Problem #1

Subject: Printing a large integer

Filename: discrete_math.py
Allowed modules: NONE

WORK

For this task, I just need to format a given number using commas. This can be done with a formatted string as the following: f”number:,”

OUTPUT

discrete_math.py : pretty_int() test

0 = 0

1 = 1

999 = 999

1000 = 1,000

65536 = 65,536

18446744073709551616 = 18,446,744,073,709,551,616

CODE

```
def pretty_int(n):
   return f'{n:,}'
```

Problem #2

Subject: Finding a Prime number

Filename: discrete_math.py
Allowed modules: time

WORK

For this problem, I just need to find any prime number factor from any other given number. For this, we don't need to check every single number, at most we need to check every number up to the square root of the given number.

OUTPUT

discrete_math.py : prime_factor() test

0 = (0)*(1) (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) (0.017 seconds)

4,698,643,325,249 = (1,264,447)*(3,715,967) (0.280 seconds)

253,049,136,761,293 = (12,957,929)*(19,528,517) (2.816 seconds)

18,241,119,882,520,216,117 = (320,019,647)*(57,000,000,011) (93.533 seconds)

37,530,294,278,910,762,919 = (612,671)*(61,256,847,931,289) (0.169 seconds)

CODE

```
def prime_factor(n):
  start = time.time()


  for i in range(2, int(n ** 0.5) + 1):
    if (time.time() - start) >= 1200:
      return (1, n)
    if n % i == 0:
      return (i, n // i)


  return (n, 1)
```

Problem #3

Subject: Finding a Prime number

Filename: discrete_math.py
Allowed modules: time, threading, discrete_math

WORK

For this section, I need 2 more functions, factor_thread(), and factor_list().

factor_thread() needs to take 3 parameters: the number to factor, a results list to append since this will not be able to return a variable, and a lock to synchronize access to the list.

factor_list() will need to use the other functions to factor the RSA numbers within a time limit. It will need to take in the list of RSA numbers, and a time limit (in seconds). This will then create threads each assigned a single RSA number, and factor until complete, or until the time limit is reached.

OUTPUT

discrete_math.py : factor_list() test

Factoring a list of RSA numbers

List length: 10 numbers

Time limit : 8 seconds

38273635989997 = (6186527,6186611) --- (53.194 sec)

263532029957197 = (6186503,42597899) --- (53.194 sec)

263531430756403 = (6186493,42597871) --- (53.195 sec)

263538196264043 = (6186619,42598097) --- (53.195 sec)

263533126545097 = (6186527,42597911) --- (53.195 sec)

1814582021563777 = (42597899,42597923) --- (53.195 sec)

1814589007617233 = (42597889,42598097) --- (53.195 sec)

1814588326047229 = (42597871,42598099) --- (53.195 sec)

1814590285559783 = (42597917,42598099) --- (53.196 sec)

1814585173809743 = (42597917,42597979) --- (53.196 sec)

Successfully factored 10 numbers.

Terminating 0 child threads.

Clean up complete, exiting program.

CODE

```
def factor_thread(number, result_list, lock):
    global time_expired
    factor = prime_factor(number)
    lock.acquire()
    try:
        result_list.append((number, factor))
    finally:
        lock.release()

def factor_list(rsa_list, time_limit):
    global time_expired
    threads = []
    results = []
    lock = threading.Lock()
    start_time = time.time()

    #create threads for each RSA number
    for number in rsa_list:
        thread = threading.Thread(target=factor_thread, args=(number, results, lock))
        threads.append(thread)
        thread.start()

    #factor numbers until time limit is reached
    while time.time() - start_time < time_limit:
```

```
        if threading.active_count() == 1:
            break
        time.sleep(0.1)

    time_expired = True

    for thread in threads:
        thread.join()

    print("Factoring a list of RSA numbers")
    print("List length:", len(rsa_list), "numbers")
    print("Time limit :", time_limit, "seconds")

    for number, factor in results:
        print("{} = ({}*{}) --- ( {:.3f} sec )".format(number, factor[0], factor[1], time.time() -
start_time))

    success_count = sum(1 for _, factor in results if factor[0] != 1)
    print("\nSuccessfully factored {} numbers.".format(success_count))

    for thread in threads:
        if thread.is_alive():
            thread.join(timeout=0.1)

    print("Terminating", threading.active_count() - 1, "child threads.")
    print("Clean up complete, exiting program.")
```