CS3080-001 Spring 2024

HW03

PROBLEM:

Problem: 1

Assignment:

Subject: Printing a large integer with thousands separators (commas).
Modules: none
Filename: discrete_math.py, HW04_01.py

Write a function, named pretty_int(n), that takes a single argument, assumed to be a non-negative
integer, and returns a string representing that number and that includes the comma separators.
Place this function in the discrete_math module and include test code that prints out the values, 0, 1,
999, 1000, 2^16, and 2^64 using your pretty_int() function

Work:

None

Output:

The results of the test cases

0

1

999

1,000

65,536

18,446,744,073,709,551,616

CODE:

```
'''
PROGRAMMER: =Carson L. King
USERNAME: cking20
PROGRAM: hw04_03.py

DESCRIPTION: Driver program for pretty int
'''

from discrete_math import pretty_int

#Puts the test codes specfied in the pdf
test_1 = pretty_int(0)
test_2 = pretty_int(1)
test_3 = pretty_int(999)
test_4 = pretty_int(1000)
test_5 = pretty_int(2**16)
test_6 = pretty_int(2**64)

#prints the results
print("The results of the test cases")
print(test_1)
print(test_2)
print(test_3)
```

```
print(test_4)

print(test_5)

print(test_6)

(in discrete_math)

def pretty_int(n):

    return "{:,.0f}".format(n)
```

Problem #2

ASSIGNMENT:

Subject: Finding a prime factor of a number

Modules: none
Filename: discrete_math.py, HW04_002.py

Add another function to your discrete_math module named prime_factor(n) that takes a single argument, n, that is assumed to be an integer and returns a 2-tuple in which the first member, p, is a prime factor of n and the second is n/p. The first number can be ANY prime factor of p, it does NOT have to be the smallest prime factor. You do not have to determine whether or not the second member is prime.
If the number is prime, then the return value should be (n, 1). If the function fails to find a prime factor, for any reason, it should return (1, n), which can be used to detect the failure since 1 is not a prime number. Reasons for returning this “error value” include the number passed not being integer-valued, the number not having any prime factors (which is the case for any integer smaller than 2, since 2 is the smallest prime number), or the function having insufficient time to find a prime factor.

Work:

None

Output:

```
    0 = (1)*(0) ( 0.000 seconds)
    1 = (1)*(1) ( 0.000 seconds)
    2 = (2)*(1) ( 0.000 seconds)
    3 = (3)*(1) ( 0.000 seconds)
    12 = (2)*(6) ( 0.000 seconds)
    97 = (97)*(1) ( 0.000 seconds)
    5,782,475,771 = (69,151)*(83,621) ( 0.005 seconds)
    4,698,643,325,249 = (1,264,447)*(3,715,967) ( 0.092 seconds)
    253,049,136,761,293 = (12,957,929)*(19,528,517) ( 0.948 seconds)
    18,241,119,882,520,215,552 = (320,019,647)*(57,000,000,011)
( 21.542 seconds)
    37,530,294,278,910,763,008 = (612,671)*(61,256,847,931,289)
( 0.040 seconds)
```

CODE:

```
def prime_factor(n):
    if not isinstance(n, int) or n < 2:
        return (1, n)
    for p in range(2, n):
        if n % p == 0:
            return (p, n // p)
    return (n, 1)
In HW04_02
```

```
'''
PROGRAMMER: =Carson L. King
USERNAME: cking20
PROGRAM: hw04_03.py

DESCRIPTION: Multithreaded RSA Factoring program
'''

import discrete_math as dm

rsa_list = []
with open("rsa_numbers.txt", "rt") as fp:
    strings = fp.readlines()

for s in strings:
    rsa_list.append(int(s))

dm.factor_list(rsa_list, 1200)
print("done")
```

Problem 3

Assignment:

Modules: none

Filename: HW04_03.py, discrete_math.py

This is where everything comes together. In this problem, the task is to factor as many of the RSA
numbers contained in the file “rsa_numbers.txt” as possible in no more

than 1200 seconds (20 minutes)
by using the functions you have already written to write a new function, factor_list(), that exploits
mul0threading.
A driver script, hw04_03.py, is provided for you

Work:

None

Output:

Factoring a list of RSA numbers

List length: 10 numbers

Time limit: 8 seconds

Results:

263,531,430,756,403 = 6,186,493*42,597,871 --- (1.106 sec)

263,533,126,545,097 = 6,186,527*42,597,911 --- (2.147 sec)

263,532,029,957,197 = 6,186,503*42,597,899 --- (2.355 sec)

38,273,635,989,997 = 6,186,527*6,186,611 --- (2.744 sec)

263,538,196,264,043 = 6,186,619*42,598,097 --- (3.037 sec)

TIME EXPIRED for 1814589007617233

TIME EXPIRED for 1814590285559783

TIME EXPIRED for 1814585173809743

TIME EXPIRED for 1814588326047229

TIME EXPIRED for 1814582021563777

Successfully factored 5 numbers.

Terminating 5 child threads.

Clean up complete, exiting program.

Done

Code:

```
results = {}
lock = threading.Lock()
def factor_thread(number):
    global results
    start_time = time.time()

    #uses the previous prime factor
    factors = prime_factor(number)

    #tracks time per factor
    end_time = time.time()
    elapsed_time = end_time - start_time

    with lock:
        results[number] = (factors, elapsed_time)

def factor_list(numbers, time_limit):
    global results
    threads = []
    count = 0

    print(f"Factoring a list of RSA numbers\nList length:
{len(numbers)} numbers\nTime limit: {time_limit} seconds")
    print("-" * 50)
```

```
    #Begins the process while starting the timer
    for number in numbers:
        t = threading.Thread(target=factor_thread, args=(number,))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

    #Checks if it took more time than required
    print("Results:")
    for number, (factors, elapsed_time) in results.items():
        if elapsed_time > time_limit:
            print(f"TIME EXPIRED for {number}")
            count += 1
        else:
            print(f"{pretty_int(number)} = {'*'.join(map(pretty_int,
factors))} --- ( {elapsed_time:.3f} sec )")

    print(f"Successfully factored {len(numbers) - count} numbers.")
    print(f"Terminating {count} child threads.")
    print("Clean up complete, exiting program.")
(In HW04_03)
PROGRAMMER: =Carson L. King
USERNAME: cking20
PROGRAM: hw04_03.py
```

```
DESCRIPTION: Multithreaded RSA Factoring program
'''

import discrete_math as dm

rsa_list = []
with open("rsa_numbers.txt", "rt") as fp:
    strings = fp.readlines()

for s in strings:
    rsa_list.append(int(s))

dm.factor_list(rsa_list, 1200)
print("done")
```