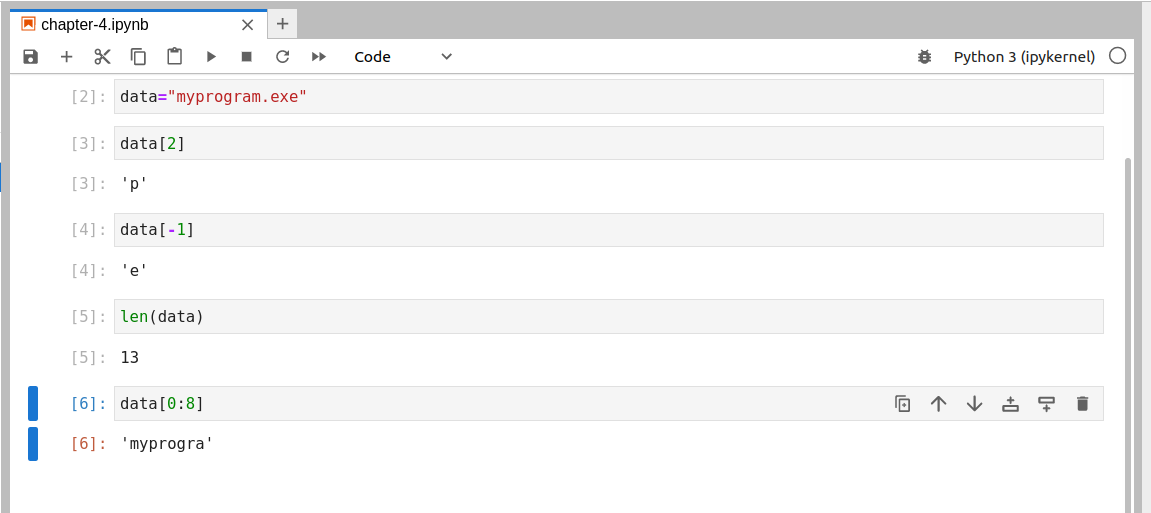
**Chapter 4 – Week 6 – Exercises**

Exercises #1 – page 106

1. **Assume that the variable data refers to the string “myprogram.exe”. Write the values of the following expressions:**
2. **data[2]** ‘p’
3. **data[-1]** ‘e’
4. **len(data)** 13
5. **data[0:8]** ‘myprogra’



1. **Assume that the variable data refers to the string “myprogram.exe”. Write the expressions that perform the following tasks:**

a) Extract the substring **“gram”** from **data**.

data[5:9]

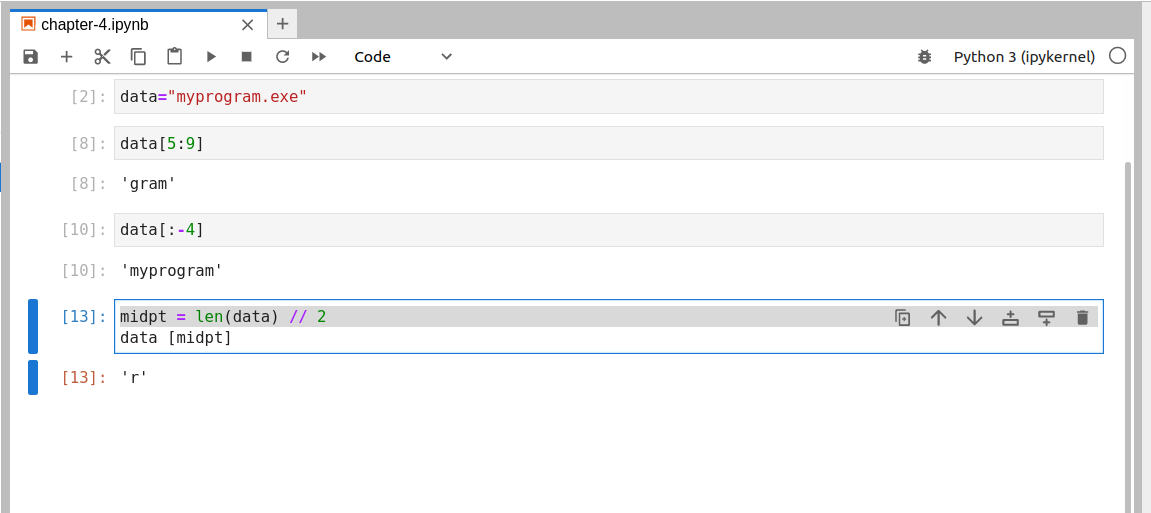
b) Truncate the extension **“.exe”** from **data**.

data[:-4]

c) Extract the character at the middle position from **data**.

midpt = len(data) // 2

data[midpt]



1. **Assume that the variable myString refers to a string. Write a code segment that uses a loop to print the characters of the string in reverse order.**

Option 1: (uses only things from book)

start = len(myString) - 1

for ch in range(start,-1, -1):

print( myString[ch], end="" )

Option 2: (uses this reversed fn I found at *https://docs.python.org/3/library/functions.html*)

for ch in reversed( myString ):

print( ch, end="" )

1. **Assume that the variable myString refers to a string, and the variable reversedString refers to an empty string. Write a loop that adds the characters from myString to reversedString in reverse order.**

Same options with a slight change.

Option 1:

for ch in reversed( myString ):

reversedString += ch

Option 2:

start = len(myString) - 1

for ch in range(start,-1, -1):

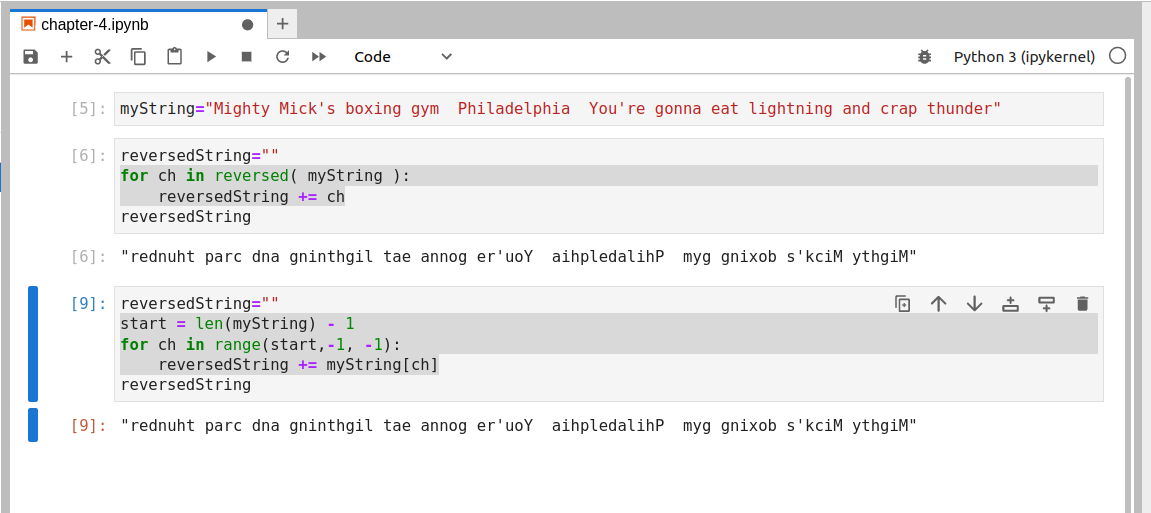
reversedString += myString[ch]

1. **Assume the variable teststring refers to a string. Write a loop that prints each character in this string followed by its ASCII value.**

teststring="Randomness"

for ch in teststring:

print( ch, end=" " )

****

Exercises #2 – page 109

1. **Write the encrypted text of each of the following words using a Caesar cipher with a distance of 3:**

a) python

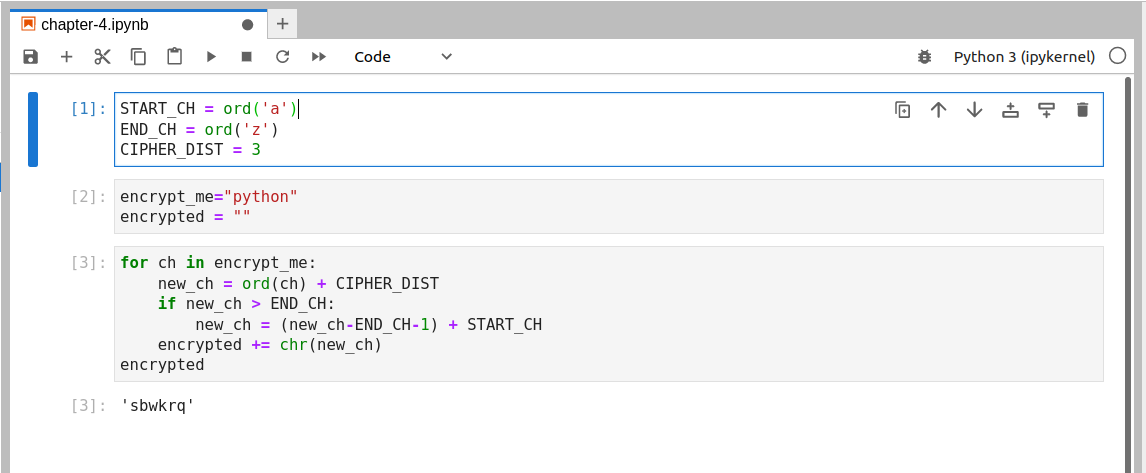
sbwkrq

b) hacker

kdfnhu

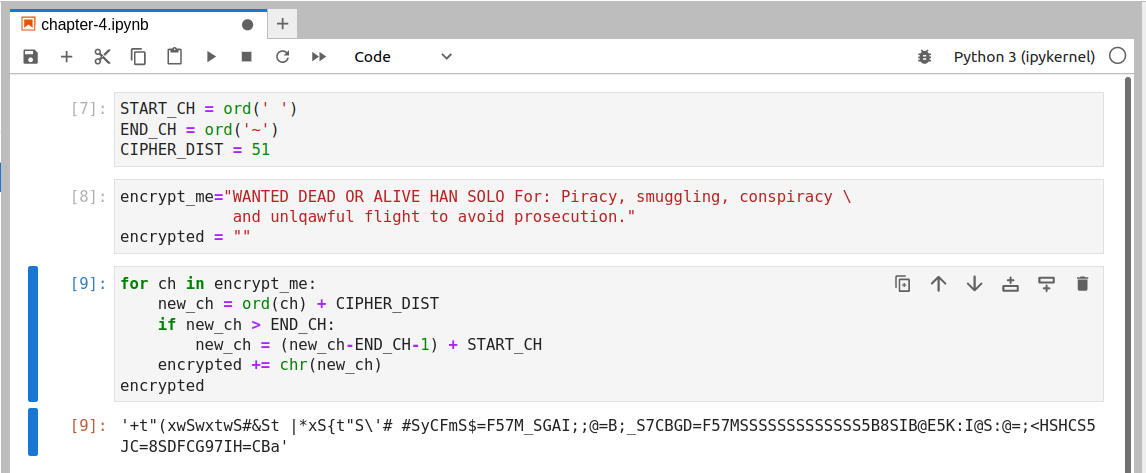
c) wow

zrz



1. **Consult the Table of ASCII values in Chapter 2 and suggest how you would modify the encryption and decryption scripts in this section to work with strings containing all of the printable characters.**

In order to “encrypt” all of the printable characters, first one has to decide what characters are considered printable. Based on the chart on page 48 of the text, I would start with ascii 32 (the space) and end with the tilde (~, ascii 126). Then our “circle of encryption” would be larger and require only minor code changes.



1. **You are given a string that was encoded by a Caesar cipher with an unknown distance value. The text can contain any of the printable ASCII characters. Suggest an algorithm for cracking the code.**

An algorithm for undoing a Caesar cipher:

1 – Examine the text for the character with the largest and smallest values.

2 – Use these values to guess at the wrapping point of the cipher.

Perhaps only one case is used. Perhaps numerals are added.

3 – Starting with an offset of 0 and incrementally going through all possible numbers up to the difference between the starting and ending values of the cipher’s “loop”.

4 – After each time step 3 is performed, determined if the message can now be read. If so, keep a copy of the results. If is mostly readable, keep this result as well.

5 – If we have saved exactly one entry from #4, we consider this a success and report the message to the General.

6 – If we have multiple readable messages, we should send them all to the General. This is probably all we can do.

7 – If we have no fully readable entries from #4, then we check our nearly readable ones. We will use this as a clue as to how we may have slightly missed the mark on the loop edges.

8 – If we have nothing to go on yet, we go to step and consider other ways the loops edges may have been constructed.

Exercises #3 – page 114-115

1. **Translate each of the following numbers to decimal numbers:**

a) 110012 = 2510

b) 1000002 = 3210

c) 111112 = 3110

1. **Translate each of the following numbers to binary numbers:**

a) 4710 = 1011112

b) 12710 = 11111112

c) 6410 = 10000002

1. **Translate each of the following numbers to binary numbers:**

a) 478 = 1001112

b) 1278 = 10101112

c) 648 = 1101002

1. **Translate each of the following numbers to decimal numbers:**

a) 478 = 3910

b) 1278 = 8710

c) 648 = 5210

1. **Translate each of the following numbers to decimal numbers:**

a) 4716 = 7110

b) 12716 = 29510

c) AA16 = 17010

Exercises #4 – page 118

1. **Assume that the variable data refers to the string “Python rules!”. Use a string method from Table 4-2 to perform the following tasks:**

a) Obtain a list of the words in the string

**data.split()**

b) Convert the string to uppercase.

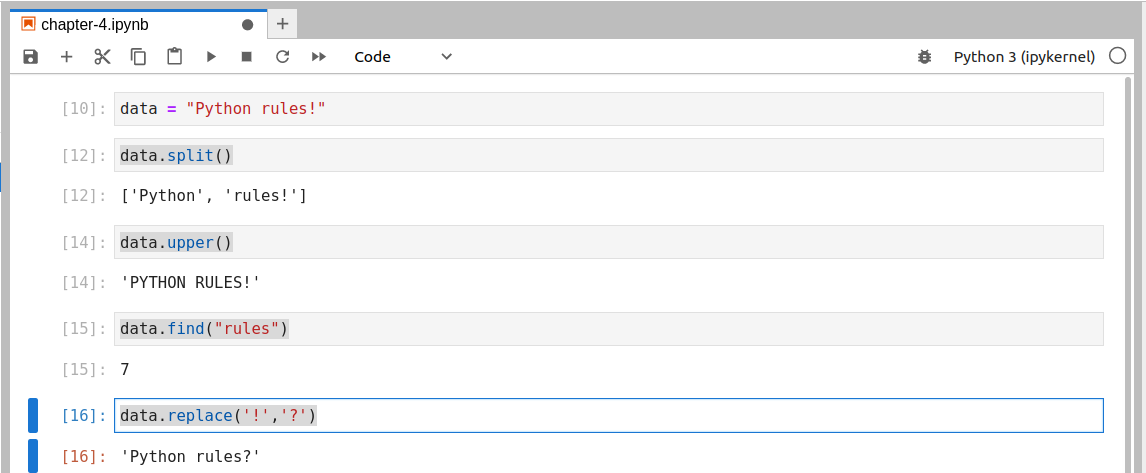
**data.upper()**

c) Locate the position of the string **“rules”**.

**data.find("rules")**

d) Replace the exclamation point with a question mark.

**data.replace('!','?')**

****

1. **Using the value of data from Exercise 1, write the values of the following expressions:**

a) data.endswith(‘i’)

**False**

b) “ totally “.join(data.split())

**'Python totally rules!'**

Exercises #4 – page 125

1. **Write a code segment that opens a file named myfile.txt for input and prints the number of lines in the file.**

file\_name = "myfile.txt"

fd = open( file\_name, 'r' )

contents = fd.read()

fd.close()

line\_count = contents.split('\n')

len(line\_count)

****

1. **Write a code segment that opens a file for input and prints the number of four-letter words in the file.**

file\_name = "Henry-VII.txt"

fd = open( file\_name, 'r' )

contents = fd.read()

fd.close()

words = contents.split()

four\_letter\_word\_count = 0

for w in words:

if len(w) == 4:

four\_letter\_word\_count += 1

print(four\_letter\_word\_count)

1. **Assume that a file contains integers separated by newlines. Write a code segment that opens the file and prints the average value of the integers.**

fd = open( file\_name, 'r' )

contents = fd.read()

fd.close()

words = contents.split()

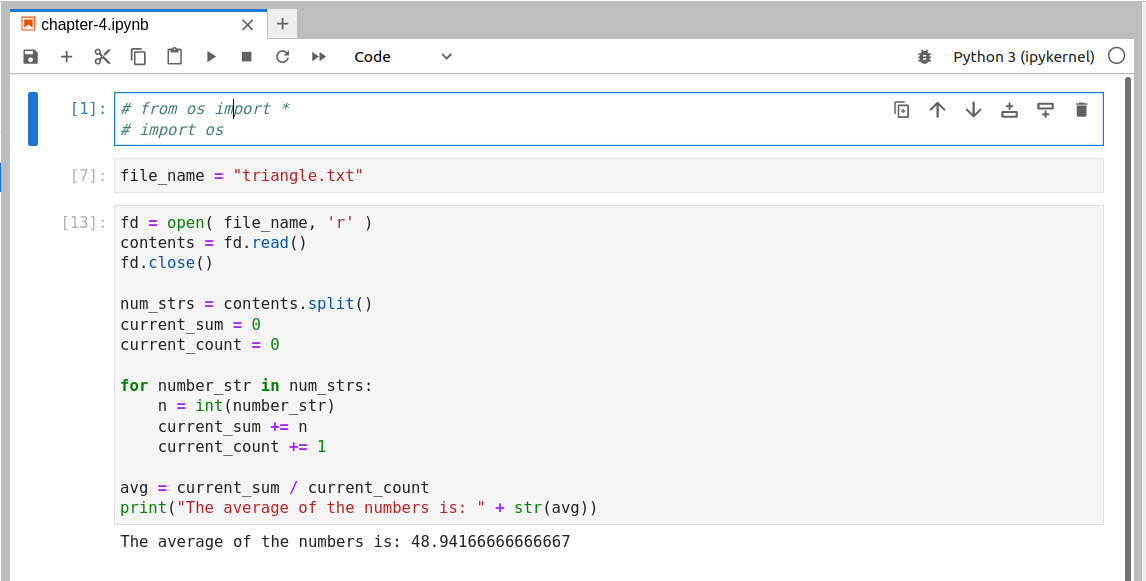
four\_letter\_word\_count = 0

for w in words:

if len(w) == 4:

four\_letter\_word\_count += 1

print(four\_letter\_word\_count)

****

1. **Write a code segment that prints the names of all the items in the current working directory.**

file\_names = os.listdir()

for fn in file\_names:

print(fn)

1. **Write a code segment that prompts the user for a filename. If the file exists, the program should print its contents on terminal. Otherwise, it should print an error message.**

import os

file\_name = input("Enter file name: ")

file\_exists = os.path.isfile(file\_name)

if file\_exists:

fd = open(file\_name,'r')

print(fd.read())

fd.close()

else:

print("Sorry, I was unable to find " + file\_name + "." )

