­­Computer Science Pre-Released Material

By Tinapat (Game) Limsila

Original Date of Submission: 01 May 2021

For Cambridge CIE Computer Science Pre-Released Assessment

Table of Content

[Task 1 – File Storage 3](#_Toc70781232)

[Task 1.1 - Pseudocode 3](#_Toc70781233)

[Design a record structure using pseudocode to store data about his books. 3](#_Toc70781234)

[Task 1.2 – 1.4 3](#_Toc70781235)

[ Task 1.2 - Write program code to: 3](#_Toc70781236)

[ Task 1.3 - The program needs to store the records in the array into a file. Write a subroutine to store the records in a serial file 3](#_Toc70781237)

[ Task 1.4 - Write a procedure to read the records from a serial file and output them with an appropriate message. 3](#_Toc70781238)

[Task 1.5- 1.6 – Python 5](#_Toc70781239)

[ Task 1.5 - Each book has a unique code. The unique code allows the book’s details to be stored in a random file using a hashing algorithm. Develop an appropriate hashing algorithm 5](#_Toc70781240)

[ Task 1.6 - Manually calculate the file location for several books using your hashing algorithm. 5](#_Toc70781241)

[Task 1.7 – Pseudocode 6](#_Toc70781242)

[Write a pseudocode algorithm to perform the hash calculation 6](#_Toc70781243)

[Task 1.8 – Pseudocode 6](#_Toc70781244)

[Write a pseudocode algorithm to store a record in its hashed location in the random file 6](#_Toc70781245)

[Task 1.9 – Pseudocode 6](#_Toc70781246)

[Write a pseudocode algorithm to read a record from its hashed location in the random file 6](#_Toc70781247)

[Task 2 – Binary Tree 7](#_Toc70781248)

[Task 2.1 7](#_Toc70781249)

[Identify the root node and leaf nodes in the binary tree. 7](#_Toc70781250)

[Task 2.2 7](#_Toc70781251)

[Add the following data to the binary tree: **pink, yellow, blue, purple, fuchsia, turquoise** 7](#_Toc70781252)

[Task 2.3 – Task 2.6 8](#_Toc70781253)

[ Task 2.3 - Write program code to store the binary tree as a 1D array of records. 8](#_Toc70781254)

[ Task 2.4 - Write program code to add a new data item to the binary tree 8](#_Toc70781255)

[ Task 2.5 - Write program code to find the position of a specific colour in the binary tree 8](#_Toc70781256)

[ Task 2.6 - Write program code to output the contents of the binary tree in alphabetical order 8](#_Toc70781257)

[Task 3 – Object-oriented programming 11](#_Toc70781258)

[Task 3.1 – 3.5 11](#_Toc70781259)

[ Task 3.1 – The program needs a class for the tools. 11](#_Toc70781260)

[The information stored about each tool must include: 11](#_Toc70781261)

[ Task 3.2 - Write program code to create a get and set method for each of the tool attributes 11](#_Toc70781262)

[ Task 3.3 - The program needs a class for the shelves in the store. 11](#_Toc70781263)

[Each shelf has the following information. position of the shelf on the wall (between 0 and 4) 11](#_Toc70781264)

[array of objects of type tool (maximum of 10 items per shelf). 11](#_Toc70781265)

[Write program code for the class shelves. The constructor should set the position of the shelf but should not set any tools. Write program code for the attributes and the constructor method. 11](#_Toc70781266)

[ Task 3.4 - Write program code for a set method to add a new tool to the shelf in the next available position. 11](#_Toc70781267)

[ Task 3.5 - Write program code to define a procedure that takes a shelf object as a parameter and outputs the name and cost of each tool on that shelf 11](#_Toc70781268)

# Task 1 – File Storage

## Task 1.1 - Pseudocode

### Design a record structure using pseudocode to store data about his books.

He wants to store the following:

* unique code for each book (between 100 and 999)
* title of the book
* main author
* year of publication.

DECLARE BookCode : INTEGER 100 to 999

DECLARE BookTitle : STRING

DECLARE BookAuthor : STRING

DECLARE BookPublication : INTEGER

## Task 1.2 – 1.4 -- Python

### Task 1.2 - Write program code to:

* + create your record structure
  + create an array to store the records about the books
  + input data for 10 books from the user
  + store each book as a separate record in an array
  + output the data in each record with an appropriate message.

### Task 1.3 - The program needs to store the records in the array into a file. Write a subroutine to store the records in a serial file

### Task 1.4 - Write a procedure to read the records from a serial file and output them with an appropriate message.

Notes: change the operator from “and” to “or” both conditions cannot be true ever

# Task 1.2

def enter\_code():

    global book\_code

    book\_code = input("Book unique code: ")

    if (int(book\_code) < 100) or (int(book\_code) > 999):

        print("invalid")

        enter\_code()

def enter\_title():

    global book\_title

    book\_title = input("Title of the book: ")

def enter\_author():

    global book\_author

    book\_author = input("Main author: ")

def enter\_year():

    global book\_year

    book\_year = input("Year of Publication: ")

    if (int(book\_code) <= 0 and int(book\_code) >= 9999):

        print("invalid")

        enter\_year()

# Task 1.3

with open("BookCollection.txt", "a") as file:

    array = []

    for i in range(10): # its 10 in the exams

        print("This is for book #" + str(i+1))

        print("format example: 101\_Harry Potter\_JK\_1997")

        enter\_code()

        enter\_title()

        enter\_author()

        enter\_year()

        book = book\_code + "\_" + book\_title + "\_" + book\_author + "\_" + book\_year + "\n"

        array.append(book)

        print(book + "is stored")

    file.writelines(array)

# Task 1.4

with open("BookCollection.txt", "r") as file:

    booklist = file.readlines()

    i = 1

    print(booklist)

    for book in booklist:

        current\_book = book.split("\_")

        print("This is book number " + str(i))

        print("Book code: " + current\_book[0])

        print("Book name: " + current\_book[1])

        print("Book author: " + current\_book[2])

        print("Book year of publication: " + current\_book[3])

        i = i + 1

## Task 1.5- 1.6 – Python

### Task 1.5 - Each book has a unique code. The unique code allows the book’s details to be stored in a random file using a hashing algorithm. Develop an appropriate hashing algorithm

### Task 1.6 - Manually calculate the file location for several books using your hashing algorithm.

MOD\_NUMBER = 20 # the algorithm

# Task 1.6

def write\_hashing\_algorithms(book\_string):

    book\_id = book\_string[0:3]

    hashed\_id = int(book\_id) % MOD\_NUMBER # Task 1.5 Hashing Algorithm

    while hash\_table[hashed\_id] != "\n":

        hashed\_id = hashed\_id + 1

    hash\_table[hashed\_id] = book\_string # Task 1.6/1.8

def read\_hashing\_algorithms(book\_id):

    found = False

    out\_of\_range = False

    hashed\_id = int(book\_id) % MOD\_NUMBER

    while not(found) and not(out\_of\_range):

        if hashed\_id < (len(hashed\_list\_from\_file)):

            if (hashed\_list\_from\_file[hashed\_id])[0:3] != book\_id:

                hashed\_id = hashed\_id + 1

            else:

                found = True

        else:

            out\_of\_range = True

    if found:

        print(hashed\_list\_from\_file[hashed\_id])

    else:

        print("Not in the hashed list")

hash\_table = []

for i in range(MOD\_NUMBER \* 2):

    hash\_table.append("\n")

with open("BookCollection.txt", "r") as file:

    book\_list = file.readlines()

    for book in book\_list:

        write\_hashing\_algorithms(book)

with open("HashedCollection.txt", "w") as hashed\_file:

    hashed\_file.writelines(hash\_table)

with open("HashedCollection.txt", "r") as hashed\_file:

    hashed\_list\_from\_file = hashed\_file.readlines()

    print(hashed\_list\_from\_file)

search1 = input("Find: ")

read\_hashing\_algorithms(search1)

search1 = input("Find: ")

read\_hashing\_algorithms(search1)

## Task 1.7 – Pseudocode

### Write a pseudocode algorithm to perform the hash calculation

FUNCTION HashCalculator(BookCode : INTEGER) RETURN HashKey

ModCoefficient 🡨 20

HashKey 🡨 MOD(BookCode, ModCoefficient)

ENDFUNCTION

## Task 1.8 – Pseudocode

### Write a pseudocode algorithm to store a record in its hashed location in the random file

PROCEDURE StoreInHashedLocation(HashKey : INTEGER, HashTable : ARRAY, BookCode : BookTitle, BookTitle : STRING, BookAuthor : STRING, BookPublication : INTEGER)

OPENFILE RandomFile.txt FOR READ

READFILE RandomFile.txt, HashTable

OPENFILE RandomFile.txt FOR WRITE

StoredFlag 🡨 FALSE

Book 🡨 “”

Book.append(BookCode, BookTitle, BookAuthor, BookPublication)

WHILE StoredFlag = FALSE

IF (HashTable[HashKey] = “”) or (HashTable[HashKey] is empty)

THEN

HashTable[HashKey] 🡨 Book

WRTIEFILE RandomFile.txt, HashTable

StoredFlag 🡨 TRUE

ELSE

HashKey 🡨 HashKey + 1

ENDIF

ENDWHILE

CLOSEFILE RandomFile.txt

ENDPROCEDURE

## ­­­Task 1.9 – Pseudocode

### Write a pseudocode algorithm to read a record from its hashed location in the random file

FUNCTION ReadInHashedLocation(HashTable : ARRAY, HashKey : INTEGER, BookCode : INTEGER) RETURN BookString

OPENFILE RandomFile.txt FOR READ

READFILE RandomFile.txt, HashTable

FoundFlag 🡨 FALSE

WHILE FoundFlag = FALSE

IF MID(HashTable[HashKey], 1, 3) = BookCode

THEN

BookString 🡨 HashTable[Hashkey]

FoundFlag 🡨 TRUE

ELSE

HashKey 🡨 HashKey + 1

ENDIF

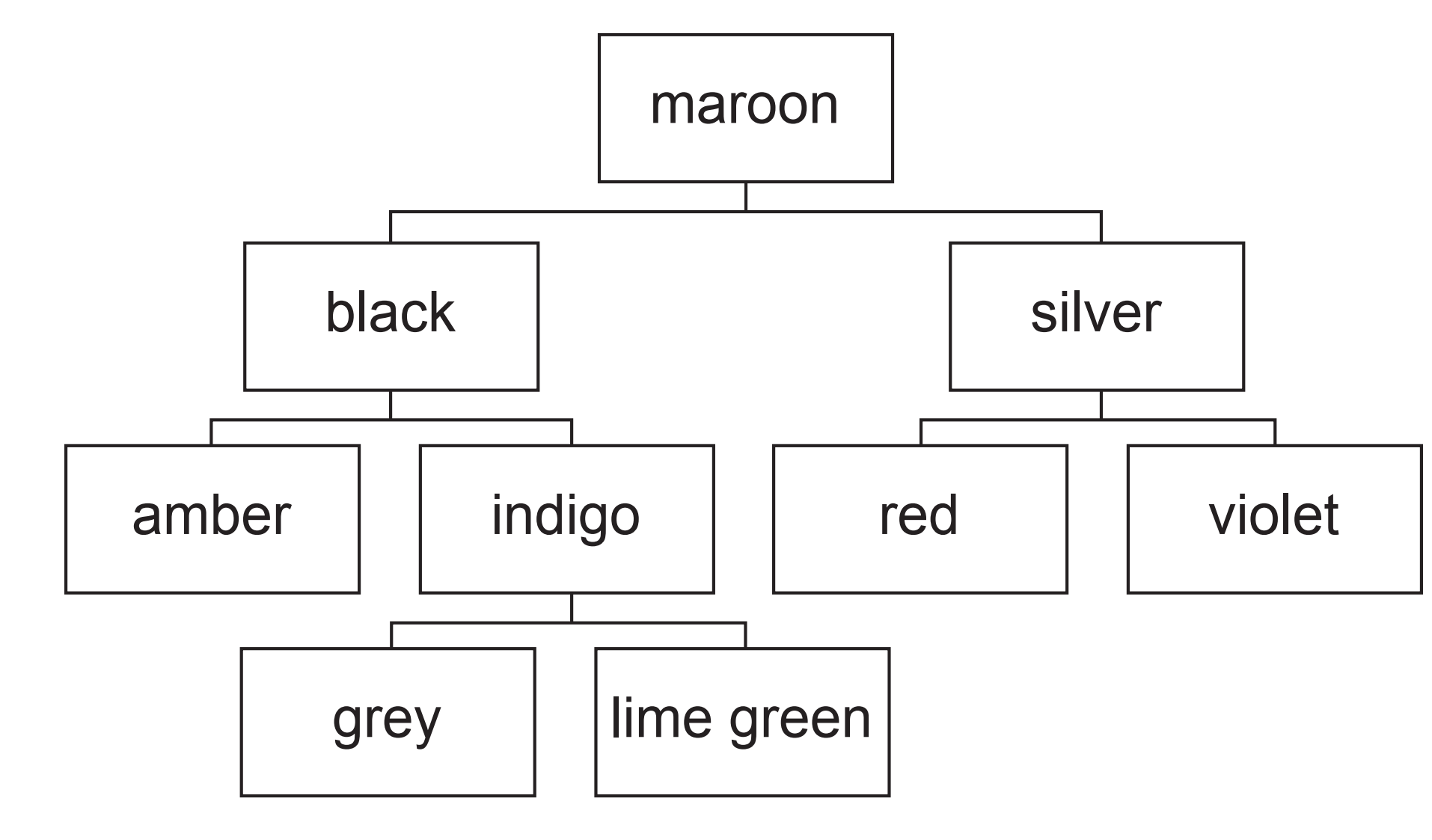
ENDWHILE

ENDFUNCTION

# Task 2 – Binary Tree

## Task 2.1

### Identify the root node and leaf nodes in the binary tree.



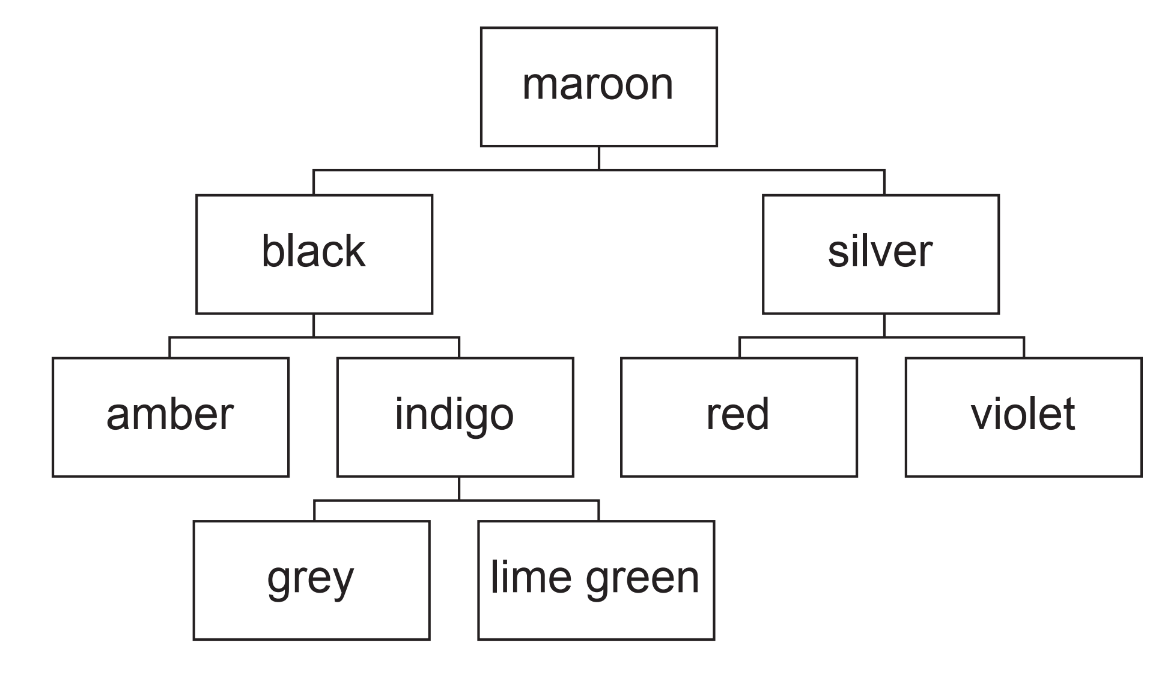
Root node is maroon

Leaf node are black, amber, indigo, grey, lime green, sliver, red, violet

## Task 2.2

### Add the following data to the binary tree: **pink, yellow, blue, purple, fuchsia, turquoise**

purple



pink

yellow

blue

fuchsia

turquoise

## Task 2.3 – Task 2.6 - Python

### Task 2.3 - Write program code to store the binary tree as a 1D array of records.

### Task 2.4 - Write program code to add a new data item to the binary tree

### Task 2.5 - Write program code to find the position of a specific colour in the binary tree

### Task 2.6 - Write program code to output the contents of the binary tree in alphabetical order

duplicate\_flag = False

duplicate\_list = []

class Node():

    def \_\_init\_\_(self, colour, position = None, left\_pointer = None, right\_pointer = None):

        self.position = position

        self.colour = colour

        self.left\_pointer = left\_pointer

        self.right\_pointer = right\_pointer

Maroon = Node('maroon', 0, 1, 2)

Black = Node('black', 1, 3, 4)

Silver = Node('silver', 2, 5, 6)

Amber = Node('amber', 3)

Indigo = Node('indigo', 4, 7, 8)

Red = Node('red', 5)

Violet = Node('violet', 6)

Grey = Node('grey', 7 )

Lime\_Green = Node('lime green', 8)

#Task 2.3

binarytree = [Maroon, Black, Silver, Amber, Indigo, Red, Violet, Grey, Lime\_Green]

def print\_node(Node):

    print('''

Node Position {}

Node Colour: {}

Node Left Pointer: {}

Node Right Pointer: {}'''.format(Node.position, Node.colour, Node.left\_pointer, Node.right\_pointer))

def asciify(Node, char\_position = 0):

    return ord(Node.colour[char\_position])

# Task 2.4

def adding\_function(Append\_Node, Parent\_Node = binarytree[0], char\_position = 0):

    global duplicate\_flag

    if (asciify(Append\_Node, char\_position)) > (asciify(Parent\_Node, char\_position)):

        if Parent\_Node.right\_pointer == None:

            Append\_Node.position = len(binarytree)

            Parent\_Node.right\_pointer = len(binarytree)

            binarytree.append(Append\_Node)

        else:

            adding\_function(Append\_Node, binarytree[Parent\_Node.right\_pointer])

    elif (asciify(Append\_Node, char\_position)) < (asciify(Parent\_Node, char\_position)):

        if Parent\_Node.left\_pointer == None:

            Append\_Node.position = len(binarytree)

            Parent\_Node.left\_pointer = len(binarytree)

            binarytree.append(Append\_Node)

        else:

            adding\_function(Append\_Node, binarytree[Parent\_Node.left\_pointer])

    else:

        if Append\_Node.colour == Parent\_Node.colour:

            duplicate\_flag = True

            duplicate\_list.append(Append\_Node.colour)

        else:

            adding\_function(Append\_Node, Parent\_Node, char\_position +1)

# Task 2.5

def find\_function(Wanted\_Node, Parent\_Node = binarytree[0], char\_position = 0):

    if (asciify(Wanted\_Node, char\_position)) > (asciify(Parent\_Node, char\_position)):

        find\_function(Wanted\_Node, binarytree[Parent\_Node.right\_pointer], char\_position)

    elif (asciify(Wanted\_Node, char\_position)) < (asciify(Parent\_Node, char\_position)):

        find\_function(Wanted\_Node, binarytree[Parent\_Node.left\_pointer], char\_position)

    else: # when they equal

        if Wanted\_Node.colour == Parent\_Node.colour:

            print("Found {} at position {} of the list(Starting at 0)".format(Parent\_Node.colour, Parent\_Node.position))

        else:

            find\_function(Wanted\_Node, Parent\_Node, char\_position +1)

def print\_in\_alphabetical\_order(Parent\_Node = binarytree[0]):

    if ((Parent\_Node.left\_pointer == None) and (Parent\_Node.right\_pointer == None)):

        print(Parent\_Node.colour)

    if Parent\_Node.left\_pointer != None:

        print\_in\_alphabetical\_order(binarytree[Parent\_Node.left\_pointer])

    if Parent\_Node.right\_pointer != None:

        print(Parent\_Node.colour)

        print\_in\_alphabetical\_order(binarytree[Parent\_Node.right\_pointer])

def \_\_main\_\_():

    Pink = Node("pink")

    adding\_function(Pink)

    Yellow = Node("yellow")

    adding\_function(Yellow)

    Blue = Node("blue")

    adding\_function(Blue)

    Purple = Node("purple")

    adding\_function(Purple)

    Fuchsia = Node("fuchsia")

    adding\_function(Fuchsia)

    Turquoise = Node("turquoise")

    adding\_function(Turquoise)

Turquoise = Node("turquoise")

    adding\_function(Turquoise)

    for node in binarytree:

        print\_node(node)

    if duplicate\_flag:

        print()

        print("""all of these colors are already on the list

Duplicated not added""")

        print(duplicate\_list)

find\_function(Yellow)

print\_in\_alphabetical\_order()

\_\_main\_\_()

# Task 3 – Object-oriented programming

## Task 3.1 – 3.5 - Python

### Task 3.1 – The program needs a class for the tools.

### The information stored about each tool must include:

* + Name
  + Cost
  + Image file name (e.g. ‘spade.jpg’)

### Task 3.2 - Write program code to create a get and set method for each of the tool attributes

### Task 3.3 - The program needs a class for the shelves in the store.

### Each shelf has the following information. position of the shelf on the wall (between 0 and 4)

### array of objects of type tool (maximum of 10 items per shelf).

### Write program code for the class shelves. The constructor should set the position of the shelf but should not set any tools. Write program code for the attributes and the constructor method.

### Task 3.4 - Write program code for a set method to add a new tool to the shelf in the next available position.

### Task 3.5 - Write program code to define a procedure that takes a shelf object as a parameter and outputs the name and cost of each tool on that shelf

import random

class Tools: # Task 3.2

    def \_\_init\_\_(self, name, cost, image\_file\_name):

        self.name = name # string

        self.cost = cost # reals

        self.image\_file\_name = image\_file\_name # string

class Shelves:

    def \_\_init\_\_(self, array):

        self.position = random.randint(0,4) # only 0-4

        self.array = array

# Task 3.4

def add\_tool\_to\_shelf(tool, shelf):

    if len(shelf.array) < 10:

        shelf.array.append(tool)

    else:

        print("shelf is full, max 10 tools")

def read\_tool\_from\_shelf(shelf):

    for i in range(len(shelf.array)):

        current\_tool = Shelf1.array(i)

        print(i)

        print(current\_tool)

# Task 3.5

def list\_item\_in\_self(shelf):

    i = 1

    for tool in shelf.array:

        print("Tool number {0}".format(i))

        print("Tool: {}".format(tool.name))

        print("Cost: ${}".format(tool.cost))

        print()

        i += 1

Object1 = Tools("Hammer", "100", "hammer.jpg")

Object2 = Tools("Premium Hammer", "200", "hammer.jpg")

BASE\_ARRAY = []

Shelf1 = Shelves(BASE\_ARRAY)

add\_tool\_to\_shelf(Object1, Shelf1)

add\_tool\_to\_shelf(Object2, Shelf1)

add\_tool\_to\_shelf(Object1, Shelf1)

add\_tool\_to\_shelf(Object2, Shelf1)

add\_tool\_to\_shelf(Object1, Shelf1)

add\_tool\_to\_shelf(Object2, Shelf1)

add\_tool\_to\_shelf(Object1, Shelf1)

add\_tool\_to\_shelf(Object2, Shelf1)

add\_tool\_to\_shelf(Object1, Shelf1)

add\_tool\_to\_shelf(Object2, Shelf1) # 10th

add\_tool\_to\_shelf(Object1, Shelf1) # 11th

input("Tool added, enter to list all item")

list\_item\_in\_self(Shelf1)