AU LYCÉE AVEC

PYTHON

Python en seconde

Aller à l'essentiel, tel est le credo de ce livre. Apprendre vite et bien grâce à un style didactique innovant.

Ce livre vous propose de découvrir le langage Python avec les notions qui sont au programme de la classe de seconde au lycée.

Le principe directeur est de réduire le texte au minimum pour gagner du temps et se concentrer sur ce qui est vraiment important. Une première partie présente les notions qui sont au programme et une seconde propose un certain nombre d'exercices et d'exemples commentés et entièrement corrigés. On y aborde des thèmes classiques comme les nombres triangulaires, les factorielles, les fractions égyptiennes, le problème de Bâle (somme des inverses des carrés), la distance de Hamming, les nombres premiers, la conjecture de Syracuse, l'algorithme de Héron, la dichotomie, etc.

Diplômé de l'ENSEA et de Supélec, Éric Pruvost a développé un style pédagogique original au travers de ses expériences dans l'industrie, la recherche et l'enseignement supérieur.

À l'été 2016, Éric Pruvost lance la chaîne Youtube MathsPlusUn qui est immédiatement remarquée par le journaliste Paul Conge pour le magazine l'Étudiant. Le canal compte aujourd'hui plus de 50 000 abonnés. Éric Pruvost est à l'origine de la série de livres *Voyages sur Maths* pour les Bac+1/Bac+2 et de la collection *Surfe les Maths !* pour les cycles 1 à 4.

Code ISBN : 9798540324373
Marque éditoriale : Independently published

Graphisme : Lylie Romaine

AU LYCÉE AVEC PYTHON – PYTHON EN SECONDE

AU LYCÉE AVEC

PYTHON

Python en seconde

Éric Pruvost

Algorithmique & informatique
Juste l'essentiel !

Au lycée avec Python

Python en seconde

Par le créateur de la chaîne YouTube mathsplusun
Éric Pruvost

Maths+1
2021

Table des matières

Avant-propos

Ce livre vous propose de découvrir le langage Python avec les notions qui sont au programme de la classe de seconde au lycée.

Le principe directeur est de réduire le texte au minimum pour gagner du temps et se concentrer sur l'essentiel. Une première partie présente les notions qui sont au programme et une seconde partie propose un certain nombre d'exercices et d'exemples commentés et corrigés.

La plupart du temps, les exercices « évidents » ont été évités. Plusieurs activités sont plus longues et abordent des thèmes classiques comme la conjecture de Syracuse par exemple.

La première partie (notions essentielles) aborde les thèmes suivants :

1. Généralités sur les langages, repères historiques, concepteur du langage, Python et les autres langages, code source, outils de développement ;
2. Rudiments de programmation : fonction `print()` et chaînes de caractères ;
3. La notion de variable : déclaration, initialisation, typage, types `int`, `float` et `bool`, échange de variables, affectations multiples, utilisation des f-strings ;
4. Les nombres et les booléens : opérateurs booléens, addition, soustraction, multiplication, division, division euclidienne, différence entre « = » et « == », opérateurs de comparaison, la fonction `bool`, le littéral `None`, opérateur % (reste de la division euclidienne) ;
5. Les structures conditionnelles : `if`, `elif` et `else`, l'importance de l'indentation et le rôle du caractère « : » en Python ;
6. Boucles de type `for in range` (on connaît à l'avance le nombre d'itérations), les boucles de type `while` (on

ne connaît pas *a priori* le nombre d'itérations), boucle `for` pour les chaînes de caractères ;

7. Fonctions mathématiques pour la seconde, diverses utilisations de l'instruction `import`, le module `random` ;
8. La notion de fonction, le mot-clé `def`, le mot-clé `return`, importance de l'indentation, différence entre *argument* et *paramètre*, exemple des années bissextiles, portée des variables, le mot-clé `global`, valeur par défaut d'un argument, les étiquettes.

La seconde partie propose trois sous-parties exclusivement consacrées à des exercices et à des exemples entièrement commentés et corrigés. C'est la partie la plus importante en ce sens qu'elle vous apporte de nombreux exemples de code source et de traitements de différents types de problèmes.

Les premiers exercices sont basiques puis sont abordés des sujets comme les nombres triangulaires, les factorielles, les fractions égyptiennes, le problème de Bâle (somme des inverses des carrés), la distance de Hamming, les nombres premiers, la conjecture de Syracuse, l'algorithme de Héron, la dichotomie, etc.

Enfin, la dernière sous-partie aborde quelques notions élémentaires relatives au module `turtle` qui permet de dessiner et de réaliser des animations à l'écran. Nous en profitons pour donner un aspect visuel à la conjecture de Syracuse et commencer à explorer la notion de fractale avec une esquisse du triangle de Sierpinski.

La statistique nécessitant la notion de liste qui n'est pas au programme de seconde est traitée dans le volume *Python en Première*.

Introduction

Langage de programmation ?

Un langage de programmation est un langage formel (artificiel) qui permet d'écrire des programmes informatiques (des logiciels).

Le texte écrit à l'aide d'un langage de programmation est appelé code source.

Quelques repères

Créateur : Guido van Rossum

Origine : langage de script du système d'exploitation Amoeba (ordinateur distribué).

Origine du nom : la série télé britannique de 45 épisodes intitulée Monty Python's Flying Circus diffusée dans les années 1970.

Repères historiques

1972 : première version du langage C

1983 : première version du langage C++

1986 : Gossum rejoint le projet Amoeba

1989 : première version de Python qui reprend le principe de l'indentation du langage ABC

1991 : première version publique (0.9.0)

1995 : Python 1.5

1996 : Java 1.0

2000 : Python 2.0

2008 : Python 3

2020 : Python 3.9

Premières notions

La fonction print()

```
print("Bonjour !")
```

```
Bonjour !
```

`print()` : fonction prédéfinie du langage

Permet d'écrire dans la console.

`"Bonjour !"` : est une chaîne de caractères

On aurait pu écrire :

```
print('Bonjour !')
```

Mais pas :

```
print("Bonjour !')
```

Il faut utiliser le même délimiteur au début et à la fin

Les « strings »

Une chaîne de caractères est appelée « string » en anglais. Python, contrairement à Java admet deux délimiteurs de string :

`'Bonjour'` ou `"Bonjour"`

Dans certaines situations on doit préférer les guillemets double :

```
print("L'aliment")
```

Mais on aurait pu également écrire :

```
print('L\'aliment')
```

\ est appelé « antislash » ou encore « backslash »

Variables

Une curiosité (1)

Si on ne met pas les guillemets :

```
print(bonjour)
```

```
Traceback (most recent call
last):
  File "jdoodle.py", line 1, in
<module>
    print(bonjour)
NameError: name 'bonjour' is not
defined
```

`'bonjour'` est une chaîne de caractères

`"bonjour"` est une chaîne de caractères

`bonjour` n'a pas de sens pour Python

Changement de type

Contrairement à des langages comme Java, le type d'une variable peut changer en Python :

```
a = 8
print(type(a))
a= 5.0
print(type(a))
a = "8"
print(type(a))
a = False
print(type(a))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
```

Tester le type

Voici une manière de faire pour tester le type d'une variable :

```
a = 5
print(type(a) is int)
print(type(a) is float)
print(type(a) is bool)
print(type(a) is str)
```

```
True
False
False
False
```

```
a = 5.0
print(type(a) is int)
print(type(a) is float)
print(type(a) is bool)
print(type(a) is str)
```

```
False
True
False
False
```

Les « f-string »

Une f-string (formatted string) est une chaîne de caractères dans laquelle chaque expression entre accolade est évaluée :

```
prenom = "Jessy"
print(f"Bonjour {prenom}")
```

```
Bonjour Jessy
```

L'évaluation de la variable `prenom` retourne la chaîne `Jessy`

Dans la suite nous utiliserons fréquemment cette syntaxe qui est très pratique.

Les booléens & les nombres en Python

Booléens

Négation logique (not)

```
print(not True) False
print(not False) True
```

Ou logique (or)

```
print(True or True) True
print(True or False) True
print(False or True) True
print(False or False) False
```

Et logique (and)

```
print(True and True) True
print(True and False) False
print(False and True) False
print(False and False) False
```

int & float

Il existe deux types de nombres en Python : les entiers et les flottants.

```
n = 5
print(type(n))
x = 5.0
print(type(x))
```

```
<class 'int'>
<class 'float'>
```

Python détermine automatiquement à l'exécution le type d'une variable en analysant la valeur qui lui est affectée.

`5` représente un entier, le type est donc `int`

`5.0` représente un flottant, le type est donc `float`

La division euclidienne

Les f-string nous permettent un affichage convivial :

```
a = 7
b = 2
print(f"{a} = {a//b}x{b} + {a%b}")
```

```
7 = 3x2 + 1
```

Mais, quand le reste est nul, nous obtenons un affichage peu cohérent :

```
a = 6
b = 2
print(f"{a} = {a//b}x{b} + {a%b}")
```

```
6 = 3x2 + 0
```

Nous allons voir comment améliorer les choses dans la prochaine partie.

Les structures conditionnelles

Un petit problème...

Ce programme présente un défaut d'affichage :

```
a = 6
b = 2
print(f"{a} = {a//b}x{b} + {a%b}")
```

```
6 = 3x2 + 0
```

Il nous faudrait distinguer deux cas :

- le reste n'est pas nul, on l'affiche
- le reste est nul, on ne l'affiche pas

Le langage Python dispose comme tout autre langage de programmation d'une structure dite « conditionnelle ».

if & indentation

```
a = 7
if a == 2:
    print("a est bien égal à 2")
    print("C'est vraiment certain !")
print("S'affiche dans tous les cas.")
```

```
S'affiche dans tous les cas.
```

```
a = 2
if a == 2:
    print("a est bien égal à 2")
    print("C'est vraiment certain !")
print("S'affiche dans tous les cas.")
```

```
a est bien égal à 2
C'est vraiment certain !
S'affiche dans tous les cas.
```

L'indentation joue donc un rôle essentiel dans la syntaxe du code source Python !

Les boucles for & while

Code ASCII

En informatique tous les caractères sont codés en binaire (avec des 0 et des 1).

Le code `ASCII` (American Standard Code for Information Interchange) est l'un des systèmes de codage le plus utilisés. Il date des années 1960.

Le code `ASCII` permet de représenter 128 caractères.

Un petit problème...

La fonction prédéfinie **chr()** retourne le caractère correspondant au code ASCII indiqué :

```
print(chr(97))
```

```
a
```

On souhaite afficher les caractères dont les codes ASCII sont compris entre 49 et 60.

On pourrait écrire :

```
print(chr(49))
print(chr(50))
print(chr(51))
print(chr(52))
print(chr(53))
etc.
```

pas très malin !

Il y a beaucoup mieux : les boucles

Fonctions mathématiques

Fonctions prédéfinies

Certaines fonctions mathématiques sont directement disponibles en Python.

```
# Arrondi décimal
print(round(1.123456, 3)) # 1.123
print(round(-1.123456, 3)) # -1.123

# Min et Max
print(min(5, 2)) # 2
print(min(3, 6, 1, 5)) # 1
print(max(5, 2)) # 5
print(max(3, 6, 1, 5)) # 6

# Valeur absolue
print(abs(2)) # 2.0
print(abs(-2)) # 2.0

# Troncature entière
# Ce n'est pas la partie entière !
print(int(5.6)) # 5
print(int(-5.6)) # -5

# Puissance
print(pow(2,3)) # 8
print(2**3) # 8
```

Module math

D'autres fonctions « usuelles » sont disponibles via le module `math`.

Pour utiliser toutes les fonctions d'un module `xxx` on écrit en début de code :

```
from xxx import *
```

```
from math import *
# Partie entière
# (Arrondi à l'entier inférieur)
print(floor(5.6)) # 5
print(floor(-5.6)) # -6

# Arrondi à l'entier supérieur
print(ceil(5.6)) # 6
print(ceil(-5.6)) # -5

# Racine carrée
print(sqrt(4)) # 2.0
```

Les fonctions

Un petit problème...

On souhaite calculer la somme S des entiers de 1 à n de deux manières différentes.

La première manière de faire utilise directement le calcul :

$$S = 1 + 2 + \cdots + n$$

Une seconde manière de procéder consiste à utiliser la formule :

$$S = \frac{n(n+1)}{2}$$

Un petit problème...

On souhaite ensuite comparer si les deux manières de faire conduisent bien au même résultat pour des valeurs de n comprises entre 1 et 20 par exemple.

Ceci nous conduit à distinguer trois blocs de code différents :

1) Un bloc qui calcule S avec la méthode 1
2) Un bloc qui calcule S avec la méthode 2
3) Un bloc qui utilise une boucle et teste si les résultats sont bien identiques.

Exercices (1)

Exercices

Exc. 1 : écrire une fonction Python qui retourne l'image de x pour la fonction définie pour tout réel par :

$$f(x) = \frac{5x + 3}{4x^2 + 1}$$

Exc. 2 : écrire une fonction Python qui retourne la vitesse connaissant la distance et le temps mis pour parcourir cette distance.

Exc. 3 : écrire une fonction Python qui retourne `True` si l'entier passé en argument est un nombre pair.

Exc. 4 : écrire une fonction Python qui affiche les multiples de trois entre deux valeurs entières a et b.

Solutions

```
# ex. 1
def f(x):
    return (5*x+3)/(4*x**2+1)

# ex. 2
def vitesse(d, t):
    return d/t

# ex. 3 (solution 1)
def est_pair(n):
    if n % 2 == 0: return True
    else: return False

# ex. 3 (solution 2)
def est_pair(n):
    return n % 2 == 0

# ex. 4
def mult_3(a, b):
    for n in range(a, b+1):
        if n%3 == 0:
            print(n)
```

Exercices

Exc. 5 : écrire une fonction Python qui retourne un prix remisé de x %.

Exc. 6 : on place un capital à x % mensuel d'intérêts composés, écrire une fonction qui retourne le nombre de mois à partir duquel le capital a doublé.

Exc. 7 : écrire une fonction qui retourne le nième nombre triangulaire.
$\text{triangulaire}(n) = 1+2+3+\ldots+n$

Fractions égyptiennes*

Objectif* : écrire une fonction qui affiche la décomposition d'une fraction en fractions égyptiennes. Exemple :

$$\frac{3}{16} = \frac{1}{6} + \frac{1}{48}$$

Dans une telle décomposition les numérateurs sont égaux à 1 et les dénominateurs sont tous différents.

On fera l'hypothèse que la fraction initiale est telle que : $0 \leq \frac{a}{b} < 1$

Fractions égyptiennes*

On utilisera le module `fractions` qui permet de manipuler facilement des fractions :

```
from fractions import Fraction
f = Fraction(8, 5) - Fraction(3/2)
print(f) # 1/10
print(f.numerator) # 1
print(f.denominator) # 10
```

On pourra utiliser l'algorithme suivant :

```
Afficher(fraction = )
a = numérateur(fraction)
b = dénominateur(fraction)
Tant que a ≠ 1
    d = b÷a + 1
    f = a/b - 1/d
    a = numérateur(fraction)
    b = dénominateur(fraction)
    Afficher(1/d +)
d = b/a
Afficher(1/d)
```

Fractions égyptiennes*

```
from fractions import Fraction

def frac_egyp(frac):
    print(f"{frac} = ")
    a = frac.numerator
    b = frac.denominator
    while a != 1:
        d = b//a + 1
        f = Fraction(a, b) -
▸Fraction(1, d)
        a = f.numerator
        b = f.denominator
        print(f"1/{d} +")
    d = b//a
    print(f"1/{d}")

frac_egyp(Fraction(35,93))
```

```
35/93 =
1/3 +
1/24 +
1/744
```

Cet exercice est plus difficile observez bien le code et essayez de bien l'assimiler.

Exercices (2)

Moyennes

Objectif : Créer deux fonctions qui retournent la moyenne arithmétique et la moyenne géométrique de deux nombres.

La moyenne arithmétique est la demi-somme des deux nombres alors que la moyenne géométrique est la racine carrée du produit des deux nombres.

On utilisera la fonction `sqrt()` du module `maths` pour calculer la racine carrée.

Moyennes

```
from math import sqrt

def moy_arith(a, b):
    return (a+b)/2

def moy_geom(a, b):
    return sqrt(a*b)

print(moy_arith(5,8))
print(moy_geom(5,8))
```

```
6.5
6.324555320336759
```

Géométrie

Objectif 1 : écrire une fonction Python qui retourne les coordonnées du milieu du segment $[AB]$ où A et B sont deux points du plan.

Objectif 2 : écrire une fonction Python qui retourne la distance entre deux points.

Un point sera représenté par un couple (x, y).

Si `A =(x, y)`

`A[0]` retourne `x`

`A[1]` retourne `y`

Géométrie

```
def milieu(A, B):
    return ((A[0]+B[0])/2,
▸(A[1]+B[1])/2)

A = (2,4)
B = (5,1)
print(milieu(A, B))
```

```
(3.5, 2.5)
```

```
from math import sqrt

def distance(A, B):
    deltaX2 = (B[0] - A[0])**2
    deltaY2 = (B[1] - A[1])**2
    return sqrt(deltaX2 + deltaY2)

A = (2,4)
B = (5,1)
print(distance(A, B))
```

```
4.242640687119285
```

Nombres premiers

Objectif : écrire une fonction Python qui retourne `True` si le nombre passé en argument est un nombre premier et `False` sinon. On supposera que le nombre passé en argument est un entier positif non nul.

Comment déterminer si un entier est premier ? Il existe différents algorithmes. Nous allons utiliser l'un des plus simples. 1 n'est pas premier, 2 est premier, tout autre nombre pair n'est pas premier. Trois premier est premier et tout autre multiple de 3 n'est pas premier, etc.

Conjecture de Syracuse*

On appelle temps de vol le premier indice à partir duquel on obtient 1. Écrire une fonction qui retourne le temps de vol pour une valeur de départ donnée.

On appelle altitude maximale la plus grande valeur atteinte par la suite. Écrire une fonction qui retourne l'altitude maximale pour une valeur de départ donnée.

Conjecture de Syracuse*

```
# exo 1
def syracuse(u):
    if u%2 ==0: u = u//2
    else: u = 3*u+1
    return u

# exo 2
def temps_vol(u):
    n = 0
    while u != 1:
        u = syracuse(u)
        n += 1
    return n

u = 15
print(f"temps de vol {u} =
▸{temps_vol(u)}")
# temps de vol 15 = 17

u = 127
print(f"temps de vol {u} =
▸{temps_vol(u)}")
# temps de vol 127 = 46
```

Conjecture de Syracuse*

. . .

```
# exo syr3
def altitude_max(u):
    max_u = 4
    while u != 1:
        n = 0
        u = syracuse(u)
        if u > max_u:
            max_u = u
    return max_u

u = 15
print(f"Altitude max {u} =
▸{altitude_max(u)}")
# Altitude max 15 = 160

u = 127
print(f"Altitude max {u} =
▸{altitude_max(u)}")
# Altitude max 127 = 4372

u = 77671
print(f"Altitude max {u} =
▸{altitude_max(u)}")
# Altitude max 77671 = 1570824736
```

Le record d'altitude pour une valeur de départ inférieure à 100 000 est 77 671

Exercices (3)

Dessiner avec turtle

Le module `turtle` permet de dessiner facilement sur l'écran.

Il s'agit d'un module didactique et l'on peut voir le dessin apparaître progressivement à l'écran.

On peut installer le logiciel `Thonny` pour utiliser `turtle`, on peut aussi utiliser un site en ligne comme :
`https://www.lelivrescolaire.fr/outils/console-python`

Exemple 1

Le code suivant trace un carré à l'écran :

```
import turtle

# Efface l'écran
turtle.clearscreen()

# Pose le stylo sur la feuille
turtle.pendown()

# Stylo noir
turtle.color('black')

# Stylo au milieu de l'écran
turtle.goto(0,0)

# Déplace le stylo de 100 pixels
turtle.forward(100)

# pivote de 90° à gauche
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
# Nécessaire avec PyCharm
turtle.exitonclick()
```

Algorithme de Héron

Cet algorithme permet de calculer la valeur approchée de la racine d'un entier n de la manière suivante :

$u = n$

Répéter

$u = 1/2\,(u+n/u)$

Écrire un programme qui détermine combien d'itérations sont nécessaires pour obtenir une approximation à 0,0001 près de racine de

2 875 219

Algorithme de Héron

```
import math

n = 2875219
u = n
k = 0

while u-math.sqrt(n) >= 0.0001:
    k += 1
    u = (1/2)*(u+n/u)

print(k) # 14
```

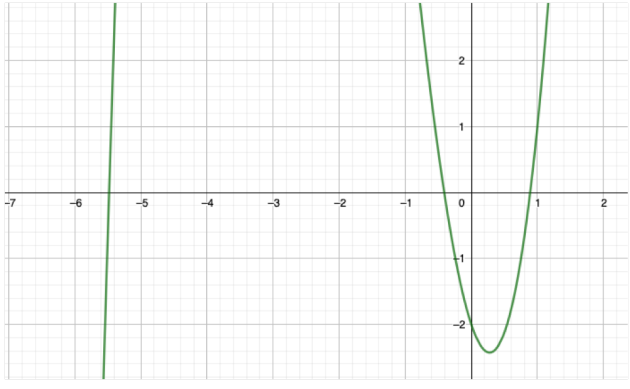
Un boucle `while` est appropriée car on ne sait pas à l'avance combien de fois il faudra parcourir la boucle.

Recherche de solutions

Le but de cet exercice est de rechercher les solutions d'une équation du troisième degré.
On prendra comme exemple l'équation :

$$x^3 + 5x^2 - 3x - 2 = 0$$

On traçant la fonction associée à l'aide de Geogebra on constate que cette équation possède trois solutions comprises en -6 et 1.



Recherche de solutions

Nous allons écrire un programme qui donne les trois solutions en calculant les valeurs de $f(x)$ par pas de 0,01.

La première solution se trouve entre -6 et -5 le principe utilisé sera transposable aux autres solutions.

L'idée est de détecter quand $f(x)$ change de signe.

Recherche de solutions

```
def f(x):
    return x**3+5*x**2-3*x-2

x = -6
while f(x) <=0:
    x += 0.01

print(x) # -5.480000000000011
```

Cet algorithme est assez primaire et naïf mais il permet d'obtenir une valeur approchée.

L'algorithme suppose que l'on part d'une valeur de $f(x)$ négative.

Recherche d'un maximum

Soit f une fonction admettant sur un intervalle donné un maximum. On cherche à déterminer ce maximum. On prendra comme exemple la fonction polynomiale du second degré :

$$f(x) = -x^2 + 4x + 15$$

On dispose comme information que le maximum cherché est dans l'intervalle

$$[0, 5]$$

Le principe sera de détecter quand la fonction change de monotonie.

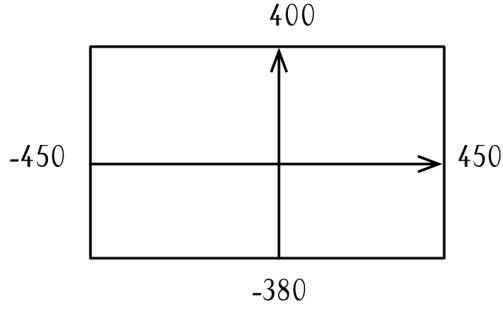
Dichotomie

Soit f une fonction monotone sur un intervalle, on cherche la valeur pour laquelle la fonction s'annule mais on n'a pas un intervalle nécessairement très bon de la solution. La méthode de la recherche par pas de 0,01 par exemple risque d'être coûteuse en temps de calcul.

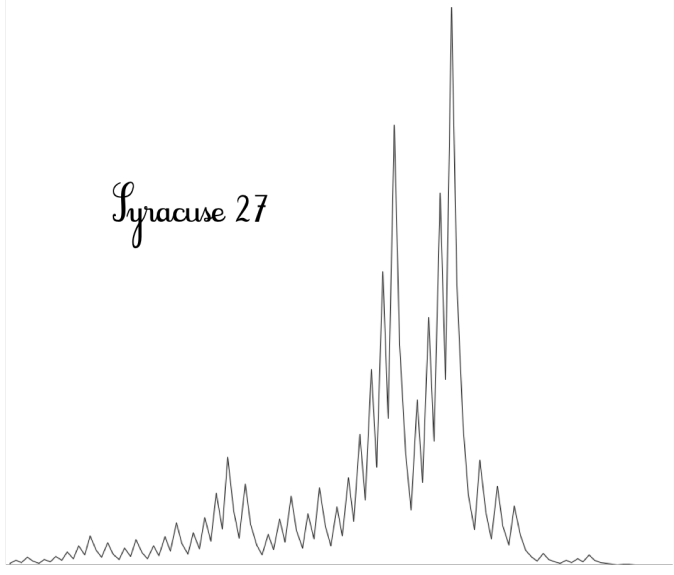
La méthode par dichotomie est plus performante. Il s'agit d'affiner progressivement l'intervalle $[a, b]$ dans lequel la fonction s'annule. Il suffit de fixer l'amplitude de l'intervalle pour avoir une condition d'arrêt.

Conjecture de Syracuse*

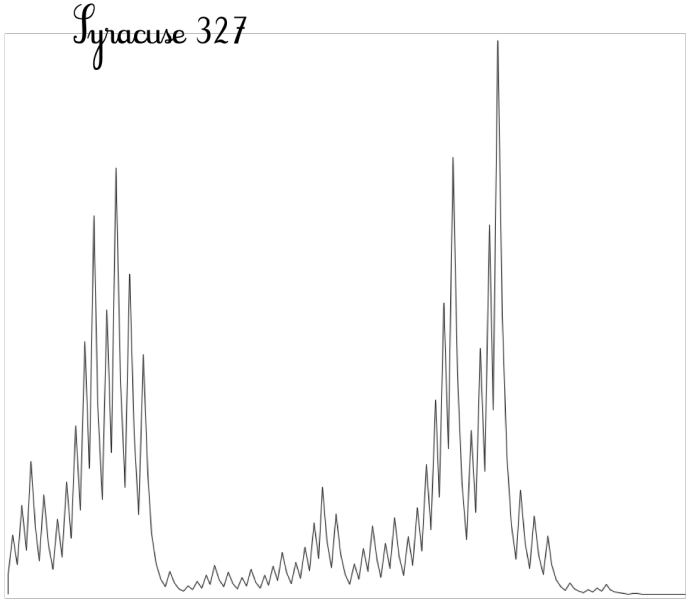
On se propose ici de représenter graphiquement à l'aide du module turtle des suites de Syracuse. On utilisera les fonctions écrites précédemment. En effet, il faut connaître le temps de vol pour adapter l'axe des abscisses et l'altitude maximale pour adapter l'axe des ordonnées. On supposera que l'on dispose de la zone d'affichage :



Conjecture de Syracuse*



Conjecture de Syracuse*



Conjecture de Syracuse*

```
import turtle
turtle.penup()
zero_X = -450
X_max = 450
Y_Max = 400
zero_Y = -380
turtle.goto(zero_X, zero_Y)
turtle.pendown()

def syracuse(u):
    . . .
def altitude_max(u):
    . . .
def temps_vol(u):
    . . .
seed = 327
u = seed
beta = (Y_Max-zero_Y-50)/altitude_max(u)
etendue = temps_vol(u) + 9
for n in range(0, etendue):
    x = zero_X + n*(X_max-zero_X)/
▸etendue
    y = zero_Y + u*beta
    turtle.goto(x,y)
    n += 1
    u = syracuse(u)
```