**Reverse**

def print\_reverse(text, print\_spaces=False):

"""

This is a function which prints out a string in reverse order

The first argument is a string, the second argument is a boolean which

determines if there are spaces added to the beginning of each line when

the string is printed out.

"""

stopPoint = -len(text) -1

count = len(text) - 1

# This for loop is going backwards starting at -1 so that the string

# is read from the end and eventually reaches the beginning

for counter in range(-1, stopPoint, -1):

#This if statement adds the spaces in front of each letter if

# print\_spaces is true

if print\_spaces == True:

startChar = " " \* count

else:

startChar = ""

count = count -1

print(startChar + text[counter])

if \_\_name\_\_ == "\_\_main\_\_":

print('Without Spaces:')

print\_reverse('reversestring', False)

print('With Spaces:')

print\_reverse('reversestring', True)

**Output:**

PS C:\Users\samra\OneDrive\CompSci Uni\Programming\CA1> python .\reverse.py

Without Spaces:

g

n

i

r

t

s

e

s

r

e

v

e

r

With Spaces:

g

n

i

r

t

s

e

s

r

e

v

e

r

PS C:\Users\samra\OneDrive\CompSci Uni\Programming\CA1>

**Longest**

from reverse import print\_reverse

def read\_words(filename):

"""

This function opens a file, reads it and then puts the contents of the file

into a list

The single argument is the name of the file

"""

file = open(filename, 'r')

lines = file.readlines()

file.close()

return lines

def longest(words):

"""

This function finds and prints the first longest word in a list

The single argument is a list

"""

longestLength = 0

for counter in range(len(words)):

# We are removing the new line part off of the end of the current word

# and saving the result in our variable

currentWord = words[counter].strip()

if len(currentWord) > longestLength:

longestWord = currentWord

longestLength = len(currentWord)

return longestWord, longestLength

def all\_longest(words):

"""

This funciton finds and prints a list of all the longest words in a list

of words

The single argument is a list

"""

longestLength = 0

longestWords = []

for counter in range(len(words)):

# We are removing the new line part off of the end of the current word

# and saving the result in our variable

currentWord = words[counter].strip('\n')

if len(currentWord) > longestLength:

# If it is a new longest word, we clear the list of longest words

# and add the new word to the end of the list

longestWords.clear()

longestWords.append(currentWord)

longestLength = len(currentWord)

elif len(currentWord) == longestLength:

# If the current word is the same length as the longest word, we

# add the current word to the list of longest words

longestWords.append(currentWord)

print(longestWords)

if \_\_name\_\_ == "\_\_main\_\_":

longW, longL = longest(read\_words('words.txt'))

print('longest word in the file:', longW, 'it has a length of', longL)

all\_longest(read\_words('words.txt'))

print\_reverse(longW)

**Output:**

PS C:\Users\samra\OneDrive\CompSci Uni\Programming\CA1> python .\longest.py

longest word in the file: formaldehydesulphoxylate it has a length of 24

['formaldehydesulphoxylate', 'pathologicopsychological', 'scientificophilosophical', 'tetraiodophenolphthalein', 'thyroparathyroidectomize']

e

t

a

l

y

x

o

h

p

l

u

s

e

d

y

h

e

d

l

a

m

r

o

f

PS C:\Users\samra\OneDrive\CompSci Uni\Programming\CA1>

**Fibonacci**

def fibonacci(n):

"""

This function calculates the fibonacci numbers and saves them in a list

The argument that is taken is the number of fibonacci numbers to be

calculated

"""

if n < 2:

F = 'n must be greater than or equal to 2'

else:

# The first two fibonacci numbers are initialised into the list so that

# the rest can be calculated

F = [0 ,1]

for count in range(2, n):

F.append(F[count-1] + F[count-2])

return F

if \_\_name\_\_ == "\_\_main\_\_":

print(fibonacci(20))

print(fibonacci(1))

**Output:**

PS C:\Users\samra\OneDrive\CompSci Uni\Programming\CA1> python .\fibonacci.py

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]

n must be greater than or equal to 2

PS C:\Users\samra\OneDrive\CompSci Uni\Programming\CA1>

**Golden Ratio**

from fibonacci import fibonacci

def goldenratio\_sequence(n):

"""

This function calculates the golden ratios between the numbers in the

fibonacci sequence.

The argument that it takes is the number of fibonacci numbers that want

to be used to estimate the golden ratio

The length of the list returned is 2 less than the number of fibonacci

numbers used

"""

fibList = fibonacci(n)

goldenRatio = []

for count in range(1, n - 1):

# Adding each of the new ratios to the end of the golden ratio list

goldenRatio.append(fibList[count + 1] / fibList[count])

return goldenRatio

def find\_minimum\_fibonacci\_numbers(tolerance=0.0000000001, maxnum=100):

"""

This function finds the minimum number of fibonacci numbers that need to

be used to estimate the golden ratio to within a certain tolerance

The two arguments are the tolerance, which takes a float value, and the

maxnum, which takes an integer value and tells the program the maximum

number of fibonacci number that will be used to estimate the golden ratio

"""

# We hard code in the actual golden ratio to compare against our estimated

# value

actualGoldenRatio = 1.6180339887498948

ratioList = goldenratio\_sequence(maxnum)

# The count starts at two because the golden ratio sequence program

# returns a list which has 2 less values than the number of fibonacci

# numbers used, therefore, we need to compensate for this

count = 2

for x in ratioList:

# This for loop works out the difference between our estimated value

# for the golden ratio and the actual value, if it is below the

# tolerance then it is the value that the user wants to recieve

count = count + 1

if abs(x - actualGoldenRatio) <= tolerance:

return (count), x

if \_\_name\_\_ == "\_\_main\_\_":

print(goldenratio\_sequence(20))

numFibNums, approxGR = find\_minimum\_fibonacci\_numbers(10\*\*-10)

print (numFibNums, 'numbers approximated to a tolerance of 10^-10.')

print('The approximated value is', approxGR)

numFibNums, approxGR = find\_minimum\_fibonacci\_numbers(10\*\*-14)

print (numFibNums, 'numbers approximated to a tolerance of 10^-14.')

print('The approximated value is', approxGR)

numFibNums, approxGR = find\_minimum\_fibonacci\_numbers(10\*\*-18)

print (numFibNums, 'numbers approximated to a tolerance of 10^-18.')

print('The approximated value is', approxGR)

**Output:**

PS C:\Users\samra\OneDrive\CompSci Uni\Programming\CA1> python .\goldenratio.py

[1.0, 2.0, 1.5, 1.6666666666666667, 1.6, 1.625, 1.6153846153846154, 1.619047619047619, 1.6176470588235294, 1.6181818181818182, 1.6179775280898876, 1.6180555555555556, 1.6180257510729614, 1.6180371352785146, 1.618032786885246, 1.618034447821682, 1.6180338134001253, 1.618034055727554]

27 numbers approximated a value to a tolerance of 10^-10.

The approximated value is 1.6180339886704431

37 numbers approximated a value to a tolerance of 10^-14.

The approximated value is 1.6180339887498896

42 numbers approximated a value to a tolerance of 10^-18.

The approximated value is 1.618033988749895

PS C:\Users\samra\OneDrive\CompSci Uni\Programming\CA1>