TP 6 : Représentation des nombres

▸ Calcul des décimales de π par la méthode d'Archimède

Au III$^{\text{ème}}$ siècle avant Jésus-Christ, Archimède de Syracuse (sans rapport avec la suite éponyme !) proposa une méthode pour calculer une valeur approchée de π, qui repose sur un principe simple : la différence entre le périmètre du polygone à n côtés inscrit au cercle de rayon 1 et le périmètre du polygone à n côtés circonscrit au même cercle tend vers 0, et lorsque n tend vers $+\infty$, le périmètre de ces deux polygones tend vers le périmètre du cercle, qui est 2π.
Plus précisément, le périmètre du cercle inscrit est toujours inférieur à 2π (permettant donc d'obtenir une approximation par défaut de 2π), quand celui du cercle circonscrit est supérieur à 2π (et donc est une approximation par excès).
Avec cette méthode, il obtient l'encadrement $3 + \dfrac{10}{71} < \pi < 3 + \dfrac{11}{71}$, soit environ $3.1408 < \pi < 3.1549$.
Cette approximation sera améliorée au fil des siècles jusqu'à fournir les 35 premières décimales au XVI$^{\text{ème}}$ siècle, avant d'être supplantée par la formule de Machin dont nous avons déjà parlé.

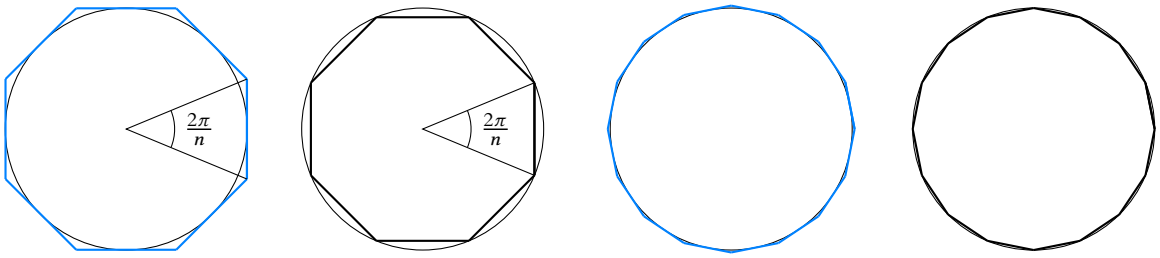

Figure 0.1 – Les polygones à 8 et 16 côtés inscrits et circonscrits à un cercle.

Pour $n \geqslant 2$, on note u_n le demi-périmètre d'un polygone à 2^n côtés inscrit dans le cercle unité, et v_n le demi-périmètre du polygone circonscrit au même cercle.
▸ **Question 1.** Montrer que $u_n = 2^n \sin\frac{\pi}{2^n}$ et $v_n = 2^n \tan\frac{\pi}{2^n}$ et que $\lim\limits_{n\to+\infty} u_n = \lim\limits_{n\to+\infty} v_n = \pi$.
On pourra admettre sans démonstration que $\forall x \in \left[0, \frac{\pi}{2}\right]$, $\sin x \leqslant x \leqslant \tan x$ *et* $\lim\limits_{x\to0} \dfrac{\sin x}{x} = \lim\limits_{x\to0} \dfrac{\tan x}{x} = 1$.

▸ **Question 2.** À l'aide des formules liant $\sin x$, $\tan x$ et $\tan\dfrac{x}{2}$, on prouve (et on ne vous demande pas de le faire) que

$$\forall n \geqslant 2,\ u_{n+1} = \sqrt{u_n v_{n+1}} \text{ et } v_{n+1} = \frac{2u_n v_n}{u_n + v_n}.$$

En utilisant ces relations et les valeurs initiales $u_2 = 2\sqrt{2}$ et $v_2 = 4$, rédiger une fonction Python qui affiche les n premières valeurs des suites u et v.
Utiliser ces fonctions avec $n = 30$, puis $n = 40$. Que constatez-vous et comment l'expliquez vous ?
À partir de quel rang est-il inutile de poursuivre l'itération de ces suites ?

Pour obtenir davantage de décimales de π, on ne peut pas continuer à utiliser le type `float`. En revanche, Python permet la manipulation d'entiers arbitrairement grands, c'est pourquoi nous allons chercher à encadrer $10^{200}\pi$ par deux entiers les plus proches possibles. Les chiffres que ces entiers auront en commun nous permettront (en décalant la virgule de 200 chiffres vers la gauche) d'obtenir un certain nombre de décimales de π.

On définit quatre suites d'entiers $(a_n), (b_n), (c_n), (d_n)$ en posant

$$a_2 = \left\lfloor \sqrt{8 \cdot 10^{400}} \right\rfloor,\ b_2 = 4 \cdot 10^{200},\ c_2 = \left\lceil \sqrt{8 \cdot 10^{400}} \right\rceil,\ d_2 = 4 \cdot 10^{200}$$

$$a_{n+1} = \left\lfloor \sqrt{a_n b_{n+1}} \right\rfloor,\ b_{n+1} = \left\lfloor \frac{2a_n b_n}{a_n + b_n} \right\rfloor,\ c_{n+1} = \left\lceil \sqrt{c_n d_{n+1}} \right\rceil,\ d_{n+1} = \left\lceil \frac{2c_n d_n}{c_n + d_n} \right\rceil$$

où $\lceil x \rceil$ désigne l'unique entier k tel que $k - 1 < x \leqslant k$ (la partie entière supérieure).

► **Question 3.** Montrer par récurrence que pour tout $n \geqslant 2$, $a_n \leqslant 10^{200} u_n \leqslant c_n$ et $b_n \leqslant 10^{200} v_n \leqslant d_n$.

Pour pouvoir calculer les valeurs exactes des quatre suites entières a, b, c, d, nous aurons besoin de quatre fonctions (définies sur $\mathbf{N} \times \mathbf{N}$ ou sur $\mathbf{N}$) :

$$\text{floor}(x, y) = \left\lfloor \frac{x}{y} \right\rfloor,\ \text{ceil}(x, y) = \left\lceil \frac{x}{y} \right\rceil,\ \text{fsqrt}(x) = \lfloor \sqrt{x} \rfloor,\ \text{csqrt}(x) = \lceil \sqrt{x} \rceil.$$

Pour éviter toute perte de précision, on souhaite définir ces fonctions sans faire appel au type `float` *et donc sans utiliser les fonctions* `floor, ceil` *ou* `sqrt`.

► **Question 4.** Définir les deux premières en Python.

Pour calculer rapidement les deux suivantes, nous allons utiliser l'algorithme de Héron, que nous généraliserons plus tard dans l'année sous le nom de méthode de Newton.

► **Question 5.** Soit $a \in \mathbf{N}^*$ et soit $f : x \mapsto x^2 - a$. On considère x_0 un entier supérieur strictement à $\sqrt{a}$ et on note x_1 l'abscisse du point d'intersection de la tangente à la courbe de f en x_0 avec l'axe des abscisses.

1. Montrer que $\sqrt{a} < x_1 < x_0$ (faire un dessin, et étudier la position de la courbe de f par rapport à sa tangente).
2. Exprimer x_1 en fonction de x_0. Vérifier alors que $x_1 = g(x_0)$, où $g : x \mapsto \frac{1}{2}\left(x + \frac{a}{x}\right)$.

On définit alors une suite (r_n) en posant $r_0 = a$, et $\forall n \in \mathbf{N}$, $r_{n+1} = \lfloor g(r_n) \rfloor$.

3. En remarquant que (r_n) est strictement décroissante, montrer qu'il existe un entier n vérifiant $r_n \leqslant \sqrt{a}$.
4. Soit N le plus petit entier vérifiant cette inégalité. Montrer que $r_N = \lfloor \sqrt{a} \rfloor$.
5. En déduire une fonction Python nommée `fsqrt` calculant $\lfloor \sqrt{a} \rfloor$ pour tout entier a.
6. Comment modifier cette fonction pour calculer $\lceil \sqrt{a} \rceil$? Rédiger la fonction `csqrt` correspondante.

► **Question 6.** Lorsqu'on calcule les valeurs des suites $(a_n), (b_n), (c_n), (d_n)$, on observe que ces suites sont stationnaires, c'est-à-dire qu'il existe un rang à partir duquel elles sont constantes. Si N désigne ce rang, l'encadrement $a_N \leqslant 10^{200}\pi \leqslant d_N$ est le meilleur que l'on puisse obtenir.
Rédiger un script Python qui déterminer cette valeur de N, ainsi que les valeurs correspondantes de a_N et de d_N. Combien de décimales de π cet encadrement fournit-il ?
Déterminer alors le nombre de décimales correctes fournies par la question 1.

► Codage de Fibonacci

Dans la suite, on considère la suite de Fibonacci, définie par $F_0 = 0$, $F_1 = 1$, $\forall n \in \mathbf{N}$, $F_{n+2} = F_{n+1} + F_n$.

► **Question 7.** Écrire une fonction Python `fibonacci` qui prend en paramètre un entier naturel p et qui renvoie la liste des $p + 1$ termes de (F_n). Par exemple `fibonacci(5)` doit renvoyer `[0,1,1,2,3,5]`.

Dans la suite, nous admettrons le théorème de Zeckendorf, qui affirme que tout entier naturel non nul n s'écrit de manière unique comme somme de nombres de Fibonacci non consécutifs.
Plus précisément : si $n \in \mathbf{N}^*$, alors il existe un entier $r \in \mathbf{N}^*$ et es entiers naturels $k_1, k_2, \ldots, k_r$ supérieurs ou égaux à 2 tels que

1. $\forall i \in [\![1, r-1]\!]$, $k_{i+1} - k_i > 1$, ce qui traduit que les nombres $F_{k_{i+1}}$ et F_{k_i} ne sont pas des termes consécutifs de la suite de Fibonacci.

2. $n = F_{k_1} + F_{k_2} + \cdots + F_{k_r}$.

Une telle écriture $n = F_{k_1} + F_{k_2} + \cdots + F_{k_r}$ sera appelée dans la suite la F-décomposition de n.
La preuve de ce théorème est en fait constructive et nous dit que pour trouver la F-décomposition de n, il faut commencer par trouver le plus grand k tel que $F_k \leqslant n < F_{k+1}$, et alors la F-décomposition de n est égale à F_k plus la F-décomposition de $(n - F_k)$.

Par exemple, cherchons la F-décomposition de 100.
Le plus grand terme de (F_n) inférieur ou égal à 100 est $89 = F_{11}$.
On a alors $100 - 89 = 11$. Le plus grand terme de (F_n) inférieur ou égal à 11 est $8 = F_6$.
Et $11 - 8 = 3 = F_4$. Donc la F-décomposition de 100 est $100 = 89 + 8 + 3 = F_{11} + F_6 + F_4$.

► **Question 8** Écrire une fonction `recherche` qui prend en paramètre un entier naturel n et renvoie le plus grand entier naturel p tel que $F_p \leqslant n$. Par exemple `recherche(7)` doit renvoyer 5.

► **Question 9** Écrire une fonction `Fdecomposition` qui prend en paramètre un entier naturel n non nul et renvoie sa F-décomposition sous forme d'une liste croissante d'entiers. Par exemple, `Fdecomposition(100)` doit renvoyer `[3,8,89]`.

Une fois qu'on dispose de la F-décomposition d'un entier naturel n non nul, on lui associe une liste composée de 0 et de 1 de la manière suivante :

- on écrit en ligne la liste de tous les termes de la suite de Fibonacci inférieurs ou égaux à n en commençant à partir de F_2 ;
- en-dessous de chaque terme de cette liste, on inscrit 1 si ce terme figure dans la F-décomposition de n et 0 s'il n'y figure pas ;
- on obtient ainsi une liste formée de 0 et de 1, qu'on «normalise» en ajoutant un 1 en dernière position.

Ce principe est appelé *codage de Fibonacci*.

► **Question 10** Vérifier que 100 est codé par la liste `[0,0,1,0,1,0,0,0,0,1,1]`.
On notera alors un tel codage $100 = \overline{00101000011}^F$.

► **Question 11** Écrire une fonction `codage` qui prend en paramètre un entier naturel n non nul et renvoie son codage de Fibonacci sous forme d'une liste de zéros et de uns. Par exemple `codage(100)` doit renvoyer la liste `[0,0,1,0,1,0,0,0,0,1,1]`.
► **Question 12** Écrire une fonction `decodage` qui réalise l'opération inverse.

► Application du codage de Fibonacci à la compression de données

Un des avantages du code de Fibonacci est que le nombre de bits nécessaires pour stocker la représentation d'un nombre n'a pas besoin d'être fixé en avance, contrairement à la représentation usuelle sur n bits, n fixé.
En effet, le codage de Fibonacci d'un entier ne peut contenir deux bits successifs égaux à 1, sauf les deux derniers (ce qui provient du 1 final que nous avons ajouté), et donc la séparation entre deux chiffres consécutifs d'une suite de bits se trouve au niveau de deux 1 consécutifs.

Par exemple, sur 8 bits, si on souhaite représenter successivement les entiers 16, 100 et 7, il nous faut $3 \times 8 = 24$ bits, et on stocke alors la suite 00010000 01100100 0000111.
En revanche, les codages de Fibonacci respectifs de 16, 100 et 7 sont $\overline{0010011}^F$, $\overline{00101000011}^F$ et $\overline{0010011}^F$.
Donc la séquence 001001100101000011 0010011, qui se «scinde» en 0010011 00101000011 0010011 représente les trois entiers 16, 100 et 7 sur 23 bits «seulement».

Pour la séquence $2, 8, 3$, le gain est plus significatif puisque $2 = \overline{011}^F$, $8 = \overline{000011}^F$ et $3 = \overline{0011}^F$, de sorte qu'on peut représenter cette suite de trois nombres sur 13 bits au lieu de 24.
En revanche, l'un des inconvénients de ce codage est que sur un nombre fixé n de bits, on représente moins de nombres différents qu'avec la représentation binaire usuelle, tout simplement parce que certaines suites de n bits ne sont le codage de Fibonacci d'aucun entier (notamment en raison du 1 final).

▸ **Question 13** Quel est le plus grand entier dont le codage de Fibonacci tient sur 8 bits ?

Dans la suite, on cherche à réduire l'espace nécessaire pour stocker l'intégralité du premier volume du *Comte de Monte-Cristo* d'Alexandre Dumas.
Vous trouverez le fichier contenant le texte sur le site http ://www.mpsi2-champo.fr
Pour l'importer dans Python, ouvrez le fichier `TP6.py`, et comme dans le TP5, complétez la valeur de la chaîne `chemin` pour qu'elle contienne le nom du répertoire contenant le fichier. Remplacez alors tous les `\` par des `\\`, et ajoutez `\\` à la fin.
Exécutez alors le programme, une variable `texte_integral` est alors créee, contenant l'intégralité du texte.

On rappelle que dans le codage Unicode, à chaque caractère correspond un entier, que l'on obtient à l'aide de la fonction Python `ord`, et que la fonction réciproque, qui à un entier associe un caractère (une lettre majuscule ou minuscule, une espace, un caractère de ponctuation ou un saut de ligne) est `chr`.

▸ **Question 14** Quel est le plus grand code utilisé par les caractères de `texte_integral` ?

Pour stocker le texte sous forme d'un fichier, il s'agit de stocker la suite des entiers associés à chaque caractère, et il est donc légitime de représenter ces entiers sur 8 bits, puisque $2^8 = 256$.

▸ **Question 15** Combien de bits faut-il alors pour stocker l'intégralité du texte ?

Nous proposons dans la suite de compresser ce texte en utilisant le codage de Fibonacci. Comme mentionné précédemment, il n'est pas nécessaire de décider à l'avance le nombre de bits alloués à chaque caractère. L'idée est alors d'utiliser les entiers dont le codage de Fibonacci nécessite le moins de bits (donc $1 = \overline{11}^F$, $2 = \overline{011}^F$, $3 = \overline{0011}^F$,$4 = \overline{1011}^F$, etc) pour stocker les caractères les plus fréquents.
Ainsi, dans un texte en français, les caractères les plus fréquents seront vraisemblablement le *e*, le `a`, et non le `W`.
Après avoir déterminé la fréquence de chaque caractère, on peut donc décider de représenter le plus fréquent par $2 = \overline{11}^F$, le second plus fréquent par $3 = \overline{011}^F$, etc.
Ainsi, si le *e* est le caractère le plus fréquent, pour chaque *e* dans le texte initial, on économisera 6 bits avec le codage de Fibonacci par rapport au codage sur 8 bits.
Ce gain sera tout de même à relativiser, car par exemple, le $90^{\text{ème}}$ caractère le plus fréquent (s'il existe) sera représenté par $90 = \overline{10000000011}^F$, qui nécessite 11 bits au lieu de 8.
Le codage dépendant de la fréquence de chaque caractère dans le texte de départ, il faudra alors stocker également la table donnant la correspondance entre caractères et codes de Fibonacci, mais pour un texte suffisamment long, l'espace nécessaire est négligeable en comparaison de celui nécessaire au stockage du texte.

▸ **Question 16** Quel sont les trois caractères les plus fréquents dans *Le comte de Monte-Cristo* ?
▸ **Question 17** En négligeant l'espace nécessaire pour stocker la table de correspondance, estimer le pourcentage de bits qu'on peut économiser en utilisant un codage de Fibonacci pour stocker le texte du *comte de Monte-Cristo* par rapport à un codage standard sur 8 bits.
On rappelle que `sorted(L,reverse=True)` *renvoie une liste contenant les mêmes valeurs que L, triées dans l'ordre décroissant.*
▸ **Question 18** Combien de caractères différents contient le texte intégral de *Monte-Cristo* ? Était-il alors nécessaire d'utiliser un codage sur 8 bits ? Le codage de Fibonacci est-il toujours intéressant ?