Chapitre 1

Solution aux défis

#1 : Distributeur automatique de pairs et d’impairs

Pour résoudre ce défi, il faut tout d’abord convertir l’entrée en entier puis déterminer si l’entier est pair ou impair. S’il est pair, il faut afficher les neuf nombres pairs suivants ; s’il est impaire, il faut afficher les neufs nombres impairs suivants. Le programme suivant effectue cette action :

'''

even\_odd\_vending.py

Print whether the input is even or odd. If even, print the next 9 even numbers

If odd, print the next 9 odd numbers.

'''

def even\_odd\_vending(num):

if (num % 2) == 0: 1

print('Even')

else:

print('Odd')

  count = 1

while count <= 9: 2

num += 2

print(num)

# increment the count of numbers printed

count += 1

  if \_\_name\_\_ == '\_\_main\_\_':

try:

  num = float(input('Enter an integer: '))

if num.is\_integer():

  even\_odd\_vending(int(num))

else:

  print('Please enter an integer')

except ValueError:

  print('Please enter a number')



L’entrée est convertie en un nombre à virgule flottante (plutôt qu’un entier) pour que l’on puisse utiliser la méthode is\_integer() pour vérifier si l’entrée est un entier. S’il est entier, nous utilisons la fonction int() pour le convertir en entier avant d’appeler la fonction even\_odd\_vending().

La fonction even\_odd\_vending() prend un entier en paramètre, utilise l’opérateur modulo (%) pour vérifier s’il est divisible par 2 1 puis affiche Even ou Odd s’il est respectivement pair ou impair.

Que le nombre soit pair ou impair, le nombre à afficher ensuite peut être obtenu en ajoutant 2 au précédent nombre. Nous effectuons cette opération dans une boucle « pour » 2 qui se répète jusqu’à ce que les neuf nombres soient affichés. Nous utilisons le label count pour garde trace de la quantité de nombres qui ont été affichés.

Si l’utilisateur saisi un nombre pair au lancement du programme, ce dernier affichera Even puis les neufs nombres pairs suivants :

Enter an integer: **2**

Even

4

6

8

10

12

14

16

18

20

Si un nombre impair est saisi, le programme affichera Odd puis les neufs nombres impairs suivants.

#2 : Générateur de tables de multiplication amélioré

La solution à ce défi est une extension du programme générateur de tables de multiplication que nous avions écrit plus avant dans le chapitre. Elle est affichée ci-après :

'''

enhanced\_multi\_table.py

Multiplication table printer: Enter the number and the number

of multiples to be printed

'''

def multi\_table(a, n):

for i in range(1, n+1):

print('{0} x {1} = {2}'.format(a, i, a\*i))

if \_\_name\_\_ == '\_\_main\_\_':

try:

a = float(input('Enter a number: '))

n = float(input('Enter the number of multiples: '))

if not n.is\_integer() or n < 0:

print('The number of multiples should be a positive integer')

else:

multi\_table(a, int(n))

except ValueError:

print('You entered an invalid input')

Notez que nous vérifions si le nombre spécifiant la quantité de multiples que nous désirons est un entier à l’aide de la méthode is\_integer(). Si la saisie est correcte, nous appelons la fonction multi\_table() avec deux paramètres : a (le nombre que nous souhaitons multiplier) et n (le nombre de multiples que nous souhaitons afficher).

Lorsque le programme est lancé, il vous demande de saisir les entrées puis affiche les multiples. Si l’utilisateur saisi une entrée invalide, un message d’erreur sera affiché et l’exécution sera arrêtée.

#3 : Convertisseur d’unités amélioré

La solution à ce défi est une amélioration du programme de conversion d’unités que nous avions écrit plus tôt afin qu’il soit capable de convertir les kilogrammes et les livres ainsi que les degrés Celsius et les degrés Fahrenheit :

'''

enhanced\_unit\_converter.py

Unit converter:

Kilometers and Miles

Kilograms and Pounds

Celsius and Fahrenheit

'''

def print\_menu():

print('1. Kilometers to Miles')

print('2. Miles to Kilometers')

print('3. Kilograms to Pounds')

print('4. Pounds to Kilograms')

print('5. Celsius to Fahrenheit')

print('6. Fahrenheit to Celsius')

def km\_miles():

km = float(input('Enter distance in kilometers: '))

miles = km / 1.609

print('Distance in miles: {0}'.format(miles))

def miles\_km():

miles = float(input('Enter distance in miles: '))

km = miles \* 1.609

print('Distance in kilometers: {0}'.format(km))

def kg\_pounds():

kg = float(input('Enter weight in kilograms: '))

pounds = kg \* 2.205

print('Weight in pounds: {0}'.format(pounds))

def pounds\_kg():

pounds = float(input('Enter weight in pounds: '))

kg = pounds / 2.205

print('Weight in kilograms: {0}'.format(kg))

def cel\_fahren():

celsius = float(input('Enter temperature in Celsius: '))

fahrenheit = celsius\*(9 / 5) + 32

print('Temperature in fahrenheit: {0}'.format(fahrenheit))

def fahren\_cel():

fahrenheit = float(input('Enter temperature in Fahrenheit: '))

celsius = (fahrenheit - 32)\*(5/9)

print('Temperature in celsius: {0}'.format(celsius))

if \_\_name\_\_ == '\_\_main\_\_':

print\_menu()

choice = input('Which conversion would you like to do? ')

if choice == '1':

km\_miles()

if choice == '2':

miles\_km()

if choice == '3':

kg\_pounds()

if choice == '4':

pounds\_kg()

if choice == '5':

cel\_fahren()

if choice == '6':

fahren\_cel()

Quatre nouvelles fonctions ont été ajoutées à notre précédent programme. La fonction kg\_pounds() prend une masse en kilogramme en entrée et retourne la masse équivalente en livres. La conversion inverse est réalisée par la fonction pounds\_kg(). Les fonctions cel\_fahren() et fahren\_cel() convertissent respectivement les Celsius en Fahrenheit et les Fahrenheit en Celsius.

Lorsque le programme est lancé, il demande à l’utilisateur de saisir un nombre entre 1 et 6 afin de choisir la conversion qu’il souhaite effectuer. Il demande ensuite de saisir la quantité à convertir puis retourne le résultat de la conversion :

1. Kilometers to Miles

2. Miles to Kilometers

3. Kilograms to Pounds

4. Pounds to Kilograms

5. Celsius to Fahrenheit

6. Fahrenheit to Celsius

Which conversion would you like to do? **5**

Enter temperature in Celsius: **37**

Temperature in Fahrenheit: 98.60000000000001

Si l’utilisateur saisi un nombre qui n’est pas reconnu par le programme, celui-ci s’arrête sans rien afficher. Vous pourriez améliorer ce programme pour qu’il retourne un message d’erreur dans ce cas.

#4 : Calculateur de fractions

Pour résoudre ce défi, nous créons des fonctions correspondant à différentes opérations mathématiques de base – add(), substract(), divide(), et multiply() – et appelons la fonction appropriée à la demande de l’utilisateur.

'''

fractions\_operations.py

Fraction operations

'''

from fractions import Fraction

def add(a, b):

print('Result of adding {0} and {1} is {2} '.format(a, b, a+b))

def subtract(a, b):

print('Result of subtracting {1} from {0} is {2}'.format(a, b, a-b))

def divide(a, b):

print('Result of dividing {0} by {1} is {2}'.format(a, b, a/b))

def multiply(a, b):

print('Result of multiplying {0} and {1} is {2}'.format(a, b, a\*b))

if \_\_name\_\_ == '\_\_main\_\_':

try:

a = Fraction(input('Enter first fraction: '))

b = Fraction(input('Enter second fraction: '))

op = input('Operation to perform - Add, Subtract, Divide, Multiply: ')

if op == 'Add':

add(a, b)

if op == 'Subtract':

subtract(a, b)

if op == 'Divide':

divide(a, b)

if op == 'Multiply':

multiply(a, b)

except ValueError:

print('Invalid fraction entered')

Lorsque le programme est lancé, il vous demande de saisir deux fractions et l’opération que vous souhaitez réaliser. Il affiche alors le résultat de cette opération. Voici un exemple de l’opération Substract :

Enter first fraction: **1/3**

Enter second fraction: **2/3**

Operation to perform - Add, Subtract, Divide, Multiply: **Subtract**

Result of subtracting 2/3 from 1/3 is -1/3

Si l’utilisateur saisi une entrée invalide, comme 1, le programme affichera un message d’erreur.

#5 : Donner à l’utilisateur le choix d’arrêter, ou non, le programme

Ce défi vous a été donné en dernier pour vous donner la chance d’essayer d’améliorer les autres programmes afin qu’ils puissent continuer leur exécution jusqu’à ce que l’utilisateur choisisse d’arrêter. Voici le programme calculateur de fractions amélioré à cet effet :

''' fractions\_operations\_exit\_power.py

Fraction operations: Do not exit until asked to '''

from fractions import Fraction

def add(a, b):

print('Result of adding {0} and {1} is {2} '.format(a, b, a+b))

def subtract(a, b):

print('Result of subtracting {1} from {0} is {2}'.format(a, b, a-b))

def: divide(a, b):

print('Result of dividing {0} by {1} is {2}'.format(a, b, a/b))

def multiply(a, b):

print('Result of multiplying {0} and {1} is {2}'.format(a, b, a\*b))

if \_\_name\_\_ == '\_\_main\_\_':

while True:

try:

a = Fraction(input('Enter first fraction: '))

b = Fraction(input('Enter second fraction: '))

op = input('Operation to perform - Add, Subtract, Divide, Multiply: ')

if op == 'Add':

add(a, b)

if op == 'Subtract':

subtract(a, b)

if op == 'Divide':

divide(a, b)

if op == 'Multiply':

multiply(a, b)

except ValueError:

print('Invalid fraction entered')

answer = input('Do you want to exit? (y) for yes ')

if answer == 'y':

break

Voici un exemple d’exécution de ce programme :

Enter first fraction: **1/3**

Enter second fraction: **4/6**

Operation to perform - Add, Subtract, Divide, Multiply: **Divide**

Result of dividing 1/3 by 2/3 is 1/2

Do you want to exit? (y) for yes **n**

Enter first fraction: **2/3**

Enter second fraction: **4/5**

Operation to perform - Add, Subtract, Divide, Multiply: **Multiply**

Result of multiplying 2/3 and 4/5 is **8/15**

Do you want to exit? (y) for yes **y**

Vous pouvez trouver, comme autre exemple, le programme de conversion d’unité amélioré de la même manière dans le fichier enhanced\_unit\_conveter\_exit\_power.py.

Chapitre 2

Solution aux défis

#1: Comment la température varie-t-elle au cours de la journée ?

Pour ce défi, vous deviez chercher les températures d’une ville à l’aide du moteur de recherche Google et recréer un graphique des températures de la journée. La Figure 1 montre un exemple de ce que vous trouvez en cherchant « New York weather ».

Figure 1.png

Figure 1

Exemple de résultat de la recherche « New York weather » dans le moteur de recherche Google.

Une fois que vous avez le graphique, faites deux listes : une pour stocker les heures de la journée et une autre pour stocker les températures correspondantes. Appelez ensuite la fonction plot() pour créer le graphique. Les heures de la journée sont des chaines de caractères. Cependant, si vous essayez de passer une liste de chaines de caractères en argument de la fonction plot(), l’exécution de votre programme va échouer et vous aurez un message d’erreur. Après tout, quel sens cela peut-il avoir de demander qu’un point soit placé à « 7 AM » ? À la place, assignez des valeurs aux différentes heures de la journée – par exemple, 1 pour 4 AM, 2 pour 7 AM, etc. Ensuite, utilisez la fonction xsticks() pour changer l’affichage des ordonnées afin d’afficher les vraies heures. Voici une solution pour ce défi :

'''

nyc\_forecast\_basic.py

Create a graph showing a city's temperature forecast for the day

'''

import matplotlib.pyplot as plt

def plot\_forecast():

time\_of\_day = ['4 AM', '7 AM', '10 AM', '1 PM', '4 PM', '7PM', '10 PM']

forecast\_temp = [71, 70, 74, 80, 82, 81, 76]

time\_interval = range(1, len(time\_of\_day) + 1) 1

plt.plot(time\_interval, forecast\_temp, 'o-')

plt.xticks(time\_interval, time\_of\_day)  plt.show() 2

if \_\_name\_\_ == '\_\_main\_\_':

plot\_forecast()

Nous créons les nombres utilisés pour représenter les différentes heures de la journée en 1. Puis nous utilisons la fonction xsticks() en 2 pour redéfinir ces nombres en des chaines de caractères, ayant pour effet d’afficher les heures en ordonnée du graphique plutôt que ces nombres (7 AM etc.). Le premier argument de cette fonction est la liste des nombres et le second est la liste des labels que vous souhaitez afficher à chaque nombre.

Lorsque vous lancez le programme, vous voyez s’afficher un graphique des températures de la journée, lequel ressemblera à celui que vous aviez vu lors de votre recherche Google.

#2 : Observer graphiquement une fonction quadratique

Pour résoudre ce défi, utilisez d’abord la fonction range() afin de générer 10 entiers compris entre -100 et 100 dans une liste 1. Les nombres -100, 80, ..., 80 seront générés. Calculez ensuite la valeur de la fonction pour chaque entier généré et stockez les valeurs obtenues dans la liste y\_values. Finalement, utilisez la fonction plot() pour paramétrer le graphique avec les listes x\_values et y\_values.

'''

quad\_function\_plot.py

Plot a quadratic function

'''

import matplotlib.pyplot as plt

def draw\_graph(x, y):

plt.plot(x, y)

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

# assume values of x

x\_values = range(-100, 100, 20) y\_values = [] 1

for x in x\_values:

# calculate the value of the quadratic

# function

y\_values.append(x\*\*2 + 2\*x + 1)

draw\_graph(x\_values, y\_values)

Lorsque vous lancez le programme, le graphique de la fonction ressemblera à celui de la Figure 2.

Figure 2.png

Figure 2

Graphique de la fonction quadratique x2+2x+1. Augmenter le nombre de points entre -100 et 100 aura pour effet de lisser la courbe.

Du fait que la fonction soit quadratique, sa variation en fonction de x n’est pas linéaire.

#3 : Programme de comparaison de trajectoires amélioré

Voici la solution à ce défi :

""" projectile\_comparison\_gen.py

Compare the projectile motion of a body thrown with various combinations of initial velocity and angle of projection

"""

import matplotlib.pyplot as plt import math

g = 9.8

def draw\_graph(x, y):

plt.plot(x, y)

plt.xlabel('x-coordinate')

plt.ylabel('y-coordinate')

plt.title('Projectile motion at different initial velocities and angles')

def frange(start, final, interval):

numbers = []

while start < final:

numbers.append(start)

start = start + interval

return numbers

def draw\_trajectory(u, theta, t\_flight):

# List of x and y coordinates

x = []

y = []

intervals = frange(0, t\_flight, 0.001)

for t in intervals:

x.append(u\*math.cos(theta)\*t)

y.append(u\*math.sin(theta)\*t - 0.5\*g\*t\*t)

# Create the graph

draw\_graph(x, y)

if \_\_name\_\_ == '\_\_main\_\_':

num\_trajectories = int(input('How many trajectories? '))

velocities = []

angles = []

for i in range(1, num\_trajectories+1):

v = input('Enter the initial velocity for trajectory {0} (m/s): '.format(i))

theta = input('Enter the angle of projection for trajectory {0} (degrees): '.format(i))

velocities.append(float(v))

angles.append(math.radians(float(theta)))

for i in range(num\_trajectories):

# Calculate time of flight, maximum horizontal distance and

# maximum vertical distance

t\_flight = 2\*velocities[i]\*math.sin(angles[i])/g

S\_x = velocities[i]\*math.cos(angles[i])\*t\_flight

S\_y = velocities[i]\*math.sin(angles[i])\*(t\_flight/2) - (1/2)\*g\*(t\_flight/2)\*\*2

print('Initial velocity: {0} Angle of Projection: {1}'.format(velocities[i], math.degrees(angles[i]))) 1

print('T: {0} S\_x: {1} S\_y: {2}'.format(t\_flight, S\_x, S\_y))

print()

draw\_trajectory(velocities[i], angles[i], t\_flight) 2

# Add a legend and show the graph

legends = []

for i in range(0, num\_trajectories):

legends.append('{0} - {1}'.format(velocities[i], math.degrees(angles[i])))

plt.legend(legends) 3

plt.show()

Lorsque vous l’exécutez, le programme commence par vous demander le nombre de trajectoires que vous souhaitez comparer. Puis il demande de saisir la vitesse initiale et l’angle de projection de chaque trajectoire. Pour chacune d’elles, le temps de vol, la distance parcourue et la hauteur atteinte sont affichés 1 puis la fonction draw\_trajectory() est appelée 2, ce qui configure le graphique pour chaque trajectoire.

Pour identifier les trajectoires, nous ajoutons une légende en 3. Finalement, nous appelons la fonction show() pour afficher le graphique.

Voici un exemple d’exécution :

How many trajectories? **3**

Enter the initial velocity for trajectory 1 (m/s): **25**

Enter the angle of projection for trajectory 1 (degrees): **60**

Enter the initial velocity for trajectory 2 (m/s): **50**

Enter the angle of projection for trajectory 2 (degrees): **50**

Enter the initial velocity for trajectory 3 (m/s): **30**

Enter the angle of projection for trajectory 3 (degrees): **60**

Initial velocity: 25.0 Angle of Projection: 59.99999999999999 T: 4.41849695808387 S\_x: 55.23121197604839 S\_y: 23.91581632653061

Initial velocity: 50.0 Angle of Projection: 50.0

T: 7.8167800318263065 S\_x: 251.22646760515516 S\_y: 74.85001133079913

Initial velocity: 30.0 Angle of Projection: 59.99999999999999

T: 5.302196349700644 S\_x: 79.53294524550968 S\_y: 34.438775510204074

La Figure 3 montre le graphique créé par ce programme. Vous pouvez voir les trois trajectoires et leur légende indiquant l’angle de projection et la vitesse initiale de chaque trajectoire.

Figure 3.png

Figure 3

Trajectoires de trois corps en chute libre lancés avec des vitesses initiales différentes à des angles différents

#4 : Visualiser vos dépenses

Pour résoudre ce défi vous devez demander à l’utilisateur le nombre de catégories qu’il souhaite avoir et ce à quoi ces catégories correspondent. Puis vous devez demander combien il a été dépensé pour chaque catégorie et créer un diagramme à bâtons à l’aide de la fonction barh(). Voici le programme :

'''

expenditures\_barchart.py

Visualizing weekly expenditure using a bar chart

'''

import matplotlib.pyplot as plt

def create\_bar\_chart(data, labels):

# number of bars

num\_bars = len(data)

# This list is the point on the y-axis where each

# bar is centered. Here it will be [1, 2, 3..]

positions = range(1, num\_bars+1)

plt.barh(positions, data, align='center') 1

# Set the label of each bar

plt.yticks(positions, labels)

plt.xlabel('Amount')

plt.ylabel('Categories')

plt.title('Weekly expenditures')

# Turns on the grid which may assist in visual estimation

plt.grid()

plt.show() 2

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input('Enter the number of categories: '))

labels = []

expenditures = []

for i in range(n):

category = input('Enter category: ')

expenditure = float(input('Expenditure: '))

labels.append(category) 3

expenditures.append(expenditure) 4

create\_bar\_chart(expenditures, labels) 5

Les catégories saisies par l’utilisateur sont stockées dans la liste labels 3, laquelle nous permettra de définir le nom des barres du diagramme. La dépense correspondant à chaque catégorie est stockée dans la liste expenditures 4. Nous appelons alors la fonction create\_bar\_chart() 5 avec les dépenses en premier argument et les noms en second argument. Cette fonction appelle alors barh() 1 pour définir les noms des axes et autres détails puis nous appelons la fonction show() 2 pour afficher le diagramme des dépenses.

Voici un exemple d’exécution du programme :

Enter the number of categories: **4**

Enter category: **Food**

Expenditure: **70**

Enter category: **Transportation**

Expenditure: **35**

Enter category: **Entertainment**

Expenditure: **30**

Enter category: **Phone/Internet**

Expenditure: **30**

La Figure 4 montre le diagramme créé par cette exécution :

Figure 4.png

Figure 4

Un diagramme à bâtons montrant les dépenses hebdomadaires dans chaque catégorie

#5 : Découvrir la relation entre la suite de Fibonacci et le nombre d’or

Pour résoudre ce défi, nous allons calculer les 100 premiers nombres de la suite de Fibonacci à l’aide de la fonction fibo() mentionnée dans la description du défi. Nous allons ensuite calculer le rapport entre les nombres successifs, stocker les résultats dans une liste puis utiliser la fonction plot() pour créer le graphique. Voici la solution :

'''

fibonacci\_goldenration.py

Relationship between Fibonacci sequence and golden ratio

'''

import matplotlib.pyplot as plt

def fibo(n):

if n == 1:

return [1]

if n == 2:

return [1, 1]

#n> 2

a= 1

b= 1

# first two members of the series

series = [a, b]

for i in range(n):

c=a+ b

series.append(c) a= b b= c

return series

def plot\_ratio(series):

ratios = []

for i in range(len(series)-1):

ratios.append(series[i+1]/series[i])

plt.plot(ratios)

plt.title('Ratio between Fibonacci numbers & golden ratio')

plt.ylabel('Ratio')

plt.xlabel('No.')

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

# Number of fibonacci numbers

num = 100

series = fibo(num)

plot\_ratio(series)

La boucle « pour » en 1 est l’étape clé de la fonction plot\_ratio(). Nous parcourrons les nombres de la suite de Fibonacci et stockons le rapport du troisième terme sur le deuxième, du quatrième sur le troisième et ainsi de suite, le tout dans la liste ratios. Nous appelons alors la fonction plot(), ajoutons un titre et des noms d’axe puis appelons la fonction show() pour afficher le graphique.

Chapitre 3

Solution aux défis

#1 : Améliorer le programme de calcul du coefficient de corrélations

Nous pouvons modifier légèrement la fonction find\_corr\_x\_y() afin qu’elle vérifie la longueur des deux listes en argument. Si elles ne sont pas de même longueur nous retournons None. Voici la nouvelle fonction, avec quelques autres changements :

'''

“

linear\_correlation\_enhanced.py

Linear correlation program

'''

def find\_corr\_x\_y(x,y):

if len(x) != len(y): 1

print('The two sets of numbers are of unequal size')

return None

n = len(x)

# find the sum of the products

prod = [xi\*yi for xi, yi in zip(x, y)] 2

sum\_prod\_x\_y = sum(prod)

  # sum of the numbers in x

sum\_x = sum(x)

# sum of the numbers in y

sum\_y = sum(y)

  # square of the sum of the numbers in x

squared\_sum\_x = sum\_x\*\*2

# square of the sum of the numbers in y

squared\_sum\_y = sum\_y\*\*2

  # find the squares of numbers in x and the

  # sum of the squares

x\_square = [xi\*\*2 for xi in x] 3

x\_square\_sum = sum(x\_square)

# find the squares of numbers in y and the

# sum of the squares

y\_square = [yi\*\*2 for yi in y]

y\_square\_sum = sum(y\_square)

# numerator

numerator = n\*sum\_prod\_x\_y - sum\_x\*sum\_y

denominator\_term1 = n\*x\_square\_sum - squared\_sum\_x

denominator\_term2 = n\*y\_square\_sum - squared\_sum\_y

denominator = (denominator\_term1\*denominator\_term2)\*\*0.5

correlation = numerator/denominator

return correlation

En plus d’ajouter la vérification de la longueur des listes au début du code 1, vous pouvez voir que nous avons aussi utilisé les listes en compréhension (comme expliqué dans l’annexe B) afin de rendre des parties du programme plus compactes (par exemple, en 2 et 3).

Lorsque nous appelons cette fonction avec deux listes de nombres contenant le même nombre d’éléments, elle retourne le coefficient de corrélation comme elle le faisait auparavant. Mais si les deux jeux de nombres sont de longueur inégale, None sera retourné. Lorsque vous utiliserez cette fonction dans vos programmes, vous devrez vérifier la valeur retournée et agir en conséquence.

corr = find\_corr\_x\_y(x,y)

if not corr:

print('Correlation correlation could not be calculated')

else:

print('The correlation coefficient between x and y is {0}'.format(corr))

Par exemple, si la valeur retournée est None, nous affichons un message disant que le coefficient de corrélation ne peut pas être calculé.

#2 : Calculateur de statistiques

Le programme lisant les nombres depuis le fichier mydata.txt et calcule les diverses valeurs statistiques est le suivant :

'''

statistics\_calculator.py

Read numbers from a file, calculate and print statistical measures:

mean, median, mode, variance, standard deviation

'''

from stats import mean, median, mode, variance\_sd 1

def read\_data(filename):

numbers = []

  with open(filename) as f:

for line in f:

  numbers.append(float(line))

return numbers

 if \_\_name\_\_=='\_\_main\_\_':

data = read\_data('mydata.txt') 2

m = mean(data)

median = median(data)

mode = mode(data)

variance, sd = variance\_sd(data)

 print('Mean: {0:.5f}'.format(m))

print('Median: {0:.5f}'.format(median))

print('Mode: {0:.5f}'.format(mode))

print('Variance: {0:.5f}'.format(variance))

print('Standard deviation: {0:.5f}'.format(sd))

Nous avons créé un module, stats.py (au même emplacement que statistics\_calculator.py), contenant les fonctions calculant la moyenne, la médiane, le mode, la variance et l’écart-type. Ces fonctions sont importées en 1. Ces fonctions sont les même que nous avions écrites dans le chapitre mais nommées différemment. L’autre différence est que la fonction variance\_sd() retourne la variance et l’écart-type ensembles dans un tuple. Placer ces fonctions dans un module séparé de notre programme nous permet de ne pas redéfinir les fonctions à chaque fois que nous souhaitons les utiliser.

Nous appelons la fonction read\_data(), laquelle lit les nombres depuis le fichier et les retourne dans une liste, data 2. Une fois que nous avons la liste de nombres, nous pouvons appeler les différentes fonctions pour calculer les valeurs statistiques correspondantes et finalement les afficher.

#3 : Essayer d’autres données CVS

Le fichier CSV USA\_SP\_POP\_TOTL.csv contient deux colonnes de données : une date et les populations totales depuis 31/12/1960 jusqu’au 31/12/2012. Nous allons lire les populations et les années depuis le fichier, calculer les valeurs statistiques et créer le graphique :

'''

us\_population\_stats.py

Read the US population data from a CSV file, calculate the growth in

population in consecutive years, and compute various statistical measures

Also creates two graphs - one showing the total population over the years and

the other showing the change between consecutive years

'''

import matplotlib.pyplot as plt

import csv

from stats import mean, median, variance\_sd

def read\_csv(filename):

years = []

population = []

with open(filename) as f:

reader = csv.reader(f)

next(reader)

summer = []

highest\_correlated = []

for row in reader:

# Extract only the year from

# date

year = row[0].split('-')[0] 1

years.append(year)

population.append(float(row[1]))

# Reverse the lists because the original data lists the

# most recent years first

population.reverse()

years.reverse()

  return population, years

 def plot\_population(population, years):

plt.figure(1) 2

xaxis\_positions = range(0, len(years))

plt.plot(population, 'r-')

plt.title('Total population in US')

plt.xlabel('Year')

plt.ylabel('Population')

  plt.xticks(xaxis\_positions, years, rotation=45) 3

def calculate\_stats(population):

# find the growth in population in consecutive years

growth = []

for i in range(0, len(population)-1):

growth.append(population[i+1] - population[i])

print('Mean growth: {0:.5f}'.format(mean(growth)))

print('Median growth: {0:.5f}'.format(median(growth)))

print('Variance/Sd growth: {0:.5f}, {1:.5f}'.format(\*variance\_sd(growth)))

return growth

 def plot\_population\_diff(growth, years):

xaxis\_positions = range(0, len(years)-1)

xaxis\_labels = ['{0}-{1}'.format(years[i], years[i+1]) for i in range(len(years)-1)] 4

plt.figure(2)

plt.plot(growth, 'r-')

plt.title('Population Growth in consecutive years')

plt.ylabel('Population Growth')

plt.xticks(xaxis\_positions, xaxis\_labels, rotation=45)

if \_\_name\_\_ == '\_\_main\_\_':

population, years = read\_csv('USA\_SP\_POP\_TOTL.csv')

plot\_population(population, years)

growth = calculate\_stats(population)

plot\_population\_diff(growth, years)

plt.show()

La fonction read\_cvs() lis les données depuis le fichier pour créer deux listes : years et population, lesquelles contiennent respectivement les années et leur population correspondante. Comme nous l’avions dit plus tôt, le fichier contient une date, mais nous ne souhaitons extraire que l’année (afin que le texte puisse rentrer dans notre graphique), et c’est ce que nous faisons en 1. La méthode split() retourne une liste de mots depuis une chaine de caractères et sépare les mots selon le délimiteur spécifié. Par exemple :

>>> d = '2012-09-10'

>>> d.split('-')

['2012', '09', '10']

Ainsi, si nous voulions obtenir 2012, nous n’aurions qu’à saisir le premier objet de la liste retournée :

>>> words = d.split('-')

>>> words[0]

'2012'

Comme nous lisons le fichier de haut en bas et que les dates les plus récentes sont les premières, nous inversons les deux listes avant de les utiliser en utilisant la méthode reverse(). Cette méthode inverse la liste qu’on lui donne. Par exemple, si liste = [1, 2, 3]. Appeler liste.reverse() aura effet de changer liste en [3, 2, 1]. Une fois que nous avons les deux listes, nous créons le premier graphique montrant l’évolution de la population à travers les années dans la fonction plot\_population(). Comme nous souhaitons créer deux graphiques séparés, nous créons une figure en appelant la fonction figure() et lui assignons le nombre 1 en 2. Nous ajoutons la population au graphique avant d’insérer les années sur l’axe Ox à l’aide de la fonction xsticks() 3. L’argument rotation nous permet d’afficher les noms sur l’axe Ox avec un certain angle afin que ceux-ci ne se piétinent pas. Ici, nous demandons que ces noms soient écrits avec un angle de 45°.

Dans la fonction calculate\_stats(), nous calculons et affichons les valeurs statistiques de la croissance de la population à l’aide des fonctions qui composent désormais le module stats. Cette fonction retourne aussi la liste growth, laquelle contient la croissance de la population au fil des années.

La fonction plot\_population\_diff() créé une seconde figure et créé le graphique de la croissance de la population au fil des années. Nous utilisons une liste en compréhension pour créer une liste de labels pour l’axe Ox de la forme 1960-1961, 1961-1962 et ainsi de suite, afin d’indiquer les années 4. Nous disons, ici encore, que nous souhaitons que les noms soient affichés avec un angle de 45 degrés à l’aide l’argument rotation.

Lorsque vous lancez le programme, vous devriez voir s’afficher les deux graphiques côte à côte comme dans la Figure 5.

Figure 5.png

Figure 5

Deux graphiques de la population

La première fenêtre sera appelée Figure 1 et l’autre Figure 2 et les valeurs statistiques seront affichées comme suit :

Mean growth: 2562366.15385

Median growth: 2476370.00000

Variance/Sd growth: 188985554755.28406, 434724.68846

Vous aurez besoin d’optimiser la taille des fenêtres pour pouvoir lire les noms de l’axe Ox.

#4 : Trouver le centile

Le programme trouvant la valeur correspondant à un certain centile utilise l’algorithme évoqué dans la description du défi et ressemble à ceci :

'''

percentile\_score.py

Calculate the number from a list of numbers that corresponds

to a specific percentile

This implements the method described at

http://web.stanford.edu/class/archive/anthsci/anthsci192/anthsci192.1064/handouts/

calculating%20percentiles.pdf

'''

def find\_percentile\_score(data, percentile):

if percentile < 0 or percentile > 100:

return None

data.sort() 1

if percentile == 0:

return data[0]

if percentile == 100:

return data[-1]

n = len(data)

i = ((n\*percentile)/100) + 0.5 2

if i.is\_integer():

real\_idx = int(i-1)

return data[real\_idx] 3

else:

k = int(i) 4

f = i –k 5

real\_idx\_1 = k - 1

real\_idx\_2 = k

return (1-f)\*data[real\_idx\_1] + f\*data[real\_idx\_2] 6

def read\_data(filename):

numbers = []

with open(filename) as f:

for line in f:

numbers.append(float(line))

return numbers

if \_\_name\_\_ == '\_\_main\_\_':

percentile = float(input('Enter the percentile score you want to calculate: '))

data = read\_data('marks.txt')

percentile\_score = find\_percentile\_score(data, percentile)

if percentile\_score:

print('The score at {0} percentile: {1}'.format(percentile, percentile\_score))

else:

print('Could not find the score corresponding to {0} percentile'.format(percentile))

La fonction find\_percentile\_score() implémente l’algorithme contenu dans la description du défi. Il commence par trier la liste de nombres placée en argument à l’aide de la méthode sort() 1. Puis il vérifie si le centile n’est pas 0 ou 100. Si c’est 0, le premier élément de la liste triée est retourné et si c’est 100, le dernier élément est retourné. En 2, nous calculons la valeur de i correspondant à l’étape 2 de l’algorithme. Si c’est un entier, nous retournons le i-ème élément de la liste de nombres 3. Comme une liste commence à l’indice 0 en Python, l’élément correspondant sera à la place i-1. Si la valeur n’est pas un entier, nous extrayons l’entier et la partie fractionnelle du nombre en 4 et 5 que nous stockons respectivement dans les labels k et f. Puis nous retournons le nombre (1-f)\*data[k] + f\*data[k+1] 6, lequel correspond à la valeur du centile. Vous trouverez un fichier nommé marks.txt dans le répertoire source, contenant une liste de nombres. Lorsque vous exécuterez ce programme en spécifiant un certain centile, la valeur correspondante sera affichée :

Enter the percentile score you want to calculate: 88

The score at 88.0 percentile: 19.5

Cependant, cet algorithme ne fonctionne pas pour des centiles supérieurs à 98 sur cet ensemble de données. Un algorithme alternatif est distribué par Microsoft Excel à l’adresse suivante : <https://en.wikipedia.org/wiki/Percentile#Microsoft_Excel_method>. Le code suivant implémente cet algorithme :

'''

percentile\_score\_microsoft\_excel.py

Calculate the number from a list of numbers that corresponds

to a specific percentile

This implements the "Microsoft Excel Method":

https://en.wikipedia.org/wiki/Percentile#Microsoft\_Excel\_method

'''

def find\_percentile\_score(data, percentile):

if percentile < 0 or percentile > 100:

return None

data.sort()

if percentile == 0:

return data[0]

if percentile == 100:

return data[-1]

n = len(data)

rank = (percentile/100)\*(n-1) + 1 1

k = int(rank) 2

d = rank – k 3

real\_idx\_1 = k-1

real\_idx\_2 = k

return data[real\_idx\_1] + d\*(data[real\_idx\_2]-data[real\_idx\_1]) 4

def read\_data(filename):

numbers = []

with open(filename) as f:

for line in f:

numbers.append(float(line))

return numbers

 if \_\_name\_\_ == '\_\_main\_\_':

percentile = float(input('Enter the percentile score you want to calculate: '))

data = read\_data('marks.txt')

percentile\_score = find\_percentile\_score(data, percentile)

if percentile\_score:

print('The score at {0} percentile: {1}'.format(percentile, percentile\_score))

else:

  print('Could not find the score corresponding to {0} percentile'.format(percentile))

L’étape clé de ce programme est en 1, lorsque nous calculons rank. Nous extrayons ensuite sa partie entière et sa partie fractionnelle en 2 et 3 et retournons le nombre correspondant au centile saisi par l’utilisateur en 4.

Voici un exemple d’exécution de ce programme :

Enter the percentile score you want to calculate: **88**

The score at 88.0 percentile: 19.5

Comme nous nous y attendions, les deux programmes retournent la même valeur pour un centile de 88. Essayons maintenant de trouver la valeur correspondant au centile 99,6 :

Enter the percentile score you want to calculate: **99.6**

The score at 99.6 percentile: 20.0

Chapitre 4

Solution aux défis

#1 : Factorisation d’expressions

Le code suivant implémente le programme de factorisation d’expressions :

'''

factorizer.py

Factor an input expression

'''

from sympy import factor, sympify, SympifyError

def factorize(expr):

return factor(expr)

if \_\_name\_\_ == '\_\_main\_\_':

expr = input('Enter an expression to factorize: ') 1

try:

expr\_obj = sympify(expr)

except SympifyError:

print('Invalid expression entered as input')

else:

print(factorize(expr\_obj))

Nous demandons à l’utilisateur de saisir une expression à factoriser en 1, la convertissons en un objet SymPy à l’aide de la fonction sympify() puis appelons la fonction factorize(), laquelle utilise la fonction factor() de SymPy. Voici un exemple d’exécution de ce programme :

Enter an expression to factorize: x\*\*2 + 5\*x + 6

(x + 2)\*(x + 3)

Si une expression invalide est saisie, un message d’erreur sera affiché.

#2 : Solutionneur graphique d’équations

Le programme suivant prend en entrée deux équations de la forme , essaye de trouver une solution aux deux équations et créé un graphique montrant les deux droites solutions des équations :

'''

graphical\_eq\_solve.py

Graphical equation solver

'''

from sympy import Symbol, sympify, solve, SympifyError

from sympy.plotting import plot

def solve\_plot\_equations(eq1, eq2, x, y):

# Solve

solution = solve((eq1, eq2), dict=True)

if solution:

print('x: {0} y: {1}'.format(solution[0][x], solution[0][y]))

else:

print('No solution found')

# Plot

eq1\_y = solve(eq1,'y')[0]

eq2\_y = solve(eq2, 'y')[0]

plot(eq1\_y, eq2\_y, legend=True)

if \_\_name\_\_=='\_\_main\_\_':

eq1 = input('Enter your first equation : ')

eq2 = input('Enter your second equation: ')

try:

eq1 = sympify(eq1)

eq2 = sympify(eq2)

except SympifyError:

print('Invalid input')

else:

x = Symbol('x')

y = Symbol('y')

# check if the expressions consist of only two variables

eq1\_symbols = eq1.atoms(Symbol)

eq2\_symbols = eq2.atoms(Symbol)

if len(eq1\_symbols)> 2 or len(eq2\_symbols) > 2:

print('The equations must have only two variables - x and y')

  elif x not in eq1\_symbols or y not in eq1\_symbols:

print('First equation must have only x and y variables')

  elif x not in eq2\_symbols or y not in eq2\_symbols:

print('Second equation must have only x and y variables')

  else:

solve\_plot\_equations(eq1, eq2, x, y)

Après avoir validé l’équation saisie, nous trouvons les variables de chaque équation à l’aide de la méthode atoms() en 1 et 2. Lorsque nous appelons la méthode atoms() avec la classe Symbol en paramètre, la méthode retourne une liste des variables de l’expression. Nous vérifions ainsi que l’expression saisie ne contient que les variables x et y. Si tel est le cas, nous appelons la fonction solve\_plot\_equations(). Celle-ci utilise la fonction solve() pour tenter de trouver la solution aux deux équations. Si aucune solution ne peut être trouvée, un message s’affiche pour nous en informer. Dans tous les cas, nous exprimons les deux équations comme des fonctions de x et appelons la fonction plot() pour créer le graphique.

Voici un exemple d’exécution de ce programme :

Enter your first equation : 3\*x + 2\*y - 3

Enter your second equation: 2\*x + 3\*y - 2

x: 1 y: 0

La Figure 6 nous montre le graphique des deux équations, ce qui nous permet de vérifier l’exactitude de la solution trouvée (1,0).

Figure 6.png

Figure 6

Graphique des deux équations en entrée du programme

#3 : Sommer une série

Le programme suivant montre comment résoudre le défi à l’aide de la fonction summation() :

'''

series\_summation.py

Sum an arbitrary series

'''

from sympy import summation, sympify, Symbol, pprint

def find\_sum(n\_term, num\_terms):

n = Symbol('n')

s = summation(n\_term, (n, 1, num\_terms))

pprint(s)

if \_\_name\_\_ == '\_\_main\_\_':

n\_term = sympify(input('Enter the nth term: '))

num\_terms = int(input('Enter the number of terms: '))

find\_sum(n\_term, num\_terms)

Nous demandons à l’utilisateur de saisir l’expression du n-ième terme de la série et le nombre de termes que nous voulons sommer. Nous appelons ensuite la fonction find\_sum(), laquelle utilise la fonction summation() de SymPy avec l’expression du n-ième terme en premier argument. Le second argument est un tuple contenant la variable de l’expression, 1, et le nombre de termes que nous voulons sommer. Nous affichons ensuite le résultat sous une forme agréable à l’aide de la fonction pprint(). Voici un exemple d’exécution :

Enter the nth term: **a+(n-1)\*d**

Enter the number of terms: **3**

3·a + 3·d

#4 : Résoudre une inégalité à une seule variable

Le programme suivant implémente un algorithme solutionneur d’inéquations :

'''

isolve.py

Single variable inequality solver

'''

from sympy import Symbol, sympify, SympifyError

from sympy import solve\_poly\_inequality, solve\_rational\_inequalities

from sympy import solve\_univariate\_inequality, Poly

from sympy.core.relational import Relational, Equality

def isolve(ineq\_obj): x = Symbol('x')

expr = ineq\_obj.lhs rel = ineq\_obj.rel\_op

if expr.is\_polynomial():

p = Poly(expr, x)

return solve\_poly\_inequality(p, rel)

elif expr.is\_rational\_function():

p1, p2 = expr.as\_numer\_denom()

num = Poly(p1)

denom = Poly(p2)

return solve\_rational\_inequalities([[((num, denom), rel)]])

else:

return solve\_univariate\_inequality(ineq\_obj , x, relational=False)

if \_\_name\_\_ == '\_\_main\_\_':

ineq = input('Enter the inequality to solve: ')

try:

ineq\_obj = sympify(ineq)

except SympifyError:

print('Invalid inequality')

else:

# We check if the input expression is an inequality here

if isinstance(ineq\_obj, Relational) and not isinstance(ineq\_obj, Equality): 1

print(isolve(ineq\_obj))

else:

print('Invalid inequality')

Le programme demande à l’utilisateur de saisir une inégalité puis la converti en un objet SymPy à l’aide de la fonction sympify(). Nous vérifions alors si l’expression est bien une inégalité et non pas une expression quelconque. La fonction isinstance() de Python nous permet de faire cela, en 1, en vérifiant si l’objet retourné par sympify() est un objet de type Relational et si ce n’est pas une égalité. Cette vérification est nécessaire care seuls les objets de type Relational ont un attribut rel\_op référant à l’opérateur relationnel (>, >=, < et <=) qui sera utilisé plus tard dans la fonction isolve(). Une fois avoir vérifié si l’entrée est une inégalité, nous appelons la fonction isolve().

Dans celle-ci, nous utilisons les fonctions is\_polynomial() et is\_rational\_function() pour déterminer si l’inégalité est une fonction polynomiale ou rationnelle et appeler la fonction adéquate pour trouver la solution. Si l’inégalité n’est d’aucun des deux types précédents, nous appelons la fonction solve\_univariate\_inequality().

Voici un exemple d’exécution pour une inégalité polynomiale :

Enter the inequality to solve: **-x\*\*2 + 4 < 0**

[(-oo, -2), (2, oo)]

Voici un exemple d’exécution pour une inégalité rationnelle :

Enter the inequality to solve: **((x-1)/(x+2)) > 0**

(-oo, -2) U (1, oo)

Et voici un exemple d’exécution pour une inégalité qui n’est ni polynomiale ni rationnelle :

Enter the inequality to solve: sin(x) - 0.6 > 0

(0.643501108793284, 2.49809154479651)

Chapitre 5

Solution aux défis

#1 : Utiliser des diagrammes de Venn pour visualiser des relations entre ensembles

Voici la solution au défi :

"""

venn\_sports.py

Is football the favorite sport in my class too?

Let's find out using a Venn diagram

"""

from sympy import FiniteSet

from matplotlib\_venn import venn2

import matplotlib.pyplot as plt

import csv

def read\_csv(filename):

football = []

others = []

with open(filename) as f:

reader = csv.reader(f)

next(reader)

for row in reader:

if row[1] == '1':

football.append(row[0])

if row[2] == '1':

others.append(row[0])

return football, others

def draw\_venn(f, o):

venn2(subsets=(f, o), set\_labels=('Football', 'Others'))

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

football, others = read\_csv('sports.csv')

f = FiniteSet(\*football) 1

o = FiniteSet(\*others) 2

draw\_venn(f, o)

L’étape clé est de créer les deux ensembles – l’ensemble des étudiants qui aime le football et l’ensemble des étudiants qui aiment un autre sport – en lisant les données depuis le fichier sports.csv qui doit être créé en assemblant les résultats de l’étude. Ce fichier est lu par la fonction read\_csv(), laquelle créé deux listes contenant des identifiants d’élèves (football et others) et les retourne.

Nous créons alors deux ensembles correspondant aux catégories représentées, en 1 et 2, puis nous appelons la fonction draw\_venn() pour tracer le diagramme de Venn.

#2 : Loi des grands nombres

Pour ce défi, nous lançons un dé un certain nombre de fois pour voir comment la valeur moyenne s’approche de la valeur théorique de 3,5. Voici le programme :

'''

law\_ln.py

Verify the law of large numbers using a six-sided die roll as an example

'''

import random

def roll(num\_trials):

rolls = []

for t in range(num\_trials):

rolls.append(random.randint(1, 6))

return sum(rolls)/num\_trials

if \_\_name\_\_ == '\_\_main\_\_':

expected\_value = 3.5

print('Expected value: {0}'.format(expected\_value))

for trial in [100, 1000, 10000, 100000, 500000]:

avg = roll(trial)

print('Trials: {0} Trial average {1}'.format(trial, avg))

La fonction roll() simule le lancer d’un dé à six faces pour un certain nombre de lancers spécifié par l’argument num\_trials et retourne la valeur moyenne obtenue. Appelons cette fonction avec 100, 1000, 10000, 100000 et 500000 lancers pour voir comment la valeur moyenne s’approche de la valeur attendue, 3,5 :

Expected value: 3.5

Trials: 100 Trial average 3.49

Trials: 1000 Trial average 3.519

Trials: 10000 Trial average 3.4966

Trials: 100000 Trial average 3.50036

Trials: 500000 Trial average 3.501474

#3 : Combien de lancers de pièce avant de n’avoir plus d’argent ?

La solution à ce défi est le programme suivant :

'''

game\_tosses.py

A player wins 1$ for every head and loses 1.5$ for every tail.

The game is over when the player's balance reaches 0$

'''

import random

def play(start\_amount):

win\_amount = 1

loss\_amount = 1.5

cur\_amount = start\_amount

tosses = 0

while cur\_amount > 0: 1

tosses += 1

toss = random.randint(0, 1) 2

if toss == 0:

cur\_amount += win\_amount

print('Heads! Current amount: {0}'.format(cur\_amount))

else:

cur\_amount -= loss\_amount

print('Tails! Current amount: {0}'.format(cur\_amount))

print('Game over :( Current amount: {0}. Coin tosses: {1}'.format(cur\_amount, tosses))

if \_\_name\_\_ == '\_\_main\_\_':

start\_amount = float(input('Enter your starting amount: '))

play(start\_amount)

#4 : Mélanger un paquet de cartes

Voici le programme mélangeant un paquet de 52 cartes :

'''

shuffle\_enhanced.py

Shuffle a deck of 52 cards

'''

import random

class Card:

def \_\_init\_\_(self, suit, rank):

self.suit = suit

self.rank = rank

def initialize\_deck():

suits = ['Clubs', 'Diamonds', 'Hearts', 'Spades']

ranks = ['Ace', '2', '3','4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King']

cards = []

for suit in suits:

for rank in ranks:

card = Card(suit, rank) 1

cards.append(card)

return cards

def shuffle\_and\_print(cards):

random.shuffle(cards) 2

for card in cards: 3

print('{0} of {1}'.format(card.rank, card.suit))

if \_\_name\_\_ == '\_\_main\_\_':

cards = initialize\_deck()

shuffle\_and\_print(cards)

Chaque carte du jeu est représentée par un objet de la classe Card. Nous initialisons le jeu de 52 cartes dans la fonction initialize\_deck() en créant un objet Card pour chaque combinaison de couleur et numéro que nous stockons ensuite dans la liste cards 1. Nous appelons la fonction shuffle\_and\_print() avec cette liste de cartes, fonction dans laquelle nous utilisons la fonction random.shuffle() pour mélanger les cartes 2. Finalement nous affichons chacune des cartes dans un ordre mélangé 3.

#5 : Estimer la surface d’un disque

Le programme simulant des lancers de fléchettes pour estimer la surface d’un disque est le suivant :

'''

estimate\_circle\_area.py

Estimate the area of a circle

'''

import math

import random

def estimate(radius, total\_points):

center = (radius, radius)

in\_circle = 0

for i in range(total\_points):

x = random.uniform(0, 2\*radius)

y = random.uniform(0, 2\*radius)

p = (x, y)

# distance of the point created from circle's center

d = math.sqrt((p[0]-center[0])\*\*2 + (p[1]-center[1])\*\*2)

if d <= radius:

in\_circle += 1

area\_of\_square = (2\*radius)\*\*2

return (in\_circle/total\_points)\*area\_of\_square

if \_\_name\_\_ == '\_\_main\_\_':

radius = float(input('Radius: '))

area\_of\_circle = math.pi\*radius\*\*2

for points in [10\*\*3, 10\*\*5, 10\*\*6]:

print('Area: {0}, Estimated ({1}): {2}'.format(area\_of\_circle, points, estimate(radius, points)))

La fonction estimate() prend deux arguments en entrée – le rayon du cercle et le nombre de fléchettes virtuelles que nous voulons lancer sur la cible. Nous simulons un lancer de fléchette sur une surface carrée de côté 2\*radius à l’aide de la fonction random.uniform() pour générer les coordonnées x et y du point p, où la fléchette atteint la cible. Si la distance entre ce point et le centre est inférieure ou égale au rayon, cela signifie que la fléchette a atterrie dans le cercle. Finalement, nous retournons la fraction du nombre de fléchettes ayant atterries dans le cercle multiplié par l’aire du carré. Ceci nous donne une estimation de la surface du disque.

Lorsque nous augmentons le nombre de fléchettes lancées, l’aire estimée devient de plus en plus proche de l’aire théorique du disque :

Radius: 1

Area: 3.141592653589793, Estimated (1000): 3.14

Area: 3.141592653589793, Estimated (100000): 3.14756

Area: 3.141592653589793, Estimated (1000000): 3.143504

Le programme estimant la valeur de π est un programme très proche du programme précédent :

'''

estimate\_pi.py

Estimate the value of pi

'''

import math

import random

def estimate(total\_points):

radius = 1

center = (radius, radius)

in\_circle = 0

for i in range(total\_points):

x = random.uniform(0, 2\*radius)

y = random.uniform(0, 2\*radius)

p = (x, y)

# distance from circle's center

d = math.sqrt((p[0]-center[0])\*\*2 + (p[1]-center[1])\*\*2)

if d <= radius:

in\_circle += 1

return (in\_circle/total\_points)\*4

if \_\_name\_\_ == '\_\_main\_\_':

for points in [10\*\*3, 10\*\*5, 10\*\*6]:

print('Known value: {0}, Estimated ({1}): {2}'.format(math.pi, points, estimate(points)))

La fonction estimate() prend le nombre total de lancers que nous souhaitons simuler. Similairement au programme estimant l’aire du disque, nous comptons le nombre de fléchettes qui atterrissent dans le cercle et retournons la fraction de ce nombre sur le nombre total de fléchettes lancées multiplié par 4. Ceci est une estimation de la valeur de π. L’estimation devient de plus en plus fidèle lorsque le nombre de lancers augmente.

Chapitre 6

Solution aux défis

#1 : Ranger des cercles dans un carré

La solution à ce défi est le programme suivant :

'''

circle\_in\_square.py

Circles in a square

'''

from matplotlib import pyplot as plt

def draw\_square():

square = plt.Polygon([(1, 1), (5, 1), (5, 5), (1, 5)], closed=True)

return square

def draw\_circle(x, y):

circle = plt.Circle((x, y), radius=0.5, fc='y')

return circle

if \_\_name\_\_ == '\_\_main\_\_':

ax = plt.gca()

s = draw\_square()

ax.add\_patch(s)

y = 1.5

while y < 5: 1

x = 1.5

while x < 5: 2

c= draw\_circle(x, y) 3

ax.add\_patch(c) 4

x += 1.0

y += 1.0

plt.axis('scaled')

plt.show()

Nous ajoutons d’abord un carré (avec des côtés d’une longueur de 4) à la figure en créant un objet Polygon à l’aide de la fonction draw\_square(). Nous utilisons alors deux boucles « tant que » en 1 et 2 pour remplir le carré de cercles d’un rayon de 0,5. En 3 nous traçons le cercle et en 4 nous ajoutons le cercle à la figure.

#2 : Tracer un triangle de Sierpiński

Le programme traçant le triangle de Sierpiński est le suivant :

'''

sierpinski.py

Draw the Sierpinski triangle

'''

import random

import matplotlib.pyplot as plt

def transformation\_1(p):

x = p[0]

y = p[1]

x1 = 0.5\*x

y1 = 0.5\*y

return x1, y1

def transformation\_2(p):

x = p[0]

y = p[1]

x1 = 0.5\*x + 0.5

y1 = 0.5\*y + 0.5

return x1, y1

def transformation\_3(p):

x = p[0]

y = p[1]

x1 = 0.5\*x + 1

y1 = 0.5\*y

return x1, y1

def get\_index(probability):

r = random.random()

c\_probability = 0

sum\_probability = []

for p in probability:

c\_probability += p

sum\_probability.append(c\_probability)

for item, sp in enumerate(sum\_probability):

if r <= sp:

return item

return len(probability)-1

def transform(p):

# list of transformation functions

transformations = [transformation\_1, transformation\_2, transformation\_3]

probability = [1/3, 1/3, 1/3]

# pick a random transformation function and call it

tindex = get\_index(probability)

t = transformations[tindex]

x, y = t(p)

return x, y

def draw\_sierpinski(n):

# We start with (0, 0)

x = [0]

y = [0]

x1, y1 = 0, 0

for i in range(n):

x1, y1 = transform((x1, y1))

x.append(x1)

y.append(y1)

return x, y

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input('Enter the desired number of points in the Sierpinski Triangle: '))

x, y = draw\_sierpinski(n)

# Plot the points

plt.plot(x, y, 'o')

plt.title('Sierpinski with {0} points'.format(n))

plt.show()

Similairement au programme que nous avions écrit pour tracer la fougère de Barnsley, ce programme contient trois fonctions (transformation\_1, transformation\_2 et transformation\_3) correspondant aux trois transformations possibles sélectionnées avec une probabilité équivalente.

Lorsque vous exécutez ce programme, il demande à l’utilisateur de saisir le nombre de points constituant le triangle. Ce nombre correspond aussi au nombre de transformations appliquées à notre point initial (0, 0).

Essayez ce programme avec des nombres de points différents pour voir l’effet sur la figure tracée.

#3 : Explorer la fonction de Hénon

Le programme créant le graphique de Hénon pour 20000 points est le suivant :

''' henon.py

Plot 20,000 iterations of the Henon function

'''

import matplotlib.pyplot as plt

def transform(p):

x,y = p

x1 = y + 1.0 - 1.4\*x\*\*2

y1 = 0.3\*x

return x1, y1

if \_\_name\_\_ == '\_\_main\_\_':

p = (0, 0)

x = [p[0]]

y = [p[1]]

for i in range(20000):

p = transform(p)

x.append(p[0])

y.append(p[1])

plt.plot(x, y, 'o')

plt.show()

Le point initial (0, 0) subit 20000 transformations. La transformation, dans ce cas, est implémentée dans la fonction transform(). Nous stockons les coordonnées de chaque nouveau point créé dans les deux listes x et y pour enfin tracer le graphique.

Le programme animant le tracé de la fonction de Hénon nécessite un peu plus d’efforts :

'''

henon\_animation.py

Animating 20000 iterations of the Henon function

'''

import matplotlib.pyplot as plt

from matplotlib import animation

def transform(p):

x,y = p

x1 = y + 1.0 - 1.4\*x\*\*2

y1 = 0.3\*x

return x1, y1

def update\_points(i, x, y, plot):

plot.set\_data(x[:i], y[:i])

return plot,

if \_\_name\_\_ == '\_\_main\_\_':

p = (0, 0)

x = [p[0]]

y = [p[1]]

for i in range(10000):

p = transform(p)

x.append(p[0])

y.append(p[1])

fig = plt.gcf()

ax = plt.axes(xlim = (min(x), max(x)), ylim = (min(y), max(y)))

plot = plt.plot([], [], 'o')[0] 1

anim = animation.FuncAnimation(fig, update\_points, fargs=(x, y, plot), frames = len(x), interval = 25)

plt.title('Henon Function Animation')

plt.show()

Nous créons d’abord deux listes pour stocker les coordonnées des points calculés. Nous créons ensuite un graphique vide en appelant la fonction plot() avec deux listes vides 1. Rappelons que la fonction plot() de matplotlib retourne une liste d’objets (Chapitre 2). Dans notre cas, cette liste ne contient qu’un seul objet, lequel nous obtenons en utilisant l’index 0. Nous créons un label, plot, pour se référer à cet objet. Nous créons ensuite un objet FuncAnimation avec les arguments adéquats. L’animation utilisera le nombre de points pour définir son nombre d’images. L’image 1 ne contiendra que le premier point, l’image 2 sera composée des deux premiers points et ainsi de suite. La fonction update\_points() actualise la figure à chaque nouvelle image. Dans cette fonction, nous utilisons la fonction set\_data() de matplotlib pour ajouter des points à plot. La i-ème image contiendra les i premiers points dont les coordonnées sont stockées dans les listes x et y.

#4 : Dessiner l’ensemble de Mandelbrot

'''

mandelbrot.py

Draw a Mandelbrot set

Using "Escape time algorithm" from:

http://en.wikipedia.org/wiki/Mandelbrot\_set#Computer\_drawings

Thanks to http://www.vallis.org/salon/summary-10.html for some important

ideas for implementation.

'''

import matplotlib.pyplot as plt

import matplotlib.cm as cm

# Subset of the complex plane we are considering

x0, x1 = -2.5, 1

y0, y1 = -1, 1

def initialize\_image(x\_p, y\_p):

image = []

for i in range(y\_p):

x\_colors = []

for j in range(x\_p):

x\_colors.append(0)

image.append(x\_colors)

return image

def mandelbrot\_set():

# Number of divisions along each axis

n = 400

# Maximum iterations

max\_iteration=1000

image = initialize\_image(n, n)

# Generate a set of equally spaced points in the region

# above

dx = (x1-x0)/(n-1)

dy = (y1-y0)/(n-1)

x\_coords = [x0 + i\*dx for i in range(n)] 1

y\_coords = [y0 + i\*dy for i in range(n)] 2

for i, x in enumerate(x\_coords):

for k, y in enumerate(y\_coords):

z1 = complex(0, 0)

iteration = 0

c = complex(x, y)

while (abs(z1) < 2 and iteration < max\_iteration): 3

z1 = z1\*\*2 + c

iteration += 1

image[k][i] = iteration 4

return image

if \_\_name\_\_ == '\_\_main\_\_':

image = mandelbrot\_set()

plt.imshow(image, origin='lower', extent=(x0, x1, y0,y1), cmap=cm.Greys\_r, interpolation='nearest') 5

plt.show()

La fonction mandelbrot\_set() trace l’ensemble de Mandelbrot dans la région délimitée par les points (-2,5, -1) et (1, 1) pour un total de 1600 points équidistants. Nous utilisons la fonction initalize\_image() comme nous en parlions dans la description du problème afin d’initialiser une liste de listes, lesquelles contiendront les couleurs appropriées.

Nous générons 400 points également espacés le long de chaque axe en 1 et 2. Nous utilisons ensuite deux boucles « pour » pour parcourir les 1600 points et leur assigner une couleur. Pour définir quelle couleur doit être assignée, nous commençons pas créer un nombre complexe pour chacun de ces points à l’aide de la fonction complex(). Puis nous utilisons une boucle « tant que » 3 pour implémenter les étapes 4 et 5 de l’algorithme décrit dans l’énoncé du problème. Lorsque la boucle « tant que » prend fin, la valeur d’itération détermine la couleur du point 4.

Finalement, nous retournons la liste de listes, image, laquelle est passée en argument de la fonction imshow() 5. L’argument extent permet de délimiter la région que nous souhaitons colorier, il suffit de placer les coordonnées x des points en premier, suivis par les coordonnées y. Lorsque vous exécutez ce programme, vous verrez s’afficher l’ensemble de Mandelbrot (Figure 7).

Figure 7.png

Figure 7

L’ensemble de Mandelbrot composé de 1600 dans la région délimitée par les points (-2,5, -1,0) et (1,0, 1,0)

Si vous changez le nombre de points divisant chaque axe et jouez avec le nombre d’itérations, vous pourrez observer l’effet de ces changements sur la figure finale.

Chapitre 7

Solution aux défis

#1 : Vérifier la continuité d’une fonction en un point

La solution à ce défi est un programme permettant de trouver la limite à droite et à gauche d’une fonction en un point donné et évalue la fonction en ce point. Voici le programme :

'''

verify\_continuity.py

Verify the continuity of a function

'''

from sympy import Limit, Symbol, sympify, SympifyError

def check\_continuity(f, var, a):

l1 = Limit(f, var, a, dir='+').doit() 1

l2 = Limit(f, var, a, dir='-').doit() 2

f\_val = f.subs({var:a})

  if l1 == l2 and f\_val == l1:

print('{0} is continuous at {1}'.format(f, a))

  else:

print('{0} is not continuous at {1}'.format(f, a))

 if \_\_name\_\_ == '\_\_main\_\_':

f = input('Enter a function in one variable: ')

var = input('Enter the variable: ')

a = float(input('Enter the point to check the continuity at: '))

try:

  f = sympify(f)

except SympifyError:

  print('Invalid function entered')

else:

  var = Symbol(var)

d = check\_continuity(f, var, a)

La fonction ssolutions prend en entrée une fonction, la variable de cette fonction et le point auquel on souhaite vérifier la limite. Elle évalue ensuite les limites à gauche et à droite à l’aide du mot-clé dir lors de la création de l’objet Limit en 1 et en 2, ainsi que la valeur de la fonction au point donné à l’aide de la méthode subs(). Enfin, on vérifie si les limites l1 et l2 sont égales et, dans le cas échéant, on vérifie si elles sont égales à la valeur de la fonction en ce point. Si toutes les conditions sont vérifiées, la fonction est continue en ce point.

#2 : Implémenter l’algorithme du gradient descendant

Comme il est dit dans l’énoncé, la seule différence entre la solution à ce défi et le programme implémentant l’algorithme du gradient ascendant est la manière avec laquelle est créé chaque nouveau point. Le programme suivant implémente l’algorithme du gradient descendant afin de trouver la valeur minimum d’une fonction :

'''

grad\_descent.py

Use gradient descent to find the minimum value of a

single variable function. This also checks for the existence

of a solution for the equation f'(x)=0 and plots the intermediate

points traversed.

'''

from sympy import Derivative, Symbol, sympify, solve

import matplotlib.pyplot as plt

def grad\_descent(x0, f1x, x):

# check if f1x=0 has a solution

if not solve(f1x):

print('Cannot continue, solution for {0}=0 does not exist'.format(f1x))

return None

epsilon = 1e-6

step\_size = 1e-4

x\_old = x0

x\_new = x\_old–step\_size\*f1x.subs({x: x\_old}).evalf() 1

# list to store the X values traversed

X\_traversed = []

while abs(x\_old - x\_new) > epsilon: 2

X\_traversed.append(x\_new) 3

x\_old = x\_new

x\_new = x\_old - step\_size\*f1x.subs({x:x\_old}).evalf()

return x\_new, X\_traversed

def frange(start, final, interval):

numbers = []

while start < final:

numbers.append(start)

start = start + interval

return numbers

def create\_plot(X\_traversed, f, var):

# First create the graph of the function itself

x\_val = frange(-1, 1, 0.01)

f\_val = [f.subs({var:x}) for x in x\_val]

plt.plot(x\_val, f\_val, 'bo')

# calculate the function value at each of the intermediate

# points traversed

f\_traversed = [f.subs({var:x}) for x in X\_traversed]

plt.plot(X\_traversed, f\_traversed, 'r.')

plt.legend(['Function', 'Intermediate points'], loc='best')

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

f = input('Enter a function in one variable: ')

var = input('Enter the variable to differentiate with respect to: ')

var0 = float(input('Enter the initial value of the variable: '))

try:

f = sympify(f)

except SympifyError:

print('Invalid function entered')

else:

var = Symbol(var)

d = Derivative(f, var).doit()

var\_min, X\_traversed = grad\_descent(var0, d, var)

if var\_min:

print('{0}: {1}'.format(var.name, var\_min))

print('Minimum value: {0}'.format(f.subs({var:var\_min})))

create\_plot(X\_traversed, f, var) 4

En 1 nous créons le point. Comme dans le programme du gradient ascendant, nous continuons d’explorer de nouveaux points jusqu’à ce que la différence entre le point actuel et le point suivant soit inférieure à 1e-6 2. Nous stockons aussi tous les points parcourus dans une liste, X\_traversed 3, afin de pouvoir les marquer sur le graphique dans la fonction create\_plot(). Une exécution de ce programme pourrait ressembler à ceci :

Enter a function in one variable: **3\*x\*\*2 + 2\*x**

Enter the variable to differentiate with respect to: **x**

Enter the initial value of the variable: **0.1**

x: -0.331668643986980

Minimum value: -0.333325019761474

La Figure 8 montre le graphique de cet exemple.

Figure 8.png

Figure 8

Graphique de la fonction 3x2+2x montrant comment l’algorithme du gradient ascendant trouve le minimum de la fonction

#3 : L’aire entre deux courbes

Voici la solution à ce défi :

'''

area\_curves.py

Find the area enclosed by two curves between two points

'''

from sympy import Integral, Symbol, SympifyError, sympify

def find\_area(f, g, var, a, b):

a = Integral(f-g, (var, a, b)).doit()

return a

if \_\_name\_\_ == '\_\_main\_\_':

f = input('Enter the upper function in one variable: ')

g = input('Enter the lower function in one variable: ')

var = input('Enter the variable: ')

l = float(input('Enter the lower bound of the enclosed region: '))

u = float(input('Enter the upper bound of the enclosed region: '))

try:

f = sympify(f)

g = sympify(g)

except SympifyError:

print('One of the functions entered is invalid')

else:

var = Symbol(var)

print('Area enclosed by {0} and {1} is: {2} '.

format(f, g, find\_area(f, g, var, l, u)))

Lorsque le programme est lancé, il demande à l’utilisateur de saisir les deux fonctions et leurs valeurs de départ et d’arrivée. Il appelle alors la fonction find\_area() avec ces valeurs. Cette fonction calcule alors l’intégrale en créant un objet Integral, représentant l’aire entre les deux courbes en entre a et b. Voici un exemple d’exécution de ce programme :

Enter the upper function in one variable: **x+1**

Enter the lower upper function in one variable: **x\*exp(-x\*\*2)**

Enter the variable: **x**

Enter the lower bound of the enclosed region: **0**

Enter the upper bound of the enclosed region: **2**

Area enclosed by x + 1 and x\*exp(-x\*\*2) is: 3.50915781944437

Je dois vous informer que ce défi décrit un cas spécifique de calcul d’aire entre deux courbes. Vous pouvez en apprendre plus sur les autres cas que vous pourrez rencontrer à l’adresse http://tutorial.math .lamar.edu/Classes/CalcI/AreaBetweenCurves.aspx et écrire des programmes pour les résoudre. Notre exemple est une résolution de l’exemple 2 du site web.

#4 : Trouver la longueur d’une courbe

Ce programme calcule la longueur d’une fonction entre deux points :

'''

length\_curve.py

Find the length of a curve between two points

'''

from sympy import Derivative, Integral, Symbol, sqrt, SympifyError, sympify

def find\_length(fx, var, a, b):

deriv = Derivative(fx, var).doit()

length = Integral(sqrt(1+deriv\*\*2), (var, a, b)).doit().evalf()

return length

if \_\_name\_\_ == '\_\_main\_\_':

f = input('Enter a function in one variable: ')

var = input('Enter the variable: ')

l = float(input('Enter the lower limit of the variable: '))

u = float(input('Enter the upper limit of the variable: '))

try:

f = sympify(f)

except SympifyError:

print('Invalid function entered')

else:

var = Symbol(var)

print('Length of {0} between {1} and {2} is: {3} '.format(f, l, u, find\_length(f, var, l, u)))

Un exemple d’exécution de ce programme est montré ci-après. Nous utilisons les données de l’énoncé du défi :

Enter a function in one variable: 2\*x\*\*2 + 3\*x + 1

Enter the variable: x

Enter the lower limit of the variable: -5

Enter the upper limit of the variable: 10

Length of 2\*x\*\*2 + 3\*x + 1 between -5.0 and 10.0 is: 268.372650946022

Voici autre exemple avec des données issues de Khan Academy (https://www .khanacademy.org/math/integral-calculus/solid\_revolution\_topic/arc-length/v/ arc-length-example-2):

Enter a function in one variable: **x\*\*3/6 + 1/(2\*x)**

Enter the variable: **x**

Enter the lower limit of the variable: **1**

Enter the upper limit of the variable: **2**

Length of x\*\*3/6 + 1/(2\*x) between 1.0 and 2.0 is: 1.41666666666667