Exercices de programmation

Table des matières

2.1 Énoncés

2.1.1 Introduction à la programmation en Python

Mode interactif

Exercice 2.1 (Opérateurs)

1. Quelle est la réponse de l'interpréteur après avoir entré les expressions suivantes ?
   - `2*3`
   - `2**3`
   - `20/3`
   - `20//3`
   - `20%3`
   - `11 == 132/12`
   - `2 <= 8 and 15.0 != 15`
2. Quel est le rôle de l'opérateur `&` sur des entiers ? (Essayer avec de petits opérandes et conclure) Même question pour les opérandes `|`, `^`, `<<` et `>>` (Indication : penser à l'écriture binaire).
3. Taper le calcul `111**2` puis `_**2`. Quel est le rôle du caractère de soulignement `_` ?

Exercice 2.2 (Types)

1. Quel est le résultat du calcul `1-1/3-2/3` ? Expliquer ce résultat (en particulier, dire son type).
2. Exécuter les calculs `88**99` puis `88.1**99`. Commenter ces résultats (mentionner leur type).
3. Taper le calcul `2**1000` suivi de `_*1.` puis en déduire l'effet du point `.` derrière `1` ?
4. Quel est le type des expressions suivantes (autocorrection avec la fonction `type()` ?
   - `123/3`
   - `123//3`
   - `123%3`
   - `123.%3`
   - `123>3`
   - `"123"+"3"`

Exercice 2.3 (Built-in Python)

1. Exécuter `int(4.2)`, `int(-4.2)`.
   Quel est l'effet de la fonction `int()` sur un flottant ?
   Exécuter `int("57")`, `int("110",2)` et `int("ff",16)`.
   Quel est l'effet de la fonction `int("n",p)` où `n` et `p` sont des entiers ?
2. Exécuter `float(4)`, `float(2**100)`.
   Quel est l'effet de la fonction `float()` sur un grand entier ?
3. Que donne `round(5/6,7)` ? `round(5/6)` ? `round(5/6,0)` ? `round(1000/3,-1)` ?
   Lire l'aide de cette fonction en tapant `help(round)`.
4. Que donne `max(1,-2,5)` ? `max([1,-2,5])` (liste) ? `max({1,-2,5})` (ensemble) ?
   Comparer `min("b","a","c")`, `min("B","a","c")`, `min("BAC")` et `min("Bac","BAC")`.
   Comment sont rangés les caractères pour Python ?
5. Examiner les résultats de ces quelques autres fonctions pré-programmées (liste non exhaustive) :
   - Conversions : `bin(2**10-1)`, `hex(125)`, `oct(9)` et `str(5)`
     Voir aussi les fonctions `ord` et `chr` : `ord("H")`, `ord("h")` et `chr(35)`
   - Affichage : `print("Il y a",3,"jours")` et `print("Il y a"+str(3)+"jours")`
     Lire l'aide `help(print)`, pour les options `sep` et `end` en particulier.
   - Tri de liste : `sorted([1,5,2,8,3])`, `sorted(["H","T","M","L])` et `sorted("HTML")`
   - Longueur de liste : `len([1,5,2,8,3])`, `len('abcd')`, `len(range(1,10))`

Exercice 2.4 (Module math)

1. Qu'obtient-on en tapant `cos(0)` ?
   Importer le module `math` (avec `import math`) puis réessayer.
   Le module `math` étant importé (avec `import math`), comment obtenir la valeur de $cos(0)$?
   Comment importer la fonction `cos` du module `math` pour que `cos(0)` soit valide ?
2. Quelle est la valeur approchée de la constante `pi` ?
   Y a t-il une autre constante dans le module `math` ?
3. Quel est le résultat des calculs suivants (après avoir écrit `from math import *`) ?
   - `floor(4.2)` et `floor(-4.2)` (comparer avec `int(-4.2)`)
   - `ceil(4.2)` (comparer avec `int(4.2)`) et `ceil(-4.2)`
   - `sqrt(49)` et `sqrt(2)` (comparer leur type)
   - `pow(2,5)` (comparer avec `2**5`) ; `pow(2,0.5)` (comparer avec `2**0.5` et `sqrt(2)`)
   - `degrees(pi)`, `radians(90)`, `asin(0.5)` et `sin(pi/6)`
     Quelle unité utilise Python pour les fonctions trigonométriques ?

Exercice 2.5 (Autres modules)

1. Importer le module `random` et examiner (avec la fonction `help`) les fonctions :
   - `randint` : que donne `randint(2,5)` ?
   - `random` : que donne `random()` ?
   - `randrange` : que donne `randrange(2,5)` ? et `randrange(1,6,2)` ?
   - `uniform` : que donne `uniform(0.5,0.8)` ?
2. Importer le module `time` puis taper `time.perf_counter()` à plusieurs reprises.
   Quel est le principe de cette fonction `perf_counter` ?
3. Importer le module `datetime` puis taper `datetime.date.today()` pour savoir la date du jour.
   Taper `datetime.datetime.now()` pour savoir le jour et l'heure actuelle.
4. Importer le module `calendar` puis taper `calendar.weekday(2019,9,2)`.
   Comment sont codés les jours de la semaine ?
5. Importer le module `turtle` puis taper `turtle.circle(100,360)`. Que fait cette fonction ?
   NB : Comme pour les précédents modules, de nombreuses autres fonctions sont disponibles...

Exercice 2.6 (Variables)
1. On exécute successivement les instructions `a=3`, `b=a*a` et `a=b-2`.
   Que contiennent les variables `a` et `b` ?
2. Sachant que les variables `a` et `b` contiennent des nombres, on exécute successivement :
   `c=a`, `a=b`, `b=c`
   Quel peut être le but de ces instructions ? Proposer une meilleure solution (sur une seule ligne).
3. Sachant que les variables `a` et `b` contiennent des nombres, on exécute successivement :
   `a=a+b`, `b=a-b`, `a=a-b`
   Quel peut être le but de ces instructions ?
4. On exécute successivement les instructions
   `a=input("Entrer un nombre :")`, `print("Le double de ce nombre est",a*2)`.
   Que produit la 2e instruction ? Proposer une correction.
5. Écrire sur deux lignes les instructions demandant le rayon d'un cercle et affichant son périmètre.
6. Écrire sur trois lignes les instructions demandant deux nombres entiers,
   le 1er étant compris comme une base < 36, le 2e étant un nombre dans cette base,
   et affichant la conversion du 2e nombre en base 10 (fonction à utiliser dans l'exercice 2.3).

Mode de programmation

Exercice 2.7 (Entrées-sorties)
1. Dans l'éditeur de programme, entrer les trois lignes de programme nécessaires pour demander la longueur et la largeur d'un rectangle (en mètres) et qui en affiche le périmètre (en mètres) et l'aire (en mètres carrés). Pour l'affichage sur deux lignes, utiliser le caractère `\n`.
   - Enregistrer ce programme (le nommer « rectangle1.py ») dans un dossier « Programmes » du dossier « Documents ».
   - Exécuter ce programme (Exécuter>démarrer le script dans Pyzo) ; entrer `12` puis `15`.
2. Transformer le programme précédent en une procédure (fonction sans valeur de retour), nommée « rectangle » qui prend deux nombres `a` et `b` en arguments et qui réalise l'affichage du périmètre et de l'aire. La 1re ligne de cette fonction s'écrit `def rectangle(a,b):`
   le reste de la fonction (ici, une seule ligne) est indenté.
   - Enregistrer ce nouveau programme (le nommer « rectangle2.py ») dans « Programmes ».
   - Exécuter ce programme. La fonction est maintenant connue de l'interpréteur Python : taper `rectangle(12,15)` dans la console pour vous en assurer.

2.1.2 Séquences conditionnelles et boucles

Séquences conditionnelles

Exercice 2.8 (Min, Max & Étendue)
Le programme ci-dessous détermine le minimum et le maximum de la distance à l'origine de `n=100` points dont les coordonnées sont tirées au hasard entre 0 et 1. Compléter les deux endroits laissés en pointillés, puis tester.

```
from math import sqrt
from random import random
n,min,max=100,.....,.....
for i in range(n):
  x,y=random(),random()
  d=sqrt(x**2+y**2)
  if ................
print('min=',min,'max=',max,'étendue=',max-min)
```

Exercice 2.9 (Procédures conditionnelles)

1. Écrire une procédure « boite » qui prend deux nombres `a` et `b` comme arguments et qui affiche le nombre de boîtes contenant `b` objets, nécessaires au rangement de `a` objets.
   Enregistrer ce programme, puis tester avec : `(a,b)=(49,10)`, `(a,b)=(50,10)`
2. Écrire une procédure « trinome » qui prend trois nombres `a`, `b` et `c` comme arguments et qui affiche la ou les solutions de l'équation $ax^2+bx+c=0$ quand elle(s) existe(nt) et qui affiche `"pas de solution"` quand il n'y en a pas.
   Enregistrer ce programme, puis tester avec :
   `(a,b,c)=(1,1,1)`, `(a,b,c)=(1,2,1)`, `(a,b,c)=(1,3,2)`
3. Écrire une procédure « intersection » qui prend quatre nombres `a1`, `b1`, `a2` et `b2` comme arguments et qui affiche, selon les cas, les coordonnées du point d'intersection des droites d'équations `y=a1*x+b1` et `y=a2*x+b2` ou un des messages suivants : `"pas d'intersection"` ou `"droites confondues"`.
   Enregistrer ce programme, puis tester avec :
   `(a1,b1,c1,d1)=(1,2,1,3)`, `(a1,b1,c1,d1)=(1,2,1,2)`,`(a1,b1,c1,d1)=(1,2,2,1)`

Exercice 2.10 (Âge)

1. Écrire une fonction « age » qui détermine l'âge d'une personne en entrant sa date de naissance sous la forme de trois nombres :`j` (jour), `m` (mois) et `a` (année).
   La fonction accède à la date du jour obtenue en important la fonction `date` du module `datetime` : le jour est donné par l'attribut `day` (écrire `date.today().day` ou mieux, si `d=date.today()` seulement `d.day`), le mois et l'année sont obtenus avec les attributs `month` et `year`. L'âge déterminé par cette fonction est le nombre entier d'années écoulées depuis la naissance, selon l'usage. La fonction retourne ce nombre si les arguments sont cohérents : `0<j<32`, `0<m<13`. La date entrée doit aussi être inférieure à la date du jour (écrire ce test à l'aide des opérateurs booléens). En cas d'incohérence retourner le mot-clé `None`.
2. Affiner la détection d'une date impossible si `j>28` en déterminant le nombre maximum de jours du mois, une variable notée `j_mois`. Confier cette détermination à une fonction « mois » qui prend en argument l'année et le mois et renvoie la longueur du mois entré.
3. Enregistrer et tester la fonction « age » avec des dates connues, puis avec des dates possibles ou impossibles (31 avril 2019, 29 février 2004, 29 février 2003, etc.).
4. Documenter votre script en ajoutant :
   - des commentaires sobres en fin de ligne (ce que fait la ligne, ce que contient une variable)
   - un entête au script contenant des informations (titre, objet, version, date, auteur)
   - un commentaire entre `"""..."""` après la 1re ligne de chaque fonction pour décrire la fonction. Ainsi rédigé, ce commentaire sera accessible dans la console, une fois le script exécuté, en tapant `help(age)` ou `help(mois)` (voir l'exemple 14, page 23 du cours)

Boucles

Exercice 2.11 (Suite de Syracuse)

Le successeur `s` d'un nombre entier `n` est déterminé par la règle de suivante :
Si `n` est pair alors `s=n//2`, sinon `s=n*3+1`
En partant d'un entier `n>0`, il s'avère qu'on arrive toujours à 1 (ce n'est qu'une conjecture).

1. Écrire une fonction qui affiche tous les successeurs de `n` jusqu'à 1.
2. Modifier la fonction précédente pour qu'elle affiche également, à la fin, la longueur `long` de la suite (nombre de termes) ainsi que la valeur maximum `maxi` qui a été atteinte.
3. Tester avec `n=15` (on trouve `maxi=160` et `long=17`). Déterminer `maxi` et `long` pour `n=127`.

Exercice 2.12 (Facteurs premiers)
Pour déterminer si un nombre entier `n` est premier, on peut essayer de le diviser par tous les entiers allant de 2 jusqu'à `p` $\geqslant \sqrt{n}$ (le premier entier qui vérifie cette inégalité convient) : si aucun de ces entiers ne le divise alors il est premier.

1. Écrire une fonction `premier` qui teste si un entier `n` est premier.
2. Modifier la fonction précédente pour qu'elle affiche les facteurs premiers de `n`.
   Avec cet algorithme naïf, la fonction `premier` s'appelle elle-même (pour déterminer si un diviseur de `n` est premier). Attention : les facteurs premiers d'un nombre peuvent apparaître plusieurs fois ! Exemple : $12 = 2 \times 2 \times 3$.
3. Soigner l'affichage pour obtenir la multiplicité d'un facteur sous la forme d'un exposant (pour `n=12` afficher par exemple `12=2^2×3`, pour `n=360` afficher `360=2^3×3^2×5`).

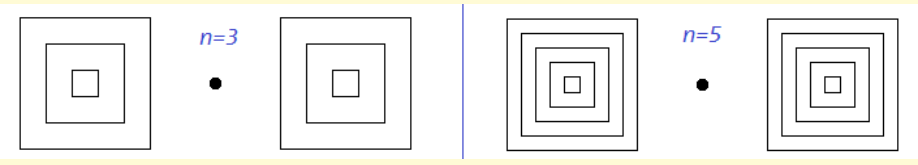
Exercice 2.13 (Conversion)
Pour convertir un nombre `n` en base `b` (avec $2 \leqslant$ `b` $\leqslant 36$), il suffit de diviser par `b`, autant de fois qu'il est possible. L'écriture en base `b` de ce nombre est la concaténation par la gauche des restes de ces divisions euclidiennes successives.

1. Écrire une fonction `convert(n,b)` qui effectue la conversion de `n` dans une base `b` $\leqslant 10$.
2. Modifier la fonction précédente pour qu'elle réalise cela avec une base $2 \leqslant$ `b` $\leqslant 36$. Pour cela, si `b>10` on peut utiliser la fonction `chr` qui donne la `p`$^{\text{e}}$ lettre de l'alphabet en écrivant `chr(96+p)` (par exemple `chr(96+1)` retourne `"a"`).
3. Améliorer la fonction précédente pour qu'elle effectue la conversion de `n` donné dans une base initiale `i` vers la base `b` (utiliser la fonction `int`). Attention : `n` pouvant contenir des lettres quand `i>10` doit être donné comme une chaîne de caractères.
   On peut rendre la base initiale optionnelle en donnant la valeur par défaut `convert(n,b,i=10)`

Tester la fonction `convert` à tous les stades de son évolution : `convert(2020,7)=5614` ; `convert(2020,16)='7e4'` ;`convert('7e4',10,16)=2020` ; `convert('7e4',36,16)='1k4'`.

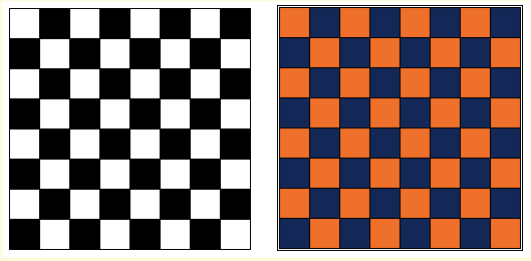
Exercice 2.14 (Pyramide 3D)
Après avoir entré `from turtle import *`, quel est le sens des instructions :
`goto(50,150)` ; `forward(100)` ; `left(90)` ; `down()` ; `up()` ;`dot(15)` ;`ht()`

1. Dessiner un carré de 100 pixels de côté dont le sommet inférieur gauche a pour coordonnées `(50,-50)`.
2. Automatiser cette construction en écrivant une fonction `carre(x,y,c)` qui trace un carré de `c` pixels de côté dont le sommet inférieur gauche a pour coordonnées `(x,y)`.
3. Dessiner trois carrés imbriqués (le 1$^{\text{er}}$de 100 px de côté, le 2$^{\text{e}}$ de 60 px, le 3$^{\text{e}}$ de 20 px) à gauche et à droite du point de coordonnées `(0,0)`, comme sur la figure ci-dessous à gauche.
4. Améliorer le programme pour créer `n` carrés imbriqués (figure de droite pour `n=5`).
   L'écart entre les carrés est calculé avec la formule `ecart=c//(2*(n-1)+1)` où `c` est le côté.



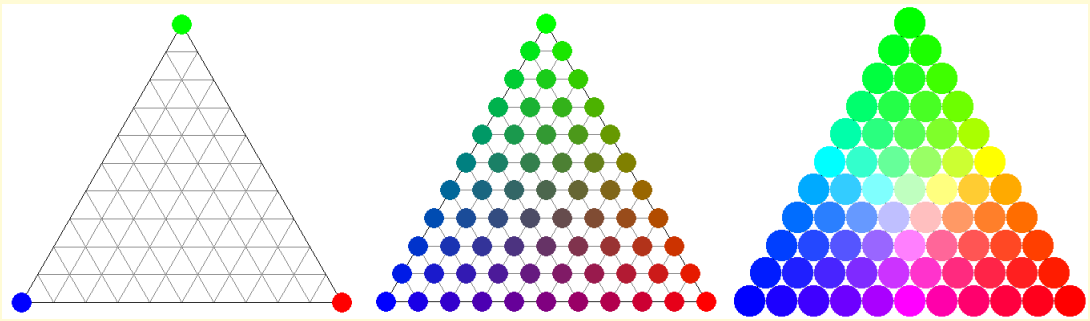
Exercice 2.15 (Damier)
La couleur du stylo (`pen`) de `turtle` peut être modifiée en utilisant un argument : `color('black','#ef702b')` colorie le bord d'un polygone avec la couleur noire et son intérieur avec la couleur codée en hexadécimal dans le système RGB (couleur mandarine).
Pour colorier l'intérieur, il faut en indiquer le début et la fin avec `begin_fill()` et `end_fill()`.

1. Écrire une fonction `carrePlein(x,y,c)` qui dessine un carré plein en recopiant la fonction `carre(x,y,c)` et en y ajoutant les instructions `begin_fill()` au début du tracé et `end_fill()` à la fin.
2. Dessiner à l'aide de la fonction `carrePlein` un damier de `2*n` cases, alternativement pleines et vides (ne dessiner que les pleines). Pour la bordure finale, utiliser la fonction `carre(x,y,c)`. La figure ci-dessous montre la réalisation pour `n=4`.
3. Améliorer le dessin en utilisant deux couleurs (exemple ci-dessous à droite).



Exercice 2.16 (Triangle des couleurs)
On veut visionner les couleurs RGB dans un triangle équilatéral de 400 pixels de côté, chaque sommet représentant une couleur primaire pure : à gauche Bleu, à droite Rouge, en haut Vert.

1. Tracer le triangle ainsi que les segments matérialisant un quadrillage triangulaire sur le modèle de la figure ci-dessous (à gauche). Placer un point de la bonne couleur à chaque sommet (utiliser le format RGB fréquence où le bleu est codé par `color(0.,0.,1.)`).
2. Écrire une fonction qui prend en argument deux fréquences (entre 0 et 1) correspondant aux pourcentages de rouge (`r`) et de vert (`v`) et qui place un point dont la couleur est donnée par (`r, v, 1-r-v`) à l'endroit correspondant sur la figure (en partant du point bleu, parcourir les `r%` du segment qui va jusqu'au point rouge, puis tourner de 60° et parcourir les `v%` du segment qui va vers le point vert). Noter que la couleur RGB fabriquée de cette manière a toujours une somme égale à 255, ce qui explique par exemple que le jaune n'est pas très jaune : (125, 125, 0) au lieu de (255, 255, 0) pour un jaune pur.
3. Á l'aide de deux boucles imbriquées, placer un point de la bonne couleur à chaque intersection des segments de la figure (figures du milieu).
4. Pour donner aux couleurs leur brillance maximum, passer en mode RGB classique avec `colormode(255)` et appliquer un coefficient de proportionnalité tel que la couleur ayant la fréquence maximum passe à 255 (figure de droite).



Exercice 2.17 (Opérations cryptées)
C'est un classique des jeux avec les nombres qui fut popularisé par un casse-tête que Henry Dudeney (1857-1930) publia dans le Strand Magazine en 1924. L'idée est d'écrire une opération dont les chiffres sont codés par des lettres, chaque lettre représentant un chiffre différent, les lettres formant des mots qui donnent un second sens à l'affirmation. D'une façon générale, un « bon » problème n'admet qu'une seule solution.

1. Dans la version de Dudeney, un jeune homme dans le besoin télégraphie à son père « SEND MORE MONEY ». L'addition SEND+MORE=MONEY n'admettant qu'une seule solution, c'est un bon problème à 7 variables (S, E, N, D, M, O, R, Y). Prouver cela en écrivant un programme qui essaie toutes les combinaisons possibles (voir l'exemple 7 du cours) et affiche toutes les solutions.
2. Dans le même esprit, voici une multiplication en français : « CINQ×SIX=TRENTE » qui peut être traitée comme une multiplication cryptée à 9 variables (C, I, N, Q, S, X, T, R, E). Programmer la recherche des solutions éventuelles et montrer que c'est un bon problème.
3. En voici une qu'on ne trouve pas sur internet : « PHILIPPE+MOUTOU=PILEPOIL ». Cette addition cryptée a également 9 variables (P, H, I, L, E, M, O, U, T) mais ce n'est pas un bon problème dans le sens où il a 4 solutions différentes. Le prouver.
4. Ce problème à 6 variables : « NEUF+UN+UN=ONZE » est-il un bon problème ?

Exercice 2.18 (Exploration numérique)
Définissons une règle de succession : on part d'un nombre de trois chiffres `abc` ; on remplace ce nombre par `bce` où `e` est le chiffre des unités de `a+c` ; on recommence le processus en partant cette fois de `bce`, ... et on ne s'arrête que lorsque le nombre obtenu est égal au nombre de départ.

1. Programmer une fonction qui détermine et retourne le successeur d'un nombre `abc`. Pour cela, on peut convertir le nombre en chaîne de caractère et utiliser la méthode de *slicing* : si `n="123"` alors `n[-1]` contient `"3"`, `n[:-1]` contient `"12"`, `n[1:]` contient `"23"`.
2. Compléter le script précédent en écrivant une fonction qui prend un nombre de trois chiffres `abc` comme argument et qui affiche la succession des nombres obtenus avec la règle de succession, la longueur de la succession, le minimum et le maximum qui ont été atteint.
3. Á l'aide du programme précédent, explorer l'ensemble des nombres à trois chiffres (il contient 10000 nombres, `001` en fait par exemple partie) pour les répartir en sous-ensembles disjoints. Montrer qu'il y a dix sous-ensembles dont un seul ne contient qu'un seul nombre `{000}`.

Exercice 2.19 (Décomposition des entiers)
On veut répondre à cette question : « les nombres entiers peuvent-ils tous être constitués par une somme d'entiers consécutifs ? » et, si la réponse est non, déterminer les nombres qui ne sont pas reconstituables ainsi. Le nombre 25, par exemple, peut être décomposé en la somme 12+13 de deux entiers consécutifs ; on remarque que la somme 3+4+5+6+7 convient aussi. Ce nombre se décompose de deux façons différentes comme somme de nombres entiers consécutifs.

1. Exploration systématique : chercher une décomposition en somme d'entiers consécutifs pour tous les nombres entiers inférieurs à 100 (le programme peut se contenter d'afficher pour chaque nombre s'il a trouvé ou non une décomposition qui convienne). Émettre alors une conjecture concernant la question posée.
2. On voudrait savoir maintenant quel est le 1er nombre à avoir `k=1,2,3,...` décomposition(s). Améliorer le programme pour qu'il détermine le nombre de décompositions possibles d'un nombre entier et affiche seulement ceux dont le total des décompositions diffère des valeurs déjà atteintes. Pousser l'exécution du programme au-delà de 100 (jusqu'à 1000 par exemple pour avoir le er nombre à avoir 15 décompositions, au-delà pour `k`$\in \{10, 12, 13, 14\}$).

2.1.3 Types de données construits

Listes

Exercice 2.20 (Cartes)

1. Écrire un programme qui renvoie une main de `N` cartes prises au hasard dans un jeu de 32 cartes. Par exemple, avec `N=2`, on doit pouvoir obtenir la liste `[As de Trêfle, Dix de Pique]`, mais pas la liste `[Dame de cœur, Dame de cœur]` puisqu'une carte ne peut être tirée qu'une fois.
2. Sur le même modèle, écrire une version du programme qui distribue les 52 cartes d'un jeu entre 4 joueurs.

Exercice 2.21 (Calendrier)

1. Écrire un programme qui affiche les jours de la semaine pour un mois `m` et une année `a` fournis en arguments. Ce sera l'occasion de réinvestir des fonctions déjà vues, en particulier la fonction « mois » qui donne le nombre de jours du mois (cf exercice 2.10) et la fonction `calendar.weekday` qui donne le jour de la semaine d'une date donnée (cf exercice 2.5).
2. Mettre en forme l'affichage de ce calendrier en écrivant, comme l'usage le veut : une ligne par semaine, une colonne par jour, la semaine commençant en France par Lundi. Ne pas oublier une ligne de titre rappelant le mois et l'année !

Exercice 2.22 (Nombres premiers)

1. Écrire une fonction qui détermine la liste des nombres premiers inférieurs ou égaux à un entier `n` donné. Selon la méthode du crible d'Ératosthène, on peut supprimer d'une liste contenant les entiers entre 2 et `n` tous les multiples de 2, puis supprimer tous les multiples du nombre suivant 2 qui n'a pas été supprimé (il s'agit de 3), puis supprimer tous les multiples du nombre suivant 3 qui n'a pas été supprimé, etc. jusqu'à avoir supprimé tous les multiples des nombres premiers inférieurs à $\sqrt{\texttt{n}}$ (cf exercice 2.12)
2. On veut déterminer le nombre de nombres premiers inférieurs ou égaux à tous les entiers inférieurs ou égaux à `n`. Si on se limite à `n=12`, par exemple, on peut se contenter du résultat suivant `L=[0,0,1,2,2,3,3,4,4,4,4,5,5]` où `L[k]` donne le nombre de nombres premiers inférieurs ou égaux à `k`. Par exemple `L[10]=4` car il y a `4` nombres premiers inférieurs ou égaux à `10` (ce sont `2,3,5,7`).

Exercice 2.23 (Écriture décimale)

1. En partant de l'écriture irréductible d'un nombre rationnel donnée par deux nombres `a` et `b` (le numérateur et le dénominateur de la fraction), écrire une fonction qui donne la nature du nombre (entiers, décimal non entier ou rationnel non décimal) ainsi que la suite des chiffres qui se répète dans l'écriture décimale d'un rationnel non décimal. Si on entre `1` et `7`, par exemple, la fonction doit renvoyer `142857`. Si le nombre est décimal, la fonction donne son écriture décimale.
2. Adapter le programme pour qu'il fasse la même chose que précédemment pour une base quelconque (on se limitera aux bases inférieures ou égales à 10).

Exercice 2.24 (Élimination cyclique)

1. Un problème ancien : Josèphe fut réduit à se disposer en cercle avec quarante personnes, sachant qu'en comptant de trois en trois à partir de l'un d'eux, ils devaient se supprimer mutuellement (le premier à être supprimé est rangé en position 3 et il se fait tuer par celui qui est rangé en position 6 ; celui-ci est le suivant à être supprimé, par celui en position 9, etc.) Ainsi, progressivement, toute la troupe est appelée à s'autodétruire (le dernier est supposé se supprimer lui-même). À quelle place doit se ranger le courageux Josèphe pour être le dernier et devoir ainsi se supprimer lui-même ?
2. Cette situation est transposée au monde de l'enfance : « plouf, plouf, une poule en or c'est toi qui sort ». À quel rang faut-il être pour être choisi (équivalent moderne de se supprimer soi-même) ? Supposer qu'il y a `n=5` bambins et `p=10` syllabes dans la comptine de sélection.

Exercice 2.25 (Énigme)

Voici l'énigme MU de Hofstadter[1] : on dispose de `MI`. À l'étape suivante, on ajoute toutes les chaînes que l'on peut obtenir par une des quatre règles suivantes :

- Si on possède une chaîne se terminant par `I`, on peut lui ajouter `U` à la fin
- Si on possède la chaîne `Mx`, on peut lui ajouter `x` pour obtenir la chaîne `Mxx`
- Si on possède une chaîne contenant `III`, on peut remplacer `III` par `U`
- Si une chaîne contient `UU`, on peut supprimer ce `UU`

L'énigme est la suivante : va t-on, à une étape quelconque, obtenir la chaîne `MU` ?

1. Proposer un programme qui détermine les chaînes s'ajoutant à la collection à chaque étape et qui s'arrête quand `MU` est trouvé (cela peut ne jamais arriver).
2. Vérifier qu'à l'étape 1 on obtient les chaînes `MIU` et `MII` et à l'étape 2 obtient `MIUIU`, `MIIU` et `MIII`.

Dictionnaires et ensembles

Exercice 2.26 (Code César)

Principe du code César : on commence par créer un alphabet de substitution en décalant toutes les lettres d'un certain nombre constituant la clé du code. On peut alors appliquer la substitution à chaque lettre du message à encoder ou bien, en sens inverse, on peut décoder un message. Par exemple `XKJFKQN` code le message `BONJOUR` avec une clé de 4 (`E` code `A`, `F` code `B`, etc.).

1. Écrire une fonction qui prenne un nombre entier `c` comme argument et un texte `t` et qui décale toutes les lettres du texte de `c` rangs vers la droite ; si `c<0` le décalage se fait vers la gauche. Cette fonction sera utilisée pour coder ou décoder un message. La tester sur `4,XKJFKQN`.
2. Le message suivant `XQEBX MUEMZ FQDUQ EXQEB XGEOA GDFQE EAZFX QEYQU XXQGD QE` a été encodé avec un code César mais on a perdu la clé. Pour le décrypter, on peut utiliser le principe suivant : en français, la lettre la plus fréquente étant le `E`, il suffit de déterminer la lettre la plus fréquente pour savoir comment est codé le E et en déduire la clé et le message décodé. Si celui-ci n'a pas de sens, on peut essayer la 2^{e} lettre la plus fréquente, ou la 3^{e}... Commencer par construire un dictionnaire avec les 26 lettres de l'alphabet comme clés et le nombre d'occurrences de chacune comme valeur. Appliquer ensuite la fonction de décodage à la clé ayant la valeur la plus importante. Décoder alors le message.

1. Douglas Hofstadter, *Gödel, Escher, Bach, les Brins d'une Guirlande Éternelle*, Dunod 2000, pp.38-47.

Exercice 2.27 (Stock)
On veut gérer le stock de livres d'un éditeur spécialisé en utilisant un dictionnaire. Pour cela, on part de l'inventaire qui définit le stock initial sous la forme d'un dictionnaire dont les clés sont les titres et les valeurs les quantités disponibles. Lors d'une commande, on examine le stock et renvoie le message "livre épuisé" s'il est nul, "référence absente" si la clé n'est pas dans le dictionnaire, "commande partiellement exécutée" si la quantité demandée excède la quantité disponible et "commande satisfaite" si la quantité demandée a pu être servie.

1. Le stock après inventaire est `stock={"Python en 100 leçons":1500, "Je programme comme un pro":2500,"Jeux en Python":900}`
   Mettre au point la fonction `commande(livre,nombre)` qui examine le stock et renvoie le message approprié après avoir mis à jour le stock. Tester en entrant successivement : `commande("Jeux en Python",750)`, `commande("Jeux en Python",500)`, `commande("Je programme en Python",100)` et `commande("Python en 100 leçons",1500)`.
2. Lorsqu'une nouvelle référence apparait ou lorsqu'une ancienne référence est réapprovisionnée, il faut pouvoir l'intégrer au stock. Écrire une fonction `ajout(livre,quantité)` qui réalise cela et renvoie le message "ajout pris en compte" suivi de l'état du stock (un affichage du dictionnaire `stock` fera l'affaire).
   Tester cette nouvelle fonction en partant de l'inventaire initial et entrant successivement `ajout("Je programme en Python",1000)` et `ajout("Jeux en Python",5000)`.
3. On veut pouvoir conserver la liste des reliquats de commandes partiellement exécutées. Pour cela, enregistrer votre programme dans une version 2. Modifier la fonction `commande` pour qu'elle prenne le nom du client comme 3e argument et, en cas de commande partiellement exécutée, enregistrer le reliquat dans une liste `reliquats` contenant des tuples de la forme `(livre,reliquat,client)`. Modifier ensuite la fonction `ajout` pour qu'elle examine si un des reliquats de commande ne peut pas être servi, même partiellement. Servir ce reliquat quand c'est possible et afficher, en plus du stock, la liste `reliquats` si elle n'est pas vide.
   Tester ces nouvelles fonctions en partant de l'inventaire initial et entrant successivement `commande("Jeux en Python",1500,"FNOC")` et `ajout("Jeux en Python",5000)`.

Exercice 2.28 (Association sportive)
Une association sportive propose trois sports à ses adhérents : natation, paddling et voile. Le responsable des inscriptions a besoin d'un programme qui établisse la liste des personnes inscrites à chaque sport. Il voudrait aussi disposer des listes des personnes inscrites à deux sports ainsi que de la liste des personnes inscrites aux trois sports. Les inscriptions arrivent sous la forme d'un couple `(nom de la personne,liste des sports pratiqués)`.

1. Écrire une fonction `ins(nom,liste)` qui complète les listes `natation`, `paddling` et `voile` en y enregistrant les noms des personnes inscrites (initialement, ces listes sont créées vides). Pour éviter les doublons (deux inscriptions avec le même nom), on utilise des ensembles ; cette structure permettra d'obtenir facilement les listes des double et triple inscriptions.
   Tester cette fonction en générant des inscriptions valides, par exemple `ins("AB",["natation","paddling"])`, `ins("BC",["natation","voile"])`, `ins("CD",["voile"])`, `ins("DE",["paddling"])`, `ins("EF",["natation"])`, `ins("FG",["paddling","natation"])`, `ins("GH",["paddling","natation","voile"])`.
2. Écrire la fonction `affichage` qui donne les trois listes simples, les trois listes des double inscriptions et la liste des triple inscriptions.
   Quelles seront ces sept listes après inscriptions des sept personnes précédentes ?
   `natation={.....}`, `paddling={.....}`, `voile={.....}`
   `natation+paddling={.....}`,`natation+voile={.....}`,`paddling+voile={.....}`
   `natation+paddling+voile={...}`

p-uplets

Exercice 2.29 (Fractions)

Une fraction peut être vue comme un couple de deux nombres `(a,b)` où `a` est un entier éventuellement négatif et `b` un entier non nul.

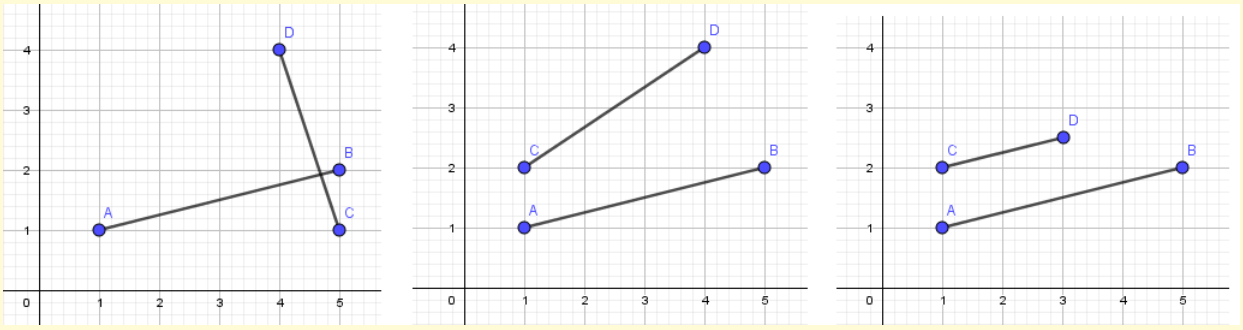
1. Écrire une fonction `irreductible(a,b)` qui transforme une fraction quelconque en sa forme irréductible. Pour cela, déterminer le PGCD des deux nombres (exemple 4 page 9 du cours) à l'aide d'une fonction `pgcd(a,b)` et simplifier la fraction initiale avec ce PGCD.
2. Écrire une fonction `operation(a1,b1,op,a2,b2)` qui effectue, sur les fractions `(a1,b1)` et `(a2,b2)`, l'opération `op` (le caractère `op` étant choisi dans le p-uplet `("+","-","*","/")`). Cette fonction retourne le résultat sous la forme d'une fraction irréductible.
3. Écrire une fonction `affichage(a1,b1,op,a2,b2)` qui écrive sur la sortie standard (la console) l'opération ainsi que son résultat sous la forme habituelle des fractions, sur 3 lignes.

```
1   1   2          1   15   31          -1   15   463          -1   15   -152          112   -125   -1750
- + - = -         -- + -- = --         --- + --- = -----        --- / --- = ----        --- * ---- = -----
2   6   3         12    6   12         123   456   18696        123   456   615         123    456    7011
```

Exercice 2.30 (Croisement)

Supposons que, connaissant déjà les coordonnées de trois points `A(xa,ya)`, `B(xb,yb)` et `C(xc,yc)`, on cherche à déterminer si le point `D(xd,yd)` est du même côté du segment `[AB]` que `C` ou s'il est de l'autre côté de ce segment. Pour répondre à cette question, on peut déterminer une équation cartésienne de la droite `(AB)` (elle est du type $ax + by + c = 0$), une équation cartésienne de la droite `(CD)` (elle est aussi du type $ax + by + c = 0$), les coordonnées du point d'intersection s'il existe, et enfin vérifier que ce point d'intersection est située entre `A` et `B` . Tout cela peut paraitre assez fastidieux aussi procèderons-nous par étapes.

1. Écrire ou retrouver la fonction `droite(xa,ya,xb,yb)` qui renvoie les coefficients `(a,b,c)` de la droite `(AB)` (cette fonction servira également pour la droite `(CD)`).
2. Écrire ou retrouver la fonction `intersection(a1,b1,c1,a2,b2,c2)` qui renvoie les coordonnées `(x,y)` du point d'intersection des droites d'équation $a_1x + b_1y + c_1 = 0$ et $a_2x + b_2y + c_2 = 0$ quand il existe et `(None,None)` sinon.
3. Utiliser la formule du produit scalaire des vecteurs $\vec{u}$`(x1,y1)` et $\vec{v}$`(x2,y2)` (ce produit vaut `x1×x2+y1×y2`) pour déterminer si le point d'intersection trouvé appartient au segment `[AB]` (si le produit scalaire de deux vecteurs colinéaires est positif, les deux vecteurs ont le même sens, sinon ils ont un sens contraire).
4. Écrire une fonction `coupe(xa,ya,xb,yb,xc,yc,xd,yd)` qui renvoie `True` si `[CD]` coupe `[AB]` et `False` sinon.
   Tester cette fonction avec les trois quadruplets de points :
   `A(1,1)`, `B(5,2)`, `C(5,1)` et `D(4,4)`
   `A(1,1)`, `B(5,2)`, `C(1,2)` et `D(4,4)`
   `A(1,1)`, `B(5,2)`, `C(1,2)` et `D(3,2.5)`



Fichiers

Exercice 2.31 (Pendu)
Principe du jeu : l'ordinateur choisit un mot au hasard dans une liste contenant des mots. Le joueur tente de trouver les lettres intérieures du mot, remplacées par un caractère de soulignement (_). À chaque coup, il saisit une lettre ; si la lettre figure dans le mot, l'ordinateur replace le ou les (_) correspondant(s) par la lettre. Le joueur a 3 chances de plus que le nombre de lettres du mot. Au delà, il a perdu et est « pendu ».

1. Écrire une fonction `tirage()` qui choisit un mot au hasard dans une des trois listes de mots enregistrées dans les fichiers `"adjectifs.txt"`, `"verbes.txt"` et `"noms.txt"` à récupérer dans le dossier partagé. La fonction affiche le type de mot à trouver : adjectif, verbe ou non. Les mots étant écrits en minuscules avec des accents, il faut les débarrasser de ces accents éventuels. La fonction opère ce nettoyage et renvoie le mot choisi sans accent.
2. Écrire le programme principal sous la forme d'une boucle infinie dont on ne sort que si le joueur ne répond pas `"y"` à la question `"Voulez-vous rejouer?"`. Dans le corps de la boucle infinie, une autre boucle permet au joueur de proposer successivement plusieurs lettres. Limiter le nombre d'erreurs à dix (ou douze selon le dessin du pendu). Le mot à deviner est affiché, au début, sous la forme `"I _ _ _ F"` où `"I"` et `"F"` sont les lettres initiales et finales, et `"_"` sont les lettres à deviner.
3. Écrire une fonction `correction()` qui prend en argument le mot affiché et qui examine si la lettre proposée par le joueur peut remplacer un au moins des `"_"`. La fonction retourne un couple composé du mot affiché, éventuellement modifié, et d'un booléen indiquant si le joueur a trouvé le mot ou non.
4. Finaliser le jeu en calculant puis en affichant un score (par exemple, on augmente le score du joueur de 10 s'il gagne, et s'il perd, on le diminue de 5), selon une méthode de calcul plus ou moins sophistiquée.
5. Facultatif : en utilisant le module `tkinter`, transformer l'affichage afin d'obtenir progressivement le dessin du pendu (un trait s'ajoute au dessin à chaque erreur). On ne doit plus interagir avec la console mais celle-ci peut servir de lieu d'enregistrement. Les lettres saisies ne sont pas affichées sur la fenêtre graphique qui ne montre que le mot, le score et le dessin.

Exercice 2.32 (Pays)
On a recueilli des informations sur les pays à partir de Wikipédia dans le fichier `"pays_sex.txt"`. Chaque ligne est constituée ainsi : nom du pays ; capitale ; latitude, longitude ; URL du drapeau ; population ; superficie ; monnaie ; langue ; extension internet ; indicatif international. Notre objectif est d'afficher certaines de ces informations à côté d'un fond de carte centré sur la capitale. Pour réaliser cela dans une page HTML, il nous faut les coordonnées géographiques du lieu au format décimal et non sexagésimal : les coordonnées de Pékin, par exemple, sont données par `"39° 55' N, 116° 23' E"` mais nous devons les traduire en degrés décimaux, soit `"39.9167, 116.3833"`, pour l'affichage de la carte.

1. Écrire la fonction `latitude_vers_decimal(lat)` qui retourne la latitude au format décimal (positive vers le Nord, négative vers le Sud). Attention, il peut y avoir des secondes de degré (Jakarta : `"6° 10' 05'' S, 106° 49' 07'' E"`) ! Recopier et modifier cette fonction pour obtenir la longitude décimale (positive vers l'Est, négative vers l'Ouest). Tester aussi avec `Paris : "48° 52' N, 2° 19' 59'' E"` (Wikipédia donne "48° 52' N, 2° 19,59' E" mais c'est ambigu et source d'erreur pour notre programme, d'où ma correction.
2. Écrire le programme qui lit, ligne après ligne, le fichier `"pays_sex.txt"` et qui écrit un fichier `"pays_dec.txt"` dans lequel les coordonnées sexagésimales ont été décimalisées.

Exercice 2.33 (Code postal)
Télécharger sur le site `data.gouv.fr` la base officielle des codes postaux (pour la France (métropole et DOM), les TOM et MONACO). L'adresse précise en est :
`https://www.data.gouv.fr/fr/datasets/base-officielle-des-codes-postaux/`

1. Écrire un programme qui ouvre ce fichier `csv` et enregistre les données dans une table de dictionnaires. Pour cela, utiliser la fonction `DictReader` du module `csv` appliqué au fichier `f` que l'on a préalablement ouvert (attention, le délimiteur est `";"` !) : en écrivant `table=list(csv.DictReader(f))`, `table` est une liste dont chaque élément est un dictionnaire. Afficher `table[0].keys()` et `len(table)` pour obtenir la liste des clés des dictionnaires (les champs de la table et la longueur de la table (le nombre d'enregistrements).
2. Compléter votre programme en écrivant la fonction `code(nom)` qui renvoie le code postal d'une commune dont on connait le `nom`. Déterminer ainsi les codes postaux des communes de STE MARIE, OUESSANT, ST REMY LES CHEVREUSE et PARIS 05. Vous noterez au passage que si on ne connait pas l'écriture exacte employée, on n'a peu de chance d'obtenir le bon code...
3. Écrire la fonction inverse de la précédente `nom(code)` qui renvoie le nom d'une commune dont on connait le `code`. Déterminer ainsi les noms des communes correspondants aux codes 92430, 97438, 66470. Constatez à cette occasion que plusieurs communes peuvent porter le même nom (en général il s'agit de communes de départements différents).
4. Écrire une fonction `affiche(dept)` qui affiche le nom des communes du département `dept` et leur code postal. L'inconvénient de cet affichage est qu'il ne va pas respecter l'ordre des numéros de code ni celui des orthographes de noms : les communes apparaissent dans l'ordre de leur inscription dans la table initiale.
5. Copier et modifier la dernière fonction pour créer la fonction `enregistre(dept)` qui génère un fichier `csv` contenant la liste des communes du département `dept` et leur code postal. On pourra utiliser la fonction `DictWriter` du module `csv` qui, pour un fichier `fw` ouvert en écriture , s'utilise ainsi :
   `w=csv.DictWriter(fw,['Code_postal','Nom_commune'],lineterminator='\n')`
   pour n'avoir qu'un saut de ligne `"\n"` après chaque enregistrement. Pour séparer les données avec `";"` au lieu de la virgule par défaut, ajouter `delimiter=";"`
   `w.writeheader()` est l'instruction qui écrit la ligne d'entête.
   `w.writerow({...})` est celle qui enregistre sur une ligne un dictionnaire.

Exercice 2.34 (Cercle)
On souhaite déterminer le nombre de points à coordonnées entières appartenant à un même cercle centré sur l'origine et de rayon $n \geqslant 1$. Ce nombre – noté $4N$ plus loin – est nécessairement supérieur ou égal à 4 (les points du cercle situés sur les axes de coordonnées). Par ailleurs, les autres points qui conviennent, pour des raisons de symétrie, vont également par 4 (par rotation successive d'un quart de tour). On va donc se limiter aux points du 1er quadrant : en faisant varier l'abscisse entière `x` des points solutions entre 0 et $n-1$, on en calcule l'ordonnée positive $y = \sqrt{n^2 - x^2}$. Si celle-ci est entière alors on a une solution.

1. Écrire la fonction `cercle(n)` qui détermine et renvoie la liste des N abscisses entières positives conduisant à une ordonnée entière positive sur le cercle de rayon `n`.
2. Utiliser la fonction `cercle` pour les rayons entiers supérieurs ou égaux à 1 jusqu'à avoir obtenu 100 points à coordonnées entières sur un même cercle.
   Enregistrer les résultats dans un fichier `"cercle.txt"` avec le codage suivant :
   n(rayon)-$4N$(nombre total de points)-x_1 x_2 ... x_N(liste des abscisses pour le 1er quadrant).
   Les cinq premières lignes de ce fichier sont : `1-4-0`, `2-4-0`, `3-4-0`, `4-4-0` et `5-12-0 3 4`.

Exercice 2.35 (Tangram)

Récupérer le fichier `tangram.txt` qui se situe dans le dossier `Documents` du professeur ou bien télécharger ce fichier qui se situe à l'adresse `http://mathadomicile.fr/Puzzles/tangram.txt`. Ce fichier est assez important en taille (220 969ko), il contient toutes les formes homogènes que l'on peut construire avec le Tangram, le célèbre jeu de dissection du carré en sept pièces. On peut lire sur la 1re ligne, l'enregistrement suivant `4.5.00200250550055100051.20.18.5.10.0` qui correspond à une des 5 583 516 formes enregistrées. Le système utilisé pour encoder ces formes est le suivant, les valeurs étant séparées par des points :

- nombre de colonnes (4) puis nombre de lignes (5)
- lignes concaténées de l'encodage de la forme (00200250550055100051)
- différentes caractéristiques de la forme (dimension, convexité, symétrie, côtés, trou)

Afin d'exploiter ce fichier, on souhaite obtenir la silhouette de la forme sélectionnée. Pour ce faire on va utiliser le module `turtle` sachant que chacun des chiffres du champs « lignes concaténées » correspond à une des six possibilités illustrées ci-dessous (code 5 : carré plein, code 0 : carré vide, codes 1 à 4 : triangle). Ainsi, le grand carré est encodé `4.4.0210255135540340...` tandis que l'image nous montre d'autres formes, dont celle de la 1re ligne.

1. Écrire la fonction `lecture(r)` qui va chercher dans le fichier `tangram.txt` la ligne de rang `r` et qui la décode, transformant notamment le champs « lignes concaténées » en une liste de listes. Rappel : pour initialiser à zéro une telle liste de `lig` lignes et `col`colonnes, on écrit `L=[col*[0] for i in range(lig)]`.
2. Écrire les fonctions `affiche(liste)` et `dessine(x,y,c)` qui, la 1re, lance les dessins correspondants aux codes `c` lus dans la liste ; la 2e réalise les dessins des polygones pleins (triangle ou carré) dans un carré dont le sommet supérieur gauche a pour coordonnées `(x,y)`.
3. Affiner la recherche des formes dans le fichier en précisant une caractéristique recherchée. Les formes ayant un élément de symétrie ont un code symétrie qui vaut `0`, `2` ou `4` pour 4, 2 ou 1 axe(s) de symétrie, `3` pour centre et `1` pour un centre quart de tour).
   - Afficher la 1re et seule forme ayant un code symétrie égal à `1`, puis
   - la 13e forme convexe (le code convexité, la 2e caractéristique, est égal à `0`),
   - la 50e forme ayant 18 côtés (côtés : avant-dernière caractéristique), et enfin
   - la 900e forme ayant un vrai trou (dernière caractéristique égale à `2`).