



Informatique S1

Cours d'Informatique S1

Initiation à l'algorithmique


QUESTIONNEMENTS DE COURS

JACQUES TISSEAU

Ecole nationale d'ingénieurs de Brest
Centre européen de réalité virtuelle
tisseau@enib.fr

Avec la participation de ROMAIN BÉNARD, STÉPHANE BONNEAUD, CÉDRIC BUCHE, GIREG DESMEULLES, CÉLINE JOST, SÉBASTIEN KUBICKI, ERIC MAISEL, ALÉXIS NÉDÉLEC, MARC PARENTHOËN et CYRIL SEPTSEULT.


Ces questionnements accompagnent les enseignements d'informatique du 1^{er} semestre (S1) de l'Ecole Nationale d'Ingénieurs de Brest (ENIB : www.enib.fr). Ils complètent les notes de cours « Initiation à l'algorithmique ».



— Cours d'Informatique S1 —

Initiation à l'algorithmique

JACQUES TISSEAU
Ecole nationale d'ingénieurs de Brest
Centre européen de réalité virtuelle
tisseau@enib.fr



Ces notes de cours accompagnent les enseignements d'informatique du 1^{er} semestre (S1) de l'Ecole Nationale d'Ingénieurs de Brest (ENIB : www.enib.fr). Leur lecture ne dispense en aucun cas d'une présence attentive aux cours ni d'une participation active aux travaux dirigés.

Avec la participation de ROMAIN BÉNARD, STÉPHANE BONNEAUD, CÉDRIC BUCHE, GIREG DESMEULLES, CÉLINE JOST, SÉBASTIEN KUBICKI, ERIC MAISEL, ALÉXIS NÉDÉLEC, MARC PARENTHOËN et CYRIL SEPTSEULT.

version du 21 octobre 2014

Tisseau J., *Initiation à l'algorithmique*, ENIB, cours d'Informatique S1, Brest, 2009-2014.

Table des matières

1	Introduction	3
2	Qu'est-ce que l'algorithmique ?	8
2.1	Exemple	8
2.2	Généralisation	9
2.3	Applications	9
2.4	Entraînement	10
2.5	Révisions	14
3	Qu'est-ce que l'affectation ?	15
3.1	Exemple	15
3.2	Généralisation	15
3.3	Applications	16
3.4	Entraînement	17
3.5	Révisions	19
4	Comment calcule-t-on avec des opérateurs booléens ?	20
4.1	Exemple	20
4.2	Généralisation	21
4.3	Applications	22
4.4	Entraînement	22
4.5	Révisions	24
5	Comment coder un nombre ?	25
5.1	Exemple	25
5.2	Généralisation	25
5.3	Applications	26
5.4	Entraînement	26
5.5	Révisions	29
6	Qu'est-ce qu'un test ?	30
6.1	Exemple	30
6.2	Généralisation	30
6.3	Applications	31
6.4	Entraînement	31
6.5	Révisions	35
7	Comment construit-on une boucle ?	36
7.1	Exemple	36
7.2	Généralisation	36
7.3	Applications	37
7.4	Entraînement	38
7.5	Révisions	41

8	Comment imbriquer des boucles ?	42
8.1	Exemple	42
8.2	Généralisation	42
8.3	Applications	43
8.4	Entraînement	44
8.5	Révisions	47
9	Comment spécifier une fonction ?	48
9.1	Exemple	48
9.2	Généralisation	49
9.3	Applications	50
9.4	Entraînement	50
9.5	Révisions	52
10	Qu'est-ce que la récursivité ?	53
10.1	Exemple	53
10.2	Généralisation	53
10.3	Applications	55
10.4	Entraînement	55
10.5	Révisions	58
11	Comment trier une séquence ?	59
11.1	Exemple	59
11.2	Généralisation	59
11.3	Applications	61
11.4	Entraînement	61
11.5	Révisions	63
12	Tout en un ?	64
12.1	Exemple	64
12.2	Généralisation	69
12.3	Applications	70
	Liste des questions	72
	Références	75

1 Introduction

Questionnement, subst. masc., Fait de poser un ensemble de questions sur un problème.

Centre national de ressources textuelles et lexicales ([CNRTL](#))

Dans toutes les disciplines, le questionnement est au cœur de l'activité pédagogique. [...] Il y aurait pourtant lieu de s'interroger sur la pratique qui consiste à interroger de but en blanc ceux que l'on place devant des objets culturels qui leur sont précisément étrangers. Que peuvent-ils dire d'autre que des banalités bienfaisantes ? L'animateur le sait, tout comme le professeur, mais il n'attend souvent du procédé qu'une occasion pour embrayer ses propres réponses, en faisant mine de les inscrire dans le fil de celles des élèves, par continuité ou par contraste. Il oublie ainsi que l'on voit moins avec ses yeux qu'avec ses neurones, et qu'il n'est pas d'observation possible en l'absence de cadres interprétatifs. Le propre de l'expert c'est de disposer d'outils conceptuels lui permettant de voir autre chose, et autrement que le commun des mortels. Ne dit-on pas que toute observation renseigne sur l'observateur ? [...] Examinons d'abord plus précisément les questions orales, qui montrent une curieuse inversion par rapport aux formes quotidiennes du questionnement, en famille ou entre amis. Dans ce cas, n'importe quelle réponse fait souvent l'affaire, et l'absence même de réponse est fréquente, l'essentiel étant de pouvoir développer un échange de points de vue. Ou alors, on pose une question à un plus expert que soi pour combler une ignorance, résoudre un problème ou lever un doute. À l'école, au contraire, c'est le professeur qui interroge ceux qui, à l'évidence, en savent moins que lui ! Les élèves comprennent vite cette bizarrerie de la « forme scolaire » et réalisent que le professeur, lui, ne cherche pas à s'informer mais à tester la classe. Dès la première question posée, flotte ainsi un parfum d'évaluation informelle, avec ce qu'elle implique de violence symbolique. Contrairement au didactique familial, le propre du didactique scolaire c'est de travailler des questions ayant déjà des réponses, les élèves sachant parfaitement que le professeur les connaît. Du coup, ils cherchent à s'y adapter en inférant la réponse souhaitée, et ils répondent ainsi davantage au professeur pour satisfaire ses attentes qu'aux questions posées en vue de résoudre une énigme. Une analyse quantitative de séquences didactiques fait de surcroît apparaître que le nombre de questions oralement posées est très élevé, ce qui renforce la nécessité pour les élèves de répondre de façon stratégique. C'est ainsi qu'on peut parler d'un véritable « métier d'élève » ! [...]

Le questionnement sur des documents ou sur des manuels pose d'autres problèmes. Il conviendrait d'abord de clarifier, pour les élèves, sur quel mode on attend d'eux qu'ils répondent. Faute de savoir le repérer, il arrive qu'ils répondent de façon très élaborée...alors qu'on cherche seulement à s'assurer de la bonne compréhension du texte ; et, inversement, qu'ils fournissent une réponse qui reprend les termes du document...alors qu'on attend d'eux, cette fois, une mise en rapport de différents éléments pour développer une analyse critique. Pour le dire autrement, ils ne savent pas quel est le niveau d'objectif visé, tel que les a distingués Bloom : est-ce que cela relève de la connaissance de faits particuliers, de la compréhension d'informations, de l'application d'une règle fournie, de l'analyse ou de la synthèse, voire d'une production critique qui combine la maîtrise objective des données avec un point de vue personnel. Les élèves seraient grandement aidés si on leur précisait ainsi l'enjeu du questionnement (repérer une ou des informations contenues dans le texte, extraire du texte la ou les informations qui contiennent la réponse, mettre en relation différents documents, interpréter le document à partir de cadres théoriques extérieurs, proposer une analyse plus subjective...). [...]

TAXONOMIE DE BLOOM			
Bloom B.S. et al., <i>Taxonomy of educational objectives. Handbook I : the cognitive domain</i> , 1956			
Niveaux	Caractéristiques	Capacités	Actions
1. connaissance	Repérer de l'information et s'en souvenir. Connaître des événements, des dates, des lieux, des faits. Connaître de grandes idées, des règles, des lois, des formules.	Etre capable de restituer des informations dans des termes voisins de ceux que l'on a appris.	citer, décrire, définir, dire, énumérer, étiqueter, examiner, nommer, cerner, répéter
2. compréhension	Saisir des significations. Traduire des connaissances dans un nouveau contexte. Interpréter des faits à partir d'un cadre donné.	Etre capable de traduire et d'interpréter de l'information en fonction de ce que l'on a appris.	associer, comparer, estimer, différencier, discuter, extrapoler, expliquer, illustrer, résumer, interpréter
3. application	Réinvestir des méthodes, des concepts et des théories dans de nouvelles situations. Résoudre des problèmes en mobilisant les compétences et connaissances requises.	Etre capable de sélectionner et transférer des données pour réaliser une tâche ou résoudre un problème.	appliquer, changer, compléter, démontrer, illustrer, montrer, modifier, rattacher, résoudre, traiter
4. analyse	Percevoir des tendances. Organiser un ensemble en différentes parties. Reconnaître les sous-entendus. Extraire des éléments.	Etre capable de distinguer, classer, mettre en relation les faits ou la structure d'un énoncé ou d'une question.	analyser, catégoriser, choisir, comparer, contraster, diviser, inférer, isoler, ordonner, séparer
5. synthèse	Utiliser des idées disponibles pour en créer de nouvelles. Généraliser à partir d'un certain nombre de faits. Mettre en rapport des connaissances issues de plusieurs domaines.	Etre capable de concevoir, intégrer et conjuguer des idées en un nouveau produit, un nouveau plan ou une nouvelle proposition.	composer, conjuguer, créer, élaborer, intégrer, inventer, mettre en rapport, planifier, réécrire, réarranger
6. évaluation	Comparer et distinguer des idées. Déterminer la valeur de théories et d'exposés. Poser des choix en fonction d'arguments raisonnés. Vérifier la valeur des preuves. Reconnaître la part de subjectivité.	Etre capable d'estimer, d'évaluer ou de critiquer en fonction de normes et de critères construits.	appuyer, argumenter, critiquer, décider, évaluer, juger, justifier, noter, recommander, tester

Plus fondamentalement, le questionnement d'apprentissage devrait être distinct du questionnement d'évaluation. Alors que le second vient souvent recouvrir le premier de façon anticipée, comme si apprendre consistait à s'entraîner aux épreuves de contrôle... En réalité, leur logique n'est pas la même. Contrairement au questionnement évaluatif, le questionnement d'apprentissage devrait s'intéresser davantage aux réponses incorrectes qu'aux réponses correctes, puisqu'il s'agit d'introduire les élèves à maîtriser des distinctions qu'ils sont en train d'apprendre. Toutes les réponses devraient être exploitées, analysées, comparées, puisqu'il s'agit de faire apprendre ! Surtout lorsqu'elles contiennent des erreurs « intéressantes » parce que régulières ou significatives. [...]

Les problèmes complexes posés par le questionnement pédagogique paraissent dus, pour une large part, à la prééminence actuelle des méthodes inductives. On répète sur le mode de l'évidence que les notions doivent toujours être introduites à partir d'exemples et d'activités concrètes, pour ne se dévoiler progressivement en tant que telles qu'au terme du scénario. Le questionnement joue un rôle essentiel dans cet artifice, puisqu'il s'agit de faire dire par les élèves ce que précisément ils ignorent, en évitant de l'imposer de façon dogmatique. On admettra que c'est là un principe préférable à celui d'un cours magistral indigeste, mais il n'y a pas de raison que cela devienne une nouvelle « pensée unique » en pédagogie. En fait, il faut distinguer l'induction logique de l'induction pédagogique.

Sur le plan de la logique l'induction est une forme de raisonnement qui vise à dégager une loi à partir de cas particuliers, à dépasser les exemples au profit d'un concept. En fait, elle n'est jamais épistémologiquement valide, car elle comporte toujours le risque qu'un contre-exemple imprévu vienne contredire la généralisation. Le seul mode de raisonnement rigoureux est la déduction, mais celle-ci ne produit rien de neuf, puisque tout est déjà inscrit dans les prémisses du syllogisme. L'induction court donc toujours le risque de la réfutation, mais elle est essentielle dans le progrès de la connaissance. Elle est à la base du raisonnement expérimental, en permettant l'élaboration d'hypothèses plausibles.

Sur le plan pédagogique, l'induction est un procédé d'enseignement partant d'exemples et d'expériences pratiques, et qui s'appuie sur eux pour introduire de façon intuitive une règle, une loi, un théorème. Cette induction, actuellement préconisée dans de nombreuses disciplines, se démarque des pratiques traditionnelles qui commençaient par énoncer la règle avant de proposer des exercices d'application. La question n'est pas ici celle de la validité logique, mais celle d'une compréhension qui serait plus progressive et naturelle. Cette induction pédagogique est attrayante parce qu'elle semble plus concrète, plus proche des faits observables, mais le passage de l'exemple à la notion rappelle souvent la prestidigitation, comme le lapin qui sort du chapeau. Elle est souvent illusoire, car seul l'enseignant voit dans l'exemple le prototype d'une règle à venir, tandis que l'élève reste souvent scotché à l'exemple. Elle n'est donc pas sans vertu, mais il n'y a guère de raisons d'en faire le « régime » unique du moteur de la classe.

Outre que cela allonge considérablement les séquences, il vaut sans doute mieux différencier les moments où l'on raisonne de façon « ascendante » (de l'exemple au concept) et ceux où l'on raisonne de façon « descendante » (du concept à l'exemple). Quoi qu'on fasse, il faut bien changer de registre à un moment ou un autre, pour dégager la « pépite » conceptuelle de sa « gangue » d'exercices à répétition.

JEAN-PIERRE ASTOLFI, Le questionnement pédagogique,
Economie et management, 128:68-73, juin 2008

Pour reprendre la terminologie de JEAN-PIERRE ASTOLFI dans l'encadré ci-dessus, ce document est conçu de façon plutôt « ascendante » (de l'exemple au concept) et se veut complémentaire des notes de cours [1] qui, elles, sont conçues plutôt classiquement de façon « descendante » (du concept à l'exemple).

Nous introduisons donc ici une dizaine de questionnements qui couvrent l'ensemble des notions abordées lors du cours d'informatique S1 de l'ENIB. Chaque questionnement concerne un point particulier du cours ; il est structuré en 5 parties de la manière suivante :

1. Exemple : dans cette partie, des questions « simples » sont posées sur un problème « connu » de la « vie courante » afin d'introduire le concept informatique sous-jacent. On y trouvera des exemples tels qu'aller au restaurant (section 2), ranger un meuble à tiroirs (3), analyser les sorties d'un circuit logique (4), compter avec BOBBY LAPOINTE en base « bibi » (5), déterminer sa mention au bac (6), planter un clou (7), ranger des rondins de bois (8), cuisiner un quatre-quarts aux pépites de chocolat (9), jouer aux tours de Hanoï (10) ou encore trier un jeu de cartes (11).

Cette partie est principalement traitée par les étudiants eux-mêmes, individuellement ou en groupe.

2. Généralisation : dans cette partie, les concepts informatiques sous-jacents sont présentés et introduits à l'aide de questions plus « informatiques ». On y aborde les concepts d'algorithmique (section 2), d'affectation (3), de calculs booléens (4), de codage des nombres (5), de tests et d'alternatives (6), de boucles (7 et 8), de spécification de fonction (9), de récursivité (10) ou encore de manipulation de séquences (11).

En général, cette partie est traitée par l'enseignant.

3. Applications : des exemples « simples » d'application sont ensuite proposés.

Le premier exemple est en général traité *in extenso* par l'enseignant, les autres par les étudiants, en groupe ou individuellement.

4. Entraînement : cette partie est une préparation à l'évaluation qui a lieu en début de séance suivante. Les étudiants y travaillent chez eux entre les deux séances, individuellement ou en groupe.

- (a) Enoncé : on présente ici le problème que l'on souhaite traiter tel que le calcul en base « Shadok » (section 2), le calcul de facteurs de conversion entre unités physiques (3), l'établissement de la table de vérité d'une expression logique (4), l'écriture d'un nombre réel selon la norme IEEE 754 (5), la détermination de la valeur d'une fonction continue et linéaire par morceaux (6), le calcul d'un développement limité selon une certaine précision (7), le dessin d'un motif géométrique composé de polygones réguliers (8), la spécification d'une fonction connue (un « grand classique » de la programmation) (9), le parcours d'un arbre binaire (10) ou encore le tri d'un annuaire selon différents critères (11).

- (b) Exemple : un exemple est traité en détail dans cette partie en insistant sur la méthode pour arriver au résultat recherché et sur une ou des méthodes de vérification du résultat obtenu.

- (c) Questions : 24 questions de même difficulté sont proposées ici pour permettre à chaque étudiant de s'entraîner sur le problème à traiter.

Le jour de l'évaluation, chaque étudiant traitera individuellement une des 24 questions tirée au sort (une question différente par élève). Il sera demandé à chaque étudiant de s'auto-évaluer selon 3 critères : la méthode pour arriver au résultat, le résultat lui-même et la vérification du résultat.

5. Révisions : Cette partie fait le lien entre ce document et les notes de cours [1, 2].

La conclusion « tout en un » reprend le premier exemple du document (exemple « aller au restaurant » de la section 2) et le traite intégralement d'un point de vue informatique afin de mettre en œuvre toutes les notions abordées dans le cours d'informatique S1 de l'ENIB. Finalement, la liste des 98 questions proposées dans ce document est donnée en annexe page 72.

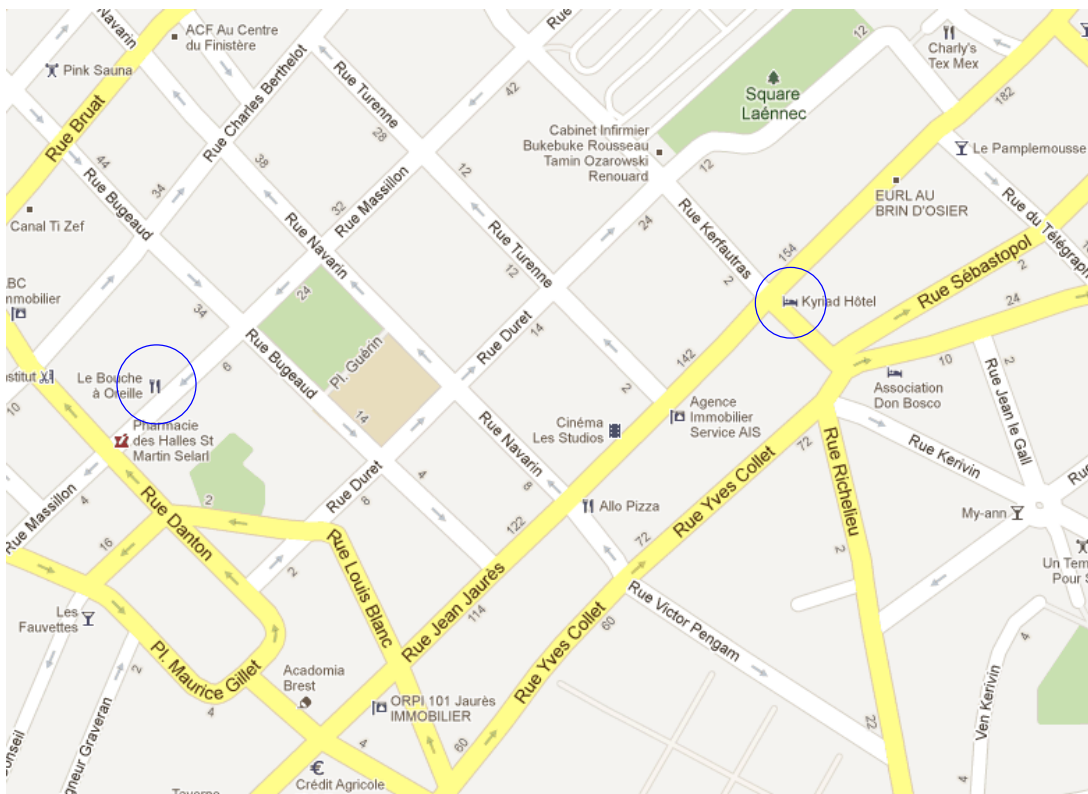
2 Qu'est-ce que l'algorithmique ?

Objectif : aborder les notions d'algorithme, d'algorithmique et de programmation.



2.1 Exemple

Objectif : Un touriste veut rejoindre le restaurant « le bouche à oreille » à partir de son hôtel « Kyriad » (voir plan ci-dessous).



Méthode : Il s'agit ici de décrire une suite ordonnée d'instructions (*aller tout droit, prenez la troisième à droite...*) qui manipulent des données (*carrefours, rues...*) pour réaliser la tâche désirée (*aller au restaurant*).

Questions

Q 2.1 (« aller au restaurant » : itinéraire) Proposer une suite d'instructions qui décrit un itinéraire pour aller de l'hôtel au restaurant.

On doit toujours vérifier que la description proposée ne contient que des instructions compréhensibles par celui qui devra l'exécuter et qu'elle réalise bien exactement la tâche pour laquelle elle a été conçue.

Q 2.2 (« aller au restaurant » : vérification) Faire vérifier l'itinéraire proposé par une tierce personne qui exécutera scrupuleusement les instructions dans l'ordre annoncé et vérifiera qu'elle se rend bien de l'hôtel au restaurant.

2.2 Généralisation

Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.

L'algorithmique est la science des algorithmes. Elle s'intéresse à l'art de construire des algorithmes ainsi qu'à caractériser leur validité, leur robustesse, leur réutilisabilité, leur complexité et leur efficacité.

Q 2.3 (algorithme : validité) *La validité d'un algorithme est son aptitude à réaliser exactement la tâche pour laquelle il a été conçu.*

L'itinéraire « aller au restaurant » proposé est-il valide ?

Q 2.4 (algorithme : robustesse) *La robustesse d'un algorithme est son aptitude à se protéger de conditions anormales d'utilisation.*

L'itinéraire « aller au restaurant » proposé s'applique-t-il aussi bien à une voiture qu'à un piéton ? L'itinéraire est-il « robuste » ?

Q 2.5 (algorithme : réutilisabilité) *La réutilisabilité d'un algorithme est son aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.*

L'itinéraire « aller au restaurant » proposé s'applique-t-il pour le restaurant « Tex Mex » du plan ? L'itinéraire est-il « réutilisable » ?

Q 2.6 (algorithme : complexité) *La complexité d'un algorithme est le nombre d'instructions élémentaires à exécuter pour réaliser la tâche pour laquelle il a été conçu.*

Pour l'itinéraire « aller au restaurant » proposé, quelle pourrait être l'instruction élémentaire pour un piéton ?

Q 2.7 (algorithme : efficacité) *L'efficacité d'un algorithme est son aptitude à utiliser de manière optimale les ressources du matériel qui l'exécute.*

Dans l'itinéraire « aller au restaurant » proposé, n'existerait-il pas un raccourci piétonnier pour aller plus vite au restaurant ? L'itinéraire est-il le plus « efficace » pour un piéton ? pour une « voiture » ?

L'algorithmique permet ainsi de passer d'un problème à résoudre à un algorithme qui décrit la démarche de résolution du problème, en caractérisant l'algorithme retenu. La programmation a alors pour rôle de traduire cet algorithme dans un langage « compréhensible » par l'ordinateur afin qu'il puisse exécuter l'algorithme automatiquement.

2.3 Applications

Q 2.8 (algorithme : tracés de polygones réguliers) *On cherche à faire dessiner une figure polygonale sur la plage à quelqu'un qui a les yeux bandés. Pour cela, on ne dispose que de 2 commandes orales : avancer de n pas en avant (n est un nombre entier de pas) et tourner à gauche d'un angle θ (rotation sur place de θ).*

1. *Faire dessiner un triangle équilatéral de 10 pas de côté.*
2. *Faire dessiner un carré de 10 pas de côté.*
3. *Faire dessiner un hexagone régulier de 10 pas de côté.*
4. *Faire dessiner un polygone régulier de n côtés de 10 pas chacun.*

Q 2.9 (algorithme : propriétés) *Quelle figure géométrique dessine-t-on en exécutant dans l'ordre la suite d'instructions ci-dessous ?*

1. avance de 2 pas,
2. tourne à gauche de 90° ,
3. avance de 3 pas,
4. tourne à gauche de 90° ,
5. avance de 4 pas,
6. tourne à gauche de 90° ,
7. avance de 5 pas,
8. tourne à gauche de 90° ,
9. avance de 6 pas.

Discuter des propriétés de cet algorithme : validité, robustesse, réutilisabilité, complexité, efficacité.

2.4 Entraînement

2.4.1 Enoncé

Contexte : « Les cerveaux des Shadoks avaient une capacité tout à fait limitée. Ils ne comportaient en tout que 4 cases. Comme ils n'avaient que 4 cases, évidemment les Shadoks ne connaissaient pas plus de 4 mots : GA, BU, ZO ET MEU. Etant donné qu'avec 4 mots, ils ne pouvaient pas compter plus loin que 4, le Professeur Shadoko avait réformé tout ça :

- Quand il n'y a pas de Shadok, on dit GA et on écrit GA.
- Quand il y a un Shadok de plus, on dit BU et on écrit BU.
- Quand il y a encore un Shadok, on dit ZO et on écrit ZO.
- Et quand il y en a encore un autre, on dit MEU et on écrit MEU.

Tout le monde applaudissait très fort et trouvait ça génial sauf le Devin Plombier qui disait qu'on n'avait pas idée d'inculquer à des enfants des bêtises pareilles et que Shadoko, il fallait le condamner. Il fut très applaudi aussi. Les mathématiques, cela les intéressait, bien sûr, mais brûler le professeur, c'était intéressant aussi, faut dire. Il fut décidé à l'unanimité qu'on le laisserait parler et qu'on le brûlerait après, à la récréation.

- Répétez avec moi : MEU ZO BU GA... GA BU ZO MEU.
- Et après ! ricanait le Plombier.
- Si je mets un Shadok en plus, évidemment, je n'ai plus assez de mots pour les compter, alors c'est très simple : on les jette dans une poubelle, et je dis que j'ai BU poubelle. Et pour ne pas confondre avec le BU du début, je dis qu'il n'y a pas de Shadok à côté de la poubelle et j'écris BU GA. BU Shadok à côté de la poubelle : BU BU. Un autre : BU ZO. Encore un autre : BU MEU. On continue. ZO poubelles et pas de Shadok à côté : ZO GA... MEU poubelles et MEU Shadoks à côté : MEU MEU. Arrivé là, si je mets un Shadok en plus, il me faut une autre poubelle. Mais comme je n'ai plus de mots pour compter les poubelles, je m'en débarrasse en les jetant dans une grande poubelle. J'écris BU grande poubelle avec pas de petite poubelle et pas de Shadok à côté : BU GA GA, et on continue... BU GA BU, BU GA ZO... MEU MEU ZO, MEU MEU MEU. Quand on arrive là et qu'on a trop de grandes poubelles pour pouvoir les compter, eh bien, on les met dans une super-poubelle, on écrit BU GA GA GA, et on continue... »

JACQUES ROUSSEL, *Les Shadoks* : GA BU ZO MEU, Circonflexe, 2000.

Remarque préliminaire : Le calcul « Shadok » repose sur les 4 chiffres GA, BU, ZO et MEU qui peuvent être assimilés aux 4 chiffres 0, 1, 2 et 3 :

- « Quand il n'y a pas de Shadok, on dit GA et on écrit GA. » $\Rightarrow \text{GA} = 0$
- « Quand il y a un Shadok de plus, on dit BU et on écrit BU. » $\Rightarrow \text{BU} = 1$
- « Quand il y a encore un Shadok, on dit ZO et on écrit ZO. » $\Rightarrow \text{ZO} = 2$
- « Et quand il y en a encore un autre, on dit MEU et on écrit MEU. » $\Rightarrow \text{MEU} = 3$

Il s'agit donc d'une **numération en base 4**.

Les différentes « poubelles Shadok » correspondent respectivement aux 4-aines (les « dizaines » de la base 4 : 4^1), aux 16-aines (les « centaines » de la base 4 : 4^2), aux 64-aines (les « milliers » de la base 4 : 4^3) :

- « Si je mets un Shadok en plus, évidemment, je n'ai plus assez de mots pour les compter, alors c'est très simple : on les jette dans une poubelle, et je dis que j'ai BU poubelle. Et pour ne pas confondre avec le BU du début, je dis qu'il n'y a pas de Shadok à côté de la poubelle et j'écris BU GA. BU Shadok à côté de la poubelle : BU BU. Un autre : BU ZO. Encore un autre : BU MEU. On continue. ZO poubelles et pas de Shadok à côté : ZO GA... MEU poubelles et MEU Shadoks à côté : MEU MEU. Arrivé là, si je mets un Shadok en plus, il me faut une autre poubelle. Mais comme je n'ai plus de mots pour compter les poubelles, je m'en débarrasse en les jetant dans une grande poubelle. J'écris BU grande poubelle avec pas de petite poubelle et pas de Shadok à côté : BU GA GA, et on continue... »

Il s'agit donc d'un **système de notation positionnelle** identique à celui qu'on utilise tous les jours. On peut donc « poser » les calculs comme en base décimale.

Rappel : Un entier positif en base b est représenté par une suite de chiffres $(r_n r_{n-1} \dots r_1 r_0)_b$ où les r_i sont des chiffres de la base b ($0 \leq r_i < b$). Ce nombre a pour valeur :

$$r_n b^n + r_{n-1} b^{n-1} + \dots + r_1 b^1 + r_0 b^0 = \sum_{i=0}^{i=n} r_i b^i$$

Objectif : exécuter les algorithmes de calcul arithmétique (+, −, ×, ÷) dans une base non décimale, ici la base « Shadok ».

Méthode : poser les opérations en base b .

Vérification : vérifier les calculs d'une part en passant par la base 10 et d'autre part, en effectuant la preuve par $b - 1$ (preuve par 3 en base 4, preuve par 9 en base 10).

2.4.2 Exemples

1. Soit à additionner $x = \text{BU BU GA MEU}$ et $y = \text{ZO BU ZO ZO}$. Compte-tenu du système de notation positionnelle en base 4 : $x = (1103)_4 = 1 \cdot 4^3 + 1 \cdot 4^2 + 0 \cdot 4^1 + 3 \cdot 4^0 = (83)_{10}$ et $y = (2122)_4 = 2 \cdot 4^3 + 1 \cdot 4^2 + 2 \cdot 4^1 + 2 \cdot 4^0 = (154)_{10}$.

On pose donc l'opération en base 4, puis on vérifie en base 10.

calcul direct en base 4	vérification en base 10
$\begin{array}{r} 1 \quad 1 \quad 10 \quad 3 \\ + \quad 2 \quad 1 \quad 2 \quad 2 \\ \hline = \quad 3 \quad 2 \quad 3 \quad 1 \end{array}$	$\begin{array}{r} \quad \quad \quad 8 \quad 3 \\ + \quad 1 \quad 5 \quad 4 \\ \hline = \quad 2 \quad 3 \quad 7 \end{array}$

$$z = x + y = (3231)_4 = 3 \cdot 4^3 + 2 \cdot 4^2 + 3 \cdot 4^1 + 1 \cdot 4^0 = (237)_{10}$$

En plus de la vérification par le passage en base 10, on peut effectuer les « preuves » par $b - 1$: « preuve » par 9 en base 10 (où 9 « vaut » 0), « preuve » par 3 en base 4 (où 3 « vaut » 0). Rappelons que si la preuve par $b - 1$ échoue, le résultat de l'opération est faux ; par contre, si la preuve par $b - 1$ réussit, le résultat de l'opération n'est pas forcément exact (2 erreurs peuvent se compenser).

preuve par 3 en base 4			preuve par 9 en base 10		
	$x = 1103$ $1 + 1 + 0 + 3 = 11$ $1 + 1 = 2$ 2			$x = 83$ $8 + 3 = 11$ $1 + 1 = 2$ 2	
$z = 3231$ $3 + 2 + 3 + 1 = 21$ $2 + 1 = 3$ 3	+	2 + 1 $2 + 1 = 3$ 3	$z = 237$ $2 + 3 + 7 = 12$ $1 + 2 = 3$ 3	+	2 + 1 $2 + 1 = 3$ 3
	$y = 2122$ $2 + 1 + 2 + 2 = 13$ $1 + 3 = 10$ $1 + 0 = 1$ 1			$y = 154$ $1 + 5 + 4 = 10$ $1 + 0 = 1$ 1	

Le passage par la base 10 et les « preuves » par 3 en base 4 et par 9 en base 10 confirment le résultat obtenu par le calcul direct. On obtient donc $(1103)_4 + (2122)_4 = (3231)_4$, soit en notation « Shadok » :

BU BU GA MEU + ZO BU ZO ZO = MEU ZO MEU BU

2. On opère à l'identique pour la soustraction des deux nombres $x = ZO ZO BU ZO$ et $y = BU ZO GA BU$.

calcul direct en base 4	vérification en base 10
$\begin{array}{r} 2 \quad 2 \quad 1 \quad 2 \\ - \quad 1 \quad 2 \quad 0 \quad 1 \\ \hline = \quad 1 \quad 0 \quad 1 \quad 1 \end{array}$	$\begin{array}{r} \quad \quad 1 \quad 16 \quad 16 \\ - \quad \quad 1 \quad 19 \quad 7 \\ \hline = \quad \quad 6 \quad 9 \end{array}$

$$z = x - y = (1011)_4 = 1 \cdot 4^3 + 0 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 = (69)_{10}$$

Pour les « preuves » par $b - 1$, on considère l'addition $z + y = x$.

preuve par 3 en base 4			preuve par 9 en base 10		
	$z = 1011$ $1 + 0 + 1 + 1 = 3$ 3			$z = 69$ $6 + 9 = 15$ $1 + 5 = 6$ 6	
$x = 2212$ $2 + 2 + 1 + 2 = 13$ $1 + 3 = 10$ 1	—	3 + 1 $3 + 1 = 10$ $1 + 0 = 1$ 1	$x = 166$ $1 + 6 + 6 = 13$ $1 + 3 = 4$ 4	—	6 + 7 $6 + 7 = 13$ $1 + 3 = 4$ 4
	$y = 1201$ $1 + 2 + 0 + 1 = 10$ $1 + 0 = 1$ 1			$y = 97$ $9 + 7 = 16$ $1 + 6 = 7$ 7	

Le passage par la base 10 et les « preuves » par 3 en base 4 et par 9 en base 10 confirment le résultat obtenu par le calcul direct. On obtient ainsi $(2212)_4 - (1201)_4 = (1011)_4$, soit en notation « Shadok » :

ZO ZO BU ZO – BU ZO GA BU = BU GA BU BU

3. La procédure reste la même pour le multiplication des deux nombres $x = \text{ZO BU BU ZO}$ et $y = \text{MEU BU GA}$.

calcul direct en base 4	vérification en base 10
$ \begin{array}{r} 2112 \\ \times 310 \\ \hline 0000 \\ + 2112 \\ + 13002 \\ \hline = 1321320 \end{array} $	$ \begin{array}{r} 150 \\ \times 52 \\ \hline 300 \\ + 750 \\ \hline = 7800 \end{array} $

$$z = x \times y = (1321320)_4 = 1 \cdot 4^6 + 3 \cdot 4^5 + 2 \cdot 4^4 + 1 \cdot 4^3 + 3 \cdot 4^2 + 2 \cdot 4^1 + 0 \cdot 4^0 = (7800)_{10}$$

preuve par 3 en base 4			preuve par 9 en base 10		
	$x = 2112$ $2 + 1 + 1 + 2 = 12$ $1 + 2 = 3$ 3			$x = 150$ $1 + 5 + 0 = 6$ 6	
$z = 1321320$ $1 + 3 + 2 + 1 + 3 + 2 + 0 = 30$ $3 + 0 = 3$ 3	×	3×1 $3 \times 1 = 3$ 3	$z = 7800$ $7 + 8 + 0 + 0 = 15$ $1 + 5 = 6$ 6	×	6×7 $6 \times 7 = 42$ $4 + 2 = 6$ 6
	$y = 310$ $3 + 1 + 0 = 10$ $1 + 0 = 1$ 1			$y = 52$ $5 + 2 = 7$ 7	

Le passage par la base 10 et les « preuves » par 3 en base 4 et par 9 en base 10 confirment le résultat obtenu par le calcul direct. On a donc $(2112)_4 \times (310)_4 = (1321320)_4$, et en notation « Shadok » :

ZO BU BU ZO × MEU BU GA = BU MEU ZO BU MEU ZO GA

4. Enfin, pour la division entière des deux nombres $x = \text{MEU ZO BU BU}$ et $y = \text{BU MEU ZO}$:

calcul direct en base 4	vérification en base 10
$ \begin{array}{r l} 3^12^111 & 132 \\ - ^11^132 & 13 \\ \hline = 123^11 & \\ - 11^122 & \\ \hline = 0103 & \end{array} $	$ \begin{array}{r l} 229 & 30 \\ - 210 & 7 \\ \hline = 019 & \end{array} $

$$q = x \div y = (13)_4 = 1 \cdot 4^1 + 3 \cdot 4^0 = (7)_{10} \text{ et } r = x \% y = (103)_4 = 1 \cdot 4^2 + 0 \cdot 4^1 + 3 \cdot 4^0 = (19)_{10}$$

$$x = (x \div y) \times y + (x \% y) = q \times y + r$$

Pour les « preuves » par $b - 1$, on considère la multiplication et l'addition $q \times y + r = x$.

preuve par 3 en base 4			preuve par 9 en base 10		
	$q = 13$ $1 + 3 = 10$ $1 + 0 = 1$ 1	$r = 103$ $1 + 0 + 3 = 10$ $1 + 0 = 1$ 1		$q = 7$ 7	$r = 19$ $1 + 9 = 10$ $1 + 0 = 1$ 1
$x = 3211$ $3 + 2 + 1 + 1 = 13$ $1 + 3 = 10$ 1	$\frac{\bullet}{\bullet}$	$1 \times 3 + 1$ $1 \times 3 + 1 = 10$ $1 + 0 = 1$ 1	$x = 229$ $2 + 2 + 9 = 13$ $1 + 3 = 4$ 4	$\frac{\bullet}{\bullet}$	$7 \times 3 + 1$ $7 \times 3 + 1 = 22$ $2 + 2 = 4$ 4
	$y = 132$ $1 + 3 + 2 = 12$ $1 + 2 = 3$ 3			$y = 30$ $3 + 0 = 3$ 3	

Le passage par la base 10 et les « preuves » par 3 en base 4 et par 9 en base 10 confirment le résultat obtenu par le calcul direct. On a finalement $(3211)_4 \div (132)_4 = (13)_4$, et en notation « Shadok » :

MEU ZO BU BU \div BU MEU ZO = BU MEU ou

MEU ZO BU BU = BU MEU ZO \times BU MEU + BU GA MEU

2.4.3 Questions

- MEU ZO GA BU \div MEU ZO ZO
- ZO BU ZO MEU \div ZO GA BU
- ZO BU BU ZO \div ZO BU GA
- ZO MEU BU ZO \div BU GA MEU
- MEU GA ZO MEU \div MEU GA ZO
- BU ZO BU ZO \div BU GA ZO
- BU GA ZO MEU \div MEU ZO ZO
- ZO MEU GA ZO \div BU MEU ZO
- BU BU ZO ZO \div BU GA BU
- ZO ZO GA MEU \div BU GA GA
- ZO GA ZO MEU \div ZO GA ZO
- BU ZO GA BU \div BU ZO ZO
- BU MEU MEU GA \div BU MEU GA
- BU MEU BU GA \div MEU ZO MEU
- MEU ZO GA BU \div ZO BU GA
- MEU GA MEU BU \div BU MEU GA
- ZO BU MEU MEU \div BU GA BU
- ZO GA MEU MEU \div BU ZO GA
- BU GA ZO MEU \div BU GA ZO
- MEU BU BU ZO \div MEU BU GA
- BU GA MEU BU \div BU GA GA
- ZO MEU MEU GA \div ZO MEU GA
- MEU ZO BU MEU \div ZO ZO ZO
- MEU MEU GA ZO \div ZO MEU MEU

2.5 Révisions

Cours	[1] : chapitre 1, section 1.1
TD	[2] : exercices 1.1 à 1.4, 1.19 à 1.28

3 Qu'est-ce que l'affectation ?

Objectif : comprendre l'instruction d'affectation en informatique.

Syntaxe Python :

- `variable = constante`
- `variable = expression`

3.1 Exemple

Objectif : on veut ranger des objets de différents types dans un meuble à tiroirs pour pouvoir les retrouver ultérieurement.

Méthode : proposer un mode de désignation des tiroirs qui permettra à une tierce personne de retrouver sans ambiguïté les objets recherchés.



Question :

Q 3.1 (« meuble à tiroirs » : désignation des tiroirs) *Proposer un mode de désignation des tiroirs qui permette de retrouver les objets.*

Q 3.2 (« meuble à tiroirs » : échange de contenus) *Utiliser ce mode de désignation pour expliquer comment échanger les contenus de deux tiroirs.*

3.2 Généralisation

Une variable est un objet informatique qui associe un nom à une valeur qui peut éventuellement varier au cours du temps. Une variable peut être vue comme une case en mémoire vive, que le programme va repérer par une étiquette (une adresse ou un nom). Pour avoir accès au contenu de la case (la valeur de la variable), il suffit de la désigner par son étiquette : c'est-à-dire soit par son adresse en mémoire, soit par son nom.

Q 3.3 (affectation : nommer les variables) *Dans l'exemple du « meuble à tiroirs », les tiroirs ont-ils un nom ? Si oui, ce nom se réfère-t-il à son contenu (ie. a-t-on une idée du contenu d'un tiroir rien qu'en connaissant son nom) ?*

L'affectation est l'opération qui consiste à attribuer une valeur à une variable.

Q 3.4 (affectation : constantes) *Donner quelques exemples de constantes possibles que l'on peut affecter à une variable.*

Q 3.5 (affectation : expressions) *Donner quelques exemples d'expressions possibles que l'on peut affecter à une variable.*

Q 3.6 (affectation : incrémentation) *Donner un exemple d'incrément d'une variable.*

Q 3.7 (affectation : égalité mathématique?) *Montrer sur un exemple simple que l'affectation informatique est une opération différente de l'égalité mathématique.*

Q 3.8 (affectation : opération commutative?) *Montrer sur un exemple simple que l'affectation n'est pas commutative.*

Q 3.9 (affectation : fonctionnement) *Décrire l'exécution d'une instruction d'affectation.*

L'affectation a ainsi pour effet de réaliser plusieurs opérations dans la mémoire de l'ordinateur :

- créer et mémoriser un nom de variable,
- lui attribuer un type bien déterminé,
- créer et mémoriser une valeur particulière,
- établir un lien (par un système interne de pointeurs) entre le nom de la variable et l'emplacement mémoire de la valeur correspondante.

3.3 Applications

Q 3.10 (unité de longueur) *L'année-lumière (al) est une unité de distance utilisée en astronomie. Une année-lumière est la distance parcourue par un photon (ou plus simplement la lumière) dans le vide, en dehors de tout champ gravitationnel ou magnétique, en une année julienne (365,25 jours).*

Ecrire une instruction qui permette de passer directement des années-lumière aux m sachant que la vitesse de la lumière dans le vide est de 299 792 458 m/s.

Q 3.11 (permutation circulaire) *Effectuer une permutation circulaire gauche entre les valeurs de 3 entiers x , y et z .*

Q 3.12 (exécution d'une séquence d'affectations) *Quelles sont les valeurs des variables a , b , q et r après les séquences d'affectations suivantes ?*

$a = 12$	$a = 19$
$b = 18$	$b = 6$
$r = a \% b$	$q = 0$
$a = b$	$r = a$
$b = r$	$r = r - b$
$r = a \% b$	$q = q + 1$
$a = b$	$r = r - b$
$b = r$	$q = q + 1$
$r = a \% b$	$r = r - b$
$a = b$	$q = q + 1$
$b = r$	

3.4 Entraînement

3.4.1 Enoncé

Contexte : Le système international d'unités (SI) en physique est composé de 7 unités de base et de 2 unités supplémentaires définies dans le tableau ci-dessous :

Grandeur	Unité	Symbole
longueur	mètre	m
masse	kilogramme	kg
temps	seconde	s
intensité de courant électrique	ampère	A
température thermodynamique	kelvin	K
quantité de matière	mole	mol
intensité lumineuse	candela	cd
angle plan	radian	rad
angle solide	stéradian	sr

Pour des raisons historiques, culturelles ou pragmatiques, un certain nombre d'unités hors-système, telles que l'heure (h), le grade (gr), le pound (lb), le nœud (kn), l'année-lumière (al) ou encore l'électronvolt (eV), peuvent être utilisées. Il est par contre nécessaire de connaître leur facteur de conversion en unités SI. Le tableau ci-dessous donne les facteurs de conversion des unités les plus connues.

Grandeur	Facteur de conversion			Domaine
année-lumière	1 al	=	9.46053×10^{15} m	astronomie
atmosphère	1 atm	=	1.01325×10^5 Pa	météorologie
baril	1 b	=	0.15891×10^0 m ³	pétrole
calorie	1 cal	=	4.184×10^0 J	thermique
cheval-vapeur	1 ch	=	735.499×10^0 W	mécanique
curie	1 Ci	=	3.7×10^{10} Bq	radioactivité
degré	1 °	=	1.745329×10^{-2} rad	géométrie
électronvolt	1 eV	=	1.602189×10^{-19} J	physique nucléaire
faraday	1 F	=	9.64870×10^4 C	électricité
foot	1 ft	=	30.48×10^{-2} m	géométrie
franklin	1 Fr	=	3.33564×10^{-10} C	électricité
frigorie	1 fg	=	4.186×10^3 J	thermique
gallon	1 gal	=	3.78541×10^{-3} m ³	volume
grade	1 gr	=	1.570796×10^{-2} rad	géométrie
heure	1 h	=	3.6×10^3 s	temps
inch	1 in	=	2.54×10^{-2} m	géométrie
lambert	1 L	=	3.183×10^3 cd · m ⁻²	photométrie
mile	1 mile	=	1.609344×10^3 m	géométrie
millimètre de mercure	1 mmHg	=	133.3224×10^0 Pa	météorologie
nœud	1 nd	=	0.514444×10^0 m · s ⁻¹	marine
oersted	1 Oe	=	79.57747×10^0 A · m ⁻¹	magnétisme
parsec	1 pc	=	3.0857×10^{16} m	astronomie
pica	1 pica	=	4.2175×10^{-3} m	typographie
torr	1 Torr	=	133.3224×10^0 Pa	météorologie

Objectif : utiliser l'affectation pour calculer un facteur de conversion entre deux unités physiques comptatibles.

Méthode : déterminer une relation entre les deux unités considérées en tenant compte de leur définition.

Vérification : vérifier le bon fonctionnement sous PYTHON en testant avec des valeurs remarquables connues.

3.4.2 Exemple

On veut convertir une température FAHRENHEIT en une température CELSIUS.

- La température CELSIUS t_C (en °C) est définie en fonction de la température thermodynamique T (en K) par la relation $t_C = T - 273.15$.
- La température FAHRENHEIT t_F (en °F) est définie en fonction de la température thermodynamique T (en K) par la relation $t_F = 9/5 \cdot T - 459.67$.

Méthode : On cherche à exprimer t_C (la température souhaitée) en fonction de t_F (la température donnée). De la définition de t_F , on peut exprimer la température thermodynamique T en fonction de la température FAHRENHEIT :

$$T = 5/9 \cdot (t_F + 459.67)$$

On reporte l'expression de T dans la définition de la température CELSIUS pour obtenir une relation liant t_C à t_F :

$$t_C = 5/9 \cdot (t_F + 459.67) - 273.15.$$

Résultat En PYTHON, une simple instruction d'affectation permet donc de passer de la température FAHRENHEIT initiale (f) à la température CELSIUS recherchée (c).

Pour la tester, on fixera f à des valeurs connues : température de l'eau glacée ($t_s = 32^\circ F$) et température de l'eau bouillante ($t_g = 212^\circ F$), et on vérifiera qu'on obtient bien respectivement $c_s = 0^\circ C$ et $c_g = 100^\circ C$.

```
1 c = 5/9*(f + 459.67) - 273.15
```

Vérification on compare le résultat obtenu avec la valeur connue de la température CELSIUS.

température de l'eau glacée

```
>>> f = 32
>>> c = 5/9*(f + 459.67) - 273.15
>>> c - 0
5.684341886080802e-14
```

température de l'eau bouillante

```
>>> f = 212
>>> c = 5/9*(f + 459.67) - 273.15
>>> c - 100
5.684341886080802e-14
```

Les différences observées sont bien quasi-nulles ($10^{-14} \approx 0$).

3.4.3 Questions

Ecrire une affectation qui calcule les facteurs de conversion suivants.

- | | |
|----------------------------|-----------------------|
| 1. année-lumière en mile | 7. inch en pica |
| 2. année-lumière en parsec | 8. foot en inch |
| 3. parsec en foot | 9. centimètre en pica |
| 4. parsec en inch | 10. mile en inch |
| 5. parsec en année-lumière | 11. mile en foot |
| 6. inch en foot | 12. année en minute |

- | | |
|-----------------------------|-----------------------------------|
| 13. année en seconde | 19. électronvolt en frigorie |
| 14. baril en litre | 20. frigorie en calorie |
| 15. baril en gallon | 21. franklin en faraday |
| 16. litre en gallon | 22. nœud en kilomètre par heure |
| 17. litre en baril | 23. torr en millimètre de mercure |
| 18. électronvolt en calorie | 24. atmosphère en torr |

3.5 Révisions

Cours	[1] : chapitre 2, section 2.2
TD	[2] : exercices 2.1 à 2.4, 2.26 à 2.29

4 Comment calcule-t-on avec des opérateurs booléens ?

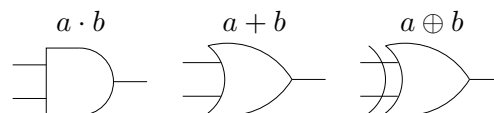
Objectif : connaître et manipuler les principaux opérateurs booléens.

Syntaxe Python :

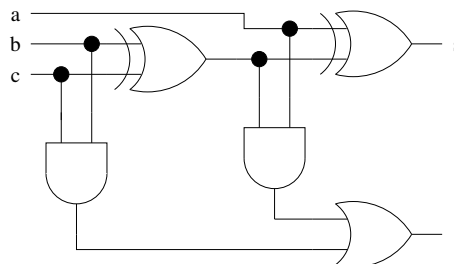
- not a
- a or b
- a and b
- a == b
- a != b

4.1 Exemple

Enoncé : On considère les conventions graphiques traditionnelles pour les opérateurs logiques and (\cdot), or ($+$) et xor (\oplus) :



On cherche à établir la table de vérité du circuit logique ci-dessous où a , b et c sont les entrées, s et t les sorties.



Questions :

Q 4.1 (« circuit logique » : combinatoire) Combien y a-t-il de combinaisons différentes possibles en entrée du circuit logique considéré (nombre de triplets (a, b, c) différents) ?

Q 4.2 (« circuit logique » : table de vérité) Déterminer les valeurs des sorties (s, t) pour chaque triplet d'entrée (a, b, c) possible.

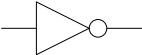

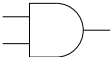

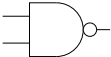
Dans la pratique, un circuit logique est conçu pour une fonction précise. Il faut alors vérifier qu'il remplit bien sa fonction.

Q 4.3 (« circuit logique » : vérification) Vérifier que ce circuit effectue l'addition des 3 bits a , b et c : s est la somme et t la retenue.

$$\begin{array}{r}
 a \\
 + \quad b \\
 + \quad c \\
 \hline
 = \quad t \quad s
 \end{array}$$

4.2 Généralisation

Les 3 opérateurs booléens de base : **not** (négation), **and** (conjonction) et **or** (disjonction), sont définis par leur table de vérité.

négation \bar{a}	disjonction $a + b$	conjonction $a \cdot b$																																				
not a <table><tr><th>a</th><th>\bar{a}</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table> 	a	\bar{a}	0	1	1	0	a or b <table><tr><th>a</th><th>b</th><th>$a + b$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> 	a	b	$a + b$	0	0	0	0	1	1	1	0	1	1	1	1	a and b <table><tr><th>a</th><th>b</th><th>$a \cdot b$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> 	a	b	$a \cdot b$	0	0	0	0	1	0	1	0	0	1	1	1
a	\bar{a}																																					
0	1																																					
1	0																																					
a	b	$a + b$																																				
0	0	0																																				
0	1	1																																				
1	0	1																																				
1	1	1																																				
a	b	$a \cdot b$																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				
	not (a or b) 	not (a and b) 																																				

On peut démontrer que ces opérateurs vérifient les propriétés suivantes :

$$\forall a, b, c \in \{0; 1\}$$

identité : $a = \bar{\bar{a}}$

élément neutre : $a + 0 = a$

élément nul : $a + 1 = 1$

idempotence : $a + a = a$

complémentarité : $a + \bar{a} = 1$

absorption : $a + (a \cdot b) = a$

simplification : $a + (\bar{a} \cdot b) = (a + b)$

commutativité : $(a + b) = (b + a)$

associativité : $(a + b) + c = a + (b + c)$

distributivité : $a + (b \cdot c) = (a + b) \cdot (a + c)$

De Morgan : $\overline{a + b} = \bar{a} \cdot \bar{b}$

$$a \cdot 1 = a$$

$$a \cdot 0 = 0$$

$$a \cdot a = a$$

$$a \cdot \bar{a} = 0$$

$$a \cdot (a + b) = a$$

$$a \cdot (\bar{a} + b) = (a \cdot b)$$

$$(a \cdot b) = (b \cdot a)$$

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

$$\overline{a \cdot b} = \bar{a} + \bar{b}$$

Q 4.4 (opérateurs booléens : distributivité) Démontrer les propriétés de distributivité :

$$\forall a, b, c \in \{0; 1\}$$

1. $a + (b \cdot c) = (a + b) \cdot (a + c)$

2. $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

On comparera pour cela les tables de vérité des deux membres de l'égalité.

Q 4.5 (opérateurs booléens : De Morgan) Démontrer les propriétés de De Morgan :


$$\forall a, b \in \{0; 1\}$$

1. $\overline{a + b} = \bar{a} \cdot \bar{b}$

2. $\overline{a \cdot b} = \bar{a} + \bar{b}$

On comparera pour cela les tables de vérité des deux membres de l'égalité.

A partir de ces 3 opérateurs de base, on peut définir des opérateurs dérivés tels que l'équivalence (\Leftrightarrow), l'implication (\Rightarrow) et la disjonction exclusive (\oplus).

équivalence $a \Leftrightarrow b$	implication $a \Rightarrow b$	ou exclusif $a \oplus b$																																													
a == b		a != b																																													
<table> <tr><th>a</th><th>b</th><th>$a \Leftrightarrow b$</th></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	a	b	$a \Leftrightarrow b$	0	0	1	0	1	0	1	0	0	1	1	1	<table> <tr><th>a</th><th>b</th><th>$a \Rightarrow b$</th></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	a	b	$a \Rightarrow b$	0	0	1	0	1	1	1	0	0	1	1	1	<table> <tr><th>a</th><th>b</th><th>$a \oplus b$</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table> 	a	b	$a \oplus b$	0	0	0	0	1	1	1	0	1	1	1	0
a	b	$a \Leftrightarrow b$																																													
0	0	1																																													
0	1	0																																													
1	0	0																																													
1	1	1																																													
a	b	$a \Rightarrow b$																																													
0	0	1																																													
0	1	1																																													
1	0	0																																													
1	1	1																																													
a	b	$a \oplus b$																																													
0	0	0																																													
0	1	1																																													
1	0	1																																													
1	1	0																																													

Q 4.6 (opérateurs booléens : opérateurs dérivés) Définir l'équivalence (\Leftrightarrow), l'implication (\Rightarrow) et la disjonction exclusive (\oplus) en fonction des 3 opérateurs de base (négation, conjonction et disjonction).

4.3 Applications

Q 4.7 (développer une expression booléenne) Développer les expressions booléennes suivantes en justifiant chaque étape du développement. Vérifier avec PYTHON.

- $t = \overline{a + b + \bar{c} \cdot \bar{d}}$
- $t = \overline{a \cdot \bar{b} \cdot c \cdot d}$
- $t = \overline{a \cdot \bar{b} \cdot c + \bar{d}}$
- $t = \overline{\bar{a} \cdot \bar{b} + c \cdot d}$
- $t = \overline{a + b \cdot c \cdot d}$
- $t = \overline{\bar{a} \cdot b + c + \bar{d}}$

Q 4.8 (fonctions booléennes binaires) Combien peut-on définir de fonctions booléennes binaires $f_i : \forall a, b, c \in \{0; 1\}, c = f_i(a, b)$? Identifier chacune de ces fonctions : on doit en particulier retrouver les opérateurs de base \cdot et $+$.

4.4 Entraînement

4.4.1 Enoncé

Objectif : établir la table de vérité d'une expression booléenne.

Méthode : introduire des variables intermédiaires pour faciliter les calculs.

Vérification : vérifier avec PYTHON les résultats obtenus pour différentes valeurs des entrées.

4.4.2 Exemple

Soit à établir la table de vérité de l'expression : $z = ((\bar{a} \Rightarrow \bar{b}) \cdot (b \Rightarrow c)) \Rightarrow ((c \Rightarrow a) \Rightarrow d)$

Méthode : On pose : $p = (\bar{a} \Rightarrow \bar{b})$, $q = (b \Rightarrow c)$, $r = p \cdot q$, $s = (c \Rightarrow a)$ et $t = (s \Rightarrow d)$. On a donc finalement : $z = (r \Rightarrow t)$.

Puis, on transforme les expression du type $(p \Rightarrow q)$ en $(\bar{p} + q)$; elles ont en effet les mêmes tables de vérité :

p	q	$p \Rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

p	\bar{p}	q	$\bar{p} + q$
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1

Résultat : La table de vérité recherchée comporte $2^4 = 16$ entrées (4 variables a, b, c, d , de 2 valeurs possibles chacune : 0 ou 1) :

a	b	c	d	p $\bar{a} \Rightarrow \bar{b}$ $a + \bar{b}$	q $b \Rightarrow c$ $\bar{b} + c$	r $p \cdot q$	s $c \Rightarrow a$ $\bar{c} + a$	t $s \Rightarrow d$ $\bar{s} + d$	z $r \Rightarrow t$ $\bar{r} + t$
0	0	0	0	1	1	1	1	0	0
0	0	0	1	1	1	1	1	1	1
0	0	1	0	1	1	1	0	1	1
0	0	1	1	1	1	1	0	1	1
0	1	0	0	0	0	0	1	0	1
0	1	0	1	0	0	0	1	1	1
0	1	1	0	0	1	0	0	1	1
0	1	1	1	0	1	0	0	1	1
1	0	0	0	1	1	1	1	0	0
1	0	0	1	1	1	1	1	1	1
1	0	1	0	1	1	1	1	0	0
1	0	1	1	1	1	1	1	1	1
1	1	0	0	1	0	0	1	0	1
1	1	0	1	1	0	0	1	1	1
1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1

Vérification : On peut vérifier le résultat obtenu avec PYTHON en utilisant les mêmes notations que précédemment.

L'expression z est simplement calculée par la séquence d'affectations suivante :

```
p = a or not b
q = not b or c
r = p and q
s = not c or a
t = not s or d
z = not r or t
```

```
>>> a, b, c, d = 0, 0, 0, 0
>>> p = a or not b
>>> q = not b or c
>>> r = p and q
>>> s = not c or a
>>> t = not s or d
>>> z = not r or t
>>> z
0
```

```
>>> a, b, c, d = 0, 1, 0, 1
>>> p = a or not b
>>> q = not b or c
>>> r = p and q
>>> s = not c or a
>>> t = not s or d
>>> z = not r or t
>>> z
1
```


L'utilisation des boucles en PYTHON permettra d'obtenir directement la table de vérité.

<pre> 1 print("a","b","c","d"," ",\ 2 "p","q","r","s","t"," ","z") 3 print(7*"-"," ",9*"-"," ",1*"-") 4 for a in [0,1] : 5 for b in [0,1] : 6 for c in [0,1] : 7 for d in [0,1] : 8 p = a or not b 9 q = not b or c 10 r = p and q 11 s = not c or a 12 t = not s or d 13 z = not r or t 14 print(a,b,c,d," ", \ 15 int(p),int(q),int(r),\ 16 int(s),int(t)," ",\ 17 int(z)) </pre>	<table border="1"> <thead> <tr> <th>a</th><th>b</th><th>c</th><th>d</th><th>p</th><th>q</th><th>r</th><th>s</th><th>t</th><th>z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	a	b	c	d	p	q	r	s	t	z	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	0	0	1	0	1	1	1	0	1	1	0	0	1	1	1	1	1	0	1	1	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	1	0	1	0	0	1	1	1	0	0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	1	0	1	1	1	0	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1
a	b	c	d	p	q	r	s	t	z																																																																																																																																																																		
0	0	0	0	1	1	1	1	0	0																																																																																																																																																																		
0	0	0	1	1	1	1	1	1	1																																																																																																																																																																		
0	0	1	0	1	1	1	0	1	1																																																																																																																																																																		
0	0	1	1	1	1	1	0	1	1																																																																																																																																																																		
0	1	0	0	0	0	0	1	0	1																																																																																																																																																																		
0	1	0	1	0	0	0	1	1	1																																																																																																																																																																		
0	1	1	0	0	1	0	0	1	1																																																																																																																																																																		
0	1	1	1	0	1	0	0	1	1																																																																																																																																																																		
1	0	0	0	1	1	1	1	0	0																																																																																																																																																																		
1	0	0	1	1	1	1	1	1	1																																																																																																																																																																		
1	0	1	0	1	1	1	1	0	0																																																																																																																																																																		
1	0	1	1	1	1	1	1	1	1																																																																																																																																																																		
1	1	0	0	1	0	0	1	0	1																																																																																																																																																																		
1	1	0	1	1	0	0	1	1	1																																																																																																																																																																		
1	1	1	0	1	1	1	1	0	0																																																																																																																																																																		
1	1	1	1	1	1	1	1	1	1																																																																																																																																																																		

4.4.3 Questions

Etablir la table de vérité des expressions booléennes suivantes en faisant apparaître des variables intermédiaires de calcul.

- | | |
|--|---|
| 1. $z = ((a \Rightarrow b) \cdot (b \Rightarrow \bar{c})) \Rightarrow (\bar{c} \Rightarrow \bar{a})$ | 13. $z = ((a \Rightarrow b) + \overline{(b \Rightarrow \bar{c})}) \Rightarrow (\bar{c} \oplus \bar{a})$ |
| 2. $z = ((\bar{a} \Rightarrow \bar{b}) \cdot (b \Rightarrow c)) \Rightarrow (c \Rightarrow a)$ | 14. $z = ((a \Rightarrow b) + \overline{(b \Rightarrow \bar{c})}) \oplus (c \Rightarrow a)$ |
| 3. $z = ((a \cdot b) \Rightarrow (b \cdot c)) \Rightarrow (c + \bar{a})$ | 15. $z = ((a + b) \oplus \overline{(b \cdot c)}) \Rightarrow (c + \bar{a})$ |
| 4. $z = ((a + b) \Rightarrow (b + c)) \Rightarrow (\bar{c} \Rightarrow \bar{a})$ | 16. $z = (\overline{(a \oplus \bar{b})} \Rightarrow (b \cdot c)) \Rightarrow (\bar{c} \oplus \bar{a})$ |
| 5. $z = ((a \Rightarrow b) + (b \Rightarrow c)) \Rightarrow (\bar{c} + \bar{a})$ | 17. $z = ((a \Rightarrow \bar{b}) \cdot \overline{(b \Rightarrow c)}) \Rightarrow (c + a)$ |
| 6. $z = ((a \Rightarrow b) + (b \Rightarrow c)) \Rightarrow (\bar{c} \Rightarrow \bar{a})$ | 18. $z = ((a \Rightarrow \bar{b}) \cdot (b \oplus c)) \Rightarrow \overline{(\bar{c} \Rightarrow \bar{a})}$ |
| 7. $z = ((a \Rightarrow b) \cdot (b \Rightarrow \bar{c})) \Rightarrow (\bar{c} \oplus \bar{a})$ | 19. $z = ((a \Rightarrow b) \cdot \overline{(b \Rightarrow \bar{c})}) \Rightarrow (\bar{c} \oplus \bar{a})$ |
| 8. $z = ((\bar{a} \Rightarrow \bar{b}) \cdot (b \Rightarrow c)) \oplus (c \Rightarrow a)$ | 20. $z = ((a \Rightarrow b) \cdot \overline{(b \Rightarrow c)}) \oplus (c \Rightarrow a)$ |
| 9. $z = ((a \cdot b) \oplus (b \cdot c)) \Rightarrow (c + \bar{a})$ | 21. $z = ((a + b) + \overline{(b \cdot c)}) \Rightarrow (c + \bar{a})$ |
| 10. $z = ((a \oplus b) \Rightarrow (b + c)) \Rightarrow (\bar{c} \oplus \bar{a})$ | 22. $z = (\overline{(a \oplus \bar{b})} \Rightarrow (b \cdot c)) \Rightarrow (\bar{c} \oplus \bar{a})$ |
| 11. $z = ((a \Rightarrow b) + \overline{(b \Rightarrow c)}) \Rightarrow (\bar{c} + \bar{a})$ | 23. $z = ((a \Rightarrow \bar{b}) \cdot \overline{(b \Rightarrow c)}) \Rightarrow (c \cdot a)$ |
| 12. $z = ((a \Rightarrow b) + (b \oplus c)) \Rightarrow (\bar{c} \Rightarrow \bar{a})$ | 24. $z = ((a \Rightarrow \bar{b}) \cdot (b + c)) \Rightarrow \overline{(\bar{c} \Rightarrow \bar{a})}$ |

4.5 Révisions

Cours	[1] : chapitre 2, section 2.3.1
TD	[2] : exercices 2.5 à 2.7, 2.29

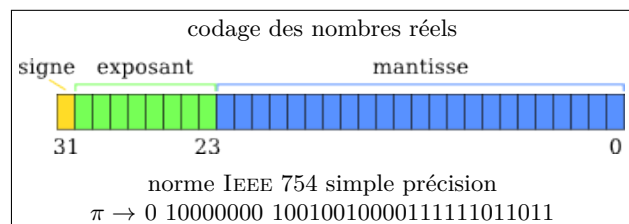
5 Comment coder un nombre ?

Objectif : savoir coder les nombres, entiers et réels.

Syntaxe Python :

```
>>> (1000).to_bytes(2, byteorder='big', signed=True)
b'\x03\xe8'
>>> (-1000).to_bytes(2, byteorder='big', signed=True)
b'\xfc\x18'
```

```
>>> pi
3.141592653589793
>>> float.hex(pi)
'0x1.921fb54442d18p+1'
>>> float.hex(-pi)
'-0x1.921fb54442d18p+1'
```



5.1 Exemple

Enoncé : On cherche à coder un entier dans une base b définie par b chiffres élémentaires : ici, $b = 16$ (base hexadécimale).

Comme on parle de binaire pour la base 2 (2^1), Bobby Lapointe (1922-1972, chanteur, fêré de mathématiques) proposa de dire « bi-binaire » pour la base 4 (2^2), et « bi-bi-binaire » pour la base 16 (2^4), terme qu'il abrégéa en « bibi ». À partir de ce postulat, Bobby Lapointe inventa une notation où à l'aide de quatre consonnes (B, D, H, K) et de quatre voyelles (A, E, I, O), il obtint les seize combinaisons nécessaires et suffisantes pour compter en base « bibi » (16) : HO (0), HA (1), HE (2), HI (3), BO (4), BA (5), BE (6), BI (7), KO (8), KA (9), KE (10), KI (11), DO (12), DA (13), DE (14), DI (15).

Questions :

Q 5.1 (« base bibi » : décodage) Décoder en base décimale l'entier $n = (HIDEKO)_{bibi}$.

Q 5.2 (« base bibi » : codage) Coder en base « bibi » l'entier $n = (538)_{10}$.

5.2 Généralisation

Un entier positif en base b est représenté par une suite de chiffres $(r_n r_{n-1} \dots r_1 r_0)_b$ où les r_i sont des chiffres de la base b ($0 \leq r_i < b$). Ce nombre a pour valeur :

$$r_n b^n + r_{n-1} b^{n-1} + \dots + r_1 b^1 + r_0 b^0 = \sum_{i=0}^{i=n} r_i b^i$$

Exemples :

$(123)_{10}$	$=$	$1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$	$=$	$(123)_{10}$
$(123)_5$	$=$	$1 \cdot 5^2 + 2 \cdot 5^1 + 3 \cdot 5^0$	$=$	$(38)_{10}$
$(123)_8$	$=$	$1 \cdot 8^2 + 2 \cdot 8^1 + 3 \cdot 8^0$	$=$	$(83)_{10}$
$(123)_{16}$	$=$	$1 \cdot 16^2 + 2 \cdot 16^1 + 3 \cdot 16^0$	$=$	$(291)_{10}$

Q 5.3 (codage binaire d'un entier positif)

1. Décoder en base décimale les nombres binaires suivants : $(10110010)_2$, $(00110001)_2$ et $(11011101)_2$.
2. Coder en base 2 les entiers décimaux suivants : $(532)_{10}$, $(493)_{10}$ et $(77)_{10}$.

Pour coder les nombres négatifs, une première idée est de réserver un bit pour le signe, les autres bits représentant la valeur absolue du nombre.

Q 5.4 (nombres relatifs) On veut coder un nombre relatif (entier négatif, positif ou nul) sur $k = 8$ bits en réservant le bit le plus à gauche pour le signe ($+$: 0, $-$: 1) et les 7 bits restant pour coder la valeur absolue du nombre.

1. Coder l'entier -57 sur 8 bits selon cette méthode.
2. Montrer qu'il y a deux manières possibles de coder 0.
3. Montrer que si l'un des nombres a ou b est négatif, alors l'addition binaire $a + b$ ne donne pas le résultat escompté.

Pour remédier aux problèmes précédents, on utilise la représentation en « complément à 2 » pour coder un entier relatif n sur k bits. Les entiers positifs sont représentés comme précédemment : bit de poids fort à 0, nombre codé sur $(k - 1)$ bits. Les nombres négatifs sont par contre obtenus en codant sur k bits $(2^k - |n|)$ de la manière suivante :

- on code la valeur absolue sur k bits puis on inverse les bits un à un (« complément à un » : les 0 deviennent des 1, les 1 deviennent des 0),
- on ajoute 1 au résultat (les dépassements à gauche sont ignorés).

Q 5.5 (complément à 2)

1. Coder le nombre $n = -41$ en complément à 2 sur 8 bits.
2. Montrer que sur k bits $(n + (-n))$ donne bien 0.
3. Montrer que sur k bits le complément à 2 de $(-n)$ redonne bien n ($n = -(-n)$).
4. Montrer que les deux inconvénients de la méthode précédente (question 5.4 2. et 3.) n'existent plus avec la représentation en complément à 2.
5. Déterminer la plage de valeurs entières possibles lorsqu'un entier positif, négatif ou nul est codé en binaire sur k chiffres dans la représentation en complément à 2.

5.3 Applications

Q 5.6 (base D'ni) D'NI est un univers imaginaire sur lequel s'appuient les jeux vidéo de la série MYST. Pour compter, les D'NI utilisent un système quinquévigésimal (base 25).

1. Décoder en base décimale l'entier $n = (21, 17, 0, 23)_{D'NI}$ exprimé en base D'NI.
2. Coder l'entier décimal $n = (3562)_{10}$ en base D'NI.

5.4 Entraînement**5.4.1 Enoncé**

Objectif : Coder un nombre réel x selon la norme IEEE 754 simple précision.

Méthode : Un nombre fractionnaire $(r_n r_{n-1} \dots r_1 r_0 . r_{-1} r_{-2} \dots)_b$ (nombre avec des chiffres après la virgule) est défini sur un sous-ensemble borné, incomplet et fini des rationnels. Un tel nombre a pour valeur dans la base b :

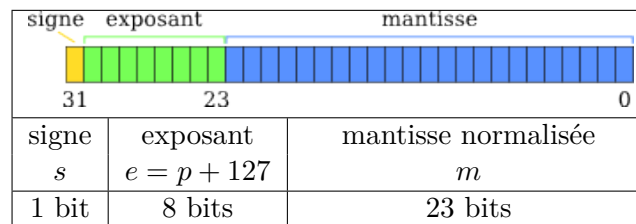
$$r_n b^n + r_{n-1} b^{n-1} + \dots + r_1 b^1 + r_0 b^0 + r_{-1} b^{-1} + r_{-2} b^{-2} + \dots$$

En pratique, le nombre de chiffres après la virgule est limité par la taille physique en machine.

$$(r_n r_{n-1} \dots r_1 r_0 . r_{-1} r_{-2} \dots r_{-k})_b = \sum_{i=-k}^{i=n} r_i b^i$$

Selon la norme IEEE 754 simple précision, un nombre x pourra être représenté en base b par un triplet $[s, m, e]$ tel que $x = (-1)^s \cdot (1.m) \cdot 2^{e-127}$ où s représente le signe de x , m sa mantisse normalisée et $e = p + 127$ son exposant p décalé de 127 :

- signe s : $s = 1$ si $x < 0$ et $s = 0$ si $x \geq 0$
- mantisse $m' = 1.m$: $m' \in [1, b[$ si $x \neq 0$ et $m' = 0$ si $x = 0$
- exposant p : $p \in [\min, \max]$



Il s'agira donc de déterminer successivement le signe s de x , les parties entière et fractionnaire de $|x|$ pour déterminer la mantisse m et l'exposant p associés.

Exemple : $x = -393.0625 = -(1.100010010001)_2 \cdot 2^8$, d'où $s = 1$, $p = 8$, $e = p + 127 = 135 = (10000111)_2$ et $m' = 1.m = (1.100010010001)_2$, soit selon la norme IEEE 754 simple précision (32 bits) : $x = |s|e|m| = |1|10000111|100010010001000000000000|$.

Vérification : On vérifiera selon deux méthodes :

1. en recalculant x par la formule $x = (-1)^s \cdot (1.m) \cdot 2^{e-127}$;
2. en consultant le site <http://babbage.cs.qc.edu/IEEE-754/Decimal.html> .

5.4.2 Exemple

Enoncé : Soit à coder le réel $x = -41.3125$ selon la norme IEEE 754 simple précision.

Méthode : Déterminer par étapes successives le signe s de x , les codes binaires de la partie entière et de la partie fractionnaire de $|x|$, la mantisse m normalisée ($m' = 1.m \in [1, 2[$) et l'exposant e relatif à 127.

Résultat : Selon les recommandations précédentes, le codage de $x = -41.3125$ en base $b = 2$ s'effectuera en 5 étapes :

1. coder le signe de x : $x = -41.3125 < 0 \Rightarrow s = 1$
2. coder la partie entière de $|x|$: $41 = (101001)_2$
3. coder la partie fractionnaire de $|x|$: $0.3125 = (0.0101)_2$

4. déterminer la mantisse m normalisée ($1.m \in [1, 2[$) :

$$|x| = (101001.0101)_2 = (1.010010101)_2 \cdot 2^5 \Rightarrow 1.m = (1.010010101)_2 \Rightarrow m = (010010101)_2$$

5. déterminer l'exposant e relatif à 127 : $e = 127 + 5 = 132 = (10000100)_2$

Ainsi, selon la norme IEEE 754 simple précision, le réel $x = -41.3125$ se code sur 32 bits de la manière suivante :

$$x = -41.3125 = |1|10000100|010010101000000000000000|$$

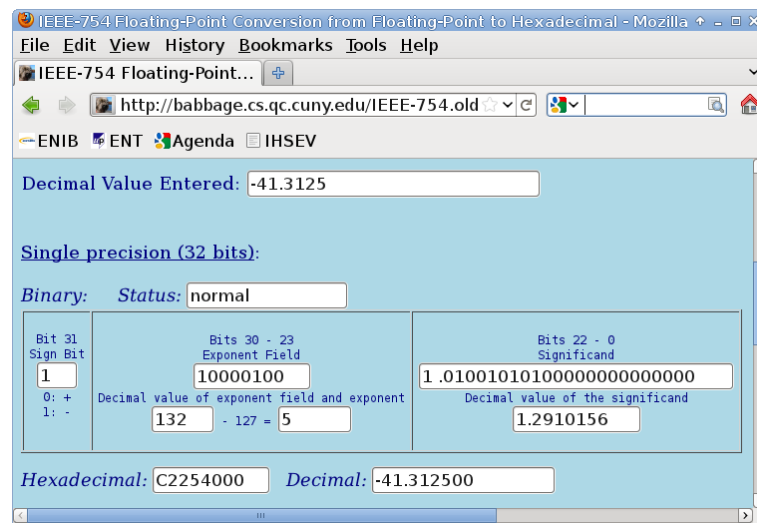
Vérifications : on vérifie selon deux méthodes :

1. en recalculant x par la formule $x = (-1)^s \cdot 1.m \cdot 2^{e-127}$:

```
>>> m = 2**(-2) + 2**(-5) + 2**(-7) + 2**(-9)
>>> e = 132
>>> s = (-1)**1
>>> x = s * (1+m) * 2**(e-127)
>>> x
-41.3125
```

Le décodage des 32 bits redonne bien $x = -41.3125$.

2. en consultant le site : <http://babbage.cs.qc.cuny.edu/IEEE-754/Decimal.html> .



Le site consulté donne bien le même résultat :

$$x = -41.3125 = |s|e|m| = |1|10000100|010010101000000000000000|.$$

5.4.3 Questions

Coder les nombres réels suivants selon la norme IEEE 754 simple précision.

- | | |
|--------------------|---------------------|
| 1. $x = 43.1875$ | 13. $x = -37.03125$ |
| 2. $x = -13.0625$ | 14. $x = 49.1875$ |
| 3. $x = 71.25$ | 15. $x = -53.0625$ |
| 4. $x = -54.375$ | 16. $x = 65.25$ |
| 5. $x = 27.75$ | 17. $x = -77.375$ |
| 6. $x = -33.625$ | 18. $x = 89.75$ |
| 7. $x = 69.5$ | 19. $x = -99.625$ |
| 8. $x = -83.125$ | 20. $x = 7.5$ |
| 9. $x = 99.3125$ | 21. $x = -19.125$ |
| 10. $x = -87.5625$ | 22. $x = 29.3125$ |
| 11. $x = 75.875$ | 23. $x = -37.5625$ |
| 12. $x = -61.25$ | 24. $x = 45.875$ |

5.5 Révisions

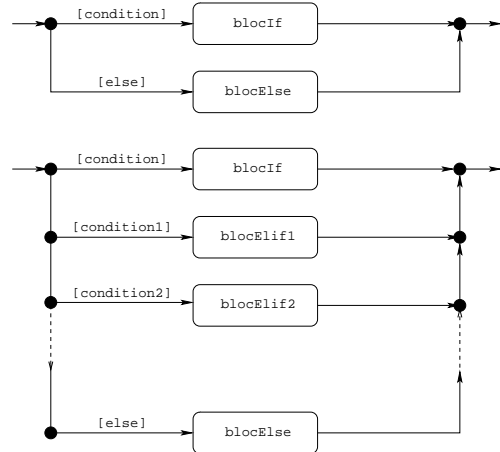
Cours	[1] : chapitre 3, section 3.1
TD	[2] : exercices 1.28, 3.2 à 3.4, 3.20 à 3.22

6 Qu'est-ce qu'un test ?

Objectif : comprendre la structure générale des tests (test simple, alternative simple, alternative multiple).

Syntaxe Python :

```
- if condition : blocIf
- if condition : blocIf
  else : blocElse
- if condition : blocIf
  elif condition1 : blocElif1
  elif condition2 : blocElif2
  ...
  else : blocElse
```



6.1 Exemple

Enoncé : on veut déterminer la mention au baccalauréat compte tenu de la moyenne générale obtenue à l'examen.

Méthode : on utilisera une alternative multiple.

Questions :

Q 6.1 (« mentions au bac » : graphe) Représenter par un graphe la relation $m = f(n)$ entre la mention m ($m \in \{\text{insuffisant, passable, assez bien, bien, très bien}\}$) et la note n obtenue ($n \in [0; 20]$).

Le test simple est une instruction de contrôle du flux d'instructions qui permet d'exécuter une instruction sous condition préalable. L'alternative simple est une instruction de contrôle du flux d'instructions qui permet de choisir entre deux instructions selon qu'une condition est vérifiée ou non. L'alternative multiple est une instruction de contrôle du flux d'instructions qui permet de choisir entre plusieurs instructions en cascade des alternatives simples.

Q 6.2 (« mentions au bac » : alternative multiple) Ecrire une alternative multiple qui calcule la mention en fonction de la note.

6.2 Généralisation

Q 6.3 (alternatives : tests ou alternative ?) Montrer à l'aide d'un contre-exemple que la séquence de tests simples (1) n'est pas équivalente à l'alternative simple (2).

1.

```
if condition : blocIf
if not condition : blocElse
```

2.

```
if condition : blocIf
else : blocElse
```

Q 6.4 (alternatives : multiples ou imbriquées ?) Montrer à l'aide d'un contre-exemple que l'alternative multiple (1) n'est pas équivalente à l'alternative multiple (2). Donner leur équivalent à l'aide d'alternatives simples imbriquées.

```
1. if condition : blocIf
    elif condition1 : blocElif1
    elif condition2 : blocElif2
    elif condition3 : blocElif3
    else : blocElse
```

```
2. if condition : blocIf
    elif condition1 : blocElif1
    elif condition3 : blocElif3
    elif condition2 : blocElif2
    else : blocElse
```

6.3 Applications

Q 6.5 (catégories sportives) *Ecrire un algorithme qui détermine la catégorie sportive d'un enfant selon son âge :*

- Poussin de 6 à 7 ans,
- Pupille de 8 à 9 ans,
- Minime de 10 à 11 ans,
- Cadet de 12 ans à 14 ans.

Q 6.6 (prix d'une photocopie) *Ecrire un algorithme qui affiche le prix de n photocopies sachant que le reprographe facture 0,10 € les dix premières photocopies, 0,09 € les vingt suivantes et 0,08 € au-delà.*

Q 6.7 (exécution d'une séquence de tests)

1. Quelle est la valeur de la variable ok après la suite d'instructions suivante ?

```
x = 2
y = 3
d = 5
h = 4
if x > 0 and x < d :
    if y > 0 and y < h : ok = 1
    else : ok = 0
else : ok = 0
```

2. Quelle est la valeur de la variable y après la suite d'instructions suivante ?

```
x = 3
y = -2
if x < y : y = y - x
elif x == y : y = 0
else : y = x - y
```

6.4 Entraînement

6.4.1 Enoncé

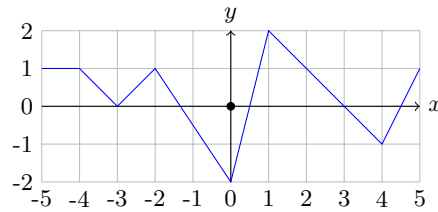
Objectif : calculer une fonction $y = f(x)$ définie par son graphe.

Méthode : utiliser l'alternative multiple.

Vérification : vérifier le bon fonctionnement sous PYTHON en testant des points caractéristiques de la fonction.

6.4.2 Exemple

On considère la fonction $y = f(x)$ définie sur $[-5; 5]$ par le graphe ci-dessous et $\forall x < -5, f(x) = f(-5)$ et $\forall x > 5, f(x) = f(5)$.

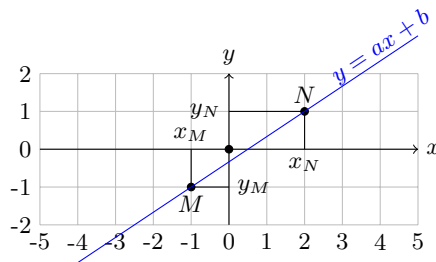


Méthode : Lorsqu'on connaît 2 points $M(x_M, y_M)$ et $N(x_N, y_N)$ d'une droite d'équation $y = ax + b$, les coefficients a (pente de la droite) et b (ordonnée à l'origine) de la droite sont obtenus par résolution du système de 2 équations : $y_M = ax_M + b$ et $y_N = ax_N + b$. On obtient alors a et b :

$$a = \frac{y_N - y_M}{x_N - x_M} \text{ et } b = \frac{y_M x_N - y_N x_M}{x_N - x_M}$$

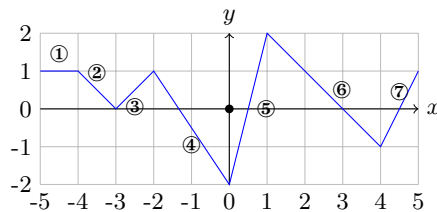
Pour la droite ci-contre :

$$a = \frac{1 - (-1)}{2 - (-1)} = \frac{2}{3} \text{ et } b = \frac{(-1) \cdot 2 - 1 \cdot (-1)}{2 - (-1)} = -\frac{1}{3}$$



On vérifie graphiquement ces résultats : pour passer de M à N , on se déplace de 3 horizontalement puis de 2 verticalement (d'où la pente $a = 2/3$), et la droite coupe bien l'axe des ordonnées en $y = -1/3$.

Il faut donc déterminer les équations de droite correspondant aux différents segments du graphe de la fonction, à savoir :



- ① $y = 1$
- ② $y = -x - 3$
- ③ $y = x + 3$
- ④ $y = -3x/2 - 2$
- ⑤ $y = 4x - 2$
- ⑥ $y = -x + 3$
- ⑦ $y = 2x - 9$

Résultat : Compte-tenu de ces équations, le code ci-contre permet de calculer $y = f(x)$, y compris pour $x < -5$ et $x > 5$. Pour le tester, on comparera les valeurs obtenues par le calcul avec celles lues directement sur le graphe pour quelques points caractéristiques.

```

1  if    x < -4 : y = 1
2  elif  x < -3 : y = -x - 3
3  elif  x < -2 : y = x + 3
4  elif  x <  0 : y = -3*x/2 - 2
5  elif  x <  1 : y = 4*x - 2
6  elif  x <  4 : y = -x + 3
7  elif  x <  5 : y = 2*x - 9
8  else                : y = 1

```

Vérifications : On peut vérifier par exemple pour $x = -1$ ($\rightarrow y = -0.5$) et $x = 3$ ($\rightarrow y = 0$).

```
>>> x = -1
>>> if x < -4 : y = 1
    elif x < -3 : y = -x - 3
    elif x < -2 : y = x + 3
    elif x < 0 : y = -3*x/2 - 2
    elif x < 1 : y = 4*x - 2
    elif x < 4 : y = -x + 3
    elif x < 5 : y = 2*x - 9
    else : y = 1

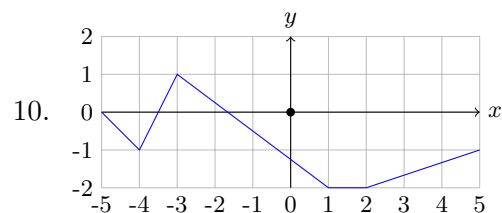
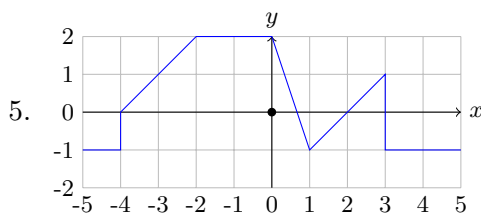
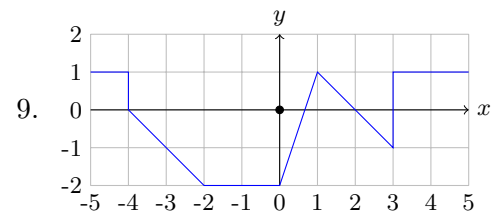
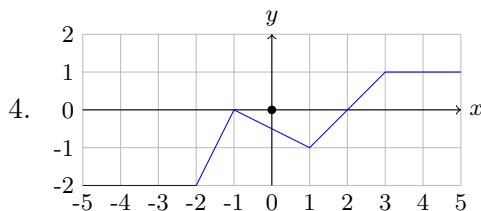
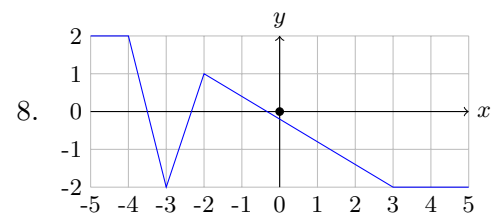
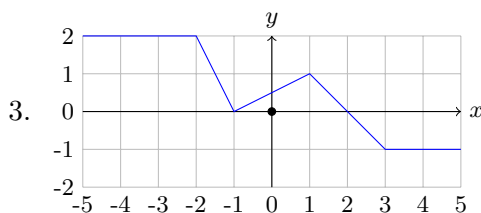
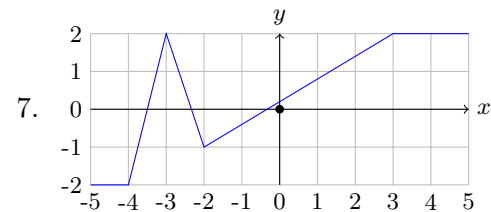
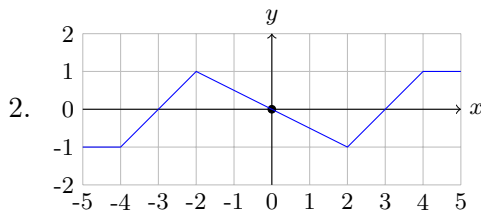
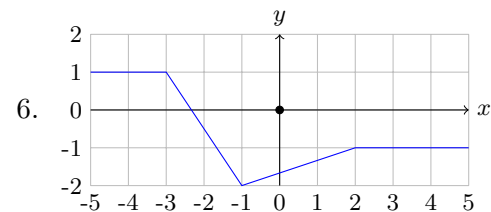
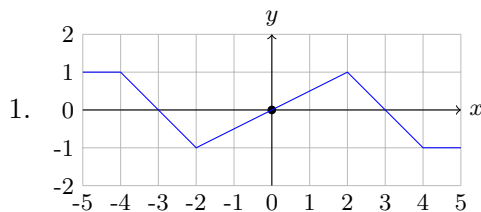
>>> y
-0.5
```

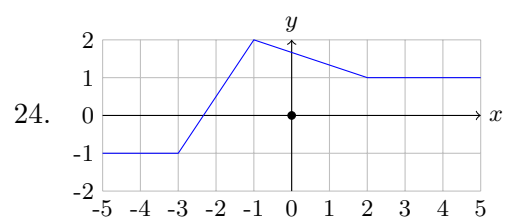
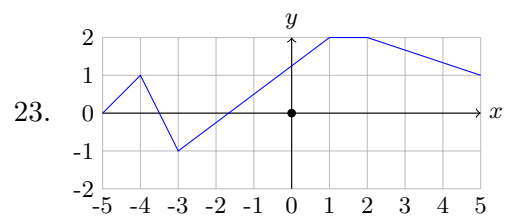
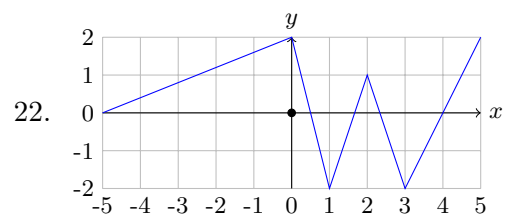
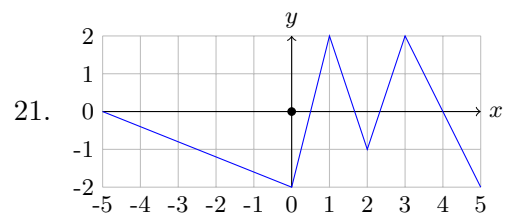
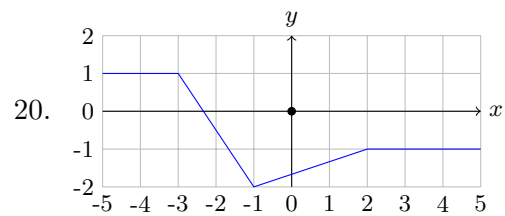
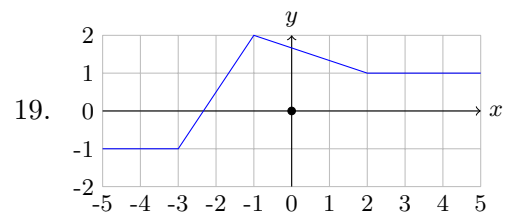
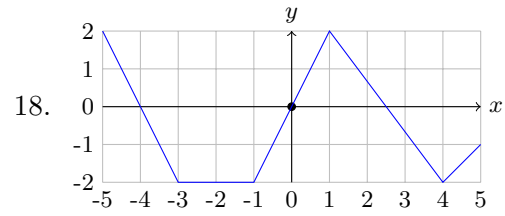
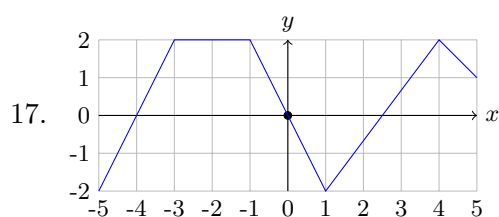
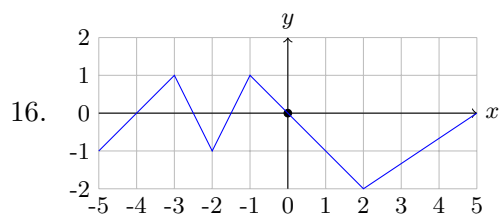
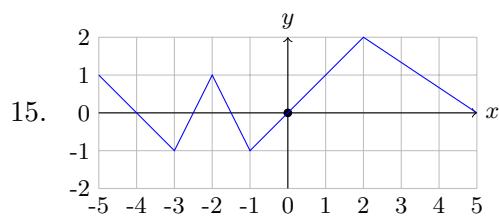
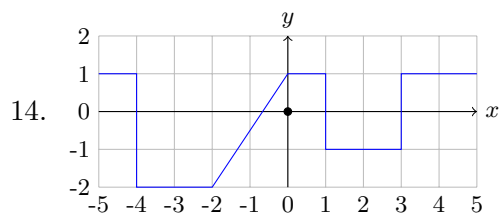
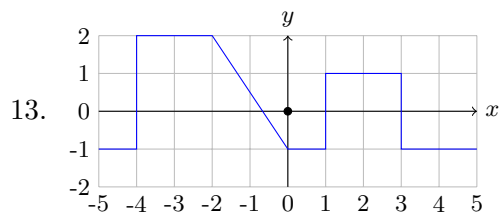
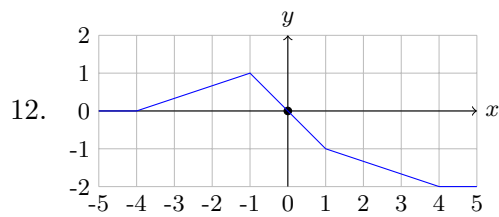
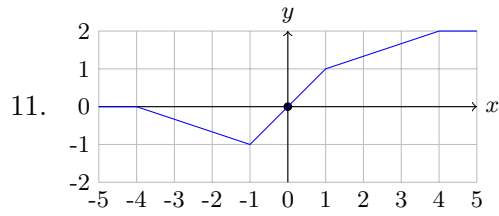
```
>>> x = 3
>>> if x < -4 : y = 1
    elif x < -3 : y = -x - 3
    elif x < -2 : y = x + 3
    elif x < 0 : y = -3*x/2 - 2
    elif x < 1 : y = 4*x - 2
    elif x < 4 : y = -x + 3
    elif x < 5 : y = 2*x - 9
    else : y = 1

>>> y
0
```

6.4.3 Questions

Ecrire une alternative multiple qui permette de déterminer $y = f(x)$ pour une fonction f définie par son graphe sur $[-5; 5]$ et $\forall x < -5, f(x) = f(-5)$ et $\forall x > 5, f(x) = f(5)$.





6.5 Révisions

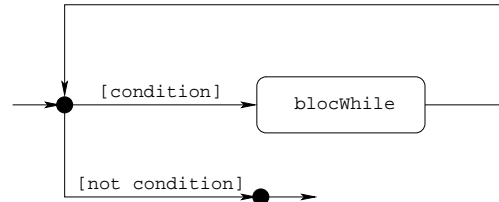
Cours	[1] : chapitre 2, section 2.3
TD	[2] : exercices 2.8 à 2.11, 2.30 à 2.32

7 Comment construit-on une boucle ?

Objectif : comprendre la structure générale des boucles pour mieux appréhender leur conception.

Syntaxe Python :

```
while condition : blocWhile
```



7.1 Exemple

Enoncé On dispose d'une planche, d'un marteau et d'un clou (situation initiale) et on veut que le clou soit enfoncé dans la planche jusqu'à la tête (situation finale).

Méthode Le travail à réaliser consiste donc à planter légèrement le clou à la main de façon qu'il tienne seul, puis à taper sur la tête du clou avec le marteau tant que la tête ne touche pas la planche. Le nombre de coups nécessaire est *a priori* inconnu.

Questions Le passage de la compréhension intuitive d'un tel énoncé à l'algorithme n'est pas toujours facile à réaliser. Pour nous y aider, nous introduisons 4 notions : l'initialisation, l'invariant, la progression et la condition d'arrêt.

Q 7.1 (« planter un clou » : initialisation) Par quelle opération commence-t-on pour « enfoncer le clou » ? Il s'agit de l'initialisation du processus « planter un clou ».

Q 7.2 (« planter un clou » : progression) Une fois l'initialisation réalisée, quelle opération doit-on effectuer pour atteindre la situation finale ? Cette opération doit faire progresser le système vers sa situation finale.

Q 7.3 (« planter un clou » : invariant) Au cours de la progression du processus d'enfoncement du clou, une situation est toujours vérifiée. Laquelle ? On parle d'invariant du processus.

Q 7.4 (« planter un clou » : condition d'arrêt) A quelle condition a-t-on atteint la situation finale ? Il s'agit de la condition d'arrêt du processus.

L'invariant et la condition d'arrêt définissent des situations tandis que l'initialisation et la progression concernent des actions. On notera les situations entre crochets ([situation]) pour les distinguer des actions (« action »).

Q 7.5 (« planter un clou » : algorithme) Disposant maintenant des 4 notions précédentes (l'initialisation, l'invariant, la progression et la condition d'arrêt), proposer un algorithme pour « planter un clou ».

7.2 Généralisation

Un algorithme est un mécanisme qui fait passer un « système » d'une « situation » dite initiale (ou [précondition]) à une « situation » finale ([postcondition] ou [but]). Le couple (situation initiale, situation finale) constitue une spécification de l'algorithme. L'algorithmique vise alors à construire rationnellement des algorithmes à partir de leur spécification.

Dans le cas d'une boucle, la construction de l'algorithme passe par les 4 étapes suivantes :

1. **Invariant** : proposer une situation générale décrivant le problème posé. Cette étape est la plus délicate car elle exige de faire preuve d'imagination.

Notation : `[invariant]`

2. **Condition d'arrêt** : à partir de la situation générale imaginée en [1], on doit formuler la condition qui permet d'affirmer que l'algorithme a terminé son travail. La situation dans laquelle il se trouve alors est appelée situation finale. La condition d'arrêt fait sortir de la boucle.

Notation : `[condition d'arrêt]`

3. **Progression** : se « rapprocher » de la situation finale, tout en faisant le nécessaire pour conserver à chaque étape une situation générale analogue à celle retenue en [1]. La progression conserve l'invariant.

Notation : « `progression` »

4. **Initialisation** : initialiser les variables introduites dans l'invariant pour que celui-ci soit vérifié avant d'entrer dans la boucle. L'initialisation « instaure » l'invariant.

Notation : « `initialisation` »

Q 7.6 (construction d'une boucle : initialisation et invariant) Dans l'exemple « planter un clou », vérifier que l'initialisation instaure l'invariant.

Q 7.7 (construction d'une boucle : condition d'arrêt et invariant) Dans l'exemple du clou, montrer à l'aide d'un contre-exemple qu'il ne suffit pas de vérifier la condition d'arrêt pour que la situation finale recherchée soit atteinte.

Q 7.8 (construction d'une boucle : cas général) Proposer la structure générale d'une boucle en combinant les 4 étapes précédentes. Dessiner le diagramme UML correspondant. Vérifier que l'algorithme « planter un clou » a bien cette structure générale.

Q 7.9 (construction d'une boucle : cas simplifié) Dans la pratique, on utilise une simplification du cas général précédent. Laquelle ? Dessiner le diagramme UML correspondant. Simplifier en conséquence l'algorithme « planter un clou ».

Cette façon de procéder permet de « prouver » la validité de l'algorithme au fur et à mesure de son élaboration. En effet la situation générale choisie en [1] est en fait l'invariant qui caractérise la boucle `while`. Cette situation est satisfaite au départ grâce à l'initialisation de l'étape [4] ; elle reste vraie à chaque itération (étape [3]). Ainsi lorsque la condition d'arrêt (étape [2]) est atteinte, cette situation nous permet d'affirmer que le problème est résolu. C'est également en analysant l'étape [3] qu'on peut prouver la terminaison de l'algorithme.

7.3 Applications

Q 7.10 (construction d'une boucle : fonction puissance) Ecrire l'algorithme de calcul de la fonction puissance (x^p) en définissant l'initialisation, l'invariant, la progression et la condition d'arrêt correspondants.

Q 7.11 (construction d'une boucle : fonction factorielle) Ecrire l'algorithme de calcul de la fonction factorielle ($n!$) en définissant l'initialisation, l'invariant, la progression et la condition d'arrêt correspondants.

Q 7.12 (construction d'une boucle : fonction pgcd) *Ecrire un algorithme qui calcule le plus grand commun diviseur (pgcd) de deux nombres en définissant l'initialisation, l'invariant, la progression et la condition d'arrêt correspondants.*

Q 7.13 (construction d'une boucle : fonction Fibonacci) *L'algorithme suivant permet de calculer le nombre f de Fibonacci à l'ordre n .*

Retrouver l'initialisation, l'invariant, la progression et la condition d'arrêt qui ont permis de construire cet algorithme.

```
i, f, f1, f2 = 3, 2, 1, 1
while i < n + 1 :
    f2 = f1
    f1 = f
    f = f1 + f2
    i = i + 1
```

7.4 Entraînement

7.4.1 Enoncé

Objectif : écrire un algorithme qui calcule $y = f(x)$ en fonction du développement en série entière de la fonction $f : f(x) = \sum u_k$, en respectant les contraintes suivantes :

- les calculs seront arrêtés lorsque la valeur absolue du terme u_k ($|u_k|$) sera inférieure à un certain seuil s (avec $0 < s < 1$) ;
- on n'utilisera ni la fonction *puissance* (x^n) ni la fonction *factorielle* ($n!$) pour effectuer le calcul du développement.

Méthode : construire l'algorithme en définissant l'initialisation, l'invariant, la progression et la condition d'arrêt correspondants.

Vérification : vérifier avec PYTHON les résultats obtenus pour différentes valeurs de x en les comparant aux calculs directs de la fonction $f(x)$.

7.4.2 Exemple

On veut calculer $\forall x \in]-1; 1[$ la fonction $f(x)$ définie par

$$\begin{aligned} y = f(x) = \frac{1}{(1+x)^a} &\approx 1 + \sum_{k=1}^n u_k = 1 + \sum_{k=1}^n (-1)^k a(a+1) \cdots (a+k-1) \frac{x^k}{k!} \\ &= 1 - ax + a(a+1) \frac{x^2}{2} + \dots + (-1)^k a(a+1) \cdots (a+k-1) \frac{x^k}{k!} \end{aligned}$$

On cherche une relation de récurrence g entre u_{k+1} et u_k afin de faciliter le calcul de u_{k+1} sans faire appel aux fonctions *puissance* (x^k) et *factorielle* ($k!$).

On a :

$$u_{k+1} = (-1)^{k+1} a(a+1) \cdots (a+(k+1)-1) \frac{x^{k+1}}{(k+1)!}$$

et on cherche donc à faire apparaître u_k dans l'expression de u_{k+1} :

$$u_{k+1} = (-1) \cdot (-1)^k \cdot a(a+1) \cdots (a+k-1) \cdot (a+k) \cdot \frac{x \cdot x^k}{k!(k+1)}$$

$$u_{k+1} = -x \cdot \frac{(a+k)}{(k+1)} \cdot (-1)^k a(a+1) \cdots (a+k-1) \frac{x^k}{k!} = -x \cdot \frac{(a+k)}{(k+1)} \cdot u_k = g(u_k)$$

Ainsi, dans la pratique, si on mémorise la valeur de u_k , on calculera sans problème la valeur de u_{k+1} grâce à cette relation de récurrence ($u_{k+1} = g(u_k)$).

$$\begin{aligned}
 y_1 &= 1 + u_1 \\
 y_2 &= 1 + u_1 + u_2 &= y_1 + u_2 &= y_1 + g(u_1) \\
 y_3 &= 1 + u_1 + u_2 + u_3 &= y_2 + u_3 &= y_2 + g(u_2) \\
 \dots & \\
 y_{m+1} &= 1 + u_1 + u_2 + \dots + u_m + u_{m+1} &= y_m + u_{m+1} &= y_m + g(u_m)
 \end{aligned}$$

Méthode Les 4 étapes de construction de l'algorithme recherché sont donc les suivantes :

- pour l'invariant, on vérifie qu'à chaque étape m , y est toujours égal à la somme des m premiers termes u_k : $\forall m, y_m = 1 + \sum_{k=1}^m u_k$;
- l'initialisation consiste à décrire le cas $m = 1$: $u_1 = -ax$ et $y = y_1 = 1 + u_1 = 1 - ax$;
- la progression consiste à calculer l'étape suivante à partir de l'étape courante ($m \rightarrow m+1$) :
 $u_{m+1} = g(u_m) = -x \cdot \frac{(a+m)}{(m+1)} \cdot u_m, y_{m+1} = y_m + g(u_m)$;
- la condition d'arrêt est contrainte par l'énoncé : $|u_m| < s$.

L'algorithme recherché prendra la forme :

« initialisation »

while not « condition d'arrêt » :

« progression »

$m = 1, u_1 = -ax, y_1 = 1 - ax$

while not $|u_m| < s$:

$u_{m+1} = g(u_m)$

$y_{m+1} = y_m + u_{m+1}$

$m \rightarrow m + 1$

Résultat En PYTHON, l'algorithme correspondant est présenté ci-contre. Pour le tester, on fixera différentes valeurs des données a , x et s , et pour le vérifier, on comparera la valeur y calculée avec le calcul direct $(1+x)^{-a}$.

```

1 m = 1
2 u = -a*x
3 y = 1 + u
4 while not fabs(u) < s :
5     u = -u*x*(a + m)/(m + 1)
6     y = y + u
7     m = m + 1

```

Vérifications on compare le résultat obtenu par l'algorithme avec le calcul direct : $y - \frac{1}{(1+x)^a}$, pour différentes valeurs des données a , x et s .

```
>>> a, x, s = 1, 0, 1.0e-9
```

```
...
>>> y - 1/(1-x)**a
0.0
```

```
>>> a, x, s = 1/2, 0.5, 1.0e-9
```

```
...
>>> y - 1/(1-x)**a
-2.65160005064e-10
```

```
>>> a, x, s = 3, 0.5, 1.0e-9
```

```
...
>>> y - 1/(1-x)**a
2.69446243095e-10
```

```
>>> a, x, s = 5, 0, 1.0e-9
```

```
...
>>> y - 1/(1-x)**a
0.0
```

```
>>> a, x, s = 5, 0.85, 1.0e-9
```

```
...
>>> y - 1/(1-x)**a
4.27782663459e-10
```

```
>>> a, x, s = 5, -0.85, 1.0e-9
```

```
...
>>> y - 1/(1-x)**a
-5.8480509324e-09
```



```
>>> a, x, s = 3, 0.5, 1.0e-4
```

```
...
```

```
>>> y = 1/(1-x)**a
```

```
2.31884143971e-05
```

```
>>> a, x, s = 5, -0.85, 1.0e-5
```

```
...
```

```
>>> y = 1/(1-x)**a
```

```
-6.00809507887e-05
```

Les différences observées sont toutes voisines de 0 et compatibles avec le seuil de précision s donné.

7.4.3 Questions

$$1. \exp(x) \approx \sum_{k=0}^n \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \dots + \frac{x^n}{n!} \quad \forall x \in \mathbb{R}$$

$$2. \exp(-x) \approx \sum_{k=0}^n (-1)^k \frac{x^k}{k!} = 1 - x + \frac{x^2}{2} + \dots + (-1)^n \frac{x^n}{n!} \quad \forall x \in \mathbb{R}$$

$$3. \log(1+x) \approx \sum_{k=1}^n (-1)^{k+1} \frac{x^k}{k} = x - \frac{x^2}{2} + \frac{x^3}{3} + \dots + (-1)^{n+1} \frac{x^n}{n} \quad \forall x \in]-1; 1]$$

$$4. \log(1-x) \approx \sum_{k=1}^n -\frac{x^k}{k} = -x - \frac{x^2}{2} - \frac{x^3}{3} - \dots - \frac{x^n}{n} \quad \forall x \in [-1; 1[$$

$$5. \log\left(\frac{1+x}{1-x}\right) = 2 \sum_{k=0}^n \frac{x^{2k+1}}{2k+1} = \quad \forall x \in [-1; 1[$$

$$6. \sinh(x) \approx \sum_{k=0}^n \frac{x^{2k+1}}{(2k+1)!} = x + \frac{x^3}{6} + \frac{x^5}{120} + \dots + \frac{x^{2n+1}}{(2n+1)!} \quad \forall x \in \mathbb{R}$$

$$7. \cosh(x) \approx \sum_{k=0}^n \frac{x^{2k}}{(2k)!} = 1 + \frac{x^2}{2} + \frac{x^4}{24} + \dots + \frac{x^{2n}}{(2n)!} \quad \forall x \in \mathbb{R}$$

$$8. \arg \sinh(x) \approx x + \sum_{k=1}^n (-1)^k \frac{1 \times 3 \times \dots \times (2k-1)}{2 \times 4 \times \dots \times 2k} \frac{x^{2k+1}}{2k+1} =$$

$$x - \frac{1}{2} \frac{x^3}{3} + \frac{1}{24} \frac{3}{4} \frac{x^5}{5} - \frac{1}{240} \frac{3 \cdot 5}{4 \cdot 6} \frac{x^7}{7} + \dots + (-1)^n \frac{1 \times 3 \times \dots \times (2n-1)}{2 \times 4 \times \dots \times 2n} \frac{x^{2n+1}}{2n+1} \quad \forall x \in]-1; 1[$$

$$9. \arg \tanh(x) \approx \sum_{k=0}^n \frac{1}{2k+1} x^{2k+1} = x + \frac{x^3}{3} + \frac{x^5}{5} + \dots + \frac{1}{2n+1} x^{2n+1} \quad \forall x \in]-1; 1[$$

$$10. \sin(x) \approx \sum_{k=0}^n (-1)^k \frac{x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{6} + \frac{x^5}{120} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} \quad \forall x \in \mathbb{R}$$

$$11. \cos(x) \approx \sum_{k=0}^n (-1)^k \frac{x^{2k}}{(2k)!} = 1 - \frac{x^2}{2} + \frac{x^4}{24} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} \quad \forall x \in \mathbb{R}$$

$$12. \arcsin(x) \approx x + \sum_{k=1}^n \frac{1 \times 3 \times \dots \times (2k-1)}{2 \times 4 \times \dots \times 2k} \frac{x^{2k+1}}{2k+1} =$$

$$x + \frac{1}{2} \frac{x^3}{3} + \frac{1}{24} \frac{3}{4} \frac{x^5}{5} + \frac{1}{240} \frac{3 \cdot 5}{4 \cdot 6} \frac{x^7}{7} + \dots + \frac{1 \times 3 \times \dots \times (2n-1)}{2 \times 4 \times \dots \times 2n} \frac{x^{2n+1}}{2n+1} \quad \forall x \in]-1; 1[$$

$$13. \arccos(x) \approx \frac{\pi}{2} - x - \sum_{k=1}^n \frac{1 \cdot 3 \cdot 5 \dots (2k-1)}{2 \cdot 4 \cdot 6 \dots (2k) \cdot (2k+1)} x^{2k+1} =$$

$$\frac{\pi}{2} - x - \frac{x^3}{2 \cdot 3} - \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} - \dots - \frac{1 \cdot 3 \cdot 5 \dots (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \dots (2n) \cdot (2n+1)} \quad \forall x \in]-1; 1[$$

$$14. \arctan(x) \approx \sum_{k=0}^n (-1)^k \frac{x^{2k+1}}{(2k+1)} = x - \frac{x^3}{3} + \frac{x^5}{5} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)} \quad \forall x \in \mathbb{R}$$

$$15. \frac{1}{1+x} \approx \sum_{k=0}^n (-1)^k x^k = 1 - x + x^2 - x^3 + \dots + (-1)^n x^n \quad \forall x \in]-1; 1[$$

$$16. \frac{1}{1-x} \approx \sum_{k=0}^n x^k = 1 + x + x^2 + x^3 + \dots + x^n \quad \forall x \in]-1; 1[$$

$$17. \frac{1}{1+x^2} \approx \sum_{k=0}^n (-1)^k x^{2k} = 1 - x^2 + x^4 + \dots + (-1)^n x^{2n} \quad \forall x \in]-1; 1[$$

$$18. \frac{1}{1-x^2} \approx \sum_{k=0}^n x^{2k} = 1 + x^2 + x^4 + \dots + x^{2n} \quad \forall x \in]-1; 1[$$

$$19. \sqrt{1+x} \approx 1 + \frac{x}{2} + \sum_{k=2}^n (-1)^{k-1} \frac{1 \times 3 \times \dots \times (2k-3)}{2^k} \frac{x^k}{k!} =$$

$$1 + \frac{x}{2} - \frac{x^2}{8} + \frac{x^3}{16} - \frac{5x^4}{128} + \dots + (-1)^{n-1} \frac{1 \times 3 \times \dots \times (2n-3)}{2^n} \frac{x^n}{n!} \quad \forall x \in]-1; 1[$$

$$20. \frac{1}{\sqrt{1+x}} \approx 1 + \sum_{k=1}^n (-1)^k \frac{1 \times 3 \times \dots \times (2k-1)}{2 \times 4 \times \dots \times 2k} x^k =$$

$$1 - \frac{1}{2}x + \frac{1}{2} \frac{3}{4} x^2 - \frac{1}{2} \frac{3}{4} \frac{5}{6} x^3 + \dots + (-1)^n \frac{1 \times 3 \times \dots \times (2n-1)}{2 \times 4 \times \dots \times 2n} x^n \quad \forall x \in]-1; 1[$$

$$21. \frac{1}{\sqrt{1-x^2}} \approx \sum_{k=0}^n \frac{(2k)!}{2^{2k} (k!)^2} x^{2k} = 1 + \frac{x^2}{2} + \frac{3x^4}{8} + \dots + \frac{(2n)!}{2^{2n} (n!)^2} x^{2n} \quad \forall x \in]-1; 1[$$

$$22. \frac{1}{(a-x)^2} \approx \sum_{k=0}^n \frac{k+1}{a^{k+2}} x^k = \frac{1}{a^2} \left(1 + \frac{2x}{a} + \frac{3x^2}{a^2} + \dots + \frac{(n+1)x^n}{a^n} \right) \quad \forall x \in]-|a|; |a|[$$

$$23. \frac{1}{(a-x)^3} \approx \sum_{k=0}^n \frac{(k+1)(k+2)}{2a^{k+3}} x^k =$$

$$\frac{1}{a^3} \left(1 + \frac{3x}{a} + \frac{6x^2}{a^2} + \dots + \frac{(n+1)(n+2)}{2a^n} x^n \right) \quad \forall x \in]-|a|; |a|[$$

$$24. \frac{1}{(a-x)^5} \approx \sum_{k=0}^n \frac{(k+1)(k+2)(k+3)(k+4)}{24a^{k+5}} x^k =$$

$$\frac{1}{a^5} \left(1 + \frac{5x}{a} + \frac{15x^2}{a^2} + \dots + \frac{(n+1)(n+2)(n+3)(n+4)}{24a^n} x^n \right) \quad \forall x \in]-|a|; |a|[$$

7.5 Révisions

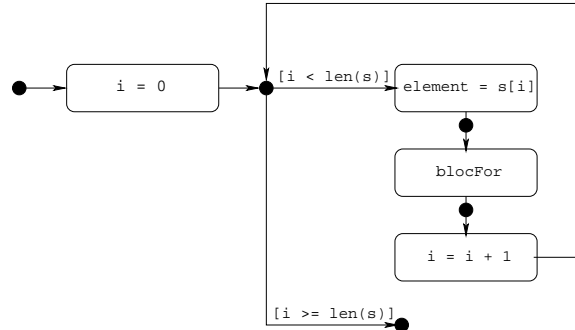
Cours	[1] : chapitre 2, sections 2.4.1 et 2.6.3
TD	[2] : exercices 2.12 à 2.16, 2.33, 2.40, 2.44 et 2.45

8 Comment imbriquer des boucles ?

Objectif : imbriquer des boucles.

Syntaxe Python :

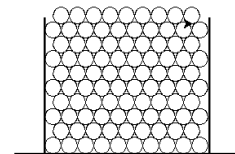
- for element in sequence :
 blocFor
- range([start,] end [, step])
 crée une liste d'entiers compris entre **start** inclus (= 0 par défaut) et **end** exclus par pas de **step** (= 1 par défaut).



8.1 Exemple

Enoncé On dispose d'un tas de bois formé de rondins d'environ 1m de long et de 10cm de diamètre, et on veut les liter en quinconce régulier pour former un stère de bois ($1m \times 1m \times 1m = 1m^3$).

On suppose que des pieux verticaux sont disposés aux quatre coins du stère pour éviter la chute des rondins périphériques.



Méthode On commence par poser sur le sol une première couche horizontale de 10 rondins ; on dispose par dessus en quinconce une deuxième rangée de 9 rondins puis une troisième rangée de 10 rondins, et ainsi de suite jusqu'à superposer 10 rangées horizontales alternativement de 10 et 9 rondins.

Questions On décompose le problème en deux sous-problèmes : empiler des rangées, aligner des rondins pour faire une rangée.

Q 8.1 (« ranger le bois » : les rangées) Proposer un algorithme pour construire un stère de bois en supposant que l'on sait manipuler directement une rangée horizontale de rondins.

Q 8.2 (« ranger le bois » : une rangée) Proposer un algorithme pour construire une rangée horizontale de rondins de bois.

Q 8.3 (« ranger le bois » : le stère) Proposer un algorithme pour construire un stère de bois.

8.2 Généralisation

Une boucle permet de traiter une séquence d'éléments un par un, les uns après les autres (placer une rangée, puis une autre, puis une autre...), et pour chaque élément, on peut vouloir traiter autre chose de manière systématique (aligner les rondins). L'algorithme sera ainsi constitué d'une boucle principale (qui place les rangées) et dans le corps de cette boucle principale, on dit aussi « à l'intérieur de la boucle », une boucle secondaire (qui aligne les rondins pour faire une rangée). La boucle secondaire peut elle-même contenir une boucle, et ainsi de suite.

Q 8.4 (boucles imbriquées : exécutions) *Qu'affichent les itérations suivantes ?*

```

for i in range(10) :
    j = 10 - i
    while j > 0 :
        print('*',end='')
        j = j - 1
    print()

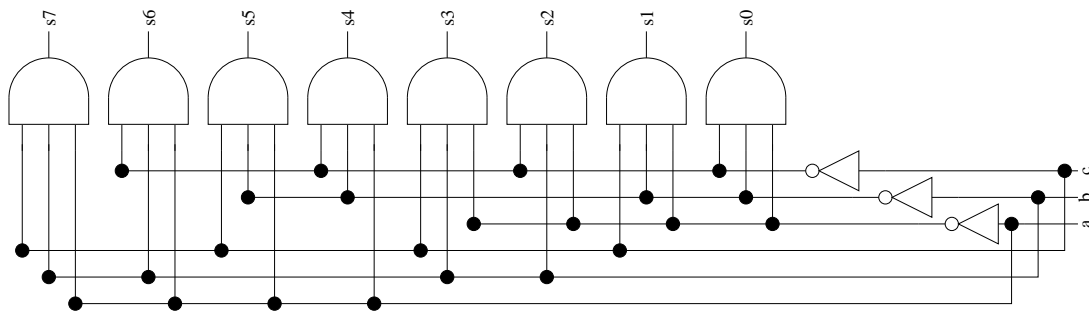
i = 0
while i < 10 :
    for j in range(i) :
        print('*',end='')
    print()
    i = i + 1
  
```

L'approche efficace pour résoudre un problème (*ranger le bois*) consiste souvent à le décomposer en plusieurs sous-problèmes plus simples (*placer une rangée...*) qui seront étudiés séparément. Ces sous-problèmes peuvent éventuellement être eux-mêmes décomposés à leur tour (*aligner les rondins...*), et ainsi de suite. Le concepteur de l'algorithme définit la structuration d'un problème en sous-problèmes : il divise le problème en sous-problèmes pour mieux le contrôler (*diviser pour régner*). Lorsque les problèmes et sous-problèmes correspondent à des boucles, on obtiendra une structuration en boucles imbriquées et/ou en boucles successives.

Q 8.5 (boucles imbriquées : imbriquées ou successives ?) *A partir de deux exemples simples et comparables, illustrer et expliquer la différence entre deux boucles imbriquées et deux boucles successives. Dessiner les diagrammes UML correspondants.*

8.3 Applications

Q 8.6 (boucles imbriquées : tables de vérité) *Ecrire un algorithme qui affiche la table de vérité du circuit logique suivant, où a , b et c sont les entrées et s_0, s_1, \dots, s_7 les sorties.*



Q 8.7 (boucles imbriquées : tables de multiplication) *Ecrire un algorithme qui affiche successivement les 10 premières tables de multiplication de deux manières différentes :*

d'abord sous la forme :	$3 \times 0 = 0$	puis sous la forme :	$0 \times 3 = 0$
	$3 \times 1 = 3$		$1 \times 3 = 3$
	$3 \times 2 = 6$		$2 \times 3 = 6$
	$3 \times 3 = 9$		$3 \times 3 = 9$
	\dots		\dots
	$3 \times 9 = 27$		$9 \times 3 = 27$

Q 8.8 (boucles imbriquées : triangle de Pascal) *Ecrire un algorithme qui affiche le triangle de Pascal jusqu'à l'ordre n . Chaque ligne i de ce triangle est composée des coefficients c_{ij} du*

binôme $(x+y)^i = \sum_{j=0}^i c_{ij} x^{i-j} y^j = \sum_{j=0}^i \frac{i!}{j!(i-j)!} x^{i-j} y^j$ où $\forall i, c_{i0} = c_{ii} = 1$ et $c_{ij} = c_{i-1,j} + c_{i-1,j-1}$ $\forall j, 0 < j < i$.

Les 8 premières lignes du triangle de Pascal sont donc les suivantes :

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

```

8.4 Entraînement

8.4.1 Énoncé

Objectif : en utilisant les instructions de la tortue LOGO (module `turtle`), écrire un algorithme qui dessine un motif géométrique régulier composé de polygones réguliers.

Méthode : construire l'algorithme en décomposant le problème en 3 niveaux :

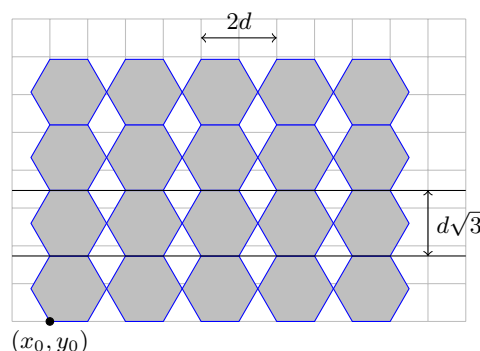
1. le tracé du polygone élémentaire,
2. le tracé d'une ligne de polygones élémentaires,
3. le tracé du motif de lignes de polygones élémentaires.

On pourra commencer soit par l'étape de plus haut niveau (le motif), soit par l'étape de plus bas niveau (le polygone élémentaire).

Vérification : vérifier l'algorithme sous PYTHON en comparant le tracé obtenu avec la figure de l'énoncé.

8.4.2 Exemple

On considère le motif composé de (5×4) hexagones réguliers de côté de longueur d représenté ci-dessous :



Méthode On écrit successivement le code qui trace un motif de m lignes d'hexagones, une ligne de n hexagones et un hexagone, en supposant à chaque étape que le niveau inférieur est réalisé :

1. Tracé d'un motif de m lignes d'hexagones
Pour chaque ligne d'indice j , on positionne le crayon en bas à gauche de la ligne : les lignes étant alignées verticalement, l'abscisse x ne change pas, l'ordonnée y doit par contre être

déplacée vers le haut de $d\sqrt{3}$ à chaque changement d'indice. Une fois positionnée, on trace la ligne d'hexagones.

```
# dessin d'un motif de m lignes d'hexagones
for j in range(m) :
    x, y = x0, y0 + j*d*sqrt(3)
    # dessin d'une ligne de n hexagones
```

2. Tracé d'une ligne de n hexagones

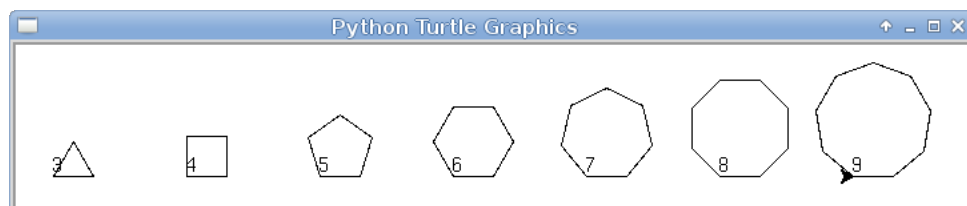
Pour chaque hexagone d'indice i , on positionne le crayon en bas à gauche de l'hexagone : les hexagones d'une même ligne étant alignés horizontalement, l'ordonnée y ne change pas, l'abscisse x doit par contre être déplacée vers la gauche de $2d$ à chaque changement d'indice. Une fois positionné, on trace l'hexagone.

```
# dessin d'une ligne de n hexagones
for i in range(n) :
    x = x0 + 2*i*d
    # dessin d'un hexagone
```

3. Tracé d'un hexagone

Les motifs considérés ici sont composés de polygones réguliers ; en utilisant les instructions de la tortue LOGO, l'algorithme suivant permet de dessiner un polygone régulier de c côtés de longueur d .

	<u>c : polygone</u>
	3 : triangle
	4 : carré
for k in range(c) :	5 : pentagone
forward(d)	6 : hexagone
left(360/c)	7 : heptagone
	8 : octogone
	9 : ennéagone
	...



On se déplace donc, crayon levé, jusqu'à l'origine en bas à gauche d'un hexagone, puis on trace l'hexagone ($c = 6$).

```
# dessin d'un hexagone
up()
goto(x,y)
down()
for k in range(c) :
    forward(d)
    left(360/c)
```

Résultat En PYTHON, l'algorithme correspondant, présenté ci-contre, est donc composé de 3 boucles imbriquées. Il faut inclure le module `math` pour utiliser la fonction `sqrt()` ainsi que le module `turtle` pour les fonctions de manipulation de la tortue LOGO (`up()`, `down()`, `goto()`, `left()` et `forward()`). Pour le tester, on fixera les valeurs de c (nombre de côtés, 6 pour un hexagone), d (longueur du côté de l'hexagone), n (nombre d'hexagones dans une ligne) et m (nombre de lignes) ainsi que l'origine du motif (x_0, y_0) . Pour le vérifier, on comparera le tracé obtenu avec celui de la figure du motif présentée plus haut en début de section.

```

1 from math import *
2 from turtle import *
3 # initialisation du motif
4 c, d = 6, 20
5 n, m = 5, 4
6 x0, y0 = 0, 0
7 # dessin du motif
8 for j in range(m) :
9     x, y = x0, y0 + j*d*sqrt(3)
10    # dessin d'une ligne d'hexagones
11    for i in range(n) :
12        x = x0 + 2*i*d
13        # dessin d'un hexagone
14        up()
15        goto(x,y)
16        down()
17        for k in range(c) :
18            forward(d)
19            left(360/c)

```

Vérifications on compare le tracé obtenu lors de l'exécution de l'algorithme précédent avec la figure donnée en début de section.

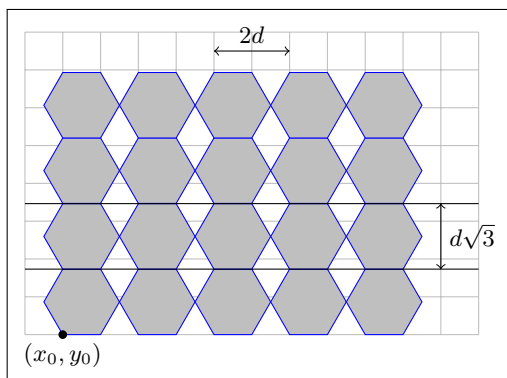
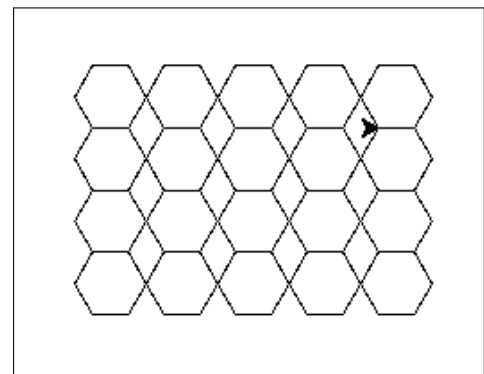


figure de l'énoncé

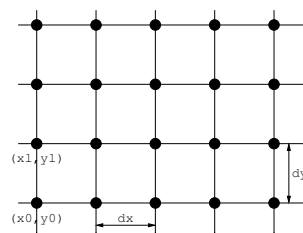


tracé PYTHON

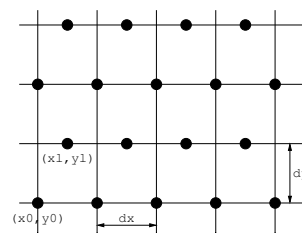
Les figures sont bien similaires.

8.4.3 Questions














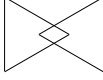

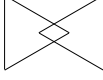
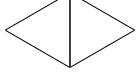

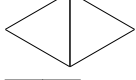





En utilisant les instructions de la tortue LOGO (module `turtle`), écrire un algorithme qui dessine un motif géométrique composé de $(n \times m)$ pavés élémentaires disposés régulièrement sur une grille ou disposés en quinconce sur la grille.



alignés



en quinconce

- | | | | |
|---|--------------|--|--------------|
| 1.  | alignés | 13.  | alignés |
| 2.  | en quinconce | 14.  | en quinconce |
| 3.  | alignés | 15.  | alignés |
| 4.  | en quinconce | 16.  | en quinconce |
| 5.  | alignés | 17.  | alignés |
| 6.  | en quinconce | 18.  | en quinconce |
| 7.  | alignés | 19.  | alignés |
| 8.  | en quinconce | 20.  | en quinconce |
| 9.  | alignés | 21.  | alignés |
| 10.  | en quinconce | 22.  | en quinconce |
| 11.  | alignés | 23.  | alignés |
| 12.  | en quinconce | 24.  | en quinconce |

8.5 Révisions

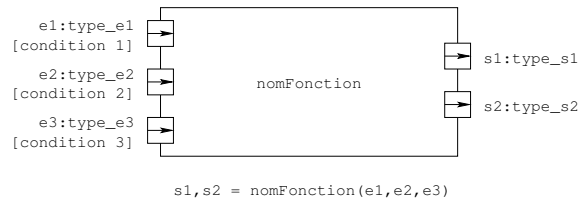
Cours	[1] : chapitre 2, sections 2.4.2 à 2.4.4
TD	[2] : exercices 2.20 à 2.24 et 2.34

9 Comment spécifier une fonction ?

Objectif : savoir spécifier une fonction.

Syntaxe Python :

```
def nom(entrées) :
    """
    description
    jeux de tests
    """
    assert préconditions
    ...
    return sorties
```



9.1 Exemple

Enoncé : On dispose d'une recette de quatre-quarts aux pépites de chocolat pour laquelle on veut séparer le « quoi » faire du « comment » faire.

Ingrédients (pour 8 personnes) :

- 100 g de pépites de chocolat,
- 200 g de beurre ramolli,
- 200 g de sucre en poudre,
- 3 œufs,
- 200 g de farine,
- 1 pincée de sel.



Préparation (≈ 30' + 45' de cuisson) :

- Préchauffer le four à 160°C (thermostat 5-6).
- Travailler le beurre ramolli, mais non fondu, avec une spatule pour le rendre crémeux.
- Ajouter progressivement le sucre et battre vigoureusement, le mélange doit être onctueux.
- Casser les œufs en séparant les jaunes des blancs. Ajouter les jaunes à la préparation et mélanger.
- Incorporer progressivement la farine tamisée puis les blancs battus en neige avec une pincée de sel, en soulevant la pâte de bas en haut.
- Ajouter les pépites de chocolat en mélangeant délicatement.
- Verser la préparation dans un moule à quatre-quarts beurré et fariné et faire cuire 45 minutes à four chaud.

Méthode : Distinguer la spécification de l'implémentation :

- la spécification d'un algorithme décrit ce que fait l'algorithme et dans quelles conditions il le fait ;
- l'implémentation d'un algorithme décrit comment fait l'algorithme pour satisfaire sa spécification.

Questions :

Q 9.1 (« quatre-quarts » : situations initiale et finale)

1. De quoi dispose-t-on avant de réaliser la quatre-quarts (situation initiale) ?
2. De quoi dispose-t-on après avoir réalisé le quatre-quarts (situation finale) ?

Un algorithme est un mécanisme qui fait passer un « système » d'une « situation » dite initiale à une « situation » finale. Le couple (situation initiale, situation finale) constitue une spécification de l'algorithme.

Q 9.2 (« quatre-quarts » : réalisation) *Quelles actions doit-on effectuer pour réaliser le quatre-quarts ?*

La suite d'instructions qui permet de passer de la situation initiale à la situation finale constitue une implémentation de l'algorithme. Il peut exister plusieurs implémentations pour une même spécification.

9.2 Généralisation

Une fonction est un bloc d'instructions nommé et paramétré, réalisant une certaine tâche. Elle admet zéro, un ou plusieurs paramètres et renvoie toujours un résultat.

Une fonction en informatique se distingue principalement de la fonction mathématique par le fait qu'en plus de calculer un résultat à partir de paramètres, la fonction informatique peut avoir des « effets de bord » : par exemple afficher un message à l'écran, jouer un son, ou bien piloter une imprimante. Une fonction qui n'a pas d'effet de bord joue le rôle d'une expression évaluable. Une fonction qui n'a que des effets de bord est appelée une procédure et joue le rôle d'une instruction.

Une procédure est un bloc d'instructions nommé et paramétré, réalisant une certaine tâche. Elle admet zéro, un ou plusieurs paramètres et ne renvoie pas de résultat.

Q 9.3 (spécification d'une fonction : nommer) *Proposer un nom pour la procédure qui réalise un quatre-quarts aux pépites.*

Les paramètres d'entrée d'une fonction sont les arguments de la fonction qui sont nécessaires pour effectuer le traitement associé à la fonction.

Les paramètres de sortie d'une fonction sont les résultats retournés par la fonction après avoir effectué le traitement associé à la fonction.

Q 9.4 (spécification d'une fonction : paramétrer) *Préciser les paramètres d'entrée et de sortie de la procédure qui réalise un quatre-quarts aux pépites.*

Les préconditions d'une fonction sont les conditions que doivent impérativement vérifier les paramètres d'entrée de la fonction juste avant son exécution.

Q 9.5 (spécification d'une fonction : protéger) *Quelles sont les conditions que doivent vérifier les paramètres d'entrée de la procédure de réalisation du quatre-quarts ?*

Un jeu de tests d'une fonction est un ensemble d'entrées-sorties associées que devra vérifier la fonction une fois implémentée.

Q 9.6 (spécification d'une fonction : tester) *Proposer un jeu de tests pour la procédure de réalisation du quatre-quarts.*

L'algorithmique s'intéresse à l'art de construire des algorithmes ainsi qu'à caractériser leur validité, leur robustesse, leur réutilisabilité, leur complexité ou encore leur efficacité. Certaines de ces caractéristiques générales, en particulier la validité, la robustesse et la réutilisabilité se concrétisent à la lumière des préconisations précédentes concernant la spécification d'une fonction.

1. La validité d'un algorithme est son aptitude à réaliser exactement la tâche pour laquelle il a été conçu.
ie : L'implémentation de la fonction doit être conforme aux jeux de tests.
2. La robustesse d'un algorithme est son aptitude à se protéger de conditions anormales d'utilisation.
ie : La fonction doit vérifier impérativement ses préconditions.
3. La réutilisabilité d'un algorithme est son aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.
ie : La fonction doit être correctement paramétrée.

9.3 Applications

Q 9.7 (prix d'une photocopie) Spécifier la fonction qui calcule le prix p de n photocopies sachant que le reprographe facture 0.10 € les 10 premières photocopies, 0.09 € les vingt suivantes et 0.08 € au-delà.

Q 9.8 (somme arithmétique)

1. Spécifier la fonction qui calcule la somme $s = \sum_0^n u_k$ des n premiers termes d'une suite arithmétique $u_k = a + kb$.
2. Proposer au moins deux implémentations différentes pour la spécification proposée. Classer ces implémentations par ordre de complexité croissante.

9.4 Entraînement

9.4.1 Enoncé

Objectif : spécifier une fonction connue (un « grand classique »).

Méthode : procéder en 4 étapes : nommer, paramétrer, protéger, proposer un jeu de tests, en faisant évoluer à chaque étape la description de la fonction.

Vérification : vérifier le jeu de tests grâce à une implémentation connue.

9.4.2 Exemple

Soit à spécifier la fonction qui détermine les racines réelles d'un trinôme du second degré à coefficients réels ($ax^2 + bx + c$).

Méthode : on avancera pas à pas en passant par les 4 étapes conseillées : nommer, paramétrer, protéger, proposer un jeu de tests, en faisant évoluer la description de la fonction en conséquence.

1. **Nommer** la fonction à l'aide d'un identificateur suffisamment explicite. Dans cet exemple, nous choisirons : `racinesTrinome`, avec pour description « détermination des racines d'un trinôme du second degré ».

```
1 def racinesTrinome() :  
2     """  
3     détermination des racines d'un  
4     trinôme du second degré  
5     """  
6     return
```

2. **Paramétrer** la fonction en entrée comme en sortie. Ici, la fonction prendra trois arguments en entrée : les coefficients a , b et c du trinôme $ax^2 + bx + c$. En sortie, nous attendons une liste des racines obtenues : « racines : liste des racines du trinôme $a*x**2 + b*x + c$ ». La longueur de la liste retournée (`len(racines)`) indiquera le nombre de racines obtenues (0, 1 ou 2).

```

1 def racinesTrinome(a,b,c) :
2     """
3     racines = racinesTrinome(a,b,c)
4     racines : liste des racines du
5                 trinome a*x**2 + b*x + c
6     """
7     racines = []
8     return racines

```

3. **Protéger** la fonction à l'aide de préconditions sur les paramètres d'entrée. Ici, a doit être un réel (`type(a) is float`) non nul (`a != 0`), b et c des réels (`type(b) is float`, `type(c) is float`). En PYTHON, ces conditions seront testées à l'aide de l'instruction `assert`. La description s'enrichit alors de ces conditions sur les paramètres d'entrée.

```

1 def racinesTrinome(a,b,c) :
2     """
3     racines = racinesTrinome(a,b,c)
4     racines : liste des racines du
5                 trinome a*x**2 + b*x + c
6                 (a: float != 0, b: float, c: float)
7     """
8     assert type(a) is float and a != 0.0
9     assert type(b) is float
10    assert type(c) is float
11
12    racines = []
13    return racines

```

4. **Proposer un jeu de tests** le plus caractéristique possible. On testera notamment quelques identités remarquables :
 $(x - d)^2 = 0 \Rightarrow x = d$,
 $(x + d)^2 = 0 \Rightarrow x = -d$,
 $(x^2 - d^2) = 0 \Rightarrow x = d$ ou $x = -d$.
On testera également un cas où il n'y a pas de racines :
 $(x^2 + d^2)$.
Ces couples d'entrées-sorties cohérentes seront testés en PYTHON à l'aide du module `doctest`.

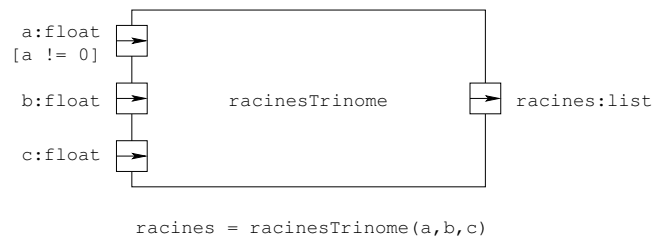
```

1 def racinesTrinome(a,b,c) :
2     """
3     racines = racinesTrinome(a,b,c)
4     racines : liste des racines du
5                 trinome a*x**2 + b*x + c
6                 (a: float != 0, b: float, c: float)
7
8     >>> racinesTrinome(1.0,0.0,1.0)
9     []
10    >>> racinesTrinome(1.0,-2.0,1.0)
11    [1.0]
12    >>> racinesTrinome(1.0,2.0,1.0)
13    [-1.0]
14    >>> racinesTrinome(1.0,0.0,-4.0)
15    [2.0, -2.0]
16    """
17    assert type(a) is float and a != 0.0
18    assert type(b) is float
19    assert type(c) is float
20
21    racines = []
22    return racines

```

Résultat : la spécification PYTHON du calcul des racines du trinôme correspond à la version obtenue à la dernière étape de la méthode précédente (étape 4 : « proposer un jeu de tests »).

Le diagramme UML associé est donné ci-contre :



Vérifications : la fonction sera testée une fois son implémentation définie (voir par exemple le TD 2.31 page 70 du cours [1]) grâce aux lignes suivantes dans le fichier source :

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Lors de la compilation du fichier, l'interpréteur signalera toutes différences observées entre le jeu de tests spécifié et les résultats effectifs obtenus en testant lui-même les instructions du jeu de tests.

9.4.3 Questions

Proposer une spécification¹ pour les fonctions suivantes :

- | | |
|--------------------------------------|----------------------------------|
| 1. algorithme d'Euclide | 13. produit mixte de vecteurs |
| 2. coefficients du binôme | 14. racines du trinôme |
| 3. conversion en base b | 15. recherche dichotomique |
| 4. courbe fractale de Koch | 16. recherche séquentielle |
| 5. crible d'Eratosthène | 17. tours de Hanoï |
| 6. développement limité de $\sin(x)$ | 18. tracé d'un polygone régulier |
| 7. fonction factorielle | 19. tri bulles |
| 8. fonction puissance | 20. tri fusion |
| 9. intégration numérique | 21. tri par insertion |
| 10. nombres de Fibonacci | 22. tri par sélection |
| 11. palindrome | 23. tri rapide |
| 12. produit de matrices | 24. zéro d'une fonction |

9.5 Révisions

Cours	[1] : chapitre 3, sections 3.1 et 3.2
TD	[2] : exercices 3.3 à 3.5,

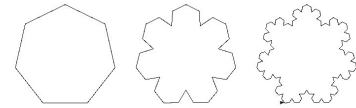
1. Les implémentations de ces fonctions ne sont pas demandées; on les trouvera dans le cours « Initiation à l'algorithmique » [1].

10 Qu'est-ce que la récursivité?

Objectif : comprendre la récursivité.

Syntaxe Python :

```
def fonction(entrées) :  
    ...  
    s = fonction(autres entrées)  
    ...  
    return sorties
```



Flocons de Koch

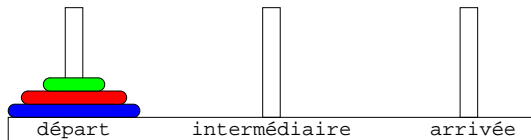
10.1 Exemple

Enoncé : Les « tours de Hanoï » est un jeu imaginé par le mathématicien français Édouard Lucas (1842-1891). Il consiste à déplacer n disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire » et ceci en un minimum de coups, tout en respectant les règles suivantes :

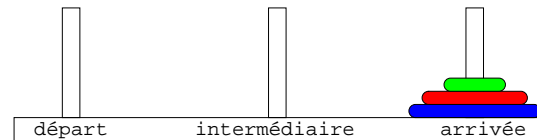
- on ne peut déplacer qu'un disque à la fois ;
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur une tour vide.

Dans l'état initial, les n disques sont placés sur la tour « départ ». Dans l'état final, tous les disques se retrouvent placés dans le même ordre sur la tour « arrivée ».

Etat initial : $n = 3$



Etat final : $n = 3$



Méthode : On cherchera à mettre en évidence une relation de récurrence entre le jeu à n disques et le jeu à $n - 1$ disques.

Q 10.1 (« tours de Hanoï » : à la main) Résoudre à la main le problème des tours de Hanoï à n disques successivement pour $n = 1$, $n = 2$ et $n = 3$.

Q 10.2 (« tours de Hanoï » : relation de récurrence)

1. Trouver une relation de récurrence entre le problème des tours de Hanoï à $n = 4$ disques avec celui à $n = 3$ disques.
2. Généraliser à un nombre de disques n quelconque. En déduire qu'il faut effectuer $(2^n - 1)$ déplacements de disque pour déplacer n disques de la tour « départ » à la tour « arrivée ».
3. Proposer un algorithme pour résoudre le problème des tours de Hanoï qui consiste à déplacer n disques d'une tour de « départ » (d) à une tour d'« arrivée » (a) en passant par une tour « intermédiaire » (i).

10.2 Généralisation

Une fonction est dite récursive si elle s'appelle elle-même : on parle alors d'appel récursif de la fonction. Un appel récursif terminal est un appel récursif dont le résultat est celui retourné par la fonction. En d'autres termes, si dans le corps d'une fonction, un appel récursif est placé

de telle façon que son exécution n'est jamais suivi par l'exécution d'une autre instruction de la fonction, cet appel est dit récursif à droite ou récursif terminal. A contrario, un appel récursif non terminal est un appel récursif dont le résultat n'est pas celui retourné par la fonction.

Q 10.3 (récursivité : exécution d'une fonction récursive)

On considère la procédure récursive f définie ci-contre.

1. Qu'affichent les appels suivants ?

```
>>> f(0, 'd', 'i', 'a') ?
>>> f(1, 'd', 'i', 'a') ?
>>> f(2, 'd', 'i', 'a') ?
>>> f(3, 'd', 'i', 'a') ?
```

```
1 def f(n,d,i,a) :
2     if n > 0 :
3         f(n-1,d,a,i)
4         print(d, '->', a)
5         f(n-1,i,d,a)
6     return
```

2. Vérifier que cette fonction f exécute correctement les déplacements des n disques des tours de Hanoï pour $n = 0$, $n = 1$, $n = 2$ et $n = 3$ respectivement.
3. Dans le code précédent, qu'est-ce qui permet d'arrêter la récursivité ?

Pour définir une fonction récursive, il faut disposer d'une relation de récurrence et d'un cas particulier (clause d'arrêt).

Q 10.4 (récursivité : fonction factorielle) Définir une fonction récursive qui calcule la fonction factorielle ($n!$). Préciser la relation de récurrence et la clause d'arrêt.

Q 10.5 (récursivité : plus grand commun diviseur) Définir une fonction récursive qui calcule le plus grand commun diviseur d de 2 entiers a et b : $\text{pgcd}(a, b) = \text{pgcd}(b, a \% b) = \text{pgcd}(d, 0) = d$. Préciser la relation de récurrence et la clause d'arrêt.

Quel que soit le problème à résoudre, on a le choix entre l'écriture d'une fonction itérative et celle d'une fonction récursive. Si le problème admet une décomposition récurrente naturelle, le programme récursif est alors une simple adaptation de la décomposition choisie. C'est le cas des fonctions `hanoi`, `factorielle` et `pgcd` par exemple.

L'approche récursive présente cependant des inconvénients : certains langages n'admettent pas la récursivité (comme le langage machine !) et elle est souvent coûteuse en mémoire comme en temps d'exécution. On peut pallier ces inconvénients en transformant la fonction récursive en fonction itérative : c'est toujours possible.

Q 10.6 (récursivité : récursivité \rightarrow itération) Comparer les versions récursive et itérative de la fonction qui calcule le plus grand commun diviseur d de 2 entiers a et b . Montrer qu'on peut passer de la version récursive à la version itérative en effectuant la transformation de code suivante :

```
1 def f(x) :
2     if cond : return y
3     else :
4         instructions
5     return f(g(x))
```

\rightarrow

```
1 def f(x) :
2     while not cond :
3         instructions
4         x = g(x)
5     return y
```

où x représente ici la liste des arguments de la fonction f , cond une condition portant sur x , instructions un bloc d'instructions qui constituent le traitement de base de la fonction f , $g(x)$ une transformation des arguments et $\text{return } y$ l'instruction de terminaison (clause d'arrêt) de la récurrence.

La méthode précédente ne s'applique qu'à la récursivité terminale. Une méthode générale existe pour transformer une fonction récursive quelconque en une fonction itérative équivalente. En particulier, elle est mise en œuvre dans les compilateurs car le langage machine n'admet pas la récursivité. Cette méthode générale fait appel à la notion de pile pour sauvegarder le contexte des appels récursifs.

10.3 Applications

Q 10.7 (récursivité : fonction d'Ackerman) Définir une fonction récursive qui calcule la fonction d'Ackerman :

$$f : \mathbb{N}^2 \rightarrow \mathbb{N} \quad \begin{cases} f(0, n) &= n + 1 \\ f(m, 0) &= f(m - 1, 1) \text{ si } m > 0 \\ f(m, n) &= f(m - 1, f(m, n - 1)) \text{ si } m > 0, n > 0 \end{cases}$$

Q 10.8 (récursivité : coefficients du binôme) Définir une fonction récursive qui calcule les coefficients du binôme $(a + b)^n = \sum_{k=0}^n \frac{n!}{k!(n-k)!} a^{n-k} b^k = \sum_{k=0}^n C_n^k a^{n-k} b^k$ où le coefficient du binôme C_n^k représente le nombre de combinaisons de k parmi n : $C_n^k = \frac{n!}{k!(n-k)!} = \binom{n}{k}$.

Q 10.9 (récursivité : flocons de Koch)

On s'intéresse ici aux programmes dont l'exécution produit des dessins à l'aide de la tortue LOGO. On considère la procédure **draw** ci-contre.

1. Dessiner le résultat des appels **draw**(*n*, 900) respectivement pour *n* = 0, *n* = 1, *n* = 2 et *n* = 3. A chaque appel, le crayon est initialement en (0,0) avec une direction de 0.
2. Définir une fonction **koch** qui dessine les flocons de Koch présentés sur la figure placée en exercice de la section 10.

```

1  def draw(n, d) :
2      if n == 0 : forward(d)
3      else :
4          draw(n-1, d/3)
5          left(60)
6          draw(n-1, d/3)
7          right(120)
8          draw(n-1, d/3)
9          left(60)
10         draw(n-1, d/3)
11     return

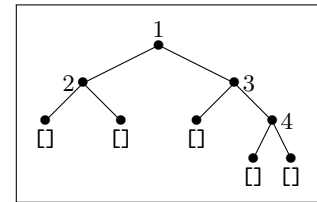
```

10.4 Entraînement

10.4.1 Enoncé

Objectif : On considère la notion d'arbre binaire. Un arbre binaire est soit vide, soit composé de 3 éléments : une racine, un arbre binaire **Gauche** et un arbre binaire **Droite**. L'arbre binaire vide sera représenté ici par la liste vide (`[]`), l'arbre binaire non vide par une liste de 3 éléments (`[racine, Gauche, Droite]`) où **Gauche** et **Droite** sont eux-mêmes des arbres binaires et **racine** de type quelconque.

Ainsi, l'arbre binaire ci-contre est représenté par la liste :
[1, [2, [], []], [3, [], [4, [], []]]].



Les deux fonctions récursives `infix` et `postfix` ci-dessous permettent d'afficher les éléments d'un arbre binaire (0 pour l'arbre vide) de deux manières différentes, respectivement en notation infixée et postfixée.

```
1 def infix(t) :
2     if t != [] :
3         infix(t[1])
4         print(t[0], end=' ')
5         infix(t[2])
6     else :
7         print(0, end=' ')
8     return
```

```
>>> t = [1, [2, [], []], [3, [], [4, [], []]]]
>>> infix(t)
0 2 0 1 0 3 0 4 0
```

```
1 def postfix(t) :
2     if t != [] :
3         postfix(t[1])
4         postfix(t[2])
5         print(t[0], end=' ')
6     else :
7         print(0, end=' ')
8     return
```

```
>>> t = [1, [2, [], []], [3, [], [4, [], []]]]
>>> postfix(t)
0 0 2 0 0 0 4 3 1
```

Question : Exécuter « à la main » les fonctions `infix` et `postfix` sur des arbres binaires.

Vérification : Représenter graphiquement l'arbre binaire et le parcourir de manière infixée et postfixée pour comparer aux résultats obtenus par les fonctions `infix` et `postfix`. Finalement, vérifier avec PYTHON.

10.4.2 Exemple

On veut parcourir l'arbre binaire `t = [1, [2, [4, [], []], []], [3, [5, [], []], [6, [], [7, [], []]]]]` de manière infixée (fonction `infix`) et de manière postfixée (fonction `postfix`).

Méthode : En notation infixée, on commence par décrire le sous-arbre de gauche `t[1]` (= `[2, [4, [], []], []]`), la racine `t[0]` (= 1) puis le sous-arbre de droite `t[2]` (= `[3, [5, [], []], [6, [], [7, [], []]]]`). Le sous-arbre de gauche `t[1]` est lui même décrit en commençant par son sous-arbre de gauche `t[1][1]` (= `[4, [], []]`), la racine `t[1][0]` (= 2) puis son sous-arbre de droite `t[1][2]` (= []). Enfin le sous-arbre de gauche `t[1][1]` est décrit en commençant par son sous-arbre de gauche `t[1][1][1]` (= []), sa racine `t[1][1][0]` (= 4) puis son sous-arbre de droite `t[1][1][2]` (= []). Lorsqu'un arbre est vide, on le décrit par 0. Ainsi, en notation infixée, le sous-arbre gauche `t[1]` (= `[2, [4, [], []], []]`) est représenté par la séquence 0 4 0 2 0. On procède de même avec le sous-arbre de droite `t[2]` (= `[3, [5, [], []], [6, [], [7, [], []]]]`); ce qui conduit à la séquence 0 5 0 3 0 6 0 7 0. L'arbre `t` complet est donc décrit par la séquence 0 4 0 2 0 1 0 5 0 3 0 6 0 7 0 en notation infixée.

En notation postfixée, on commence par décrire le sous-arbre de gauche `t[1]` (= `[2, [4, [], []], []]`), le sous-arbre de droite `t[2]` (= `[3, [5, [], []], [6, [], [7, [], []]]]`), puis la racine `t[0]` (= 1). Le sous-arbre de gauche `t[1]` est lui même décrit en commençant par son sous-arbre de gauche `t[1][1]` (= `[4, [], []]`), son sous-arbre de droite `t[1][2]` (= []) puis la racine

$t[1][0]$ (= 2). Enfin le sous-arbre de gauche $t[1][1]$ est décrit en commençant par son sous-arbre de gauche $t[1][1][1]$ (= []), son sous-arbre de droite $t[1][1][2]$ (= []) puis sa racine $t[1][1][0]$ (= 4). Lorsqu'un arbre est vide, on le décrit par 0. Ainsi, en notation postfixée, le sous-arbre gauche $t[1]$ (= [[2, [4, [], []], []]]) est représenté par la séquence 0 0 4 0 2. On procède de même avec le sous-arbre de droite $t[2]$ (= [3, [5, [], []], [6, [], [7, [], []]]); ce qui conduit à la séquence 0 0 5 0 0 0 7 6 3. L'arbre t complet est donc décrit par la séquence 0 0 4 0 2 0 0 5 0 0 0 7 6 3 1 en notation postfixée.

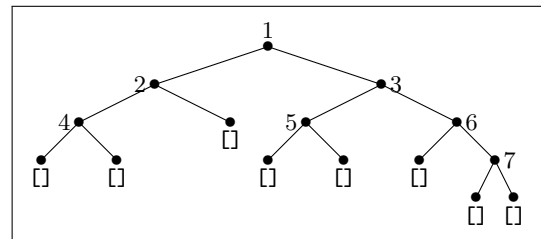
Résultats : Les résultats sont reportés dans le tableau ci-dessous.

$t = [1, [2, [4, [], []], []], [3, [5, [], []], [6, [], [7, [], []]]]]$	
notation infixée : <code>infix(t)</code>	$\rightarrow 0\ 4\ 0\ 2\ 0\ 1\ 0\ 5\ 0\ 3\ 0\ 6\ 0\ 7\ 0$
notation postfixée : <code>postfix(t)</code>	$\rightarrow 0\ 0\ 4\ 0\ 2\ 0\ 0\ 5\ 0\ 0\ 0\ 7\ 6\ 3\ 1$

Vérification : On vérifie selon deux méthodes :

- en s'aidant de la représentation graphique de l'arbre binaire,
- en exécutant le code sous PYTHON.

1. La représentation binaire ci-contre permet de se faire visuellement une meilleure idée de l'arbre binaire t et de repérer plus facilement les sous-arbres gauche et droite à tous les niveaux de l'arbre. On vérifie aisément par exemple que le sous-arbre de gauche est représenté par la séquence 0 4 0 2 0 en notation infixée et par la séquence 0 0 4 0 2 en notation postfixée, et ainsi de suite...



2. L'exécution sous PYTHON confirme nos premiers résultats :

```
>>> t = [1, [2, [4, [], []], []], [3, [5, [], []], [6, [], [7, [], []]]]]
>>> infix(t)
0 4 0 2 0 1 0 5 0 3 0 6 0 7 0
>>> postfix(t)
0 0 4 0 2 0 0 5 0 0 0 7 6 3 1
```

10.4.3 Questions

On reprend ici les fonctions `infix` et `postfix` définies plus haut (section 10.4.1). Exécuter à la main les appels suivants.

1. `>>> infix([1, [3, [5, [], []], []], [2, [], [4, [], []]])`
2. `>>> postfix([1, [3, [5, [], []], []], [2, [], [4, [], []]])`
3. `>>> infix([2, [4, [6, [], []], []], [1, [], [3, [], []]])`
4. `>>> postfix([2, [4, [6, [], []], []], [1, [], [3, [], []]])`
5. `>>> infix([1, [3, [], [5, [], []], [2, [4, [], []], []]])`
6. `>>> postfix([1, [3, [], [5, [], []], [2, [4, [], []], []]])`
7. `>>> infix([2, [4, [], [6, [], []], [1, [3, [], []], []]])`

```

8. >>> postfix([2, [4, [], [6,[],[]]] , [1, [3,[],[]], []]))
9. >>> infix([1, [3, [5,[],[]], [4,[],[]]] , [2, [], []]))
10. >>> postfix([1, [3, [5,[],[]], [4,[],[]]] , [2, [], []]))
11. >>> infix([2, [4, [], []] , [1, [], [6,[],[3,[],[]]]]))
12. >>> postfix([2, [4, [], []] , [1, [], [6,[],[3,[],[]]]]))
13. >>> infix([1, [], [2, [], [3, [5,[],[]], [4,[],[]]]]))
14. >>> postfix([1, [], [2, [], [3, [5,[],[]], [4,[],[]]]]))
15. >>> infix([2, [4, [], []] , [1, [], [6,[],[3,[],[]]]]))
16. >>> postfix([2, [4, [], []] , [1, [], [6,[],[3,[],[]]]]))
17. >>> infix([1, [2, [4,[],[]], [3, [5,[],[]], []]] , [])
18. >>> postfix([1, [2, [4,[],[]], [3, [5,[],[]], []]] , [])
19. >>> infix([2, [], [1, [4, [], []], [6,[],[3,[],[]]]]))
20. >>> postfix([2, [], [1, [4, [], []], [6,[],[3,[],[]]]]))
21. >>> infix([5, [3, [1,[],[]], []] , [4, [], [2,[],[]]]))
22. >>> postfix([5, [3, [1,[],[]], []] , [4, [], [2,[],[]]]))
23. >>> infix([6, [4, [2,[],[]], []] , [3, [], [1,[],[]]]))
24. >>> postfix([6, [4, [2,[],[]], []] , [3, [], [1,[],[]]]))

```

10.5 Révisions

Cours	[1] : chapitre 3, section 3.3
TD	[2] : exercices 3.9 à 3.12, 3.16 à 3.19

11 Comment trier une séquence ?

Objectif : savoir trier une séquence.

Syntaxe Python :

```
>>> t = [6,4,1,3,5,2]
>>> trier(t)
>>> t
[1, 2, 3, 4, 5, 6]
```

6	4	1	3	5	2
4	6	1	3	5	2
1	4	6	3	5	2
1	3	4	6	5	2
1	3	4	5	6	2
1	2	3	4	5	6

Tri de la liste
[6,4,1,3,5,2] :
en gras, les valeurs
insérées dans la par-
tie gauche à chaque
étape.

11.1 Exemple

Enoncé : On cherche à trier un jeu de n cartes selon les valeurs décroissantes des cartes et des couleurs ($\spadesuit > \heartsuit > \diamondsuit > \clubsuit$).

Exemple : $A\heartsuit, 10\clubsuit, 9\spadesuit, 7\diamondsuit, V\spadesuit, R\clubsuit, D\heartsuit, 7\heartsuit \rightarrow V\spadesuit, 9\spadesuit, A\heartsuit, D\heartsuit, 7\heartsuit, 7\diamondsuit, R\clubsuit, 10\clubsuit$

Méthode : Prendre les cartes mélangées une à une sur la table, et former une « main » en insérant chaque carte saisie à sa place : on parle de tri par insertion. Dans le tri d'une séquence par insertion, on parcourt la séquence à trier du début à la fin. Au moment où on considère le $i^{\text{ème}}$ élément, les éléments qui le précèdent sont déjà triés. Dans le cas du jeu de cartes, le $i^{\text{ème}}$ élément est la carte saisie, les éléments précédents sont la main triée et les éléments suivants correspondent aux cartes encore mélangées sur la table.



Questions :

Q 11.1 (« jeu de cartes » : tri par insertion) On considère la séquence de 8 cartes suivante : $A\heartsuit, 10\clubsuit, 9\spadesuit, 7\diamondsuit, V\spadesuit, R\clubsuit, D\heartsuit, 7\heartsuit$. Décrire, étape par étape, le passage de cette séquence à la séquence triée : $V\spadesuit, 9\spadesuit, A\heartsuit, D\heartsuit, 7\heartsuit, 7\diamondsuit, R\clubsuit, 10\clubsuit$, par la méthode du tri par insertion.

Q 11.2 (« jeu de cartes » : complexité)

1. Combien de comparaison effectuera-t-on en moyenne pour trier un jeu de n cartes par la méthode du tri par insertion ?
2. Si le jeu de n cartes est déjà trié en ordre inverse, combien de comparaison effectuera-t-on pour trier les cartes par la méthode du tri par insertion ?

11.2 Généralisation

Dans la suite, nous supposons qu'il existe une relation d'ordre total, entre les éléments de la liste à trier, notée \leq et qui vérifie les propriétés habituelles de réflexivité, d'antisymétrie et de transitivité :

1. réflexivité : $x \leq x$
2. antisymétrie : $(x \leq y) \text{ and } (y \leq x) \Rightarrow x = y$
3. transitivité : $(x \leq y) \text{ and } (y \leq z) \Rightarrow (x \leq z)$

Q 11.3 (relation d'ordre) Définir une fonction *enOrdre* qui teste si une liste *t* est triée par ordre croissant de ses éléments entre les rangs *debut* (inclus) et *fin* (inclus), en utilisant la relation d'ordre \leq : $t[i-1] \leq t[i] \leq t[i+1]$.

Le tri par sélection d'une liste consiste à rechercher le minimum de la liste à trier, de le mettre en début de liste en l'échangeant avec le premier élément et de recommencer sur le reste de la liste.

Tri par sélection					
6	4	1	3	5	2
1	4	6	3	5	2
1	2	6	3	5	4
1	2	3	6	5	4
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6

en gras, les valeurs
échangées à chaque
étape

```

1 def triSelection(t,debut,fin):
2     assert type(t) is list
3     assert 0 <= debut <= fin < len(t)
4     if debut < fin:
5         mini = minimum(t,debut,fin)
6         t[debut],t[mini] = t[mini],t[debut]
7         triSelection(t,debut+1,fin)
8     return

```

Q 11.4 (tri par sélection)

1. Définir la fonction *minimum* qui retourne le rang du plus petit élément de la séquence *t* entre les rangs *debut* (inclus) et *fin* (inclus).
2. Proposer une version itérative de la fonction *triSelection* qui trie la liste *t* par ordre croissant entre les rangs *debut* (inclus) et *fin* (inclus).

Le principe du tri rapide est le suivant : on partage la liste à trier en deux sous-listes telles que tous les éléments de la première soient inférieurs à tous les éléments de la seconde. Pour partager la liste en deux sous-listes, on choisit un des éléments de la liste (par exemple le premier) comme pivot. On construit alors une sous-liste avec tous les éléments inférieurs ou égaux à ce pivot et une sous-liste avec tous les éléments supérieurs au pivot. On trie les deux sous-listes selon le même processus jusqu'à avoir des sous-listes réduites à un seul élément.

Tri rapide					
6	4	1	3	5	2
2	4	1	3	5	6
1	2	4	3	5	6
1	2	3	4	5	6
1	2	3	4	5	6

en gras, les valeurs
des pivots à chaque
étape

```

1 def triRapide(t,debut,fin):
2     assert type(t) is list
3     assert 0 <= debut
4     assert fin <= len(t)
5     if debut < fin:
6         pivot = t[debut]
7         place = partition(t,debut,fin,pivot)
8         triRapide(t,debut,place-1)
9         triRapide(t,place+1,fin)
10    return

```

Pour partitionner la liste, on utilise 2 compteurs *inf* et *sup* qui partent des 2 extrémités de la liste en évoluant l'un vers l'autre. Le compteur de gauche *inf* part du début de la liste et lorsqu'il atteint un élément supérieur au pivot, on arrête sa progression. De même, le compteur de droite *sup* part de la fin de la liste et s'arrête lorsqu'il atteint un élément inférieur au pivot. On échange alors ces deux éléments ($t[\text{sup}], t[\text{inf}] = t[\text{inf}], t[\text{sup}]$) puis on continue à faire progresser les compteurs et à faire des échanges jusqu'à ce que les compteurs se croisent.

Q 11.5 (tri rapide)

1. Définir la fonction **partition** qui partitionne la liste **t** entre les rangs **debut** (inclus) et **fin** (inclus) par rapport au **pivot**.
2. Vérifier le bon fonctionnement de la version récursive de la fonction **triRapide** qui trie la liste **t** par ordre croissant entre les rangs **debut** (inclus) et **fin** (inclus).

11.3 Applications

Q 11.6 (tri bulles) Dans le tri bulles, on parcourt la liste en commençant par la fin, en effectuant un échange à chaque fois que l'on trouve deux éléments successifs qui ne sont pas dans le bon ordre.

1. Définir une procédure **triBulles** qui trie une liste **t** par ordre croissant entre les rangs **debut** (inclus) et **fin** (inclus) selon la méthode du tri bulles.
2. Évaluer la complexité en nombre de comparaisons du tri bulles.

Q 11.7 (tri fusion) Dans le tri fusion, on partage la liste à trier en deux sous-listes que l'on trie, puis on interclasse (on fusionne) ces deux sous-listes.

Définir une procédure **triFusion** qui trie une liste **t** par ordre croissant entre les rangs **debut** (inclus) et **fin** (inclus) selon la méthode du tri fusion.

11.4 Entraînement

11.4.1 Enoncé

Objectif : trier un annuaire selon différents critères, l'annuaire étant représenté par une liste de quadruplets (nom, age, ville, téléphone).

Exemple d'annuaire :

```
item1 = ('dupont', 23, 'brest', '06789656')
item2 = ('abgral', 61, 'lille', '06231298')
item3 = ('dupont', 23, 'brest', '02989656')
item4 = ('abgral', 67, 'brest', '06556438')
item5 = ('martin', 38, 'paris', '01674523')
item6 = ('abgral', 67, 'lille', '06231298')
```

```
annuaire = [item1, item2, item3, item4, item5, item6]
```

Méthode : utiliser un algorithme de tri (sélection, bulles, insertion, fusion ou rapide).

Vérification : vérifier à l'aide de différents annuaires et différents critères.

11.4.2 Exemple

Soit à trier un jeu de cartes stocké sous la forme d'une liste de couples (couleur, valeur) où la valeur est un entier compris entre 2 et 14 (valet = 11, dame = 12, roi = 13 et as = 14) et la couleur un autre entier tel que ♠ = 4, ♥ = 3, ♦ = 2 et ♣ = 1.

Ainsi le jeu A♥, 10♣, 9♠, 7♦, V♠, R♣, D♥, 7♥ est représenté par la liste [(3,14), (1,10), (4,9), (2,7), (4,11), (1,13), (3,12), (3,7)].

Méthode : On utilise un tri par insertion où la fonction `cmp` de comparaison est passée en argument. Par défaut, le tri est effectué par ordre croissant des éléments : `cmp = (lambda x,y : x < y)`.

```

1 def triInsertion(t,cmp=(lambda x,y : x < y)) :
2     assert type(t) is list
3     for i in range(1,len(t)):
4         x = t[i]
5         k = i
6         for j in range(i-1,-1,-1):
7             if not cmp(t[j],x) :
8                 k = k-1
9                 t[j+1] = t[j]
10            t[k] = x
11    return

```

Résultat : Il suffit ici d'appeler la fonction de tri avec la liste `t` de cartes voulue en effectuant le tri par ordre décroissant des éléments : `cmp = (lambda x,y : x > y)`.

```
t = [A♥, 10♣, 9♠, 7♦, V♠, R♣, D♥, 7♥]
```

```
>>> t = [(3,14), (1,10), (4,9), (2,7), (4,11), (1,13), (3,12), (3,7)]
```

```
>>> triInsertion(t,(lambda x,y : x > y))
```

```
>>> t
```

```
[(4, 11), (4, 9), (3, 14), (3, 12), (3, 7), (2, 7), (1, 13), (1, 10)]
```

```
t = [V♠, 9♠, A♥, D♥, 7♥, 7♦, R♣, 10♣]
```

Vérifications : la fonction sera testée avec différents jeux de cartes.

1. `t = []`

```
>>> t = []
```

```
>>> triInsertion(t,(lambda x,y : x > y))
```

```
>>> t
```

```
[]
```

2. `t = [A♥, A♣, A♠, A♦]`

```
>>> t = [(3,14), (1,14), (4,14), (2,14)]
```

```
>>> triInsertion(t,(lambda x,y : x > y))
```

```
>>> t
```

```
[(4, 14), (3, 14), (2, 14), (1, 14)]
```

3. `t = [3♣, 5♣, 8♣, 9♣, R♣]`

```
>>> t = [(1,3), (1,5), (1,8), (1,9), (1,13)]
```

```
>>> triInsertion(t,(lambda x,y : x > y))
```

```
>>> t
```

```
[(1, 13), (1, 9), (1, 8), (1, 5), (1, 3)]
```

4. `t = [R♣, 9♣, 8♣, 5♣, 3♣]`

```
>>> t = [(1,13), (1,9), (1,8), (1,5), (1,3)]
```

```
>>> triInsertion(t,(lambda x,y : x > y))
```

```
>>> t
```

```
[(1, 13), (1, 9), (1, 8), (1, 5), (1, 3)]
```

11.4.3 Questions

Un annuaire est représenté ici par une liste de quadruplets (nom, age, ville, téléphone).
Exemple d'annuaire :

```
item1 = ('dupont', 23, 'brest', '06789656')
item2 = ('abgral', 61, 'lille', '06231298')
item3 = ('dupont', 23, 'brest', '02989656')
item4 = ('abgral', 67, 'brest', '06556438')
item5 = ('martin', 38, 'paris', '01674523')
item6 = ('abgral', 67, 'lille', '06231298')
```

```
annuaire = [item1, item2, item3, item4, item5, item6]
```

Trier l'annuaire, par ordre croissant ou décroissant, selon des critères donnés par une liste des clés successives. L'ordre des critères est précisé par une liste des rangs successifs des champs du quadruplet (nom, age, ville, téléphone). Par exemple, la liste [3,0,2,1] indique qu'il faut d'abord (clé primaire) trier selon les numéros de téléphone (champ n° 3 dans le quadruplet), puis (clé secondaire) selon les noms (champ n° 0 dans le quadruplet), puis selon la ville (champ n° 2 dans le quadruplet) et enfin selon les âges (champ n° 1 dans le quadruplet).

- | | |
|---|---|
| 1. <code>cles = [2,0,1,3]</code> , croissant | 13. <code>cles = [0,1,2,3]</code> , décroissant |
| 2. <code>cles = [2,0,3,1]</code> , croissant | 14. <code>cles = [0,1,3,2]</code> , décroissant |
| 3. <code>cles = [2,1,0,3]</code> , croissant | 15. <code>cles = [0,2,1,3]</code> , décroissant |
| 4. <code>cles = [2,1,3,0]</code> , croissant | 16. <code>cles = [0,2,3,1]</code> , décroissant |
| 5. <code>cles = [2,3,0,1]</code> , croissant | 17. <code>cles = [0,3,1,2]</code> , décroissant |
| 6. <code>cles = [2,3,1,0]</code> , croissant | 18. <code>cles = [0,3,2,1]</code> , décroissant |
| 7. <code>cles = [3,0,1,2]</code> , croissant | 19. <code>cles = [1,0,2,3]</code> , décroissant |
| 8. <code>cles = [3,0,2,1]</code> , croissant | 20. <code>cles = [1,0,3,2]</code> , décroissant |
| 9. <code>cles = [3,1,0,2]</code> , croissant | 21. <code>cles = [1,2,0,3]</code> , décroissant |
| 10. <code>cles = [3,1,2,0]</code> , croissant | 22. <code>cles = [1,2,3,0]</code> , décroissant |
| 11. <code>cles = [3,2,0,1]</code> , croissant | 23. <code>cles = [1,3,0,2]</code> , décroissant |
| 12. <code>cles = [3,2,1,0]</code> , croissant | 24. <code>cles = [1,3,2,0]</code> , décroissant |

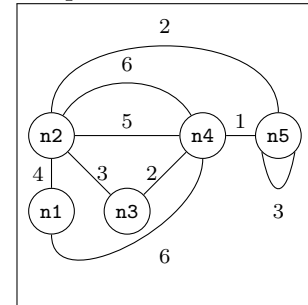
11.5 Révisions

Cours	[1] : chapitre 4, section 4.4
TD	[2] : exercices 4.15 à 4.19, 4.28

12 Tout en un ?

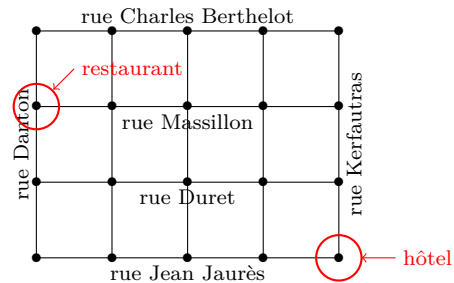
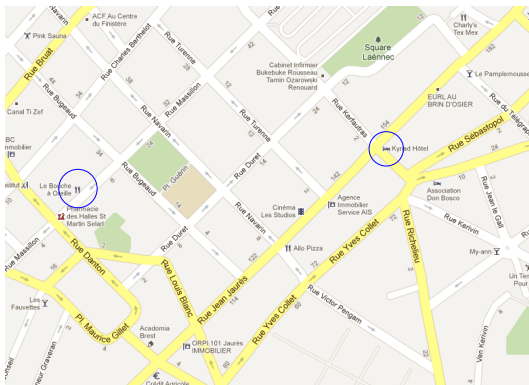
Objectif : mettre en œuvre l'ensemble des notions abordées dans ce document à travers l'exemple général de la recherche d'un chemin dans un graphe.

Graphe

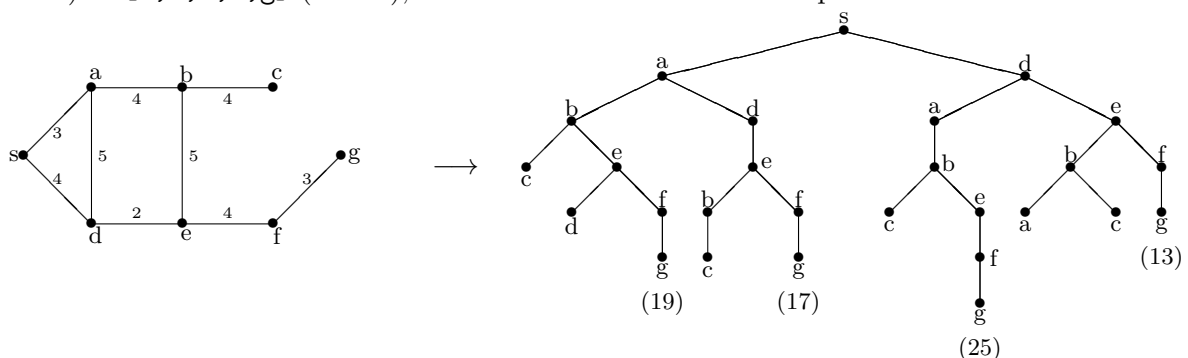


12.1 Exemple

Enoncé : Dans le premier exemple introductif (section 2, exemple 2.1 page 8 : « aller au restaurant ») un touriste devait rejoindre un restaurant depuis son hôtel. On peut, en première approximation, assimiler le plan de la ville à un graphe dont les nœuds (•) sont les carrefours et les arcs (—) les tronçons de rues entre deux carrefours, comme suggéré sur la figure ci-dessous. Le problème ainsi transposé devient la recherche d'un chemin d'un carrefour à un autre au sein d'un réseau urbain connu sans passer deux fois par le même carrefour.



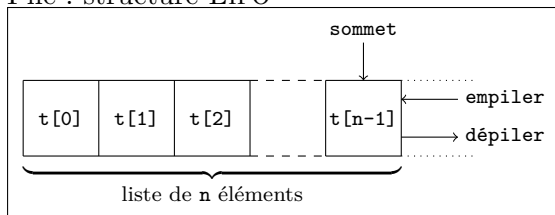
Méthode : Rechercher un chemin entre deux nœuds d'un graphe sans passer deux fois par le même nœud revient à parcourir un arbre de recherche comme le montre la figure ci-dessous. Dans le réseau routier (à gauche de la figure), on cherche à se rendre de s à g , ce qui revient à parcourir l'arbre de recherche (à droite de la figure) dans lequel on observe qu'il y a 4 chemins différents pour se rendre de s à g : $[s, a, b, e, f, g]$ de 19 km, $[s, a, d, e, f, g]$ (17 km), $[s, d, a, b, e, f, g]$ (25 km) et $[s, d, e, f, g]$ (13 km), les autres chemins étant des impasses dans ce contexte.



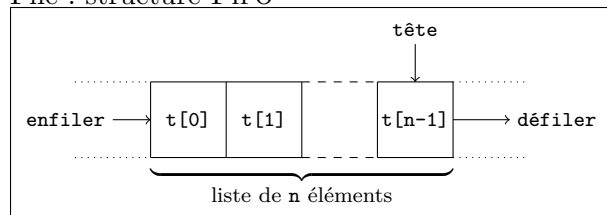
On utilisera donc des méthodes de parcours d'arbre pour ce type de recherche (recherche en profondeur, en largeur ou du meilleur chemin). Dans un premier temps, nous introduirons un certain nombre de notions (piles, files, graphes) qui seront utiles à la recherche d'un chemin dans un graphe.

12.1.1 Piles et files

Pile : structure LIFO



File : structure FIFO



Q 12.1 (« tout en un » : piles) Une pile (structure LIFO : last in, first out) sera représentée ici par une liste. On empilera et dépilera en fin de liste.

1. Définir la fonction `pileVide` qui teste si une pile `p` est vide (`True`) ou non (`False`).

```
>>> pileVide([])
True
```

```
>>> pileVide([1,2,3])
False
```

2. Définir la fonction `sommet` qui retourne le sommet `s` d'une pile `p` non vide.

```
>>> sommet([1,2,3])
3
```

```
>>> sommet([])
assert not pileVide(p)
AssertionError
```

3. Définir la fonction `empiler` qui empile un élément `e` au sommet d'une pile `p`.

```
>>> p = [1,2,3]
>>> empiler(p,7)
>>> p
[1, 2, 3, 7]
```

```
>>> p = []
>>> empiler(p,7)
>>> p
[7]
```

4. Définir la fonction `depiler` qui retourne le sommet `s` d'une pile `p` non vide après l'avoir dépilée.

```
>>> p = [1,2,3]
>>> depiler(p)
3
>>> depiler(p)
2
```

```
>>> depiler(p)
1
>>> depiler(p)
assert not pileVide(p)
AssertionError
```

Q 12.2 (« tout en un » : files) Une file (structure FIFO : first in, first out) sera représentée ici par une liste. On enfilera en début de liste et défilera en fin de liste.

1. Définir la fonction `fileVide` qui teste si une file `f` est vide (`True`) ou non (`False`).

```
>>> fileVide([])
True
```

```
>>> fileVide([1,2,3])
False
```

2. Définir la fonction `tete` qui retourne la tête `t` d'une pile `f` non vide.

```
>>> tete([1,2,3])
3
```

```
>>> tete([])
assert not fileVide(f)
AssertionError
```

3. Définir la fonction `enfiler` qui enfle un élément `e` dans une file `f`.

```
>>> f = [1,2,3]
>>> enfiler(f,7)
>>> f
[7, 1, 2, 3]
```

```
>>> f = []
>>> enfiler(f,7)
>>> f
[7]
```

4. Définir la fonction *defiler* qui retourne la tête *t* d'une file *f* non vide après l'avoir défilée.

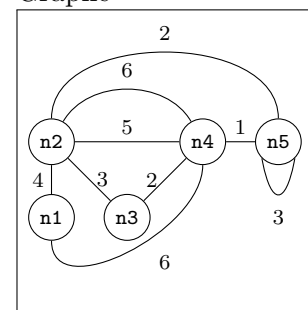
```
>>> f = [1,2,3]
>>> defiler(f)
3
>>> defiler(f)
2
```

```
>>> defiler(f)
1
>>> defiler(f)
assert not fileVide(f)
AssertionError
```

12.1.2 Graphes

Un graphe peut être vu comme un ensemble de nœuds reliés entre eux par des arcs comme dans l'exemple ci-contre. Les nœuds et les arcs peuvent être étiquetés. Pour les arcs, l'étiquette est appelée le poids de l'arc : l'arc qui relie ci-contre les nœuds *n1* et *n4* a un poids de 6. Un graphe peut avoir des arcs multiples : plusieurs arcs différents relient la même paire de nœuds comme c'est le cas pour les nœuds *n2* et *n4* ci-contre. Un arc peut ne relier qu'un nœud à lui-même comme la boucle sur le nœud *n5* avec un poids de 3.

Graphe



Q 12.3 (« tout en un » : graphes)

1. Définir la fonction *Arc* qui teste si un arc du graphe est bien un triplet (*n1*,*n2*,*p12*) où *n1* et *n2* sont 2 nœuds du graphe et *p12* le poids de l'arc qui relie ces 2 nœuds.

```
>>> Arc(('a','b',5))
True
>>> Arc([1,2],[3,4],'p')
True
```

```
>>> Arc(('a','b'))
False
>>> Arc(['a','b',5])
False
```

2. Définir la fonction *Graphe* qui teste si un graphe est une liste d'arcs (*True*) ou non (*False*).

```
>>> g = [('n1','n1',4),('n1','n2',3)]
>>> Graphe(g)
True
```

```
>>> Graphe([1,2,3])
False
>>> Graphe(('n1','n2',3))
False
```

3. Définir la fonction *adjacents* qui teste si 2 nœuds *n1* et *n2* d'un graphe *g* sont reliés par un arc du graphe (*True*) ou non (*False*).

```
>>> g = [('n1','n2',4),('n2','n3',3)]
>>> adjacents('n1','n2',g)
True
>>> adjacents('n1','n3',g)
False
>>> adjacents('n1','n4',g)
False
```

```
>>> adjacents('n2','n1',g)
True
>>> adjacents('n2','n3',g)
True
>>> adjacents('n3','n1',g)
False
>>> adjacents('n3','n2',g)
True
```

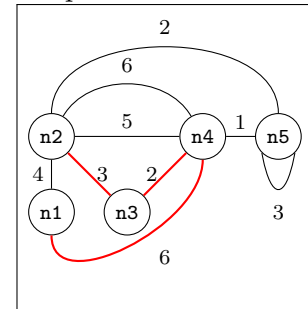
4. Définir la fonction *listeAdjacents* qui retourne la liste de tous les nœufs adjacents à un nœud *n* du graphe *g*.

```
>>> g = [('n1','n2',4),('n2','n3',3)]
>>> listeAdjacents('n1',g)
['n2']
>>> listeAdjacents('n4',g)
[]
```

```
>>> listeAdjacents('n2',g)
['n1', 'n3']
>>> listeAdjacents('n3',g)
['n2']
```

Un chemin entre deux nœuds d'un graphe est un doublet composé d'une liste de nœuds successivement 2 à 2 adjacents (une « route ») et de la somme cumulée des poids des arcs successifs qui relient ces nœuds (la « distance »). Dans le graphe ci-contre le chemin en rouge est un des chemins possibles pour « aller » de 'n2' à 'n1' sans passer deux fois par le même nœud : il passe successivement par 'n2', 'n3', 'n4' et 'n1' pour un coût cumulé de $3 + 2 + 6 = 11$, d'où le doublet (distance,route) = (11,['n2','n3','n4','n1']).

Graphe



Q 12.4 (« tout en un » : chemins dans un graphe)

1. Définir la fonction *Chemin* qui teste si un chemin est bien un doublet (distance,route) où distance est un nombre et route une liste non vide de nœuds.

```
>>> Chemin((7,['n1','n2','n3']))
True
```

```
>>> Chemin((7,[]))
False
```

2. Définir la fonction *extremite* qui retourne le dernier nœud de la route (liste des nœuds) maintenue dans un chemin.

```
>>> extremite((7,['n1','n2','n3']))
'n3'
```

```
>>> extremite((7,[]))
assert Chemin(chemin)
AssertionError
```

3. Définir la fonction *cheminsSuivants* qui retourne la liste des chemins possibles à partir du nœud extrémité d'un chemin d'un graphe donné sans repasser par l'un des nœuds du chemin.

```
>>> g = [('n1','n2',4),('n2','n3',3),\
        ('n1','n4',6),('n2','n4',5),\
        ('n2','n4',6)]
>>> cheminsSuivants((0,['n1']),g)
[(4, ['n1', 'n2']), (6, ['n1', 'n4'])]
>>> cheminsSuivants((6,['n1','n4']),g)
[(11, ['n1', 'n4', 'n2']),
 (12, ['n1', 'n4', 'n2'])]
```

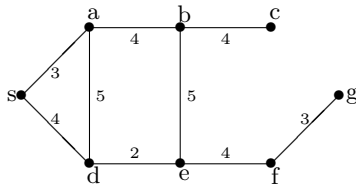
```
>>> cheminsSuivants((4,['n1','n2']),g)
[(7, ['n1', 'n2', 'n3']),
 (9, ['n1', 'n2', 'n4']),
 (10, ['n1', 'n2', 'n4'])]
>>> cheminsSuivants((7,['n1','n2','n3']),g)
[]
>>> cheminsSuivants((9,['n1','n2','n4']),g)
[]
```

12.1.3 Recherches de chemins dans un graphe

Le principe de la recherche d'un chemin dans un graphe nécessite de stocker les chemins partiels déjà explorés dans un tableau afin de vérifier qu'on ne passe pas deux fois par le même nœud. On initialise le tableau avec le chemin (0,['depart']) : on est situé sur le nœud de départ et on ne s'est pas encore déplacé dans le graphe. A chaque étape, on choisit un élément de ce tableau et si l'extrémité du chemin ainsi choisi est le nœud d'arrivée, alors ce chemin est une solution du problème. Si l'extrémité du chemin ne correspond pas au nœud d'arrivée, on remplace ce chemin par l'ensemble de ses successeurs dans le graphe (voir *cheminsSuivants* de la question 12.4 précédente) et on recommence ainsi de suite jusqu'à ce que le tableau de stockage soit vide.

Différentes méthodes peuvent être mises en œuvre selon la procédure utilisée pour stocker, choisir et remplacer les chemins partiels dans le tableau de stockage. Dans ce qui suit, on distinguera trois méthodes : la recherche en profondeur qui utilise une pile, la recherche en largeur qui utilise une file et la recherche du meilleur chemin qui utilise une pile (ou une file) triée à chaque étape.

Pour tester les différents algorithmes de recherche, on utilisera le réseau routier ci-dessous.

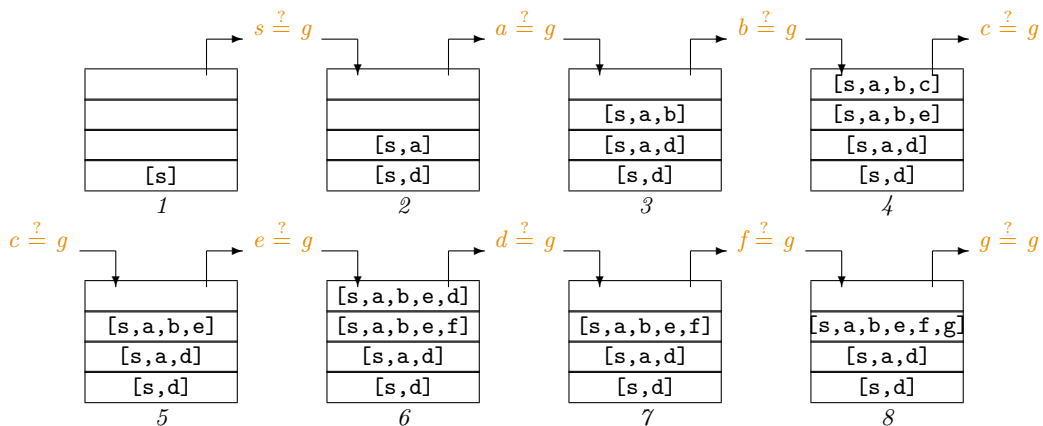


```

graphe = [('s','d',4),
          ('s','a',3),
          ('d','e',2),
          ('d','a',5),
          ('a','b',4),
          ('b','c',4),
          ('b','e',5),
          ('e','f',4),
          ('f','g',3)]

```

Q 12.5 (« tout en un » : recherche en profondeur) La recherche en profondeur consiste à stocker les chemins dans une pile comme l'illustre la figure ci-dessous.



1. Définir ainsi la fonction *profondeur* qui retourne la liste de tous les chemins qui mènent d'un nœud *depart* à un nœud *arrivee* dans un graphe *g*.

```

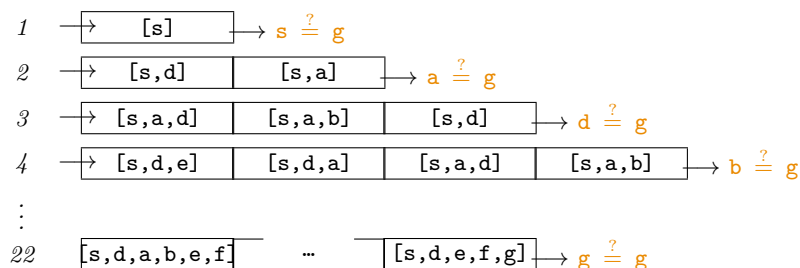
>>> profondeur('s','g',graphe)
[(19, ['s', 'a', 'b', 'e', 'f', 'g']),
 (17, ['s', 'a', 'd', 'e', 'f', 'g']),
 (25, ['s', 'd', 'a', 'b', 'e', 'f', 'g']),
 (13, ['s', 'd', 'e', 'f', 'g'])]

>>> profondeur('d','b',graphe)
[(9, ['d', 'a', 'b']),
 (7, ['d', 'e', 'b']),
 (11, ['d', 's', 'a', 'b'])]

```

2. Justifier pourquoi cette méthode est appelée recherche en profondeur.

Q 12.6 (« tout en un » : recherche en largeur) La recherche en largeur consiste à stocker les chemins dans une file comme l'illustre la figure ci-dessous.



1. Définir ainsi la fonction *largeur* qui retourne la liste de tous les chemins qui mènent d'un nœud *depart* à un nœud *arrivee* dans un graphe *g*.

```
>>> largeur('s','g',graphe)
[(13, ['s', 'd', 'e', 'f', 'g']),
 (19, ['s', 'a', 'b', 'e', 'f', 'g']),
 (17, ['s', 'a', 'd', 'e', 'f', 'g']),
 (25, ['s', 'd', 'a', 'b', 'e', 'f', 'g'])]

>>> largeur('d','b',graphe)
[(9, ['d', 'a', 'b']),
 (7, ['d', 'e', 'b']),
 (11, ['d', 's', 'a', 'b'])]
```

2. Justifier pourquoi cette méthode est appelée recherche en largeur.

Q 12.7 (« tout en un » : recherche du meilleur chemin) La recherche du meilleur chemin consiste à stocker les chemins dans une pile (ou file) et à trier cette pile (ou file) à chaque étape.

1. Définir ainsi la fonction *meilleur* qui retourne la liste de tous les chemins qui mènent d'un nœud *depart* à un nœud *arrivee* dans un graphe *g*.

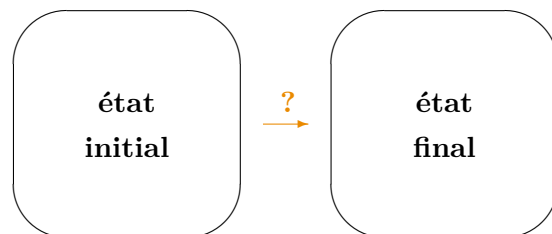
```
>>> meilleur('s','g',graphe)
[(13, ['s', 'd', 'e', 'f', 'g']),
 (17, ['s', 'a', 'd', 'e', 'f', 'g']),
 (19, ['s', 'a', 'b', 'e', 'f', 'g']),
 (25, ['s', 'd', 'a', 'b', 'e', 'f', 'g'])]

>>> meilleur('d','b',graphe)
[(7, ['d', 'e', 'b']),
 (9, ['d', 'a', 'b']),
 (11, ['d', 's', 'a', 'b'])]
```

2. Justifier pourquoi cette méthode garantit que les chemins sont obtenus par ordre croissant de la distance parcourue.

12.2 Généralisation

Soit un système dans un état initial donné. On veut le faire passer dans un état final donné connaissant les opérations élémentaires que l'on peut appliquer sur le système. Quelle(s) suite(s) d'opérations élémentaires doit-on appliquer au système pour le faire passer de l'état initial à l'état final ?



Dans le cas du réseau routier de la section 12.1.3 précédente, l'état initial est simplement le fait d'être en *s* et l'état final d'être en *g*. La seule opération élémentaire consiste à suivre une route qui part de la ville où l'on se trouve. Dans le cas d'un réseau routier réel, on imagine aisément que le nombre de possibilités pour aller d'une ville à une autre est considérable et qu'en conséquence l'arbre de recherche des solutions est immense : on parle alors d'« explosion combinatoire ».

Q 12.8 (« tout en un » : une ou plusieurs solutions) Modifier les fonctions *profondeur*, *largeur* et *meilleur* précédentes de telle manière qu'elles ne retournent qu'un nombre *n* maximum de solutions.

```
>>> meilleur('s','g',graphe,10)
[(13, ['s', 'd', 'e', 'f', 'g']),
 (17, ['s', 'a', 'd', 'e', 'f', 'g']),
 (19, ['s', 'a', 'b', 'e', 'f', 'g']),
 (25, ['s', 'd', 'a', 'b', 'e', 'f', 'g'])]

>>> meilleur('s','g',graphe,1)
[(13, ['s', 'd', 'e', 'f', 'g'])]

>>> meilleur('s','g',graphe,2)
[(13, ['s', 'd', 'e', 'f', 'g']),
 (17, ['s', 'a', 'd', 'e', 'f', 'g'])]
```

Toujours dans le cas d'un réseau routier, le graphe du réseau est explicite : il est connu à l'avance et on peut ainsi le « passer » à la fonction de recherche. Mais il existe des problèmes où ce n'est pas le cas : il suffit de penser au « Rubik's cube » et ses 43 trillions (milliards de

milliards : 10^{18}) de combinaisons possibles ! Dans ces cas là, plutôt que de passer explicitement le graphe à la fonction de recherche, on lui transmet la fonction qui, à partir d'un nœud donné, applique les différentes opérations élémentaires autorisées pour obtenir les nœuds adjacents et ainsi construire le graphe au fur et à mesure des étapes.

Q 12.9 (« tout en un » : explicite versus implicite)

1. Définir la fonction **reseau** qui retourne les arcs issus d'un nœud du réseau routier de la section 12.1.3 précédente.

```
>>> reseau('a')
[('s', 'a', 3),
 ('d', 'a', 5),
 ('a', 'b', 4)]

>>> reseau('s')
[('s', 'd', 4), ('s', 'a', 3)]
>>> reseau('c')
[('b', 'c', 4)]
```

2. Modifier les fonctions **profondeur**, **largeur** et **meilleur** précédentes afin de leur transmettre la fonction **f** qui retourne les arcs issus d'un nœud donné.

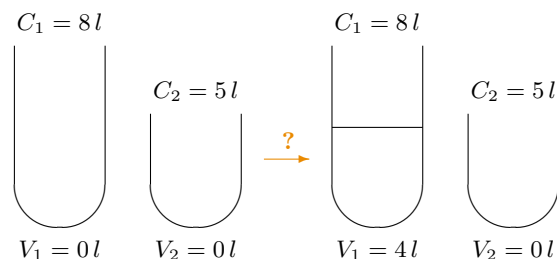
```
>>> meilleur('s','g',reseau,1)
[(13, ['s', 'd', 'e', 'f', 'g'])]

>>> meilleur('d','b',reseau,1)
[(7, ['d', 'e', 'b'])]
```

12.3 Applications

Q 12.10 (« tout en un » : vases) Soit un système composé de deux récipients non gradués R_1 et R_2 , respectivement de capacité C_1 et C_2 . L'état du système sera représenté par la paire (V_1, V_2) caractérisant la quantité d'eau contenue dans chacun des récipients. On veut faire passer le système de l'état initial (V_1^i, V_2^i) à l'état final (V_1^f, V_2^f) sachant que les seules opérations élémentaires autorisées sont : remplir complètement un récipient, vider complètement un récipient ou transvaser un récipient dans l'autre sans perdre une seule goutte. Pour fixer les idées, on prendra $C_1 = 8l$, $C_2 = 5l$, $V_1^i = V_2^i = 0l$, $V_1^f = 4l$ et $V_2^f = 0l$.

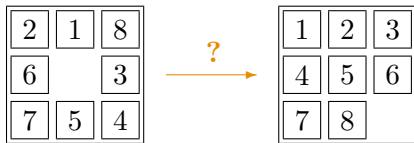
1. Définir la fonction **vases** qui applique les opérations élémentaires possibles (vider, remplir, transvaser) à un état donné (v_1, v_2) .
2. Trouver les 5 meilleures suites d'opérations élémentaires pour passer de l'état $(0, 0)$ à l'état $(4, 0)$.



```
>>> vases((0,0))
[[(0, 0), (0, 5), 1), ((0, 0), (8, 0), 1)]
>>> vases((8,0))
[[(8, 0), (3, 5), 1), ((8, 0), (8, 5), 1), ((8, 0), (0, 0), 1)]
>>> vases((0,5))
[[(0, 5), (5, 0), 1), ((0, 5), (8, 5), 1), ((0, 5), (0, 0), 1)]

>>> meilleur((0,0),(4,0),vases,5)
[(12, [(0, 0), (0, 5), (5, 0), (5, 5), (8, 2), (0, 2), (2, 0), (2, 5), (7, 0), (7, 5), (8, 4), (0, 4), (4, 0)]),
 (13, [(0, 0), (8, 0), (3, 5), (3, 0), (0, 3), (8, 3), (6, 5), (6, 0), (1, 5), (1, 0), (0, 1), (8, 1), (4, 5), (4, 0)]),
 (14, [(0, 0), (8, 0), (3, 5), (0, 5), (5, 0), (5, 5), (8, 2), (0, 2), (2, 0), (2, 5), (7, 0), (7, 5), (8, 4), (0, 4), (4, 0)]),
 (14, [(0, 0), (8, 0), (8, 5), (0, 5), (5, 0), (5, 5), (8, 2), (0, 2), (2, 0), (2, 5), (7, 0), (7, 5), (8, 4), (0, 4), (4, 0)]),
 (15, [(0, 0), (0, 5), (5, 0), (8, 0), (3, 5), (3, 0), (0, 3), (8, 3), (6, 5), (6, 0), (1, 5), (1, 0), (0, 1), (8, 1), (4, 5), (4, 0)])]
```

Q 12.11 (« tout en un » : taquin) *Le taquin est un jeu en forme de damier ($n \times n$). Il est composé de $(n^2 - 1)$ petits carreaux numérotés de 1 à $(n^2 - 1)$ qui glissent dans un cadre prévu pour n^2 carreaux. Il consiste à remettre dans l'ordre les $(n^2 - 1)$ carreaux à partir d'une configuration initiale quelconque.*



taquin (3×3)

Le carreau vide est numéroté 0 :

état initial : $[[2,1,8], [6,0,3], [7,5,4]]$

↓ ?

état final : $[[1,2,3], [4,5,6], [7,8,0]]$

1. Définir la fonction **taquin** qui applique les opérations élémentaires possibles à partir d'une configuration donnée pour un damier ($n \times n$).

```
>>> jeu = [[3,1],[0,2]]
>>> taquin(jeu)
([([3, 1], [0, 2]), ([0, 1], [3, 2]), 1),
 ([3, 1], [0, 2]), ([3, 1], [2, 0]), 1])
```

```
>>> jeu = [[2,1,8],[6,0,3],[7,5,4]]
>>> taquin(jeu)
([([2, 1, 8], [6, 0, 3], [7, 5, 4]), ([2, 0, 8], [6, 1, 3], [7, 5, 4]), 1),
 ([2, 1, 8], [6, 0, 3], [7, 5, 4]), ([2, 1, 8], [6, 5, 3], [7, 0, 4]), 1),
 ([2, 1, 8], [6, 0, 3], [7, 5, 4]), ([2, 1, 8], [0, 6, 3], [7, 5, 4]), 1),
 ([2, 1, 8], [6, 0, 3], [7, 5, 4]), ([2, 1, 8], [6, 3, 0], [7, 5, 4]), 1])
```

2. Trouver la meilleure suite d'opérations élémentaires pour passer de l'état $[[3,1],[0,2]]$ à l'état $[[1,2],[3,0]]$ dans un taquin (2×2).

```
>>> depart = [[3,1],[0,2]]
>>> arrivee = [[1,2],[3,0]]
>>> meilleur(depart,arrivee,taquin,10)
(3, ([([3, 1], [0, 2]), ([0, 1], [3, 2]), ([1, 0], [3, 2]), ([1, 2], [3, 0])]),
 (9, ([([3, 1], [0, 2]), ([3, 1], [2, 0]), ([3, 0], [2, 1]), ([0, 3], [2, 1]),
      ([2, 3], [0, 1]), ([2, 3], [1, 0]), ([2, 0], [1, 3]), ([0, 2], [1, 3]),
      ([1, 2], [0, 3]), ([1, 2], [3, 0])]))
```

3. Trouver la meilleure suite d'opérations élémentaires pour passer de l'état $[[2,1,8],[6,0,3],[7,5,4]]$ à l'état $[[1,2,3],[4,5,6],[7,8,0]]$ dans un taquin (3×3).

```
>>> depart = [[2,1,8],[6,0,3],[7,5,4]]
>>> arrivee = [[1,2,3],[4,5,6],[7,8,0]]
>>> meilleur(depart,arrivee,taquin,1)
...
```

Ce dernier exemple, qui n'en finit pas de calculer, du taquin (3×3) illustre clairement le problème évoqué plus haut de l'« explosion combinatoire ». Il révèle ainsi la limite des stratégies de recherche développées ici (profondeur, largeur, meilleur chemin). Il faudrait en effet envisager de nouvelles stratégies, moins « aveugles » et mieux « informées » du problème considéré. Mais l'étude de telles stratégies dépasse largement la cadre d'une simple initiation à l'algorithmique.

Liste des questions

2.1	« aller au restaurant » : itinéraire	8
2.2	« aller au restaurant » : vérification	8
2.3	algorithme : validité	9
2.4	algorithme : robustesse	9
2.5	algorithme : réutilisabilité	9
2.6	algorithme : complexité	9
2.7	algorithme : efficacité	9
2.8	algorithme : tracés de polygones réguliers	9
2.9	algorithme : propriétés	10
3.1	« meuble à tiroirs » : désignation des tiroirs	15
3.2	« meuble à tiroirs » : échange de contenus	15
3.3	affectation : nommer les variables	15
3.4	affectation : constantes	15
3.5	affectation : expressions	15
3.6	affectation : incrémentation	15
3.7	affectation : égalité mathématique ?	16
3.8	affectation : opération commutative ?	16
3.9	affectation : fonctionnement	16
3.10	unité de longueur	16
3.11	permutation circulaire	16
3.12	exécution d'une séquence d'affectations	16
4.1	« circuit logique » : combinatoire	20
4.2	« circuit logique » : table de vérité	20
4.3	« circuit logique » : vérification	20
4.4	opérateurs booléens : distributivité	21
4.5	opérateurs booléens : De Morgan	21
4.6	opérateurs booléens : opérateurs dérivés	22
4.7	développer une expression booléenne	22
4.8	fonctions booléennes binaires	22
5.1	« base bibi » : décodage	25
5.2	« base bibi » : codage	25
5.3	codage binaire d'un entier positif	26
5.4	nombres relatifs	26
5.5	complément à 2	26
5.6	base D'NI	26
6.1	« mentions au bac » : graphe	30
6.2	« mentions au bac » : alternative multiple	30
6.3	alternatives : tests ou alternative ?	30
6.4	alternatives : multiples ou imbriquées ?	30
6.5	catégories sportives	31
6.6	prix d'une photocopie	31
6.7	exécution d'une séquence de tests	31
7.1	« planter un clou » : initialisation	36
7.2	« planter un clou » : progression	36
7.3	« planter un clou » : invariant	36
7.4	« planter un clou » : condition d'arrêt	36

7.5	« planter un clou » : algorithme	36
7.6	construction d'une boucle : initialisation et invariant	37
7.7	construction d'une boucle : condition d'arrêt et invariant	37
7.8	construction d'une boucle : cas général	37
7.9	construction d'une boucle : cas simplifié	37
7.10	construction d'une boucle : fonction puissance	37
7.11	construction d'une boucle : fonction factorielle	37
7.12	construction d'une boucle : fonction pgcd	38
7.13	construction d'une boucle : fonction Fibonacci	38
8.1	« ranger le bois » : les rangées	42
8.2	« ranger le bois » : une rangée	42
8.3	« ranger le bois » : le stère	42
8.4	boucles imbriquées : exécutions	43
8.5	boucles imbriquées : imbriquées ou successives ?	43
8.6	boucles imbriquées : tables de vérité	43
8.7	boucles imbriquées : tables de multiplication	43
8.8	boucles imbriquées : triangle de Pascal	43
9.1	« quatre-quarts » : situations initiale et finale	48
9.2	« quatre-quarts » : réalisation	49
9.3	spécification d'une fonction : nommer	49
9.4	spécification d'une fonction : paramétrer	49
9.5	spécification d'une fonction : protéger	49
9.6	spécification d'une fonction : tester	49
9.7	prix d'une photocopie	50
9.8	somme arithmétique	50
10.1	« tours de Hanoï » : à la main	53
10.2	« tours de Hanoï » : relation de récurrence	53
10.3	récursivité : exécution d'une fonction récursive	54
10.4	récursivité : fonction factorielle	54
10.5	récursivité : plus grand commun diviseur	54
10.6	récursivité : récursivité \rightarrow itération	54
10.7	récursivité : fonction d'Ackerman	55
10.8	récursivité : coefficients du binôme	55
10.9	récursivité : flocons de Koch	55
11.1	« jeu de cartes » : tri par insertion	59
11.2	« jeu de cartes » : complexité	59
11.3	relation d'ordre	60
11.4	tri par sélection	60
11.5	tri rapide	61
11.6	tri bulles	61
11.7	tri fusion	61
12.1	« tout en un » : piles	65
12.2	« tout en un » : files	65
12.3	« tout en un » : graphes	66
12.4	« tout en un » : chemins dans un graphe	67
12.5	« tout en un » : recherche en profondeur	68
12.6	« tout en un » : recherche en largeur	68
12.7	« tout en un » : recherche du meilleur chemin	69

12.8 « tout en un » : une ou plusieurs solutions	69
12.9 « tout en un » : explicite versus implicite	70
12.10 « tout en un » : vases	70
12.11 « tout en un » : taquin	71

Références

- [1] Tisseau J., *Initiation à l'algorithmique. Cours*, ENIB, 2009-2014
- [2] Tisseau J., *Initiation à l'algorithmique. Travaux dirigés*, ENIB, 2009-2014
- [3] Tisseau J., Nédélec A., Parenthoën M., *Initiation à l'algorithmique. La démarche MVR : méthode, vérification, résultat*, ENIB, 2011-2014