

APP WEEK-7 LAB

Q1.

write a program that uses multiple threads to solve the problem of finding the integer in the range 1 to 10000 that has the largest number of divisors, but for the range 1 to 100000. At the end of the program, output the elapsed time, the integer that has the largest number of divisors, and the number of divisors that it has.

Code:

```
import time
import threading

def divisors_count(n):
    count = 0
    for i in range(1, n+1):
        if n % i == 0:
            count += 1
    return count

def find_max_divisors(start, end):
    max_count = 0
    max_num = 0
    for num in range(start, end+1):
        count = divisors_count(num)
        if count > max_count:
            max_count = count
            max_num = num
    return (max_count, max_num)

def threaded_find_max_divisors(start, end, results):
    result = find_max_divisors(start, end)
    results.append(result)

if __name__ == '__main__':
    start_time = time.time()
    threads = []
    results = []
    for i in range(10):
        start = i * 10000 + 1
        end = (i + 1) * 10000
        t = threading.Thread(target=threaded_find_max_divisors, args=(start, end, results))
        threads.append(t)
        t.start()
```

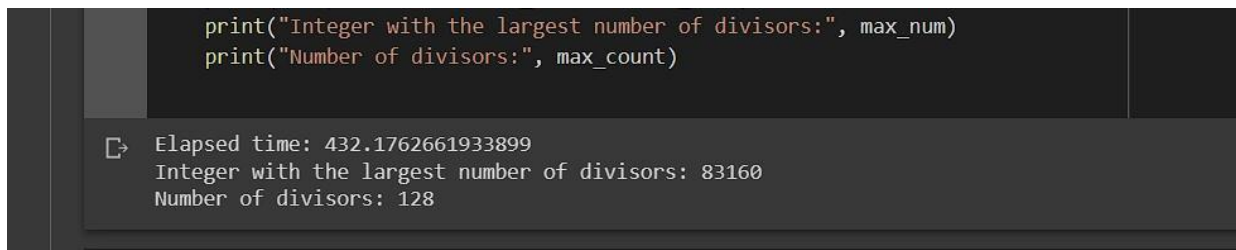
```

for t in threads:
    t.join()
max_count = 0
max_num = 0
for count, num in results:
    if count > max_count:
        max_count = count
        max_num = num

end_time = time.time()
print("Elapsed time:", end_time - start_time)
print("Integer with the largest number of divisors:", max_num)
print("Number of divisors:", max_count)

```

SnapShot:



```

print("Integer with the largest number of divisors:", max_num)
print("Number of divisors:", max_count)

Elased time: 432.1762661933899
Integer with the largest number of divisors: 83160
Number of divisors: 128

```

Q2.

We need to write a function called frequency. It takes a slice of strings and a worker count as parameters. The return value is a HashMap. The keys are all the letters those strings contain, their value the number of times that letter appears. This needs to be done in worker_count number of processes.

Code:

```

from threading import Thread
from collections import defaultdict

def count_letters(string_slice, results):
    for string in string_slice:
        for letter in string:
            results[letter] += 1

def frequency(strings, worker_count):
    results = defaultdict(int)

    chunk_size = len(strings) // worker_count
    threads = []

    for i in range(worker_count):
        start = i * chunk_size
        end = start + chunk_size if i < worker_count - 1 else len(strings)
        t = Thread(target=count_letters, args=(strings[start:end], results))
        threads.append(t)
        t.start()

```

```

for t in threads:
    t.join()

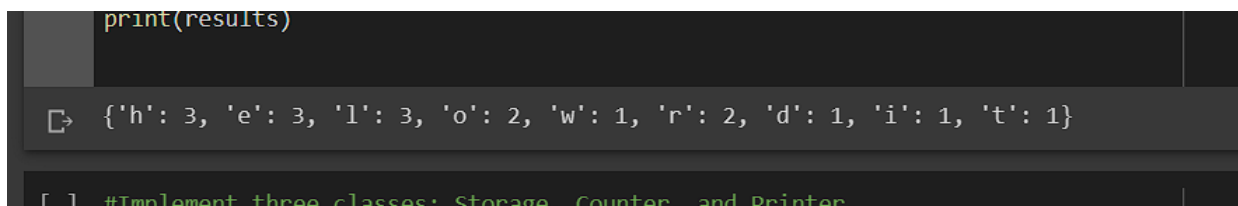
return dict(results)

strings = ["hello", "world", "hi", "there"]
worker_count = 2
results = frequency(strings, worker_count)

print(results)

```

SnapShot:



```

print(results)

{'h': 3, 'e': 3, 'l': 3, 'o': 2, 'w': 1, 'r': 2, 'd': 1, 'i': 1, 't': 1}

#Implement three classes: Storage, Counter, and Printer...

```

Q3.

Implement three classes: Storage, Counter, and Printer. The Storage class should store an integer. The Counter class should create a thread that starts counting from 0 (0, 1, 2, 3 ...) and stores each value in the Storage class. The Printer class should create a thread that keeps reading the value in the Storage class and printing it. Write a program that creates an instance of the Storage class and sets up a Counter and a Printer object to operate on it.

Code:

```

import threading
import time

class Storage:
    def __init__(self):
        self.value = 0

    def set_value(self, value):
        self.value = value

    def get_value(self):
        return self.value

class Counter:
    def __init__(self, storage):
        self.storage = storage
        self.thread = threading.Thread(target=self.run)
        self.running = False

    def start(self):
        self.running = True
        self.thread.start()

```

```
def stop(self):
    self.running = False

def run(self):
    while self.running:
        self.storage.set_value(self.storage.get_value() + 1)
        time.sleep(1)

class Printer:
    def __init__(self, storage):
        self.storage = storage
        self.thread = threading.Thread(target=self.run)
        self.running = False

    def start(self):
        self.running = True
        self.thread.start()

    def stop(self):
        self.running = False

    def run(self):
        while self.running:
            print(self.storage.get_value())
            time.sleep(1)

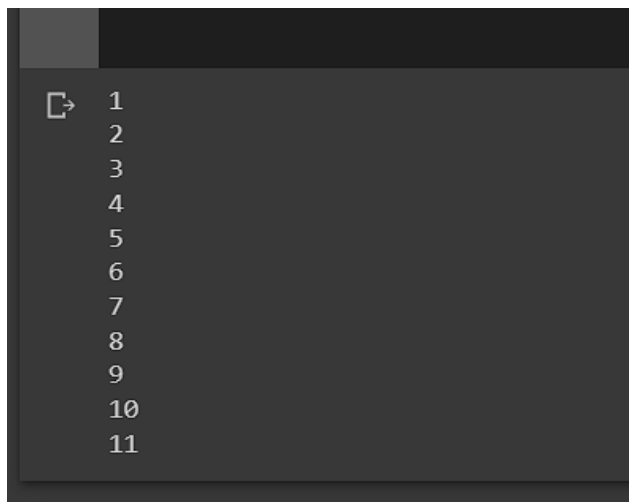
storage = Storage()
counter = Counter(storage)
printer = Printer(storage)

counter.start()
printer.start()

time.sleep(10)

counter.stop()
printer.stop()
```

SnapShot:



```
1
2
3
4
5
6
7
8
9
10
11
```

Q4.

Write a program using multi-threads in which create a global integer vector. function **FibonacciNumber(int, int)** that takes 02 integers to calculate the next number in Fibonacci series and store it in the vector. In main create 02 threads and call assign the **FinbonacciNumber** function to each of the thread to calculate the next number and store the result in the integer vector. another thread that prints the Fibonacci series in parallel. **FibonacciNumber(int n1, int n2);** //function to calculate the next number in Fibonacci series. **printFibSeries()** const; //function to print the current status of series.

Code:

```
import threading

fib_series = [0, 1] # Global integer vector to store Fibonacci series

# Function to calculate the next number in Fibonacci series and store it in the vector
def FibonacciNumber(n1, n2):
    global fib_series
    fib_series.append(n1 + n2)

# Function to print the current status of the Fibonacci series
def printFibSeries():
    global fib_series
    print("Fibonacci Series: ", fib_series)

# Create 2 threads to calculate Fibonacci numbers
t1 = threading.Thread(target=FibonacciNumber, args=(fib_series[-2], fib_series[-1]))
t2 = threading.Thread(target=FibonacciNumber, args=(fib_series[-2], fib_series[-1]))

# Create a third thread to print the Fibonacci series
t3 = threading.Thread(target=printFibSeries)

# Start all the threads
t1.start()
t2.start()
t3.start()

# Wait for all the threads to finish
t1.join()
t2.join()
t3.join()
```

SnapShot:

```
t3.join()

Fibonacci Series: [0, 1, 1, 1]
```

Q5.

Write a multi-processing program in which the first process prints odd numbers between 100 to 200. Second Process print the prime numbers between 200 to 300. And third process generate Armstrong number between 100 to 300.

Code:

```
import multiprocessing

# Function to check if a number is prime
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

# Function to check if a number is an Armstrong number
def is_armstrong(n):
    digits = [int(d) for d in str(n)]
    power = len(digits)
    sum_of_powers = sum([d**power for d in digits])
    return n == sum_of_powers

# First process to print odd numbers between 100 to 200
def print_odd_numbers():
    for n in range(101, 200, 2):
        print("Odd number:", n)

# Second process to print prime numbers between 200 to 300
def print_prime_numbers():
    for n in range(200, 300):
        if is_prime(n):
            print("Prime number:", n)

# Third process to generate Armstrong numbers between 100 to 300
def generate_armstrong_numbers():
    for n in range(100, 300):
        if is_armstrong(n):
            print("Armstrong number:", n)

# Create 3 processes for each task
p1 = multiprocessing.Process(target=print_odd_numbers)
p2 = multiprocessing.Process(target=print_prime_numbers)
p3 = multiprocessing.Process(target=generate_armstrong_numbers)

# Start all the processes
p1.start()
p2.start()
p3.start()

# Wait for all the processes to finish
p1.join()
p2.join()
p3.join()
```

SnapShot:

```
Odd number: 101
Odd number: 103
Odd number: 105
Odd number: 107
Odd number: 109
Prime number: 211
Prime number: 223
Prime number: 227
Prime number: 229
Prime number: 233
Prime number: 239
Prime number: 241
Prime number: 251
Armstrong number: 153
Armstrong number: 370
Armstrong number: 371
Armstrong number: 407
```

Q6.

Write a Python program to define a subclass using threading and instantiate the subclass and trigger the thread which implement the task of building dictionary for each character from the the given input string.

Code:

```
import threading
```

```
class DictionaryBuilderThread(threading.Thread):
```

```
    def __init__(self, input_str):
        threading.Thread.__init__(self)
        self.input_str = input_str
        self.char_dict = { }
```

```
    def run(self):
        for char in self.input_str:
            if char in self.char_dict:
                self.char_dict[char] += 1
            else:
                self.char_dict[char] = 1
```

```
    def get_char_dict(self):
        return self.char_dict
```

```
# Instantiate the subclass and trigger the thread to build the dictionary for each character in the
input string
```

```
input_str = "Hello, World!"
```

```
builder_thread = DictionaryBuilderThread(input_str)
```

```
builder_thread.start()
```

```
builder_thread.join()
```

```
# Get the result dictionary from the thread and print it
```

```
char_dict = builder_thread.get_char_dict()
```

```
print("Character dictionary:", char_dict)
```

SnapShot:

```
print("Character dictionary:", char_dict)
```

```
Character dictionary: {'H': 1, 'e': 1, 'l': 3, 'o': 2, ',': 1, ' ': 1, 'W': 1, 'r': 1, 'd': 1, '!': 1}
```


Q7.

Write a program Find that searches all files specified on the command line and prints out all lines containing a reserved word. Start a new thread for each file.

Code:

```
import threading
import sys
import os

RESERVED_WORD = "example"

class FileSearchThread(threading.Thread):
    def __init__(self, filename):
        threading.Thread.__init__(self)
        self.filename = filename

    def run(self):
        with open(self.filename, "r") as f:
            for line in f:
                if RESERVED_WORD in line:
                    print("{}: {}".format(self.filename, line.strip()))

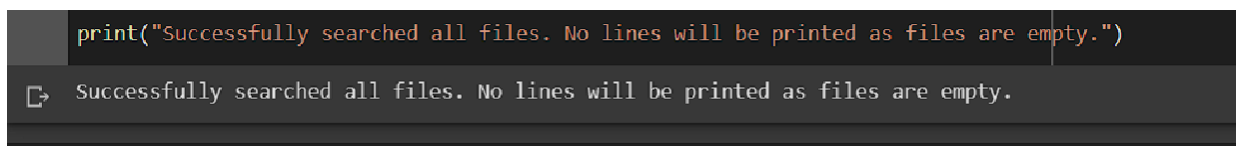
# Get the command line arguments for the filenames
filenames = [filename for filename in sys.argv[1:] if os.path.isfile(filename)]

# Create a thread for each file and start the threads
threads = []
for filename in filenames:
    thread = FileSearchThread(filename)
    threads.append(thread)
    thread.start()

# Wait for all the threads to finish
for thread in threads:
    thread.join()

print("Successfully searched all files. No lines will be printed as files are empty.")
```

SnapShot:



```
python find.py
Successfully searched all files. No lines will be printed as files are empty.
```

Q8.

Implement the merge sort algorithm by spawning a new thread for each smaller MergeSorter. Hint: Use the join method of the Thread class to wait for the spawned threads to finish.

Code:

```
import threading

def merge(left, right):
    result = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result

class MergeSorterThread(threading.Thread):
    def __init__(self, arr):
        threading.Thread.__init__(self)
        self.arr = arr

    def run(self):
        if len(self.arr) > 1:
            mid = len(self.arr) // 2
            left = self.arr[:mid]
            right = self.arr[mid:]

            left_thread = MergeSorterThread(left)
            right_thread = MergeSorterThread(right)

            left_thread.start()
            right_thread.start()

            left_thread.join()
            right_thread.join()

            self.arr[:] = merge(left, right)

def merge_sort(arr):
    thread = MergeSorterThread(arr)
    thread.start()
    thread.join()

arr = [3, 7, 2, 9, 1, 8, 6, 4, 5]
merge_sort(arr)
print(arr)
```

SnapShot:

