HWA CHONG INSTITUTION
C2 BLOCK TEST 2020

COMPUTING
Higher 2

29 June 2020 **Paper 2 (9569 / 02)** **0815 -- 1015 hrs**

Additional Materials:

Electronic version of `INSERTTOTREE.txt` data file

Electronic version of `SEARCHINTREE.txt` data file

Electronic version of `BOOK.txt` data file

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

The number of marks is given in brackets [ ] at the end of each question or part question. The total number of marks for this paper is **70**.

This document consists of **7** printed pages and **1** blank page.

1. The task is to create a NoSQL database for a library to manage its book records.

Task 1.1

Write a python program to:

- Create a MongoDB database named `library` and a new collection named `books`.
- Insert the book documents into the `books` collection in the `library` database. Use the sample dataset given in the file `BOOK.txt`. You should paste the contents of this file into your program.
- Display all book documents in the `books` collection.

Save your program code as `TASK1_1_<your name>_<ct>.py`

Run the program. Produce a screenshot of the output and save it as
`TASK1_1_<your name>_<ct>.jpg` [3]

Task 1.2

Write program code which make use of `books` collection in `library` database to:

- display the `book id`, `title` and `author` for those books published in 2015.
- display all book documents with `page_count` greater than or equal to 100 and less than 400.
- update `page_count` field to `'Less Than 100 Pages'` for those books where `page_count` field does not exist.
- display the `book title`, `page_count` and `year` of publication according to the `year` in descending order.

All outputs should have appropriate messages to indicate what you are showing.

Save your program code as `TASK1_2_<your name>_<ct>.py`

Produce a screenshot of the output and save it as
`TASK1_2_<your name>_<ct>.jpg` [7]

Solution Guide”

Task 1.1

```
from pymongo import MongoClient
try:
        client = MongoClient('localhost', 27017)
        print ('Connected Successfully!!!\n')
except:
        print ('Could not connect to MongoDB')

db = client["library"]
bkCol = db["books"]

bkList = [
        {"book_id": 234, "title": "Father Night", "author": "Kurt", "publisher": "APress",
"page_count": 433, "year": "2018"},
        {"book_id": 134, "title": "Mother Night", "author": ["Kurt", "Dan"], "publisher":
"APress", "year": "2015"},
        {"book_id": 334, "title": "Programming C## 6.0", "author": ["Andrew", "Dan"],
"page_count": 300, "year": "2000"},
        {"book_id": 534, "title": "Introduction to Python", "publisher": "MPH", "year": "1999"},
        {"book_id": 434, "title": "Travel with Dogs", "author": "Andy", "publisher": "APress",
"page_count": 100, "year": "2017"}
       ]

bkCol.insert_many(bkList)

docs = bkCol.find()
for doc in docs:
      print (doc)

print()
print ('List of Collections:')
print (db.list_collection_names())
print ('List of Databases:')
print (client.list_database_names())

client.close()
```

```
# Task 1.2

from pymongo import MongoClient
try:
        client = MongoClient('localhost', 27017)
        print ('Connected Successfully!!!\n')
except:
        print ('Could not connect to MongoDB\n')

#db = client.get_database("library")
db = client["library"]
bkCol = db["books"]

print ("All books in books collection")
for x in bkCol.find():
      print (x)
print()

#  query 1
print ("Books published in 2015")
criteria = {"year": "2015"}
docs = bkCol.find(criteria, {"book_id": 1, "title": 1, "author": 1, "_id": 0})
for doc in docs:
      print (doc)
print()

# query 2
print ("100 <= page_count < 400")
criteria = {"page_count": {"$gte":100, "$lt":400}
#criteria = {"$and":[{"page_count":{"$gte":100}}, {"page_count":{"$lt":400}}]}
docs = bkCol.find(criteria)
for doc in docs:
      print (doc)
print()

# query 3
print ("Update page_count")
criteria = {"page_count": {"$exists": False}}
newValues = {"$set":{"page_count": 'Less Than 100 Pages'}}
bkCol.update_many(criteria, newValues)
print()

# query 4
print ("Books in descending order by year of publication")
docs = bkCol.find({}, {"title":1,"page_count":1, "year":1,"_id": 0}).sort("year", -1)
for doc in docs:
```

```
    print (doc)
print()


client.close()
```

```
Books published in 2015
{'book_id': 134, 'title': 'Mother Night', 'author': ['Kurt', 'Dan']}

100 <= page_count < 400
{'_id': ObjectId('5e96651f6d02aaf40aee446a'), 'book_id': 334, 'title': 'Programm
ing C## 6.0', 'author': ['Andrew', 'Dan'], 'page_count': 300, 'year': '2000'}
{'_id': ObjectId('5e96651f6d02aaf40aee446c'), 'book_id': 434, 'title': 'Travel w
ith Dogs', 'author': 'Andy', 'publisher': 'APress', 'page_count': 100, 'year': '
2017'}

Update page_count

Books in descending order by year of publication
{'title': 'Father Night', 'page_count': 433, 'year': '2018'}
{'title': 'Travel with Dogs', 'page_count': 100, 'year': '2017'}
{'title': 'Mother Night', 'year': '2015', 'page_count': 'Less Than 100 Pages'}
{'title': 'Programming C## 6.0', 'page_count': 300, 'year': '2000'}
{'title': 'Introduction to Python', 'year': '1999', 'page_count': 'Less Than 100
 Pages'}
```

2. A prime number is an integer greater than 1 that is only divisible by 1 and itself. For example, the first five prime numbers are 2, 3, 5, 7 and 11.

   The task is to generate all prime numbers and display the total number of prime numbers up to a specified number.

   For each of the subtasks, add a comment statement, at the beginning of the code using the hash symbol '#', to indicate the sub-task the program code belongs to, for example,

In [1]:
```
# Task 2.1
Program Code
```

In [2]:
```
# Task 2.2
Program Code
```

output:

In [3]:
```
# Task 2.3
Program Code
```

output:

Task 2.1

Write program code for a function to determine whether a given integer `n` is prime. Use the following specification.

```
FUNCTION isPrime(n: INTEGER) RETURNS BOOLEAN
```

The function has a single parameter `n` and returns `TRUE` if `n` is prime, and `FALSE` otherwise.

[4]

Task 2.2

Write program code to:

- use the `isPrime()` in Task 2.1
- display all the prime numbers up to 100
- print out the total number of prime numbers up to 100. [4]

The most efficient way to find all the prime numbers less than or equal to an integer value n is by using **The Sieve of Eratosthenes**. It was developed by Eratosthenes, an ancient mathematician.

The algorithm is as follows:

- Write down all numbers from 0 to n
- Cross out 0 and 1 because they are not prime
- Set p equal to 2
- **While** p is less than n **do**
    - Cross out all multiples of p (but not p itself)
    - Set p to the next number in the list that is not crossed out
- Output all of the numbers that have not been crossed out as prime

Example

To find all the prime numbers less than or equal to n, say 20.

Generate a list of integers from 0 to 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Cross out 0 and 1

~~0~~	~~1~~	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Cross out all multiples of 2

		2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~

Cross out all multiples of 3

		2	3		5		7		~~9~~		11		13		~~15~~		17		19	

5, 7, 11, 13, 17 and 19 contain no multiples in the list

The numbers not crossed out at this point, [2, 3, 5, 7, 11, 13, 17, 19], are all the prime numbers less than or equal to 20

Hint: You may simulate crossing out a number by replacing it with 0. Then, once the algorithm completes, all of the non-zero values in the list are prime.

Task 2.3

Write program code to:

- implement this algorithm
- display all the prime numbers between 2 and 100. [7]

Download your program code and output for Task 2 as

`TASK2_<your name>_<ct>.ipynb`

Solution Guide

```
# Task 2.1 & 2.2
from math import sqrt
def isPrime(num):
    if num == 2:
        return True
    if (num < 2 or num % 2 == 0):
        return False

    # test factors 3, 5, 7, 9 ... up to square root of num
    for test in  range(3, int(sqrt(num))+1 ,2):
        if num % test == 0:
            return False
    return True

def main():
    primeCnt = 0
    print ("The primes up to 100 are: ")
    for i in range(1, 100+1):
        if isPrime(i):
            primeCnt += 1
```

```
        print (i, end = ' ')


    print ("\n\nTotal number of prime numbers up to 100:", primeCnt)


main()
```

[1] correct prime numbers

[1] correct count

```
The primes up to 100 are:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

Total number of prime numbers up to 100: 25
```

Task 2.3

```
def main():
    n = int(input("Generate all primes up to "))
    nums = []
    for i in range(0, n+1):
        nums.append(i)
      nums[0] = nums[1] = 0
     p = 2        # the smallest prime
    while p < n:
         #  cross out every multiple of p
        for i in range(p*2, n+1, p):
            nums[i] = 0


        # move to the next higher prime
        p = p + 1
        while p < n and nums[p] == 0:
            p += 1


    print ("The primes up to", n, "are: ")
    for i in nums:
```

```
    if nums[i] != 0:
        print (i, end = ' ')

main()
```

[1] correct output

```
Generate all primes up to 100
The primes up to 100 are:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

---

3. Design the server program and client program for an asymmetric guess-the-letter game where a server generates a random capital letter from A to Z and a client tries to guess it. After each incorrect guess, the server returns whether the answer is before or after the guessed letter. Once the client guess the letter correctly, both the client and server quit the game. Below is a sample run of the client.

```
Enter guess (A-Z): H
The answer is after your guess.
Enter guess (A-Z): S
The answer is before your guess.
Enter guess (A-Z): M
The answer is after your guess.
Enter guess (A-Z): O
The answer is after your guess.
Enter guess (A-Z): P
You win!
```

The socket protocol is to use `\n` to detect the end of a message and you do not need to handle invalid input of client. [10]

Save your program code as
`TASK3_client_<your name>_<ct>.py and`
`TASK3_server_<your name>_<ct>.py`

Run the program. Produce a screenshot of the output and save it as

`TASK3_<your name>_<ct>.jpg`

client.py

```
import socket

s = socket.socket()
s.connect(('127.0.0.1', 9999))

data = b''
while True:
    while b'\n' not in data:
        data += s.recv(1024)
    received = data[:data.find(b'\n')]
    data = data[len(received) + 1:]

    if received == b'LOW':
        print('The answer is after your guess.')
    elif received == b'HIGH':
        print('The answer is before your guess.')
    elif received == b'GUESS':
        guess = input('Enter guess (A-Z): ')
        s.sendall(guess.encode() + b'\n')
    elif received == b'WIN':
        print('You win!')
        break

s.close()
```

server.py

```
import socket
from random import randint

s_listen = socket.socket()
s_listen.bind(('127.0.0.1',9999))
s_listen.listen()
s, addr = s_listen.accept()

lower = ord('A')
upper = ord('Z')
answer = chr(randint(lower, upper))

guessed = False

while True:
    s.sendall(b'GUESS\n')

    data = b''
    while b'\n' not in data:
        data += s.recv(1024)
    guess = data[:data.find(b'\n')].decode()

    if guess < answer:
        s.sendall(b'LOW\n')
    elif guess > answer:
        s.sendall(b'HIGH\n')
    else:
```

```
            guessed = True
            s.sendall(b'WIN\n')
            break

s.close()
s_listen.close()
```

4. A programmer is writing a program to manipulate Binary Search Tree using Object-Oriented Programming (OOP).

   A class, `TreeNode`, will store the following data for each node in the tree:
   - Value
   - Left pointer, pointing to the left node
   - Right pointer, pointing to the right node

   A class, `BSTree`, will store the pointer to the root node of the tree. It has the following methods:

   - `Insert(value)` inserts the parameter to the correct position in the tree. Values in the tree are arranged such that the value stored in the left subtree of a given node is less than that node, and the value stored in the right subtree is more than that node.

   - `ReverseOrder()` displays all the values in descending order

   - `Search(value)` returns whether the parameter is found in the tree

   - `Count()` returns the total number of nodes in the tree

   You may apply recursion to implement the methods.

   For each of the sub-tasks, add a comment statement, at the beginning of the code using the hash symbol '#', to indicate the sub-task the program code belongs to, for example:

   In [1]:

```
# Task 4.1
Program Code
```

In [2]:

```
# Task 4.2
Program Code
```

In [3]:

```
# Task 4.3
Program Code
```

output

Task 4.1

Write program code for the class `TreeNode`. [2]

Task 4.2

Write program code for the class `BSTree`. [14]

Task 4.3

The files `INSERTTOTREE.txt` and `SEARCHINTREE.txt` store data to test your program.

Write program code to:

- Create a new tree and add the data in the file `INSERTTOTREE.txt` to the tree
- Output the current contents in descending order, and the total number of nodes in the tree
- For each data in the file `SEARCHINTREE.txt`, output whether it is in the tree

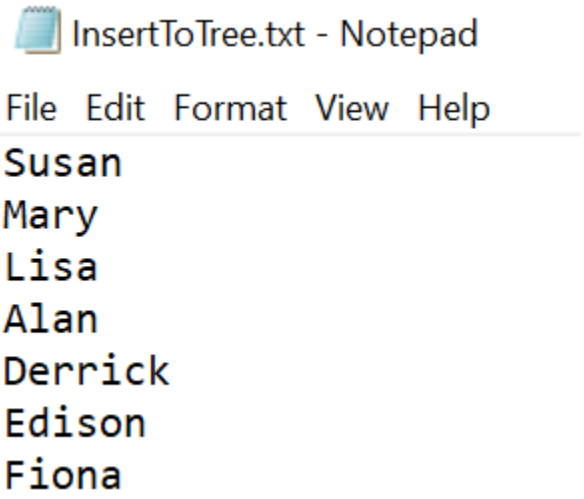
All outputs should have appropriate messages to indicate what they are showing. [9]

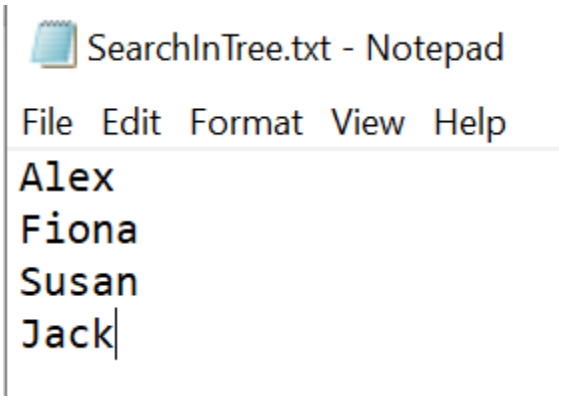
Download your program code and output for Task 5 as

`TASK4_<your name>_<ct>.ipynb`

```python
# Task 1
class TreeNode:
    def __init__(self, value):
        self.data = value
        self.left = None
        self.right = None
```

```python
# Task 3
t = BSTree()

with open('InsertToTree.txt', 'r') as f:
    for line in f:
        value = line.strip()
        t.Insert(value, t.root)

print("Contents in the tree in descending order:")
t.ReverseOrder(t.root)
print()

print("Total number of nodes:", t.Count(t.root))

with open('SearchInTree.txt', 'r') as f:
        for line in f:
            value = line.strip()
            if t.Search(value, t.root):
                print(value, 'is found in the tree.')
            else:
                print(value, 'cannot be found in the tree.')
```





Expected Output

```
Contents in the tree in descending order:
Susan Mary Lisa Fiona Edison Derrick Alan
Total number of nodes: 7
Alex cannot be found in the tree.
Fiona is found in the tree.
Susan is found in the tree.
Jack cannot be found in the tree.
```

5. Quicksort is an efficient sorting algorithm to order values in an array.

Task 5

Write program code to:

- prompt the user to enter the size of an array and check that it is greater than 5 and less than 12
- prompt the user to enter integer numbers into the array
- sort all elements of the array in descending order using a recursive quicksort algorithm
- display the sorted elements in the array. [8]

Test your program with the following test data in an array of size 10:

```
33 11 22 66 55 11 8 66 77 44
```
[2]

Download your program code and output for Task 4 as

`TASK5_<your name>_<ct>.ipynb`

Solution Guide

```
# Quick Sort

def quicksort(list, low, high):
    if (low < high):
        pivot = low
        i = low
        j = high

        while (i < j):
            while (list[i] >= list[pivot] and (i < high)):
                i += 1

            while (list[j] < list[pivot]) and (j >= low):
                j -= 1

            if (i < j):
                temp = list[i]
                list[i] = list[j]
                list[j] = temp

        temp = list[j]
        list[j] = list[pivot]
        list[pivot] = temp
```

```
        quicksort(list, low, j-1)
        quicksort(list, j+1, high)


def main():
    size = int(input("Enter the size of an array: "))
    while not (5 < size < 12):
        print ("Size of array has to be greater than 5 and less than 12. Re-enter!")
        size = int(input("Enter the size of an array: "))

    list =[-1]*size
    print('Enter the elements to be sorted:')
    for i in range(size):
        list[i]= int(input())

    quicksort(list, 0, size-1)

    print('After applying quick sort: ')
    for i in range(size):
        print(list[i], end=' ')

main()
```

```
Enter the size of an array: 10
Enter the elements to be sorted:
33
11
22
66
55
11
8
66
77
44
After applying quick sort:
77 66 66 55 44 33 22 11 11 8
```