriquer un module personnalisé que l'on pourra ensuite utiliser dans plusieurs scripts ou partager avec d'autres personnes. Pour aller au plus simple, effectuons un test basique. Tapez le code suivant :

```
# module "social"

# fonction "saluer"
def saluer(nom):
    print("Salut à vous, " + nom + ".")
```

Sauvegardez le fichier sous le nom « social.py » dans le répertoire de votre choix.

Dans le même répertoire, créez un nouveau fichier python que vous nommerez par exemple « test_social.py » et dans ce script vous pouvez ajouter le code suivant :

```
import salutation

salutation.saluer("Otto")
```

Exécutez ce dernier script. S'il est placé dans le même dossier que l'autre script, tout devrait bien se passer et vous devriez obtenir le résultat suivant :

```
>>>
Salut à vous, Otto.
>>>
```

Et voilà !

Vous avez créé votre propre module, et avez pu l'appeler depuis un nouveau script. Il est dès lors facile d'imaginer ajouter un tas d'autres fonctions dans ce module « social ».

Allons un peu plus loin, en modifiant notre module social pour obtenir ceci :

```
# module "social"

def saluer(nom):
    print("Salut à vous, " + nom + ".")

def discuter():
    print("Bla bla bla...")

def partir():
    print("Il est l'heure de se dire au revoir.")
```

Pour en tirer parti, nous pouvons modifier notre script de test de la sorte :

```
import social

social.saluer("Sarah")
social.discuter()
social.partir()
```

Ce qui devrait donner un résultat de ce type :

```
*** Console de processus distant Réinitialisée ***
>>>
Salut à vous, Sarah.
Bla bla bla...
Il est l'heure de se dire au revoir.
>>>
```

Cette méthode est tout à fait correcte, mais il faut dire qu'ici nous avons un tout petit module avec très peu de fonctions. Imaginons que nous soyons en train d'avancer sur un script complexe faisant appel à de nombreuses bibliothèques. Il pourrait être intéressant alors (pour optimiser l'usage de la mémoire, pour avoir un code plus simplet, plus lisible, etc.) de ne faire appel qu'à ce que l'on va vraiment utiliser.

Voyons comment procéder !

Nous n'allons pas modifier le fichier de module, mais seulement celui qui y fait appel (à savoir le fichier « test_social.py ».

Modifions-le ainsi :

```
from social import saluer, discuter, partir

saluer("Sarah")
discuter()
partir()
```

Quelles différences constatez-vous ?

Avec « from social import » nous indiquons que nous allons puiser dans un module nommé « social » un certain nombre de fonctions ou données. Suit la liste de ce que l'on prend. Ici, nous avons tout mis, rien de moins (pour montrer une transition simple).

Remarquez aussi que dans ce cas de figure, on appelle ensuite les fonctions directement sans préciser le nom du module à chaque fois.

Maintenant, exploitons les véritables avantages de cette façon de faire :

```
from social import saluer

saluer("Sarah")
```

Désormais, seule la fonction saluer est importée, et si vous appelez les autres, vous obtiendrez un bug !

Documenter et commenter

Le module que nous venons de créer est simple, mais n'oublions pas qu'il ne représente qu'un exemple destiné à comprendre ces notions plus facilement. **Dans les faits, les modules que l'on crée sont souvent plus complexes, et les personnes qui s'en serviront pourraient avoir besoin d'aide.**

C'est là que la documentation intervient !

Heureusement, Python permet d'utiliser facilement ce qu'on appelle des **docstrings** qui renverront des informations lorsque les utilisateurs utilisent la fonction **help**.

Modifiez le module de la sorte :

```
# module "social"

def saluer(nom):
    """Effectue une salutation
    Paramètre attendu : nom (de type chaîne)"""
    print("Salut à vous, " + nom + ".")

def discuter():
    """Lance une discussion sur des sujets divers"""
    print("Bla bla bla...")

def partir():
    """Pour quitter l'interaction et se préparer à partir"""
    print("Il est l'heure de se dire au revoir.")
```

Notez deux types d'informations apportées :

- ✓ **La ligne commençant par un dièse est un « commentaire »** et seules les personnes ayant accès au code pourront le lire ;
- ✓ **Les lignes entourées de guillemets doubles** (on aurait pu mettre des apostrophes) **sont des docstrings** qui pourront être affichés dans une console Python par quelqu'un qui veut comprendre le fonctionnement du module.

Vous pouvez tester ceci en chargeant le module dans votre console. Admettons que votre fichier « social.py » soit situé dans le répertoire suivant : « C:\python\modules ».

Vous pouvez utiliser la fonction « chdir » du module « os » pour indiquer à Python qu'il faut prendre en compte ce dossier. Lorsque ce sera fait, nous pourrons charger le module « social » et demander l'affichage de la documentation de certaines de ses fonctions.

Dans notre exemple, cela donnerait ceci :

```
from os import chdir
chdir("C:\python\modules")
help(social.saluer)
```

Vous devriez obtenir le résultat suivant :

```
>>> import social
>>> help(social.saluer)
Help on function saluer in module social:

saluer(nom)
    Effectue une salutation
    Paramètre attendu : nom (de type chaîne)
```

Comme vous le voyez, il est donc possible de renseigner les utilisateurs et utilisatrices de nos modules, et ce sur plusieurs lignes si nécessaire. Voilà de quoi réaliser des fonctions complexes sans craindre de les voir incomprises (à condition d'écrire une documentation claire et concise, bien sûr).

MODÈLES ET PARADIGMES DE PROGRAMMATION

Paradigmes

Qu'est-ce qu'un programme ?

Un paradigme de programmation désigne une façon d'appréhender le code, une « philosophie » de gestion des algorithmes et autres. Chaque paradigme a ses avantages et inconvénients, des choses qu'il facilite au prix d'autres aspects qui peuvent devenir plus compliqués.
Nous avons jusqu'à présent eu recours au paradigme dit de **programmation impérative**, qui repose sur les notions suivantes :

- ✓ Séquence d'instructions (**commandes qui se suivent** dans un programme) ;
- ✓ Affectations (le fait de placer une **valeur dans une variable**) ;
- ✓ Instructions conditionnelles (**if… else**) ;
- ✓ Boucles (**for, while**).

D'autres paradigmes existent, qui font appel à certains de ces mêmes éléments (on y retrouvera les if, les boucles, etc.) mais en les organisant différemment. Nous pouvons par exemple évoquer la **programmation objet** (expliquée dans ses grandes lignes au sein du premier module de cette année) ainsi que la **programmation procédurale** ou la **programmation fonctionnelle** (deux concepts assez proches que nous considérerons ici comme équivalents). Mais il en existe des dizaines d'autres, dont certains sont tout de même assez rares. Ces paradigmes peuvent être utilisés de manière exclusive (un seul paradigme dans tout le programme) ou bien mélangés.

Il sera plus facile de comprendre ceci avec des exemples !
Imaginons que nous voulions créer un programme qui calcule l'aire d'un disque.

La version la plus basique sera ce qu'on peut faire en **programmation impérative**, ce qui donnerait ceci (on utilisera ici du pseudo-code, pas spécifiquement du Python) :

Définir la valeur de R (le rayon)
R2 = R * R
Résultat = R2 * 3,142 (valeur approximative de Pi)
Afficher Résultat

Voilà qui va droit au but, c'est rapide, cela fonctionne, mais cela peut manquer de flexibilité.

Examinons le même calcul, mais en **programmation fonctionnelle ou procédurale** :

Procédure calcul_aire(R)
R2 = R * R
aire = R2 * 3,142
Renvoyer R2

Programme principal
Récupérer la valeur de R
Résultat = calcul_aire(R)
Afficher Résultat

Même résultat, avec plus de code, mais le fait de disposer d'une fonction peut permettre de faciliter les traitements complexe ou la maintenance du code et fur et à mesure de sa complexification.

Voyons enfin un traitement de la même information, mais en **programmation objet** :

Cercle.calcul_aire(R)
R2 = R * R
aire = R2 * 3,142
Renvoyer R2

Programme principal
Récupérer la valeur de R
Résultat = Cercle.calcul_aire(R)
Afficher Résultat

On voit ici peu de différences avec la version précédente. Dans les « coulisses » des langages, il y en aurait plus, mais pour ce qui nous intéresse, elles sont en effet peu nombreuses à ce stade. Notons cependant qu'ici les fonctions comme « calcul_aire » sont rattachées à l'objet Cercle (et prennent alors le nom de « méthodes »).
Si on l'on veut traiter plusieurs cercles, la création (instanciation) de plusieurs objets de type Cercle peut se révéler intéressante.

Du bon choix d'un paradigme

Pour rester sur les quelques paradigmes que nous avons évoqués, réfléchissons à la façon dont on pourrait en choisir un plutôt qu'un autre.

Pour ce qui est de la programmation impérative, rappelons tout d'abord qu'elle constitue une brique fondamentale de la programmation, et qu'on ne retrouve des caractéristiques dans presque tous les autres paradigmes. Nous pouvons néanmoins ajouter que l'on peut dans certains cas se contenter de ce type de fonctionnement, lorsqu'on doit résoudre un problème qui peut être résolu simplement par une suite d'instructions séquentielles simples. C'est quelque chose que l'on pourra fréquemment utiliser pour du prototypage ou des scripts python destinés à automatiser certaines tâches basiques de notre quotidien (par exemple un redimensionnement d'image, etc.).

La programmation fonctionnelle représente elle aussi un élément fondateur du codage moderne. De très nombreux programmes ont recours à ce qu'on appelle des fonctions ou des procédures afin de ne pas avoir à répéter inutilement des morceaux de codes qui sont appelés plusieurs fois lors de l'exécution. Ce type d'organisation et donc tout naturellement adaptée à la création de bibliothèques comparables à certains que nous avons déjà vus, par exemple des lots de fonctions mathématiques permettant de transformer des valeurs à travers des opérations comme des calculs d'aire, etc. mais bien sûr, ceci n'est qu'un exemple parmi d'autres, comme souvent en programmation les possibilités sont énormes.

La programmation orientée objet (parfois abrégée POO, ou OOP en anglais) permet quant à elle de gérer des objets interdépendants pour lesquels on définit des attributs et des méthodes. Cette mécanique peut être un peu plus lourde au moment de la création, mais peut se révéler très puissante et extrêmement flexible ensuite. En effet, elle permet de ranger les différentes parties du code de manière logique et facile à maintenir. Elle est aussi très pratique lorsque l'on souhaite partager du code tout en offrant une interface claire et limitée aux utilisateurs finaux, tandis que les rouages internes restent cachés dans des méthodes et attributs privés. C'est ce que l'on retrouve d'ailleurs dans de nombreux modules que nous avons déjà utilisés : en général (pour simplifier), lorsque vous trouvez des mots-clés dans le code séparé par un point, ce qui se trouve à gauche est un objet et ce qui se trouve à droite est un attribut ou une méthode de cet objet.

Pour illustrer certains des points que nous venons d'évoquer, reconsidérons un script déjà évoqué à propos des API :

```
1  import requests
2  import urllib
3
4  r = requests.post(
5      "https://api.deepai.org/api/deepdream",
6      files={
7          'image': open('img/nomimage.jpg', 'rb'),
8      },
9      headers={'api-key': '**votre clé API**'}
10 )
11 print(r.json())
12
13 resultat = r.json()
14 nom_img = resultat['output_url']
15 print(nom_img)
16
17 urllib.request.urlretrieve(nom_img,"nomsauvegarde.jpg")
18
```

Ce code importe le module « requests » qui permet d'effectuer une requête sur le web.
Il est utilisé dès la ligne 4, où l'on définit une variable nommée simplement « r » qui va accueillir le résultat d'une requête effectuée grâce à l'instruction requests.post(…). Ici la méthode post de l'objet requests va permettre d'aller vers l'URL spécifiée et d'y envoyer un fichier jpg.
La réponse donnée par le serveur de deepai.org sera elle-même traitée comme un objet contenant plusieurs attributs, et tout ceci va dans la variable « r » définie en début de ligne 4.
On peut ensuite afficher le contenu json (json désigne un format particulier, un peu comme le HTML) en tant qu'attribut de l'objet « r », c'est ce qui est fait en ligne 11.
En ligne 17, on utilise un autre module, urllib, et son objet request (ici dans le but d'aller chercher l'image modifiée proposée par deepdream, et dont seule l'URL est renvoyée dans le json au-dessus).
On voit donc quelques bribes d'utilisation de la programmation objet, du côté des utilisateurs finaux (bien sûr il y a aussi l'autre versant où l'on définit le code de chaque méthode, etc.). **Mais dans l'ensemble ce code est organisé de manière impérative, très basique.**

Si l'on voulait modifier ce script pour qu'il traite plusieurs images, on aurait recours à une boucle (for ou while), et il y aurait possibilité de rester en paradigme impératif, ou bien de créer une fonction pour adopter une posture plus procédurale/fonctionnelle.

LE TEMPS DU BILAN

NOTIONS ET CONCEPTS A RETENIR

Le processeur d'une machine attendant du langage machine constitué de zéros et de uns (binaire), ultimement **tout est une donnée, y compris les instructions d'un programme, et donc un programme lui-même**.

Les langages informatiques que l'on a créés pour permettre aux humains de transmettre efficacement des instructions à ordinateur permet de réaliser toutes sortes de tâches, **chaque langage ayant ses propres forces et faiblesses. Nous avons vu d'ailleurs qu'il existe aussi différents paradigmes de programmation**, ce qui donne encore la possibilité d'adapter l'approche de chaque problématique rencontrée.

Parmi ces paradigmes, la programmation impérative représente en quelque sorte la base de tout le reste, puisqu'il s'agit d'enchaîner des instructions en séquence. **La programmation fonctionnelle ou procédurale** offre la possibilité d'appeler facilement un morceau de code qui est destiné à être utilisé souvent au sein d'un programme. **La programmation orientée objet**, quant à elle, autorise une approche très organisée qui facilite le maintien du code et le partage de la bibliothèque

Ce dernier point fait écho au concept de modularité qui désigne, comme nous l'avons vu, la capacité qu'ont de nombreux langages de programmation (dont Python fait partie) à faire appel à des ensembles de variables et fonctions codées par d'autres personnes. On peut ainsi importer des modules livrés avec le langage, ou aller en chercher de nouveaux. Certains programmes, installés sur nos machines ou accessibles via des serveurs Web, proposent des interfaces publiques que l'on appelle **API (Application Programming Interface)**. Elles permettent de faire appel à des fonctions spécifiques qui sont forcément installées sur nos machines. C'est aussi par ce biais que certains sites privés ou gouvernementaux proposent l'accès à des données relevant de l'open data.

Enfin, pour revenir sur les **différentes pratiques de programmation et paradigmes, nous avons exploré les possibilités offertes par la récursivité.** Il s'agit d'une pratique rendue possible par la programmation fonctionnelle, puisqu'on y utilise **des fonctions qui s'appellent elles-mêmes** jusqu'à obtenir la réponse qu'elles cherchent.

Concluons :

La programmation est un domaine très riche, en perpétuelle évolution. Il existe une multitude de langages, et de nouveaux apparaissent chaque année même si tous ne sont pas promis à un avenir radieux. Certains disparaissent, faute d'engouement, d'autres subsistent malgré une adoption modérée, et ce grâce aux avantages qu'ils apportent dans des situations très spécifiques. Quelques-uns emportent l'adhésion d'un grand nombre de programmeurs et programmeuses, jusqu'à devenir des standards.

Python fait partie de ces langages qui ont su s'imposer, aux côtés d'autres poids-lourds comme JavaScript ou C# (prononcé C Sharp). On peut compter bien sûr de nombreux autres langages très populaires. Pourquoi tant de possibilités ?

Comme nous l'avons vu, **il existe différents paradigmes** de programmation, autant d'approches possibles pour un même problème. **Ces philosophies d'approche sont plus ou moins compatibles avec certains langages**. Parmi ces derniers, certains peuvent être à l'aise avec de nombreuses méthodes de programmation différentes, ou se voir réservés à des applications très précises.

Nous ne pouvons quitter ce chapitre en connaissant tous les paradigmes ou tous les langages existants, bien sûr. **Retenons néanmoins la présence et l'importance d'un tel paysage très hétérogène et en permanente évolution.**

LES CLÉS DU BAC by COURS PI

PROGRAMMATION ET RÉCURSIVITÉ

Au cours de cette clé du bac va être étudié un exercice type bac. La méthodologie de réponse va être abordée.

méthodos clés bac

- On considère un tableau de nombres de n lignes et p colonnes.
- Les lignes sont numérotées de 0 à n − 1 et les colonnes sont numérotées de 0 à p − 1. La case en haut à gauche est repérée par (0,0) et la case en bas à droite par (n − 1, p − 1).
- On appelle chemin une succession de cases allant de la case (0,0) à la case (n−1, p−1), en n'autorisant que des déplacements case par case : soit vers la droite, soit vers le bas.
- On appelle somme d'un chemin la somme des entiers situés sur ce chemin.

Par exemple, pour le tableau T suivant :

4	1	1	3
2	0	**2**	1
3	1	**5**	**1**

- Un chemin est (0,0), (0,1), (0,2), (1,2), (2,2), (2,3) (en gras sur le tableau) ;
- La somme du chemin précédent est 14.
- (0,0), (0,2), (2,2), (2,3) n'est pas un chemin.

L'objectif de cet exercice est de déterminer la somme maximale pour tous les chemins possibles allant de la case (0, 0) à la case (n − 1, p − 1).

exos clés bac

1) On considère tous les chemins allant de la case (0, 0) à la case (2, 3) du tableau T donné en exemple.

1.1) Un tel chemin comprend nécessairement 3 déplacements vers la droite. Combien de déplacements vers le bas comprend-il ?

1.1) La longueur d'un chemin est égale au nombre de cases de ce chemin. Justifiez que tous les chemins allant de (0,0) à (2,3) ont une longueur égale à 6.

2) En listant tous les chemins possibles allant de (0, 0) à (2, 3) du tableau T, déterminez un chemin qui permet d'obtenir la somme maximale et la valeur de cette somme.

3) On veut créer le tableau T' où chaque élément T'[i][j] est la somme maximale pour tous les chemins possibles allant de (0,0) à (i,j).

3.1) Complétez le tableau T', donné ci-dessous, associé au tableau :

T' =

4	5	6	.....
6	.....	8	10
9	10	.....	16

T =

4	1	1	3
2	0	2	1
3	1	5	1

3.2) Justifiez que si j est différent de 0, alors : T' [0] [j] = T [0] [j] + T' [0] [j-1]

4) Justifiez que si i et j sont différents de 0, alors : T' [i] [j] = T [i] [j] + max(T' [i-1] [j], T' [i] [j-1]).

5) On veut créer la fonction récursive somme_max ayant pour paramètres un tableau T, un entier i et un entier j. Cette fonction renvoie la somme maximale pour tous les chemins possibles allant de la case (0, 0) à la case (i, j).

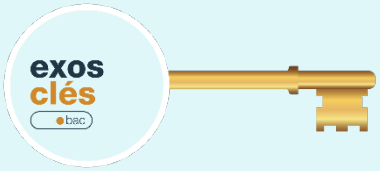
5.1) Quel est le cas de base, à savoir le cas qui est traité directement sans faire appel à la fonction somme_max ? Que renvoie-t-on dans ce cas ?

5.2) À l'aide de la question précédente, écrire en Python la fonction récursive somme_max.

5.3) Quel appel de fonction doit-on faire pour résoudre le problème initial ?

<fin de l'exercice>

BASES DE DONNÉES RELATIONNELLES ET LANGAGE SQL



L'énoncé de cet exercice utilise les mots du langage SQL suivant :

SELECT, FROM, WHERE, JOIN, INSERT INTO, VALUES, COUNT, ORDER BY.

Dans un lycée imaginaire, les données relatives aux élèves de secondes sont regroupées dans un fichier nommé seconde_lyc.csv. Un extrait de son contenu est représenté figure 1.

Fig. 1

num_eleve	nom	prenom	datenaissance	langue1	langue2	option	classe
133310FE	ACHIR	Mussa	01/01/2005	anglais	espagnol		2A
156929JJ	ALTMEYER	Yohan	05/05/2005	allemand	anglais	théâtre	2D
500633KH	BELEY	Thibaut	05/05/2005	anglais	espagnol		2A
911887GA	BELEY	Marie	05/05/2005	anglais	espagnol		2A
906089JJ	BELEY	Manon	10/01/2005	anglais	allemand		2E
488697GA	CAILLE	Marie	30/03/2004	italien	anglais		2D
193514FB	CHARPENTIER	Jules	26/12/2005	espagnol	anglais		2C
321188FA	CLAUDEL	Benjamin	09/09/2005	espagnol	anglais		2E
081282GF	EISEN	Carla	23/06/2004	anglais	allemand		2A
026946KB	EL AYAR	Amir	11/09/2005	anglais	arabe	cinéma	2D
108303KG	GEHIN	Arthur	26/02/2005	allemand	anglais		2D
057934BK	GROSJEAN	Alexandre	09/11/2005	anglais	espagnol		2C
571113KE	HENRY	Paul	12/03/2005	allemand	anglais		2E
488820DE	JACQUEY	Marc	13/11/2005	anglais	italien		2D
024810CE	JULIANO	Alberto	21/04/2005	anglais	espagnol		2C
249992EJ	KLEIBER	Gusti	20/02/2005	anglais	espagnol	cinéma	2E
492698AF	LACOUR	Julie	06/04/2005	italien	anglais		2D
026454FA	LARBI	Nourdine	14/07/2005	espagnol	anglais		2C
309341GD	LEFZA	Yasmina	26/11/2005	espagnol	anglais		2E
076725HD	MARTIN	Victor	13/03/2005	anglais	espagnol		2A
815183CB	NGUYEN	Ngong	16/03/2005	anglais	espagnol		2D
094002FC	PELTIER	Romane	14/06/2005	allemand	anglais		2D
321262HD	RENAULT	Zoé	06/08/2005	anglais	espagnol	latin	2E
075421AK	ROTH	Ursule	03/01/2005	anglais	allemand		2A
121001CK	SERHANI	Sabrina	01/09/2005	italien	anglais		2D
538965DJ	TIJDJANE	Yourk	31/01/2005	espagnol	anglais		2D
389873GC	VIALET	Priscille	28/02/2005	espagnol	anglais		2C
980306CA	WADE	Marcelin	03/05/2005	allemand	anglais		2E
807158DH	WENGER	Alexandre	20/08/2005	allemand	anglais		2A
666702FA	YAMAN	Elamine	23/04/2005	anglais	arabe		2D

Rappel : les fichiers csv sont avant tout des fichiers textuels, avec des données séparées par des virgules ou des points-virgules. Ici, le fichier est affiché tel qu'il le serait après importation dans un tableur comme LibreOffice Calc ou Microsoft Excel.

Pour les besoins de l'organisation du lycée, le chef d'établissement exploite la base de données par des requêtes en langage SQL. Il a pour cela créé une table (ou relation) SQL dénommée seconde dans son système de gestion de bases de données dont la structure est la suivante :

L'attribut num_eleve est un entier, les autres sont des chaînes de caractère (de type CHAR).

seconde
num_eleve (clef primaire)
langue1
langue2
option
classe

1.1) Dans le modèle relationnel, quel est l'intérêt de l'attribut num_eleve.

1.2) Écrire une requête SQL d'insertion permettant d'enregistrer l'élève ACHIR Mussa dans la table seconde. Les informations relatives à cet élève sont données dans la ligne 1 du fichier seconde_lyc.csv.

1.3) Lors de l'insertion de l'élève ALTMEYER Yohan (ligne 2 du fichier seconde_lyc.csv), une erreur de saisie a été commise sur la première langue, qui devrait être allemand. Écrire une requête SQL de mise à jour corrigeant les données de cet élève.

2) On suppose maintenant que la table seconde contient les informations issues de la figure 1 (ni plus, ni moins, même si la figure 1 n'est qu'un extrait du fichier seconde_lyc.csv).

2.1) Quel est le résultat de la requête suivante ?

SELECT num_eleve FROM seconde ;

2.2) On rappelle qu'en SQL, la fonction d'agrégation COUNT() permet de compter le nombre d'enregistrements dans une table. Quel est le résultat de la requête suivante ?

SELECT COUNT(num_eleve) FROM seconde;

2.3) Écrire la requête permettant de connaître le nombre d'élèves qui font allemand en langue1 ou langue2.

3) Le chef d'établissement souhaite faire évoluer la structure de sa base de données. Pour ce faire, il créé une nouvelle table élève dont la structure est la suivante :

eleve
num_eleve (clef primaire, clef étrangère de la table seconde)
nom
prenom
datenaissance

Là encore, l'attribut num_eleve est un entier, les autres sont des chaînes de caractère (de type CHAR).

3.1) Expliquer ce qu'apporte l'information clef étrangère pour l'attribut num_eleve de cette table en termes d'intégrité et de cohérence.

3.2) On suppose la table élève correctement créée et complétée. Le chef d'établissement aimerait lister les élèves (nom, prénom, date de naissance) de la classe 2A.
Écrire la commande qui permet d'établir cette liste à l'aide d'une jointure entre élève et seconde.

4) Proposer la structure d'une table coordonnée dans laquelle on pourra indiquer, pour chaque élève, son adresse, son code postal, sa ville, son adresse mail. Préciser la clef primaire et/ou la clé étrangère en vue de la mise en relation avec les autres tables.

<fin de l'exercice>

CORRECTION :

Programmation et récursivité

1.1) Le chemin de (0, 0) à (2, 3) contient nécessairement deux déplacements vers le bas.
1.2) Pour ce chemin de (0, 0) à (2, 3), nous avons vu qu'il y avait nécessairement trois déplacements vers la droite, ainsi que deux déplacements vers le bas, pour un total de 5 cases à traverser (en comptant la case de destination). Mais si la longueur de chemin est égale au nombre de cases de ce chemin, il faut aussi compter la case de départ, ce qui rajoute une case. Nous avons donc une longueur de 6.

2) Commençons à lister les chemins de (0, 0) à (2, 3).
Pour plus de lisibilité, on ne notera que les cases intermédiaires, sans noter à chaque fois le départ (0, 0) et l'arrivée (2, 3). On notera aussi en fin de ligne la somme des entiers situés sur la portion variable du chemin.

Pour récapitulatif, voici les coordonnées possibles sur le tableau :

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)

Chemins (simplifiés, sans départ et arrivée) qui partent d'abord vers la droite :

(0, 1), (0, 2), (0, 3), (1, 3) => 1 + 1 + 3 + 1 = 6
(0, 1), (0, 2), (1, 2), (1, 3) => 1 + 1 + 2 + 1 = 5
(0, 1), (0, 2), (1, 2), (2, 2) => 1 + 1 + 2 + 5 = 9
(0, 1), (1, 1), (1, 2), (1, 3) => 1 + 0 + 2 + 1 = 4
(0, 1), (1, 1), (1, 2), (2, 2) => 1 + 0 + 2 + 5 = 8
(0, 1), (1, 1), (2, 1), (2, 2) => 1 + 0 + 1 + 5 = 7

Chemins (simplifiés) qui partent d'abord vers le bas :

(1, 0), (1, 1), (1, 2), (1, 3) => 2 + 0 + 2 + 1 = 5
(1, 0), (1, 1), (1, 2), (2, 2) => 2 + 0 + 2 + 5 = 9
(1, 0), (1, 1), (2, 1), (2, 2) => 2 + 0 + 1 + 5 = 8
(1, 0), (2, 0), (2, 1), (2, 2) => 2 + 3 + 1 + 5 = 11

Le dernier chemin considéré est celui à la somme la plus élevée. Si nous prenons le chemin complet, cela donne : (0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (2, 3) => 4 + 2 + 3 + 1 + 5 + 1 = 16

3.1) Complétion du tableau T'.

Pour T'[1][1] : on peut passer par (0,0) ou (1,1) avec des sommes de 6 et 5, donc on garde 6.

Pour T'[2][2] : on voit sur notre liste des chemins qu'il y en a six qui passent par cette case, celui qui a la plus grande somme est celui-ci :
(0,0), (1, 0), (2, 0), (2, 1), (2, 2) => 4 + 2 + 3 + 1 + 5 = 15

Pour T'[0][3] : il n'y a qu'un seul chemin possible, en ligne droite, sa somme est de 9.

Tableau complété :

4	**5**	**6**	**9**
6	**6**	**8**	**10**
9	**10**	**15**	**16**

3.2) Justifiez que si j est différent de 0, alors T'[0][j] = T[0][j] + T'[0][j-1].

Pour arriver à une case (0, j) (l'équivalent d'un tableau appelé en [0][j]), il faut nécessairement passer par la case (0, j-1).

Le tableau T' contient la somme incrémentale des valeurs des cases traversées. On peut dire que :

- T[0][j] contient la valeur correspondant à la case (0,j) ;
- T'[0][j-1] contient la somme des valeurs de toutes les cases traversées (y compris le départ).

Donc la somme des deux correspond à la somme des valeurs de tout le chemin, destination comprise, ce qui est contenu dans T'[0][j].

Donc T'[0][j] = T[0][j] + T'[0][j-1].

4) Justifiez que si i et j sont différents de 0, alors : T'[i][j] = T[i][j] + max(T'[i −1][j], T'[i][j −1]).

Nous sommes sur le même principe que précédemment, mais avec une case (i,j) pour laquelle il existe plusieurs chemins d'arrivée possibles. Pour arriver à une case (i, j), il faut passer par la case (i, j-1) ou par la case (i-1, j). La fonction max(a, b) renverra la valeur la plus grande entre les deux passées en paramètre.

Comme le tableau T' contient les sommes de valeurs de chemins, nous n'avons pas besoin de connaître le détail du chemin emprunté.

Grâce à la fonction max appelée pour savoir quelle case obligatoirement traversée précédemment contient la plus grande valeur, nous nous retrouvons dans la même situation que pour la question précédente, avec :

- T[i][j] contient la valeur correspondant à la case (i,j) ;
- max(T'[i −1][j], T'[i][j −1]) contient la somme maximale des valeurs de toutes les cases traversées (y compris le départ).

Donc la somme des deux correspond à la somme maximale des valeurs de tout le chemin, destination comprise, ce qui est contenu dans T'[i][j].

Donc T'[i][j] = T[i][j] + max(T'[i −1][j], T'[i][j −1]).

5.1) La fonction récursive va partir de (i,j) pour revenir vers (0,0). Le cas de base (qui permet d'éviter une boucle infinie car il ne rappelle pas la fonction somme_max) est celui où l'on se trouve sur (0,0), on renvoie alors la valeur de cette case de la matrice, au lieu de rappeler la fonction.

5.2) En appliquant ce que nous avons démontré dans les questions précédentes, nous pouvons créer une fonction récursive avec les caractéristiques suivantes :

- elle vérifie les cases voisines en remontant vers (0,0) et les compare pour garder les valeurs maximales ;
- si on arrive sur un bord, elle ajuste le déplacement pour ne pas sortir du tableau ;
- elle renvoie une valeur sans récursivité lorsqu'elle est appelée sur les coordonnées (0,0).

Cela peut donner le code suivant :

```
def somme_max(T, i, j):
    if i == 0 and j == 0:
        return(T[0][0])
    elif i == 0:
        A = somme_max(T, 0, j-1)
        B = A
    elif j == 0:
        B = somme_max(T, i-1, 0)
        A = B
    else:
        A = somme_max(T, i-1, j)
        B = somme_max(T, i, j-1)
    return T[i][j] + max(A, B)
```

5.3) On peut tester notre fonction avec la matrice de l'exercice, de la façon suivante :

```
tabTest = [
    [4, 1, 1, 3],
    [2, 0, 2, 1],
    [3, 1, 5, 1]
]
print(somme_max(tabTest, 2, 3))
```

<fin du corrigé de l'exercice>

1.1) L'attribut num_eleve est défini comme clef primaire.
Ce champ permet donc d'identifier de manière unique chaque uplet (ou enregistrement) de la collection. Ainsi, en connaissant une clé primaire, on peut identifier une ligne de la table sans confusion possible, car chaque identifiant sera unique.

1.2) Requête permettant d'insérer le nouvel élève :

```
INSERT INTO seconde (num_eleve, langue1, langue2, classe)
  VALUES (133310, 'anglais', 'espagnol', '2A');
```

Note : l'énoncé parle d'un attribut num_eleve au format entier, tandis que dans la table affichée (le fichier CSV) on voit des lettres à la fin. Considérons donc que ces lettres constituent une information non essentielle et que les chiffres suffisent à identifier l'élève. En outre, on a ignoré dans la requête le champ « option » car ce champ est vide pour cet élève dans le fichier CSV.

1.3) Pour modifier et changer la langue sur la ligne demandée :

```
UPDATE seconde SET langue1 = 'allemand' WHERE num_eleve = 156929;
```

Note : lorsque nous effectuons une seule requête, le point-virgule final n'est pas obligatoire (il sert à distinguer plusieurs instructions).

2.1) Cette requête renvoie tous les numéros d'élève en colonne.

2.2) Cette requête renvoie le nombre de lignes de la table, à savoir 30.

2.3) Nous pouvons utiliser la requête suivante :

```
SELECT COUNT(num_eleve) FROM seconde
  WHERE langue1 = 'allemand' OR langue2 = 'allemand';
```

Note : nous aurions pu utiliser COUNT(*) à la place de COUNT(num_eleve).

3.1) Le fait que num_eleve soit ici déclaré en clef étrangère permet de s'assurer que les tables sont reliées aux yeux du SGBDR. Cela permet d'éviter la redondance des données, chacune est stockée une fois dans la base, ce qui permet d'éviter les erreurs de désynchronisation ou de doublons.

3.2) Nous pouvons recourir à une jointure pour obtenir ces informations :

```
SELECT nom, prenom, datenaissance FROM eleve
  JOIN seconde ON eleve.num_eleve = seconde.eleve
  WHERE classe = '2A';
```

4) Nous pourrions adopter la structure suivante :

Coordonnees	Types
num_eleve (clef primaire, clef étrangère de la table eleve)	INT
adresse	CHAR
codepostal	INT (ou CHAR)
ville	CHAR
adressemail	CHAR

<fin du corrigé de l'exercice>

Vous pouvez maintenant faire et envoyer le devoir n°1