COMPS258 (Autumn 2021) Practice Assignment Suggested Answers

Authored by Andrew Kwok Fai Lui
Revised Version 10/5/2022

Part I

Note that programming questions, unless otherwise specified, may be answered with various implementations. Python is a programming language that offers a number of ways to do the same operation. You are welcome to use your own way, based on the built-in modules and the third-party modules covered in the Study Units. You should NOT use other third-party modules, unless otherwise specified.

Question 1

(a) Any two from mouse, keyboard, keypad, touchpad, scanner, etc.

(b)

```
x = int(input('Enter an integer: '))
x += 1
print(x)
```

The output message can change

(c)

The programming language uses the format of an identifier or token to differentiate between constants, types, and variables. An identifier consisting of only numerals is recognized as an integer.

(d) 8

(e) 8 20

(f) True

(g) 2.5 2

(h) 0 10

(i) 6 6

(j) 3 3

(k) 3 9

(l) True

(m) [-1, 1]

(n) (i) 2 (ii) 6 x 5 = 30 (iii) clist is holding a string that is immutable and dlist is holding a list that is mutable

(o) (5, -1)

(p)

```
sum = 0
```

```
i = a
while i < b:
    sum += i
    i += 1
print(sum)
```

This is only one way to rewrite a for loop into a while loop. Keep in mind that write in a way the marker will find it easier to read, and it is to your advantage.

Question 2

(a)

The result of an integer division is an integer, which is the quotient (result of the division) rounded down to the nearest smaller integer.

```
print(7 // 3)  # gives 2 instead of 2.333
print(-7 // 3) # gives -3 instead of -2.333
```

The result of an regular division is a floating point number.

```
print(7 / 3)  # result is 2.3333
print(-7 / 3) # result is -2.3333
```

(b)

```
canPurchased = input ('Input number of cans purchased :')
canPurchased = int (canPurchased)

noOfFreeCan = canPurchased // 5

if canPurchased >= 20 :
    noOfFreeCan += 2

print (noOfFreeCan, "free cans to be offered.")
```

(c)

```
month = input ('Input the month: ')
month = int (month)

if month == 2 :
    print ("Number of day in month", month, "is 28 days")
elif month == 4 or month == 6 or month == 9 or month == 11 :
    print ("Number of day in month", month, "is 30 days")
```

```
elif month == 1 or month == 3 or month == 5 or month == 7 or month == 8 or month
 == 10 or month == 12 :
    print ("Number of day in month", month, "is 31 days")
else :
    print ('Error: The input month', month, 'is an incorrect value.')
```

(d)

```
while True:
    month = input ('Input the month: ')
    month = int (month)
    if month >= 1 and month <= 12:
        break
    print ('Error: The input month', month, 'is an incorrect value.')

if month == 2 :
    print ("Number of day in month", month, "is 28 days")
elif month == 4 or month == 6 or month == 9 or month == 11 :
    print ("Number of day in month", month, "is 30 days")
elif month == 1 or month == 3 or month == 5 or month == 7 or month == 8 or month
 == 10 or month == 12 :
    print ("Number of day in month", month, "is 31 days")
```

(e)

There are 2 problems in the given program.

First, the variable answer references to a string input by the user. The comparison of a string with a floating point value 2 / 3 is always different.

Second, even though if we change the input statement to

answer = float(input ("Enter the result of 2 divided by 3 :"))

to convert the string input by a user to a floating point value, the result output by the program is also “Wrong!”

Since 2 / 3 gives a floating-point value. Floating-point values cannot be represented exactly in computer memory.

So the comparison of 2 floating-point values by means of == operator are always different. A reasonable way to compare 2 floating point values is to use comparison <= and >= to test within an acceptable range such as 0.66 to 0.67.

(f)

```
size = int(input('Enter size: '))
for row in range(size):
    for col in range(size):
        if row == 0 or col == 0 or row == size - 1 or col == size - 1:
            print('*', end='')
        else:
            print(' ', end='')
    print()
```

(g)

```
def convert (meters, centimeters) :
    meters = int (meters)
    centimeters = float (centimeters)

    centimeters = meters * 100 + centimeters

    inches = centimeters / 2.54
    feet = inches // 12
    inches = inches - feet * 12

    yards = feet // 3
    yards = int (yards)
    feet = feet - yards * 3
    feet = int (feet)

    return yards, feet, inches
```

(h)

```
miles = float(input('Enter a distance in miles: '))
kilo = miles * 1.61
print('Same distance in kilometers =', kilo)
```

The conversion from string to miles should use float() because normally a distance can be fractional

(i)

```
def isNegative(numlist, startindex = 0, endindex = None):
    if endindex == None:
        endindex = len(numlist)
    endindex = min(endindex, len(numlist))
    if endindex < startindex:
        return None
    for i in range(startindex, endindex):
        if numlist[i] >= 0:
            return False
    return True
```

Question 3

(a)

Characteristics for good modules such as the following:

Manageable size: the size of the module is small enough for conceptual understanding and ease of implementation

Perform a single task: the purpose of the module is focused on one task.

Independent of other modules: reduced the logical relation between modules as much as possible so that changing one module will not affect another module.

(b)

(i)

The different data sizes tested the scalability of the algorithms.

(ii)

Sorting algorithms must involve comparisons between data, and this is normally a good estimation of how much work the algorithm is doing

(iii)

SortingDoris has a better scalability than sortingChris because the rate of growing of the number of comparisons is slower with respect to the data sizes

(iv)

	Answers
`sortingChris`	**`O(N^2)`**
`sortingDoris`	**`O(N)`**

(c)

	Your answers
Is traversing a binary tree same as searching a binary tree? Explain your answer	No. Traversing is to assess each record exactly once so that certain items in the record may be processed. Searching is to find the location of the record with a given key value or to find of all records which satisfy one or more conditions. In searching, it is not necessary to evaluate every record. The search may terminate immediately once the search item is found.
List the nodes visited in pre-order traversal	30, 23, 5, 1, 8, 32, 31, 33, 35
List the nodes visited in in-order traversal	1, 5, 8, 23, 30, 31, 32, 33, 35
List the nodes visited in post-order traversal	1, 8, 5, 23, 31, 35, 33, 32, 30

Write down the nodes in the right branch of 32	33, 35
Write down the height of the tree	3
Is it a binary search tree? Justify your answer.	Yes. The golden rule that all nodes in the left branch is smaller and all nodes in the right branch is larger is true for every node.

(d)

[3]

Pass 1 **Begin = 0, End = 12, Mid = 6 Not Found**

11	22	30	33	40	44	**<u>55</u>**						

Pass 2 **Begin = 0, End = 5, Mid = 2 Not Found**

		<u>30</u>	33	40	44							

Pass 3 **Begin = 3, End = 5, Mid = 4 Not Found**

				<u>40</u>								

(e)

(i)

```
          Cat
        /     \
     Ant       Dog
                  \
                   Fly
                   /  \
               Fish    Pig
                       /
                    Mouse
```

(ii)

Perfect binary tree minimizes the height of a binary tree with the same number of nodes.

Key search involves steps that are equal to the height of the tree. Perfect binary tree therefore minimizes the steps involved in key searching

Non-key searching needs to traverse the whole tee anyway. The number of steps is equal to the number of nodes.

(iii)

It is technically challenging to ensure the built binary search tree is always a perfect tree. The shape of the binary tree depends on the data sequence itself, which is unpredictable. The method is to adjust the shape of the binary tree through manipulation of nodes by de-linking and linking nodes, and to keep the binary search tree balanced and even perfect. The process is to review every node and see whether the number of nodes on the left and right subtree are similar.

One method is described.

The trick is to insert the “middle” data first.

First sort the data: Ant Cat Dog Fish Fly Mouse Pig

Similar to what a binary tree to do, find the middle data, insert the data, and split the data into left and right. For each left and right, repeat the above. So one possible insertion order is Fish, Cat, Ant, Dog, Mouse, Fly, Pig

```
           Fish

          /    \

        Cat      Mouse

      /  \       / \
   Ant   Dog  Fly  Pig
```

(iv)

Hashtable can be used to store the animal information. Hashtable stores data as (key, data) pairs in a table. The animal names are the key and the animal information is the data part. So the structure of the data is suitable.

To add data, use the put(key, data) function. To retrieve data use the key and the function get(key).

(v)

Yes, Hashtable can perform better than binary search tree, with addition and retrieval can be constant time O(1), while the best case of binary search tree is O(lg N).

However, on the other hand, hashtable worst case is O(N) if there are lots of collision happening due to table size too small. So to ensure good performance, the table size must be sufficiently large.

Accept both agree and disagree as long as your justification is relevant and correct

(f)

```
class ListNode:
    def __init__(self, data):
        self.data = data
        self.nextobj = None

front = ListNode(5)
newnode = ListNode(2)
newnode.nextobj = front

front = newnode

temp = front
while temp is not None:
    print(temp.data)
    temp = temp.nextobj
```

Question 4

(a)

```
def add1Odd(numlist);
  count = 0
  for i in range(len(numlist)):
     if numlist[i] % 2 == 1:
        numlist[i] += 1
        count += 1
  return count
```

(b)

Function calls	Return value or error occurred
add1Odd([1, 2, 3, 4])	2
add1Odd(['A', 'B'])	TypeError
add1Odd(1)	TypeError
add1Odd()	TypeError

(c)

```
def isNegative(numlist, startindex = 0):
    if startindex >= len(numlist):
        return True
    if numlist[startindex] >= 0:
        return False
    return isNegative(numlist, startindex + 1)
```

(d)

(i)

theType, x, count, numlist, top

(ii)

Variables	Lines
`count` used on line 9	**5**
`x` used on line 7	**4**
`numlist` used on line 15	**13**
`calTop` used on line 15	**4**

(iii)

A local variable is created when the local scope (such as a function) is executed

(iv)

One important benefit of local variables is that their names and their use have a local scope, which is limited to the function definition itself. Each function definition has the freedom to choose the names for their variables without interfering with other function definitions. Each function definition can be developed independently, without the need to coordinate the naming of the variables, if the function definitions are developed by different developers. Without local variables, the function definitions' variable name has to be coordinated or additional qualified with scoped names.

(e)

```
def evaluateSequence(n):
    if n <= 0:
        return 0
    if n == 1:
        return 1
    return (1/(2 * n - 1)**2) + evaluateSequence(n-1)
```

(f)

```
def removeConSpaces(text):
    newstr = ""
    lastch = 'a' # any non-space character is ok
    for ch in text:
        if ch != ' ' or lastch != ' ':
            newstr = newstr + ch
        lastch = ch
    return newstr
```

(g)

Slicing examples	Remarks
`str[1:4]`	Extract the 3 elements from index 1, 2, and 3 (4 not included)
`str[:-2]`	Extract all elements from index 0 to the second last index (-2)
`str[1::2]`	Extract one every 2 elements from index 1 to the end
`str[::-1]`	Extract every element in reverse (-1) order

The slicing operation creates a new sequence (list, string, or tuple) according to 3 parameters, start index, end index, and step size. It has the format [start:end:step]. The start and end index are positive value if the index position starts from 0 as the first element, or negative value if the index position starts from -1 as the last element.

Question 5

(a)

```
try :
    infile = open ('district.txt', 'r')
except :
    print ('File district.txt cannot be found. Execution is
terminated!')
else :
    count = totalEmployer = totalRetiredWanChai = maxStudent = 0
```

```
infile.readline()
maxStudentDistrict = None
for line in infile :
        line = line.rstrip('\n')
        dataList = line.split(',')

        count += 1
        totalEmployer += int(dataList[2])

        if dataList[0][0 : 8] == 'Wan Chai':
            totalRetiredWanChai += int(dataList[4])

        if maxStudent < int(dataList[3]) :
            maxStudent = int(dataList[3])
            maxStudentDistrict = dataList[0]

print ('The total number of districts in the raw data set is', count)
print ('The total number of employers in all districts is',
totalEmployer)
print ('The total number of retired people in the “Wan Chai” large
district is', totalRetiredWanChai)
print ('The name of the district with the highest number of students
is :', maxStudentDistrict)

infile.close()
```

(b)

(i)

```
import random

stack = Stack()
for i in range(10):
    stack.push(random.randrange(0, 10))

for i in range(10):
    print(stack.pop(), end=' ')
```

(ii)

The stack is first in last out.

(iii)

The assert statements are used to ensure a condition is True or if otherwise an AssertionError will be raised and the program will terminate.

(iv)

```
    def swap(self):
        if self.size() < 2:
            return
        temp = self.items[-1]
        self.items[-1] = self.items[-2]
        self.items[-2] = temp
```

(v)

It should be constant time, O(1), because none of the operations is related to the number of data items in the list.

(c)

(i)

```
msft_df['2018-10-11','Close']
```

(ii)

```
msft_df.nlargest(5, 'Close')
```

(iii)

```
msft_df[msft_df.Close > 240]
```

(iv)

```
import matplotlib.pyplot as plt

plt.plot(msft_df['Close'])

plt.show()
```

(e)

(i) The following shows the essential operations functions in the two ADT.

```
 The Book ADT
class Book:
      def borrowBook(self, person)
       ...
      def returnBook(self, person)
       ...

 The Library ADT
 class Library:
      def addBook(self, newbook)
       ...
       def deleteBook(self, book)
       ...
       def listBorrowedBooks(self)
       ...
```

```
def listBorrowedByFriend(self, person)
...
```

There are many possible acceptable solutions to this question, but the above shows one set of essential functions. The types of parameters need not be specified. The self parameter may be ignored for this question

(ii) The ADT will be used in applications that require a lot of searching. A binary search tree for holding the Book objects within the Library class would be efficient for searching.

Again there is no standard answer. The important part is to justify or argue that your answer is reasonable and support the listed features.