**Name: Session:**

**Programming I**

**Lab Exercise 10.18.2024**

It is often important to determine the execution time of a program. Here is an example of a simple way to do that.

import time

def main():

total = 0

for i in range(1000000):

total += i

start\_time = time.time()

main()

stop\_time = time.time()

print 'Program execution time =', stop\_time - start\_time, 'seconds'

The program does one million additions. On my nine year old computer it takes about 0.3 seconds. Notice I do not do any prints, as this would significantly slow down things and all I am testing the efficiency of the algorithm.

Hint: On problem 1 and 2 you will be using a nested for loop to get all combinations.

1. Write a program to determine how long it takes to do all of the multiplications from 1 x 1 to 1000 x 1000.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ seconds

1. Write a program to determine how long it takes to do all of the multiplications from 1 x 1 to 10000 x 10000.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ seconds

1. What can you generalize about the efficiency of your algorithm?
2. Write a program that will ask the user to enter a monetary amount up to $9,999.99 and prints it out in words. For example, if the user entered 5343.79 they program would generate the string

Five thousand three hundred forty three dollars and seventy nine cents

There are many ways to tackle this problem but you might consider building dictionaries.

We will start by taking the amount of money and breaking it down into dollars and cents as such:

dollars = int(amount)

cents = 100\*((amount - dollars)+ 0.005) #take care of rounding problem

cents = int(cents)

You will have to add a half of cent (0.005) to your cents calculation because of the rounding errors in the three conversions (int to float to int)

Next we will break apart the dollars part as thus:

thousands = dollars // 1000

hundreds = dollars // 100 % 10

tens = dollars // 10 % 10

ones = dollars % 10

Now we will break apart the cents part the same way:

dimes = cents // 10

pennies = cents % 10

Now we will build the dollar string:

m4 = money1000[thousands]

m3 = money100[hundreds]

if tens\*10 + ones < 10 or tens\*10 + ones > 19: #not a teen

m2 = money10[tens]

m1 = money1[ones]

else: #a teen

m2 = moneyTeens[10+ones]

m1 = ''

Notice that we will have to handle the teens in a separate manner. A number is not a teen if it is less than 10 or more than 19. I know 10 is technically not a teen but treating it in the same category as a teen will simplify our code.

Now we will build the cents string. It is very similar to the two least significant digits of the dollar string. We will also have the same issue with the teens.

if dimes\*10 + pennies < 10 or dimes\*10 + pennies > 19: #not a teen

c2 = cents10[dimes]

c1 = cents1[pennies]

else: #a teen

c2 = centsTeens[10+pennies]

c1 = ''

Finally, we will put everything together and print out our complete string.

**When you have completed problems 1, 2 and 4, submit your documented source code.**