**Previous Test Questions**

**Question 1:**

**Write a Python function that has one parameter, an integer, and print the sum of all its divisors. A divisor of an integer number n is a number which divides n without remainder.**

**If a negative number is passed a message should be printed instead.**

**Your program should also print the divisors that were added. Consider the blueprint code below:**

***# Function implementation***

***# Main code***

**sum\_divisors(20)**

**sum\_divisors(33)**

**sum\_divisors(42)**

**sum\_divisors(-1)**

**The output in this case should be:**

**1 + 2 + 4 + 5 + 10 + 20 = 42**

**1 + 3 + 11 + 33 = 48**

**1 + 2 + 3 + 6 + 7 + 14 + 21 + 42 = 96**

**Negative number. Only positive numbers accepted.**

**Hint: Find out first the sum of the divisors. After that use a string to store the sequence of numbers being added that needs to be printed.**

def sum\_divisors(n):

if n < 0:

print(“Negative number. Only positive numbers accepted.”)

return

sum = 0

equation = “”

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

if n % i == 0:

sum += i

equation += str(i) + “ + “

equation = equation[:-2] + “= ” + str(sum)

print(equation)

*# Main code*

sum\_divisors(20)

sum\_divisors(33)

sum\_divisors(42)

sum\_divisors(-1)

**Question 2:**

**Write a program to encrypt a string. The encryption should work as follows: every character in the string gets replaced with the character with ASCII code plus the index value of the index.**

**Example: “Monty Python” becomes “Mppw}%V|qyy”.**

**First character M (index 0) adds 0 to its ASCII code, resulting in the same character M**

**Second character o (index 1) adds 1 to its ASCII code, resulting in character p**

**Third character n (index 2) adds 2 to its ASCII code, resulting in character p**

**Last character n (index 11)addes 11 to its ASCII code, resulting in character y.**

**Your program should prompt the user for a string, encrypt it, and print the encrypted result.**

**Hint: Use the functions chr() and ord()**

my\_str = input(“Please enter a string: “)

# for i in range(len(my\_str)):

print(i, my\_str[i])

# or

encrypted = “”

for i, ch, in enumerate(my\_str):

encrypted += chr(ord(ch) + 1)

print(encrypted)

**Question 3:**

**Write a Python function that takes as input a string that stores date and time (24-hour clock) in the following format:“DD/MM/YYYY HR:MIN:SEC” and prints the following:**

**-> DD/MM/YYYY**

**-> HR:MIN:SEC**

**-> MM/YYYY**

**-> Whether the time is a.m. or p.m.**

**Validation of the input in the function is necessary. For example, if the user gives an input of “04/122/1990 13:12:12”, the given string is invalid, as there can be only 12 months in a year. In this case prints an error message instead. Think of all possible erroneous inputs and write code to handle them. The function doesn’t return anything. Consider the blueprint below:**

**import string**

**def format\_time(date\_time):**

***# function implementation***

**format\_time("21/02/2020 18:06:00")**

**print()**

**format\_time("37/05/1950 12:00:00")**

**print()**

**format\_time("01/01/1900 25:06:00")**

**The output of this code could be:**

**-> 21/02/2020**

**-> 18:06:00**

**-> 02/2020**

**-> p.m.**

**Days are wrong**

**Hour is wrong**

**Hints: (1) Use the split() function and its parameter to break down the problem. Use it multiple times if necessary. (2) Once you find an error you can print a message and use the keyword return to leave the function.**

import string

def format\_time(date\_time):

date, time = date\_time.split()

if len(date.split(“/”) != 3):

print(“Date is wrong”)

return

if len(time.split(“/”) != 3):

print(“Time is wrong”)

return

day, month, year = date.split(“/”)

hour, minutes, seconds = time.split(“:”)

if int(day) < 1 or int(day) > 31:

print(“Days are wrong”)

return

if int(hour) < 0 or int(hour) > 24:

print(“Hours are wrong”)

return

print(“->” + date)

print(“->” + time)

print(“->” + month + “/” + “year”)

if hour > 11:

print(“->p.m”)

else:

print(“-> a.m.”)

# Main code

format\_time("21/02/2020 18:06:00")

print()

format\_time("37/05/1950 12:00:00")

print()

format\_time("01/01/1900 25:06:00")

**Question 4:**

**In mathematics a positive number is called a Kaprekar number if the representation of the square of the number can be split into two parts (not necessarily of the same length) that add up to the original number (the second part is allowed to start with a leading zero).**

**If a number is smaller than 10, it is not a Kaprekar number.**

**For example, 45 is a Kaprekar number as 45\*45 = 2025, and 20+25 = 45.**

**Another Kaprekar number is 55, as 55\*55 = 3025, and 30+25 = 55**

**On the other hand 12 is NOT a Kaprekar number, as 12\*12 = 144.**

**We can partition 144 as 1 and 44, or as 14 and 4.**

**1 + 44 = 45 ≠ 12**

**14 + 4 = 18 ≠ 12**

**So there doesn’t exist a partition of 144 where the sum of both parts add up to 12.**

**The Kaprekar numbers from 10 to 1000 are:**

**10 45 55 99 100 297 703 999**

***(Note: this is an adapted definition of Kaprekars numbers for the purpose of this lab test.)***

**YOUR TASK:**

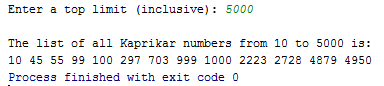
**Write a Python program that will prompt the user for a number, and print all the Kaprekar numbers from 10 up to that number.**

**Use functions where appropriate to structure your code.**

**For example, you may want to implement the following functions:**

* **A function that will take a number and check if it is a Kaprekar number**
* **A function that will take as a parameter a number n and will print all Kaprekar numbers from 10 to n**

**Sample output:**

****

def is\_kaprekar(n):

n2 = n \* n

n2\_str = str(n2) # 100 -> 10 + 0

first\_part = int(n2\_str[:len(n2\_str) // 2])

second\_part = int(n2\_str[len(n2\_str) // 2:])

if first\_part + second\_part == n;

return True

first\_part = int(n2\_str[:len(n2\_str) // 2 + 1])

second\_part = int(n2\_str[len(n2\_str) // 2 + 1:])

if first\_part + second\_part == n;

return True

# for i in range(1, len(n2\_str)):

# first\_part = int(n2\_str[:i])

# second\_part = int(n2\_str[i:])

#

# if first\_part + second\_part == n;

# return True

return False

def all\_kaprekars(upper\_limit):

for i in range(10, upper\_limit + 1):

if is\_kaprekar(i):

print(i, end=“ ”)

upper\_limit = int(input(“Enter an upper limit: ”))

all\_kaprekars(upper\_limit)

**Question 5:**

**Write a Python function that will take a text (sentence) and will reverse every second word. For example, for the input:**

**I love you for a lifetime**

**Not only for a day**

**I love you for who you are**

**Not what you do or say**

**the program should produce:**

**I evol you rof a emitefil**

**Not ylno for a day**

**I evol you rof who uoy are**

**Not tahw you od or yas**

text = “I love you for a lifetime”

words = text.split()

reversed = “”

for i, word in enumerate(words):

if i % 2 == 1:

reversed += word[::-1] + “ ”

else:

reversed += word + “ ”

print(reversed)

**Question 6:**

**Many word puzzles can be solved by iterating through a list of words while checking for characteristics specified by the puzzle. A list of words word\_list.txt is provided in Brightspace. The file has one word per line.**

**A blueprint file is also provided in the file puzzles.py. Your overall goal is to implement the functions in this file and achieve the same output as below. You MUST NOT change the code in the main scope or the functions’ headers.**

**\*Puzzle a\***

**consumptively**

**counterexamples**

**improvisational**

**manipulators**

**multiprocessing**

**\*Puzzle b\***

**wei**

**usher**

**ushers**

**\*Puzzle c\***

**brightly**

**churchly**

**knightly**

**lengthly**

**mccarthy**

**mcknight**

**promptly**

**rhythmic**

**\*Puzzle d\***

**juxtaposing**

**Expected output**

**Hints:**

**i) The word\_list.txt has 45,425 words. Do not write more than two nested loops to iterate through all the words. For example:**

**for word1 in word\_list:**

**for word2 in word\_list:**

**for word3 in word\_list:**

**This will result in 91 trillion operations and your code won’t finish running. All the questions can be answered by iterating over the list of words once.**

**ii) You will need to iterate over lists and strings. Some good friends: lenfunction, in operator, count and find methods from strings.**

**Task 1**

**Complete each puzzle implementing its respective function. Print the result of each puzzle inside its function.**

**(a) Find 5 uncapitalized, unhyphenated words that contain 9 of the letters of the alphabet from l to v ("lmnopqrstuv").  
  
(b) What words consist of two consecutive pronouns? This list of pronouns will be helpful.**

**pronouns = ['thou', 'thee', 'thine', 'thy', 'i', 'me','mine', 'my', 'we', 'us', 'ours', 'our', 'you', 'yours', 'your','he','him','his', 'she', 'her', 'hers', 'it', 'its', 'they', 'them', 'theirs', 'their']**

**(c) Find all uncapitalized, seven-letter words, containing just a single vowel that does not have the letter s anywhere within it.**

**(d) When you are writing in script, there are four letters of the alphabet that cannot be completed in one stroke: i and j (which require dots) and t and x (which require crosses). Find a word that uses each of these letters exactly once.**

def get\_word\_list(file\_name):

""" Read the word\_list.txt file and returns a list with one word per element.

Each word in the list is in lower case.

For example: [aarhus, aaron, ababa, aback, ...]. """

try:

data\_file = open(file\_name, "r")

except IOError:

print("File could not be opened.")

exit()

word\_list = [] # start with an empty word list

for word in data\_file: # for every word (line) in the file

# strip off end−of−line characters and make each word lowercase

# then append the word to the word list

word\_list.append(word.strip().lower())

return word\_list

def puzzle\_a(words):

""" Find 5 uncapitalized, unhyphenated words that contain 9

of the letters of the alphabet from l to v ("lmnopqrstuv"). """

for word in words:

if “-” not in word:

sum = 0

for c in word:

if c in "lmnopqrstuv":

sum += 1

if sum == 9:

print(word)

break

def puzzle\_b(words):

""" What words consist of two consecutive pronouns? """

pronouns = ['thou', 'thee', 'thine', 'thy', 'i', 'me', 'mine', 'my',

'we', 'us', 'ours', 'our', 'you', 'yours', 'your', 'he', 'him',

'his', 'she', 'her', 'hers', 'it', 'its', 'they', 'them',

'theirs', 'their']

def puzzle\_c(words):

""" Find all uncapitalized, seven-letter words, containing just

a single vowel that does not have the letter s anywhere within it. """

def puzzle\_d(words):

""" When you are writing in script, there are four letters of the

alphabet that cannot be completed in one stroke: i and j (which require dots)

and t and x (which require crosses). Find a word that uses each of

these letters exactly once. """

# Main code

word\_list = get\_word\_list("word\_list.txt")

print("\*Puzzle a\*")

puzzle\_a(word\_list)

print("\n\*Puzzle b\*")

puzzle\_b(word\_list)

print("\n\*Puzzle c\*")

puzzle\_c(word\_list)

print("\n\*Puzzle d\*")

puzzle\_d(word\_list)