Chapitre 1

Premiers pas dans l'environnement TI-Nspire™ CX

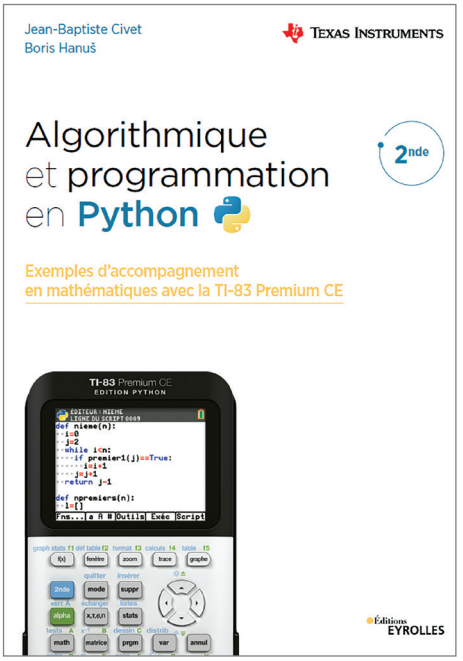
Nous allons d'abord nous familiariser avec la nouvelle interface de développement en Python de la famille TI-Nspire™ CX, puis nous nous intéresserons à la communication entre les différentes pages d'une activité. Nous verrons ensuite comment utiliser des cartes BBC micro:bit dans notre environnement de travail et comment coder un message avec le système RSA à l'aide de la TI-Nspire™ CX II-T CAS ou du logiciel TI-Nspire™ CX CAS Premium Teacher Software.

Découverte de l'interface

SE FORMER À LA PROGRAMMATION EN PYTHON

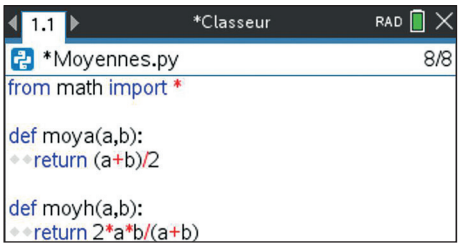
Cet ouvrage va vous présenter différents contenus d'algorithmique qui s'inscrivent dans le cadre des programmes de Mathématiques ou de NSI.

Nous en profitons pour vous rappeler l'existence d'un autre livre des mêmes auteurs intitulé *Algorithmique et programmation en Python*, qui vous permettra d'aborder les notions utiles à la programmation en Python : affectation de variables, structures de contrôle, utilisation de la console en Python, etc.



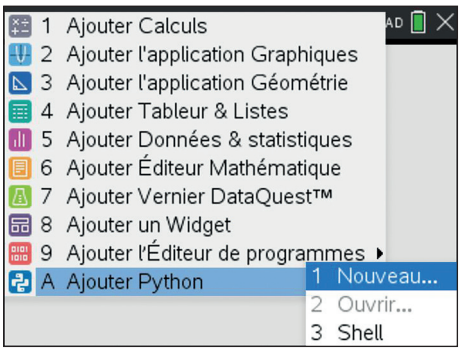
Par ailleurs, vous trouverez sur le site français de Texas Instruments (*https://education.ti.com/fr*) des ressources qui pourront compléter vos connaissances : les modules de formation « TI Code » à réaliser en ligne ou à télécharger, et la liste des différents webinaires auxquels vous pourrez vous inscrire librement. Enfin, nous vous signalons l'existence de la chaîne YouTube de Texas Instruments Education France (*https://www.youtube.com/user/TIedtechFR*) et, en particulier, la playlist dédiée à la TI-Nspire™ CX (*https://www.youtube.com/playlist?list=PL4V-Xo0EMx4inD-LJL0ZaA4cbi16Tw-id*).

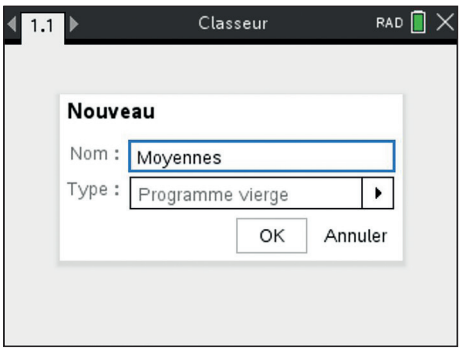
UN PREMIER SCRIPT



Commençons par créer un script **Moyennes** dans lequel nous saisissons le code ci-contre.

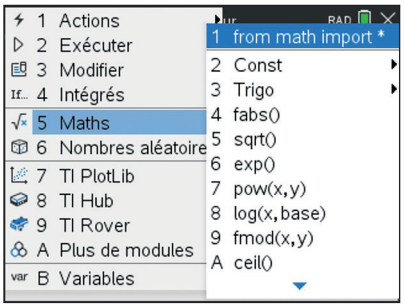
Pour cela, on ajoute une page **Python** et on crée un nouveau script qu'on nomme **Moyennes**.

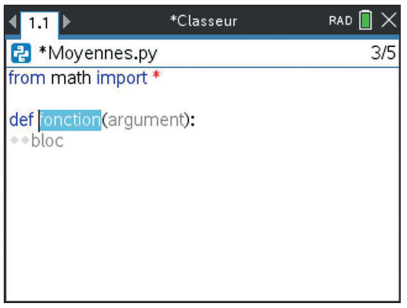


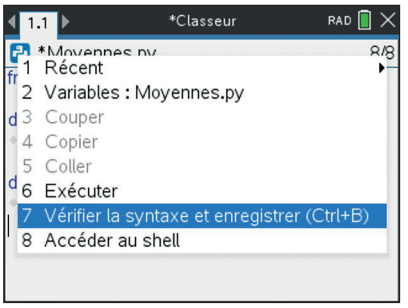


L'interface de l'éditeur Python de la famille des TI-Nspire™ CX (calculatrice ou logiciel version 5.3 afin de pouvoir utiliser Python et la connectivité avec la carte micro:bit) apporte énormément de facilité dans la saisie des scripts. Ainsi, les commandes pour importer les bibliothèques, précharger des structures de contrôle ou encore définir des fonctions sont regroupées dans des menus. De plus, il est possible d'utiliser des fonctions de copier/couper/coller.

Une fois le script saisi, il faut vérifier sa syntaxe et l'enregistrer via **Ctrl + B** ou clic droit (le clic droit étant obtenu en appuyant sur **Ctrl** puis **menu**) et sous-menu **Vérifier la syntaxe et enregistrer**. Enfin, on lance le script (clic droit puis **Exécuter**).

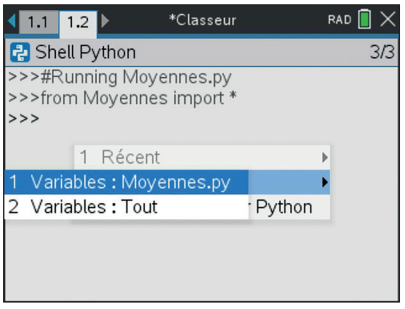


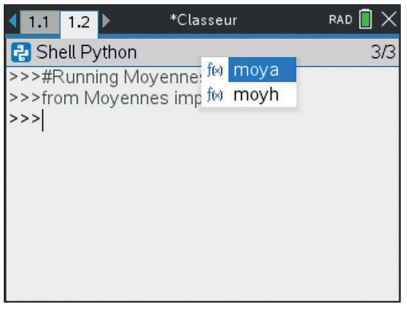


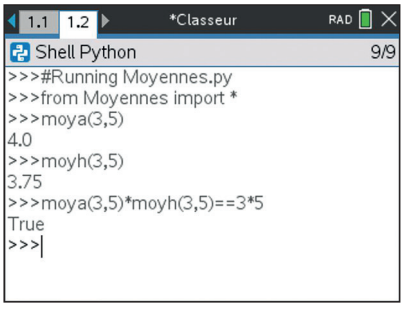


Un nouvel onglet apparaît, numéroté 1.2 (le 2 correspondant au numéro de la page et le 1 au numéro de l'activité). Il contient la console (Shell) Python pour exécuter nos commandes. N'hésitez pas à abuser du clic droit et du sous-menu **Variables** qui permet d'accéder très rapidement aux variables et aux autres fonctions disponibles. Il suffit alors de les sélectionner pour les utiliser.

On vérifie que le produit de la moyenne arithmétique et de la moyenne harmonique de deux nombres est égal au produit de ces deux nombres à l'aide de la commande **moya(3,5)*moyh(3,5)==3*5**.







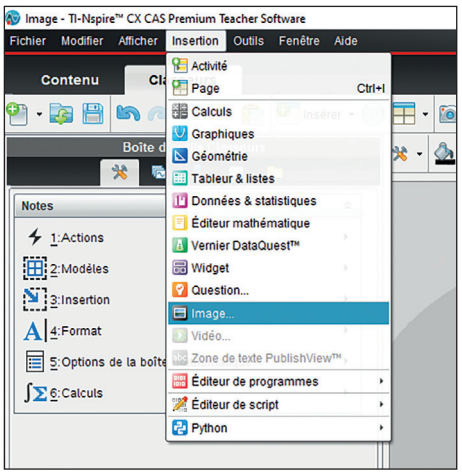
Exercice 1

Complétez le script précédent en créant la fonction `moyg(a,b)` qui retourne la moyenne géométrique des nombres `a` et `b` passés en paramètre.

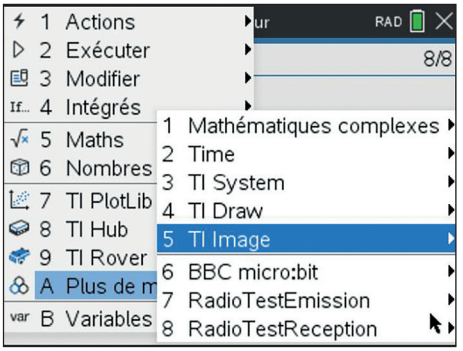
DE NOUVELLES LIBRAIRIES

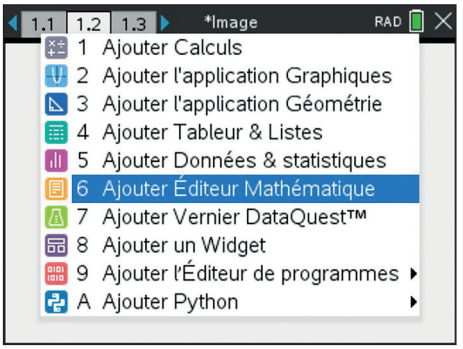
Parmi les nouveautés de l'environnement Python, notons la présence de nouvelles librairies (appelées aussi bibliothèques), accessibles nativement depuis l'interface, ou bien téléchargeables comme nous le verrons dans la suite du chapitre avec la librairie `microbit`.

Pour illustrer ces nouveautés, intéressons-nous à la librairie `ti_image`, en travaillant avec la version logicielle. Commençons par créer un classeur intitulé `Image` et ajoutons une première page `Éditeur Mathématique` (et non `Python`) à notre activité. Puis allons dans le sous-menu `Image…` du menu `Insertion`. Une fenêtre nous invite à sélectionner une image stockée sur notre ordinateur. Ici, nous avons choisi l'une des premières images en couleurs prises par le rover Perseverance sur Mars et publiées par la NASA.



Sur l'image désormais présente sur notre page, cliquons droit et sélectionnons la commande `Nommer l'image`. Appelons cette image `Rover`. Puis ajoutons une page `Python` et créons le script `Image` suivant.





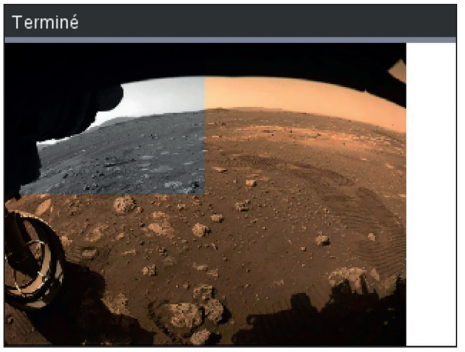
Script

```
from ti_image import *

def niveau_gris(image):
    original = load_image(image)
    transformed = copy_image(original)
    original.show_image(0,0)
    for x in range (original.w//2):
        for y in range (original.h//2):
            rgb = original.get_pixel(x,y)
            r,g,b = .21255,0.71525,.07215
            rgb = int(rgb[0]*r+rgb[1]*g+rgb[2]*b)
            pixel = (rgb,rgb,rgb)
            transformed.set_pixel(x,y,pixel)
    transformed.show_image(0,0)
```

Après avoir recopié l'image originale dans un nouvel objet `transformed`, nous pouvons l'afficher, manipuler des pixels en récupérant les composantes RGB de l'image d'origine, les modifier arithmétiquement et appliquer ces nouvelles composantes à l'image `transformed`. Ensuite, il ne reste plus qu'à l'afficher.

Dans la console Python, exécutons la commande `niveau_gris("Rover")`.

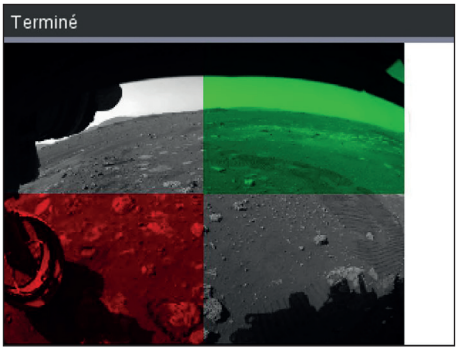


Rover est le nom que nous avons donné à l'image lorsque nous l'avons insérée dans notre classeur. Elle fait donc partie de notre environnement de travail et, bien qu'insérée dans la page **Éditeur Mathématique**, elle est bien accessible dans notre environnement **Python**.

Ce script permet de convertir le premier quart de l'image en niveaux de gris. Les instructions range (`original.h//2`) et range (`original.w//2`) permettent en effet de travailler sur la moitié de la hauteur et de la largeur de l'image originale.

Exercice 2

Complétez le script précédent pour extraire et représenter, dans les trois autres quarts de l'image, la composante rouge de l'image originale, la composante verte et enfin la conversion en niveaux de gris en effectuant la moyenne des trois composantes RGB.



Pour en savoir plus sur les différentes fonctions d'une librairie, consultez le guide correspondant de Texas Instruments sur *https://education.ti.com*.

Un environnement global de travail

Avec les manipulations de cette image, nous avons eu un premier aperçu des échanges possibles entre les contenus générés dans les différentes pages des classeurs. Allons maintenant plus loin en explorant les possibilités offertes par la librairie `ti_system`.

DE L'ENVIRONNEMENT PYTHON VERS L'ENVIRONNEMENT TI-NSPIRE™ CX

Créons d'abord un classeur `Chapitre1`, dans lequel nous ajoutons une première page `Python`. Puis créons le script `LancerDes` suivant.

Script

```
from math import *
from random import *
from ti_system import *

def lancer_des():
    de1 = randint(1,6)
    de2 = randint(1,6)
    return de1 + de2

def repeter_lancer(n):
    resultat = [0]*13
    for i in range(n):
        resultat[lancer_des()]+=1
    return resultat
```

Chapitre 2

Des rationnels aux racines carrées

Dans ce chapitre, nous allons écrire plusieurs scripts pour obtenir autant de décimales que l'on souhaite d'un nombre rationnel. Puis nous procéderons de même dans le cas d'une racine carrée.

Les fractions et Python

D'UNE FRACTION VERS SA FORME DÉCIMALE

On sait que la gestion des entiers en Python est excellente et que Python est capable d'effectuer des calculs avec de grands nombres, la limite étant pratiquement la mémoire de la machine. La situation est très différente avec les fractions (sans avoir recours à une bibliothèque), car dès que l'on écrit $a = \frac{47}{13}$, a sera de type `float`, c'est-à-dire représenté par un décimal, avec un nombre fini de chiffres après la virgule. a ne sera donc pas égal à $\frac{47}{13}$ mais seulement à une valeur approchée de ce nombre (voir chapitre 3).

Commençons par écrire quelques algorithmes pour obtenir un nombre souhaité de décimales d'un rationnel.

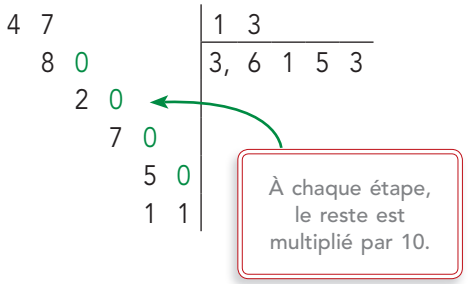
Objectif 1 Écrire une fraction sous forme décimale avec un nombre de décimales quelconque.

Lorsqu'on effectue une division de deux entiers pour aboutir à une écriture sous forme décimale (avec une virgule), on utilise l'algorithme de division classique, dit « de la potence ».

Si le résultat n'admet pas une écriture avec un nombre fini de décimales, cet algorithme ne s'arrêtera donc jamais…

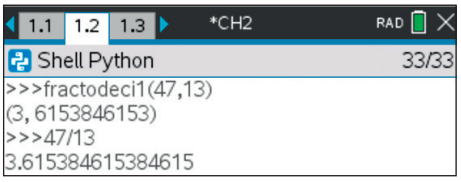
Écrivons une fonction `fractodeci1` qui prend comme arguments deux entiers naturels non nuls a et b et qui renvoie le couple (partie entière, partie décimale) des 10 premières décimales de $\frac{a}{b}$ en utilisant l'algorithme classique de division.

Puis utilisons cette fonction avec le nombre $\frac{47}{13}$.

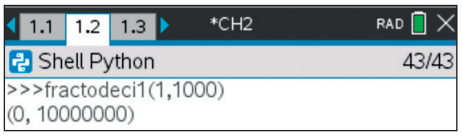


Script

```
def fractodeci1(a,b):
    q,r = a//b , a%b
    deci=0
    for i in range(10):
        q,r = 10*r//b , 10*r%b
        deci = 10*deci+q
    return a//b,deci
```



Si ce script semble bien fonctionner, l'exemple ci-contre soulève un problème de taille… Cela provient du fait que la partie décimale 001 est confondue avec 1, ce qui est normal puisque l'entier 001 va naturellement s'écrire 1. Afin de maintenir les zéros devant 1, utilisons alors un format `string` pour la partie décimale.



Exercice 1

Écrivez une fonction `fractodeci2` qui reprend la fonction `fractodeci1` mais qui renvoie la partie décimale sous forme de `string`. Application : vérifiez-le avec $\frac{1}{1000}$.

Exercice 2

Écrivez une fonction `fractodeci3` qui prend comme arguments trois entiers naturels non nuls a, b et n et qui renvoie le couple (partie entière, partie décimale) sous forme de `string` pour la partie décimale de $\frac{a}{b}$ avec n décimales de précision. Application : donnez les 100 premières décimales du nombre $\frac{47}{13}$.

On remarque que la dernière écriture du corrigé de l'exercice 2 est périodique : intéressons-nous de plus près à ce phénomène.

RECHERCHE DE LA PÉRIODE

Objectif 2 L'écriture sous forme décimale des rationnels est finie ou périodique. Lorsqu'elle est périodique, déterminer cette période.

Théorème 1 Soit a et b deux entiers non nuls tels que $pgcd(a,b)=1$. L'écriture décimale du rationnel $\frac{a}{b}$ est finie si et seulement si les diviseurs premiers de b sont 2 ou 5.

Démonstration : cette écriture est finie si et seulement si $\exists n \in \mathbb{N}^*$ tel que $\frac{a}{b}\times 10^n \in \mathbb{N}$. Étant donné que a et b sont premiers entre eux, cela signifie que b divise 10^n donc les diviseurs premiers de b sont seulement 2 ou 5. Réciproquement, si les diviseurs de b sont 2 ou 5, alors $\exists(\alpha,\beta)\in\mathbb{N}^2$ tel que $b=2^\alpha\times 5^\beta$. En prenant $n=max(\alpha,\beta)$, on a b divise 10^n donc $\frac{a}{b}\times 10^n \in \mathbb{N}$.

Théorème 2 Soit a et b deux entiers non nuls tels que $pgcd(a,b)=1$. L'écriture décimale du rationnel $\frac{a}{b}$ est périodique, la période pouvant éventuellement être réduite à 0 (dans le cas où la fraction admet un nombre fini de décimales).

Dans l'algorithme de division, le reste est multiplié à chaque étape par 10, puis on effectue la division euclidienne de ce nouveau nombre par b.

Le reste obtenu est un entier compris entre 0 et $b-1$. Il y a donc au plus b restes possibles. Ainsi, après $b+1$ étapes, on va forcément aboutir à un reste obtenu lors d'une étape précédente, ce qui assure la périodicité. Si on obtient un reste nul, la division s'arrête et on peut considérer que l'écriture est périodique de période 0.

Dans cet exemple, la liste des restes successifs commence par [8,2,7,5,11].

Pour déterminer cette période, nous allons stocker à chaque étape le reste dans une liste. S'il est déjà dans la liste, cela signifie que la période a été trouvée : il faudra donc s'arrêter. La démonstration qui précède nous garantit de l'arrêt de l'algorithme après au plus $b+1$ étapes.

Écrivons une fonction `fractoperiode1` qui prend comme arguments deux entiers naturels non nuls a et b et qui renvoie l'écriture décimale de $\frac{a}{b}$, en s'arrêtant dès que la période a été trouvée (ce cas inclut aussi celui où l'écriture décimale est finie, la période se terminant alors par 0) et en renvoyant le couple e,d avec e la partie entière et d la partie décimale jusqu'à afficher une période complète.

Ainsi, pour la fraction $\frac{7}{3}=2,6666\ldots$ qu'on notera $2,\underline{6}$, on renverra le couple 2,"6".

Pour la fraction $\frac{47}{13}=3,615384\ 615384\ldots$ notée $3,\underline{615384}$, le script va renvoyer le couple 3,"615384".

Pour la fraction $\frac{189}{50}=3,78$ (écriture décimale finie), on renverra le couple 3,"780".

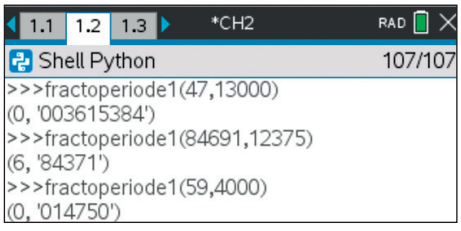
Et enfin pour la fraction $\frac{29713}{4950}=6,00\ 26\ 26\ 26\ 26\ldots$ qu'on note $6,00\underline{26}$, on renverra le couple 6,"0026".

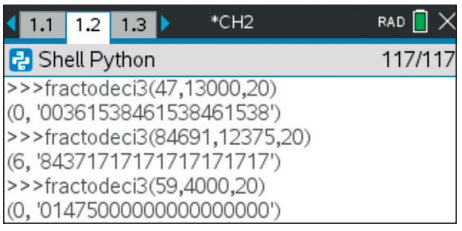
Script

```
def fractoperiode1(a,b):
    q,r=a//b,a%b
    liste=[]
    deci=""
    while r not in liste:
        liste.append(r)
        q,r = 10*r//b,10*r%b
        deci = deci+str(q)
    return a//b,deci
```

Application : utilisons ce script pour les fractions $\frac{47}{1300}$, $\frac{84691}{12375}$ et $\frac{59}{4000}$.

Nous obtenons les résultats ci-après. On peut vérifier nos résultats en affichant les 20 premières décimales de ces fractions à l'aide de la fonction `fractodeci3`.





La faiblesse de ce script est de ne pas isoler la période. Ainsi, avec ces résultats, on ne peut pas affirmer que la fraction $\frac{47}{13}$ admet comme écriture décimale $3,\underline{615384}$, ou bien que $\frac{84691}{12375}$ admet comme écriture décimale $6,84\underline{371}$. Améliorons donc nos scripts !

Objectif 3 Écrire un script qui permet d'afficher la partie entière d'une fraction, puis le début de sa partie décimale et enfin sa période. On renverra donc un triplet.
Si la partie décimale n'est constituée que de la partie périodique, on renverra un couple (partie entière, période).

Donnons quelques exemples afin de bien comprendre notre objectif.

Ainsi, le script devra, à partir de $\frac{59}{40} = 1{,}475$, renvoyer le triplet `1,"475","0"` pour signifier que la période est 0 et donc que l'écriture décimale est finie.

Pour $\frac{84691}{12375} = 6{,}843\underline{71}$, il devra renvoyer `6,"843","71"`, indiquant que le début de la partie décimale est 6,843 suivi de la période 71.

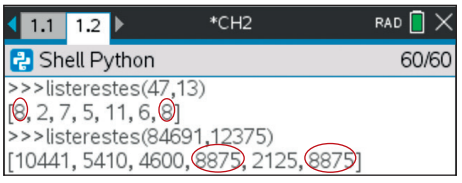
Et enfin pour $\frac{47}{13} = 3{,}\underline{615384}$, il devra renvoyer seulement le couple `3,"615384"`.

On pourra ainsi reconstituer facilement l'écriture décimale complète de n'importe quelle fraction.

La période ne commençant pas toujours par le premier chiffre après la virgule, il faut pouvoir la repérer, puis l'isoler. Écrivons dans un premier temps une fonction `listerestes` qui prend comme arguments deux entiers naturels non nuls *a* et *b* et qui renvoie la liste des restes successifs dans l'algorithme de la division de *a* par *b* jusqu'à obtenir un reste déjà présent.

Script

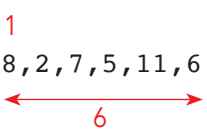
```
def listerestes(a,b):
    q,r=a//b,a%b
    liste=[]
    while r not in liste:
        liste.append(r)
        q,r = 10*r//b,10*r%b
    liste.append(r)
    return liste
```



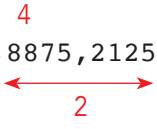
La périodicité des restes va nous permettre d'en déduire l'écriture décimale complète, l'emplacement du premier doublon jouant un rôle important dans notre recherche.

Soit p et $q \in \mathbb{N}^*$. Afin de bien identifier la période et sa taille, écrivons une fonction `cherchedoublon` qui prend comme arguments une liste d'entiers naturels du type $[x_0,\ldots,x_{p-1},n,x_p,\ldots,x_{p+q},n]$ avec les $(x_i)_{0\leq i\leq p+q}$ distincts deux à deux (le dernier terme de la liste apparaît une autre fois dans la liste, c'est le doublon !) et qui renvoie un couple $p+1$ (la **place** où le **doublon** apparaît la **première fois**), $q+2$ (la **distance** entre les deux emplacements de *n*, ce qui correspondra à la taille de la période).

Par exemple, `cherchedoublon([8,2,7,5,11,6,8])` va renvoyer 1,6.

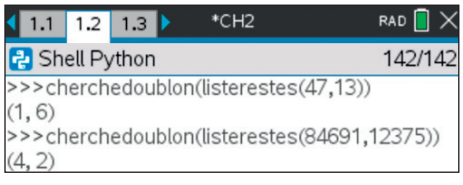


De même, `cherchedoublon([10441,5410,4600,8875,2125,8875])` va renvoyer 4,2.



Script

```
def cherchedoublon(liste):
    a=liste[-1]
    u=liste.index(a)
    liste.remove(a)
    v=liste.index(a)
    return u+1,v-u+1
```



Chapitre 3

0.1 + 0.1 + 0.1 = 0.3 ?

Dans ce chapitre, nous allons nous pencher sur un cas classique de l'arithmétique qui fait intervenir des réels en informatique. Nous nous intéresserons à la représentation des nombres en virgule flottante en Python pour répondre à la question posée dans le titre.

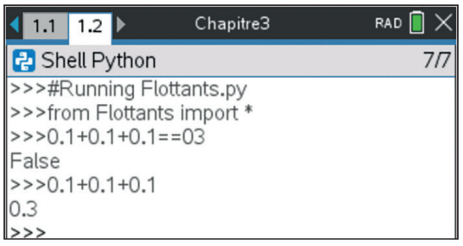
0.1 + 0.1 + 0.1 n'est pas égal à 0.3 ?

BO PREMIÈRE SPÉ NSI
Calculer sur quelques exemples la représentation de nombres réels

Objectif 1 Une première approche de la comparaison de 0.1 + 0.1 + 0.1 avec 0.3.

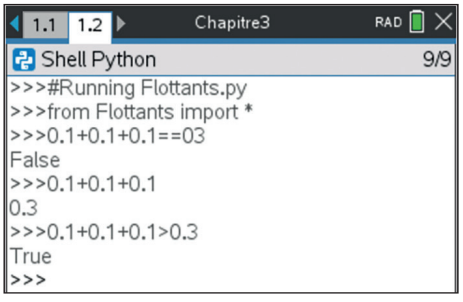
Que nous utilisions Scratch ou Python, nous sommes parfois confrontés en tant que professeurs de mathématiques à des comportements de ces langages qui peuvent nous interpeller. Ainsi, dans Scratch 3.0, réalisons le très court programme ci-contre et observons le résultat : `false` !

De même, en Python, saisissons dans la console la commande équivalente `0.1+0.1+0.1==0.3` : `False` ! Pourtant, si nous sollicitons Python (et il en serait de même dans Scratch) pour effectuer le calcul `0.1+0.1+0.1`, il retourne bien 0.3. Ainsi, le shell Python et Scratch renvoient `False` si l'on teste l'égalité.



Allons un tout petit peu plus loin dans notre essai en saisissant la commande, toujours dans la console Python, `0.1 + 0.1 + 0.1 > 0.3`. Python renvoie alors `True`.

Nous allons donc essayer de comprendre pourquoi ces langages renvoient ces résultats. Sont-ils mathématiquement prévisibles ?



Représentation binaire des nombres réels

Voici quelques premiers éléments de réflexion au sujet de cette représentation.

REPRÉSENTATION D'UN DÉCIMAL EN BASE 10

Au collège, on étudie, en cycle 3, la décomposition des nombres entiers et des nombres décimaux. Celle-ci a été également abordée dans le chapitre 2 de ce livre.

Ainsi pour un nombre entier à $n \in \mathbb{N}^*$ chiffres, on a :

$$a_{n-1}\ldots a_1a_0 = a_{n-1}\times 10^{n-1} + \ldots + a_1\times 10^1 + a_0\times 10^0.$$

Et $a_{n-1}, \ldots, a_1, a_0 \in \{0, 1, 2, \ldots, 9\}$.

Par exemple :

$$275 = 2\times 10^2 + 7\times 10^1 + 5\times 10^0.$$

Et par extension pour un nombre à $m \in \mathbb{N}^*$ décimales :

$$a_{n-1}\ldots a_1a_0, b_1b_2\ldots b_m$$

$$= a_{n-1}\times 10^{n-1} + \ldots + a_1\times 10^1 + a_0\times 10^0 + b_1\times\frac{1}{10^1} + b_2\times\frac{1}{10^2} + \ldots + b_m\times\frac{1}{10^m}.$$

Par exemple :

$$174{,}3125 = 1\times 10^2 + 7\times 10^1 + 4\times 10^0 + 3\times\frac{1}{10^1} + 1\times\frac{1}{10^2} + 2\times\frac{1}{10^3} + 5\times\frac{1}{10^4}.$$

REPRÉSENTATION D'UN DÉCIMAL EN BINAIRE

Le principe est le même pour la représentation binaire :

$$a_{n-1}\ldots a_1a_0 \;=\; a_{n-1}\times 2^{n-1} + \ldots + a_1\times 2^1 + a_0\times 2^0.$$

Et $a_{n-1}, \ldots, a_1, a_0 \in \{0, 1\}$ éléments que l'on appellera « bits ».

Par exemple :

$$10101110 = 1\times 2^7 + 0\times 2^6 + 1\times 2^5 + 0\times 2^4 + 1\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0$$

$$= 128 + 32 + 8 + 4 + 2 = 174$$

Et par extension pour un nombre à m décimales :

$$a_{n-1}\ldots a_1a_0, b_1b_2\ldots b_m = a_{n-1}\times 2^{n-1} + \ldots + a_1\times 2^1 + a_0\times 2^0 + b_1\times\frac{1}{2^1} + b_2\times\frac{1}{2^2} + \ldots + b_m\times\frac{1}{2^m}$$

Par exemple :

$$10101110.0101 = 1\times 2^7 + 1\times 2^5 + 1\times 2^3 + 1\times 2^2 + 1\times 2^1 + 1\times\frac{1}{2^2} + 1\times\frac{1}{2^4}$$

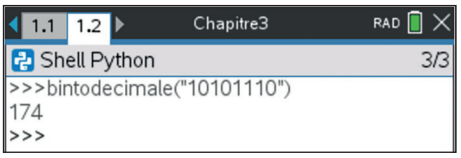
$$= 174 + 0{,}25 + 0{,}0625 = 174{,}3125$$

Exercice 1

Écrivez une fonction `bintodecimale` qui prend comme argument `b`, de type `string` (représentant un entier écrit en binaire) et qui renvoie son écriture décimale. Vous pourrez utiliser les commandes

`len` qui renverra la longueur de la chaîne de caractères (et donc le nombre de bits) et `int(k)` qui renvoie le nombre entier représenté par la chaîne `k`.

Vérifions la conversion de 10101110, on doit retrouver le résultat obtenu dans notre premier exemple.



DE LA BASE 2 VERS LA BASE 10

Objectif 2 Montrer qu'un nombre décimal en base 2 a une représentation finie en base 10 et écrire une fonction qui permet d'effectuer cette conversion.

Revenons maintenant à la représentation des nombres décimaux en base 2 et leur conversion en base 10.

Théorème 1 Un nombre binaire avec une partie entière nulle et une partie décimale non nulle, c'est-à-dire de la forme $0,b_1b_2b_3\ldots b_n$ avec $b_1, b_2, \ldots, b_n \in \{0, 1\}$ et $n \in \mathbb{N}^*$ est, en base 10, une fraction de la forme $\frac{a}{2^n}$ avec $a \in \mathbb{N}$ et $a < 2^n$.

Démonstration : soit $x = 0,b_1b_2b_3\ldots b_n{}_{\ 2} = \sum_{k=1}^{n} \frac{b_k}{2^k} = \sum_{k=1}^{n} \frac{2^{n-k}b_k}{2^n}$. On pose $a = \sum_{k=1}^{n} 2^{n-k}b_k$.

Étant donné que $\forall k \in \mathbb{N}\ \ 1 \leq k \leq n$, on a $b_k \in \{0;1\}$, cela entraîne que $0 \leq a \leq \sum_{k=1}^{n} 2^{n-k}$.

Or $\sum_{k=1}^{n} 2^{n-k} = \sum_{k=0}^{n-1} 2^k = \frac{1-2^n}{1-2} = 2^n - 1$. On a donc bien $0 \leq a < 2^n$.

Corollaire 1 Tout nombre binaire de la forme $a_{n-1}\ldots a_1a_0,b_1b_2\ldots b_m$ avec $n \in \mathbb{N}^*$ et $m \in \mathbb{N}^*$ et $a_0, \ldots a_{n-1}, b_1, \ldots, b_m \in \{0;1\}$ a en base 10 un nombre fini de chiffres après la virgule.

Démonstration : en considérant le théorème précédent et le premier théorème du chapitre 2, le résultat est immédiat.

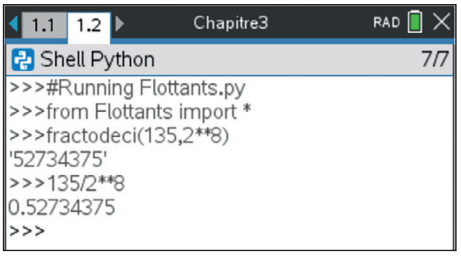
Afin d'obtenir l'écriture en base 10 exacte de ces nombres en base 2 avec décimales, restons en base 10 et écrivons une fonction `fractodeci(a,b)` qui prend comme arguments deux entiers a et b (en base 10 avec a<b) et qui renvoie la partie décimale exacte de $\frac{a}{b}$ au format `string`.

Attention ! On suppose ici que $\frac{a}{b}$ a une écriture décimale finie sinon la boucle ci-après ne s'arrêtera jamais.

Grace au corollaire précédent, étant donné que par la suite nous manipulerons des nombres de la forme $\frac{a}{2^n}$ avec a et n des entiers, cela sera toujours le cas.

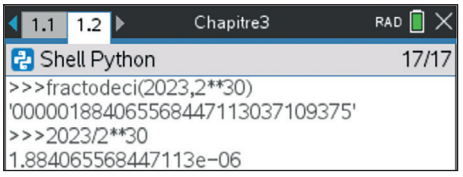
Script

```
def fractodeci(a,b)
    q,r = a//b,a%b
    deci = ""
    while r > 0:
        q,r = 10*r//b,10*r%b
        deci = deci + str(q)
    return deci
```



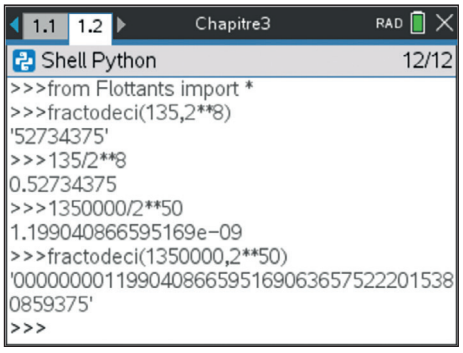
Testons notre script en cherchant la partie décimale exacte de $\frac{135}{2^8}$.

On vérifie en console notre résultat en comparant les résultats obtenus à partir de notre fonction et le résultat renvoyé par la console lorsqu'on effectue le calcul directement.



On rappelle l'importance de l'usage du format `string` pour tenir compte d'éventuels 0 au début de la partie décimale.

L'un des intérêts de ce script est qu'il va nous permettre d'afficher, en Python, plus de décimales que ne le permet l'affichage des nombres réels qu'il manipule.



Essayons d'obtenir, en Python, la valeur exacte du quotient $\frac{1350000}{2^{50}}$.

Ainsi $\frac{1350000}{2^{50}}$ vaut exactement

0, 000000001199040866595169063657522201538 0859375.

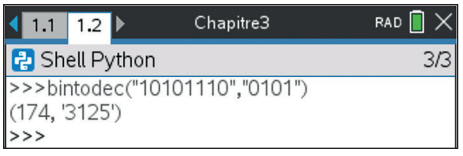
Python, à l'aide d'un `float`, ne nous affiche qu'une valeur approchée : $1{,}199040866595169 \times 10^{-9}$.

Passons maintenant à la conversion d'un nombre binaire avec des chiffres après la virgule en un nombre décimal (exact) en base 10.

Exercice 2

Écrivez une fonction `bintodec(e,d)` qui prend comme arguments `e` et `d`, de type `string`, représentant la partie entière et la partie décimale d'un nombre binaire, et qui renvoie sa valeur (donc un nombre décimal) en base 10 sous la forme `(partie entière, partie décimale)`. Vous utiliserez la fonction `fractodeci(a,b)` précédente pour la représentation de la partie décimale.

Reprenons le nombre de notre exemple précédent 10101110.0101. On retrouve bien 174 et 3125.



Mais alors quel est le lien avec la représentation des nombres binaires ? Et pourquoi 0.1 + 0.1 + 0.1 est supérieur à 0.3 ? Nous y venons.

Conversion de la représentation décimale vers la représentation binaire

Objectif 3 Déterminer la représentation de 0.1_{10} en base 2 et les problèmes engendrés par des représentations décimales infinies en base 2.

Pour aborder ce problème de la conversion, nous allons dans un premier temps convertir la partie entière en sa représentation binaire, puis dans un second temps convertir la partie décimale en sa représentation binaire. Nous mettrons en œuvre les deux algorithmes de conversion.

Chapitre 4

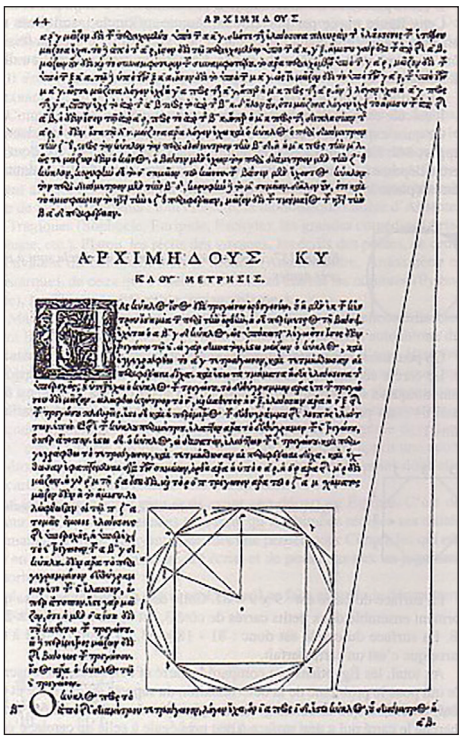
Pi dans tous ses états

Le nombre π a fait l'objet de nombreuses recherches depuis l'Antiquité. Étudions de plus près certaines de ces découvertes qui ont jalonné l'histoire des mathématiques et utilisons-les pour obtenir quelques-unes des fameuses décimales.

L'approche géométrique d'Archimède

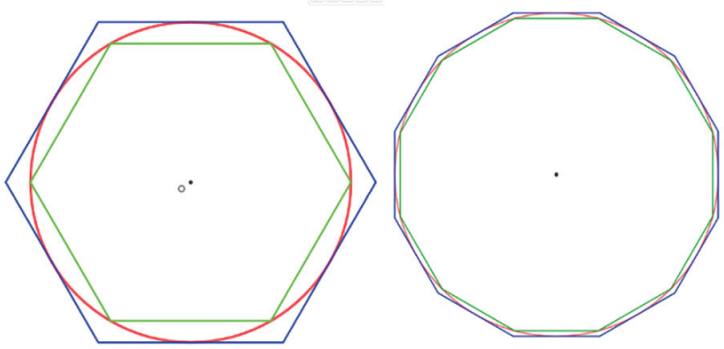
Objectif 1 Découvrir la méthode d'Archimède.

Archimède de Syracuse (287-212 av. J.-C.) est l'un des plus grands savants de l'Antiquité. À la fois ingénieur, physicien et mathématicien, il est notamment célèbre pour ses principes du levier et de la vis sans fin, encore utilisée de nos jours, sans oublier son légendaire « eurêka ».


44 ΑΡΧΙΜΗΔΟΥΣ

ΑΡΧΙΜΗΔΟΥΣ ΚΥ
ΚΑΙ ΜΕΤΡΗΣΙΣ.

L'algèbre et la trigonométrie n'existaient pas à son époque, aussi Archimède a imaginé une méthode géométrique pour déterminer une valeur approchée de π, comme le montre l'extrait du manuscrit ci-contre. Elle consiste à encadrer le demi-périmètre d'un cercle de rayon 1 qui vaut π par a_n le demi-périmètre du polygone régulier inscrit dans le cercle de 6×2^n côtés et par b_n le demi-périmètre du polygone régulier circonscrit au cercle à 6×2^n côtés avec n un entier naturel.



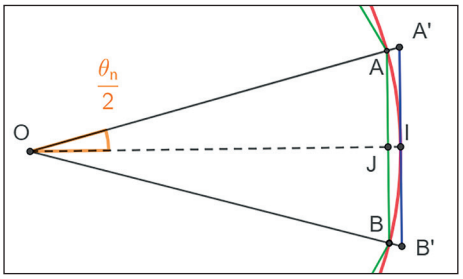
Nous allons utiliser une approche géométrique et trigonométrique pour le calcul de π.

Soit θ_n l'angle $\widehat{AOB}$ avec O le centre du cercle et [AB] un côté du polygone. $\theta_n = \frac{2\pi}{6.2^n} = \frac{\pi}{3.2^n}$.

On admettra par la suite que l'encadrement des polygones réguliers et du cercle permet d'écrire AB < $\widehat{AB}$ < A'B'.

Notons u_n = AB la longueur d'un côté du polygone à 6.2^n côtés inscrit et v_n = A'B' la longueur d'un côté du polygone à 6.2^n côtés circonscrit, ainsi $\forall n \in \mathbb{N}\ u_n < \theta_n \times 1 < v_n$

$\Leftrightarrow u_n < \frac{\pi}{3.2^n} < v_n \Leftrightarrow a_n < \pi < b_n$ avec $a_n = 3.2^n u_n$ et $b_n = 3.2^n v_n$.

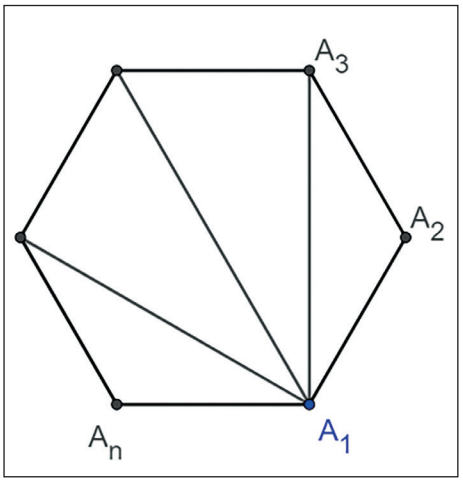


Lemme 1 $\forall n \in \mathbb{N}\ u_n = 2\sin\left(\frac{\pi}{6.2^n}\right)$ et $v_n = 2\tan\left(\frac{\pi}{6.2^n}\right)$

Démonstration : $\forall n \in \mathbb{N}\ \ u_n = 2AJ = 2\sin\left(\frac{\theta_n}{2}\right) = 2\sin\left(\frac{\pi}{6.2^n}\right)$ et $v_n = 2A'I$. Or $\tan\left(\frac{\theta_n}{2}\right) = \frac{A'I}{OI} = A'I$ donc $v_n = 2\tan\left(\frac{\pi}{6.2^n}\right)$.

Lemme 2 Soit $n \in \mathbb{N}$, $n \geq 4$ et $A_1A_2\ldots A_n$ les sommets d'un polygone régulier. L'angle $\widehat{A_1A_2A_3}$ entre deux côtés consécutifs vaut $\frac{n-2}{n}\pi$.

Démonstration : à partir du sommet A_1, on trace les $n-3$ segments $[A_1A_p]$ pour $3 \leq p \leq n-1$ pour former $n-2$ triangles. La somme des angles de tous ces triangles est $(n-2)\pi$, ce qui correspond à n fois l'angle $\widehat{A_1A_2A_3}$. Ainsi l'angle $\widehat{A_1A_2A_3}$ entre deux côtés consécutifs vaut $\frac{n-2}{n}\pi$.



Théorème 1 $\forall n \in \mathbb{N}\ \ u_{n+1} = \sqrt{\frac{u_n v_{n+1}}{2}}$

Démonstration 1 à l'aide de la trigonométrie :

$$\forall n \in \mathbb{N}\ \ u_n v_{n+1} = 2\sin\left(\frac{\pi}{6\times 2^n}\right) \times 2\tan\left(\frac{\pi}{6\times 2^{n+1}}\right) = 4\sin\left(\frac{\pi}{6\times 2^n}\right) \times \frac{\sin\left(\frac{\pi}{6\times 2^{n+1}}\right)}{\cos\left(\frac{\pi}{6\times 2^{n+1}}\right)}$$

$$= 8\sin\left(\frac{\pi}{6\times 2^{n+1}}\right)\cos\left(\frac{\pi}{6\times 2^{n+1}}\right) \times \frac{\sin\left(\frac{\pi}{6\times 2^{n+1}}\right)}{\cos\left(\frac{\pi}{6\times 2^{n+1}}\right)} = 8\sin^2\left(\frac{\pi}{6\times 2^{n+1}}\right) = 2u_{n+1}^2$$

Ce qui prouve que $u_{n+1} = \sqrt{\frac{u_n v_{n+1}}{2}}$.

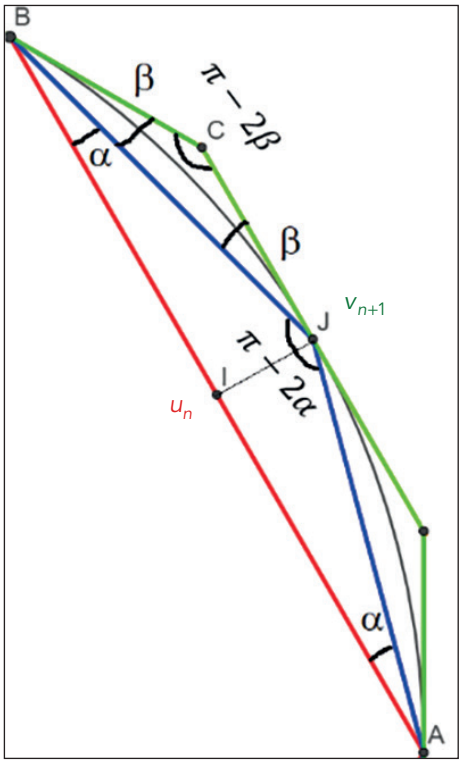
Démonstration 2 à l'aide de la géométrie :

α représente l'angle $\widehat{ABJ}$ et $\pi - 2\alpha$ l'angle $\hat{J}$ du polygone régulier à $6\times 2^{n+1}$ côtés inscrit. Ainsi d'après le lemme 2, $\pi - 2\alpha = \dfrac{6.2^{n+1}-2}{6.2^{n+1}}\pi$ donc $\alpha = \dfrac{\pi}{6\times 2^{n+1}}$.

De même $\beta = \dfrac{\pi}{6\times 2^{n+1}}$ car il s'agit de l'angle d'un polygone régulier à $6\times 2^{n+1}$ côtés circonscrit. Ainsi $\alpha = \beta = \dfrac{\pi}{6\times 2^{n+1}}$.

Les triangles AJB et BCJ sont donc semblables, ce qui prouve que $\dfrac{BJ}{BC} = \dfrac{BA}{BJ}$.

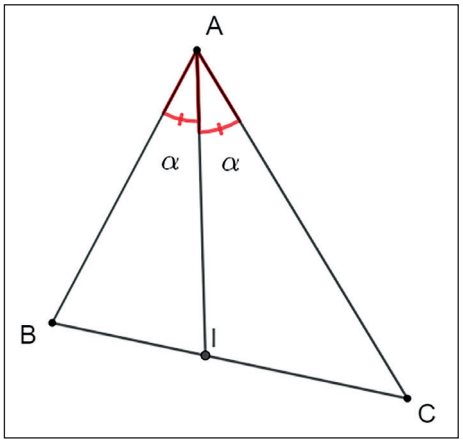
Ainsi $\dfrac{u_{n+1}}{\frac{1}{2}v_{n+1}} = \dfrac{u_n}{u_{n+1}}$ ce qui nous donne $u_{n+1}^2 = \dfrac{1}{2}u_n v_{n+1}$ soit $u_{n+1} = \sqrt{\dfrac{u_n v_{n+1}}{2}}$.

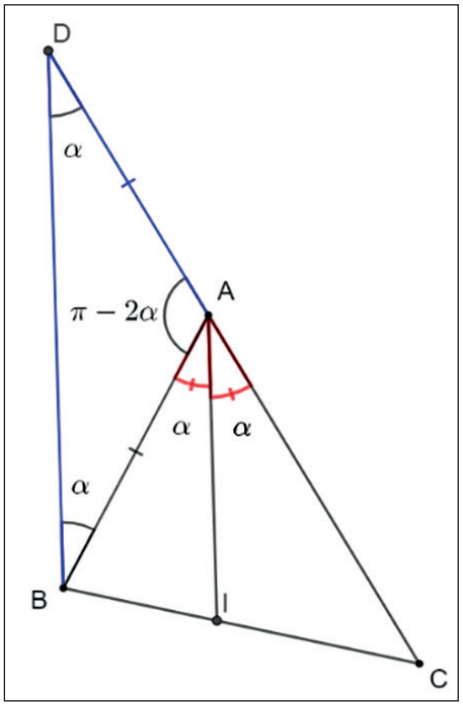


En rouge, polygone régulier à 6×2^n côtés inscrit dans le cercle.
En bleu, polygone régulier à $6\times 2^{n+1}$ côtés inscrit dans le cercle.
En vert, polygone régulier à $6\times 2^{n+1}$ côtés circonscrit au cercle.

Lemme 3 Soit ABC un triangle et $I \in [BC]$ tel que (AI) soit la bissectrice du triangle ABC issue de A.
Alors on a : $\dfrac{AC}{AB} = \dfrac{IC}{IB}$.

Démonstration : cette égalité nous faisant penser au théorème de Thalès, nous allons créer une configuration de Thalès.





Soit (Δ) la parallèle à (AI) passant par B. Soit D le point d'intersection de (Δ) et (AC). Par construction, les triangles AIC et DCB sont semblables.

Par construction, $\widehat{BDA} = \widehat{IAC} = \alpha$ et $\widehat{BAD} = \pi - 2\alpha$. On en déduit que $\widehat{ABD} = \alpha$ donc le triangle ABD est isocèle en A, ce qui prouve que AD = AB.

En utilisant le théorème de Thalès dans le triangle CBD, on a $\dfrac{CI}{CB} = \dfrac{CA}{CD}$ ce qui nous donne $\dfrac{CI}{CI+IB} = \dfrac{CA}{CA+AD}$.

Soit $CI(CA+AD) = CA(CI+IB)$

$\Leftrightarrow CI\times CA + CI\times AD = CA\times CI + CA\times IB$

$\Leftrightarrow CI\times AD = CA\times IB$

D'où $\dfrac{IC}{IB} = \dfrac{AC}{AD}$. Or AD = AB d'où le résultat.

Théorème 2 $\forall n \in \mathbb{N} \quad v_{n+1} = \dfrac{u_n v_n}{u_n + v_n} \Leftrightarrow \dfrac{1}{v_{n+1}} = \dfrac{1}{u_n} + \dfrac{1}{v_n}$

Démonstration 1 à l'aide de la géométrie :

Dans le triangle OB'J, la droite (OC) est la bissectrice issue de O. D'après le lemme 3, on obtient donc : $\dfrac{CB'}{CJ} = \dfrac{OB'}{OJ}$.

$CJ = \dfrac{1}{2}v_{n+1}$ et $CB' = B'J - CJ = \dfrac{1}{2}v_n - \dfrac{1}{2}v_{n+1}$

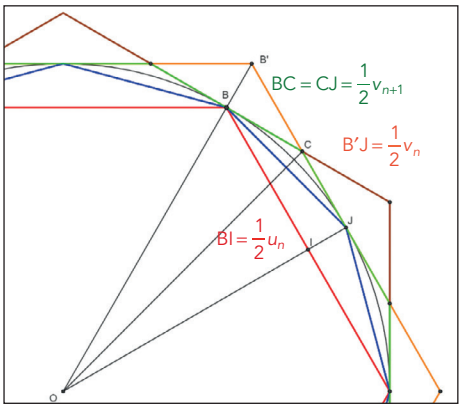
donc $\dfrac{CB'}{CJ} = \dfrac{\frac{1}{2}v_n - \frac{1}{2}v_{n+1}}{\frac{1}{2}v_{n+1}} = \dfrac{v_n - v_{n+1}}{v_{n+1}} = \dfrac{OB'}{OJ}$.

Or $OJ = OB$ donc $\dfrac{OB'}{OJ} = \dfrac{OB'}{OB}$. De plus, dans le triangle OB'J, les droites (B'J) et (BI) sont parallèles donc, d'après le théorème de Thalès, $\dfrac{OB'}{OB} = \dfrac{B'J}{BI}$ ainsi

$\dfrac{OB'}{OJ} = \dfrac{B'J}{BI}$ or $B'J = \dfrac{1}{2}v_n$ et $BI = \dfrac{1}{2}u_n$ donc

$\dfrac{OB'}{OJ} = \dfrac{v_n}{u_n}$. Ainsi nous avons l'égalité

$$\frac{v_n - v_{n+1}}{v_{n+1}} = \frac{v_n}{u_n} \Leftrightarrow u_n(v_n - v_{n+1}) = v_n v_{n+1}$$
$$\Leftrightarrow u_n v_n - u_n v_{n+1} = v_n v_{n+1}$$
$$\Leftrightarrow u_n v_n = v_n v_{n+1} + u_n v_{n+1}$$
$$\Leftrightarrow v_{n+1}(u_n + v_n) = u_n v_n \Leftrightarrow v_{n+1} = \frac{u_n v_n}{u_n + v_n}$$



En rouge, polygone régulier à 6×2^n côtés inscrit dans le cercle.
En bleu, polygone régulier à $6 \times 2^{n+1}$ côtés inscrit dans le cercle.
En vert, polygone régulier à $6 \times 2^{n+1}$ côtés circonscrit au cercle.
En marron, polygone régulier à 6×2^n côtés circonscrit au cercle.
En orange, polygone régulier à 6×2^n côtés circonscrit au cercle.

Démonstration 2 à l'aide de la trigonométrie :

$$\forall n \in \mathbb{N} \quad \frac{1}{u_n} + \frac{1}{v_n} = \frac{1}{2\sin\left(\frac{\pi}{6\times 2^n}\right)} + \frac{1}{2\tan\left(\frac{\pi}{6\times 2^n}\right)} = \frac{1}{2}\left(\frac{1}{\sin\left(\frac{\pi}{6\times 2^n}\right)} + \frac{\cos\left(\frac{\pi}{6\times 2^n}\right)}{\sin\left(\frac{\pi}{6\times 2^n}\right)}\right) = \frac{1}{2}\left(\frac{1+\cos\left(\frac{\pi}{6\times 2^n}\right)}{\sin\left(\frac{\pi}{6\times 2^n}\right)}\right)$$
$$= \frac{1}{2}\left(\frac{1+2\cos^2\left(\frac{\pi}{6\times 2^{n+1}}\right)-1}{2\sin\left(\frac{\pi}{6\times 2^{n+1}}\right)\cos\left(\frac{\pi}{6\times 2^{n+1}}\right)}\right) = \frac{1}{2}\left(\frac{2\cos^2\left(\frac{\pi}{6\times 2^{n+1}}\right)}{2\sin\left(\frac{\pi}{6\times 2^{n+1}}\right)\cos\left(\frac{\pi}{6\times 2^{n+1}}\right)}\right) = \frac{1}{2}\times\frac{1}{\tan\left(\frac{\pi}{6\times 2^{n+1}}\right)} = \frac{1}{v_{n+1}}$$

En résumé, les suites (u_n) et (v_n) sont définies par :

$\forall n \in \mathbb{N} \quad v_{n+1} = \dfrac{u_n v_n}{u_n + v_n}$ et $u_{n+1} = \sqrt{\dfrac{u_n v_{n+1}}{2}}$.

Calculons u_0 et v_0 : d'après le lemme 1, on a vu que $u_0 = 2\sin\left(\dfrac{\pi}{6}\right) = 1$ et $v_0 = 2\tan\left(\dfrac{\pi}{6}\right) = \dfrac{2\sqrt{3}}{3}$. En outre, on a : $\forall n \in \mathbb{N} \quad a_n = 3.2^n u_n$ et $b_n = 3.2^n v_n$ avec $\forall n \in \mathbb{N} \; a_n < \pi < b_n$.

Objectif 2 Déterminer un algorithme pour déterminer une valeur approchée de π à l'aide de la méthode d'Archimède.

Écrivons alors une fonction `archimede` qui prend comme argument n un entier naturel et qui renvoie le couple a_n, b_n qui encadre π.