CSE 101 Introduction to Programming, Section A END-SEM Exam, Nov 30, 2023

Part 1: There are 5 questions of 6 marks each

1. We are given a Python function to identify all words consisting of English letters in given text. The program returns: `word_list = ['now', 'you', 'are', 'required', 'to', 'write', 'a', 'python', 'program']` if `text = 'Now,, you are required to write a Python program'`

   I have introduced 3 bugs in the function code. **They are all logical errors**. Can you identify them?

```
def words(text):
    j = 0
    while j < len(text):
        while j < len(text) and text[j] not in 'abcdefghijklmnopqrstuvwxyz':
            j = j+1
        word = ''
        while j < len(text) and text[j] in 'abcdefghijklmnopqrstuvwxyz':
            word = word + text[j]
        word_list.append(word)
    return(word_list)
text = 'Now,, you are required to write a Python program'
print(text)
print(words(text))
```

The correct answer is:

```
def words(text):
    word_list = []
    j = 0
    while j < len(text):
        while j < len(text) and text[j] not in 'abcdefghijklmnopqrstuvwxyz':
            j = j+1
        word = ''
        while j < len(text) and text[j] in 'abcdefghijklmnopqrstuvwxyz':
            word = word + text[j]
            j = j +1
        word_list.append(word)
    return(word_list)
text = 'Now,, you are required to write a Python program'
text = text.lower()
print(text)
print(words(text))
```

2 marks

2 marks

2 marks

6 marks

2. This question is about aliasing of Lists. What is the output of the program given below once it executes completely?

```
Techs = ['MIT', 'Caltech']
Ivys = ['Harvard', 'Yale', 'Brown']
Univs = [Techs, Ivys]
Univs1 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]
print('Is Univs "==" Univs1 ', Univs == Univs1)
print('Is Univs "is" Univs1 ', Univs is Univs1)
Univs1 = Univs
print('Is Univs "is" Univs1 ', Univs is Univs1, 'post aliasing')
print('Univs before adding RPI ', Univs)
Techs.append('RPI')
print('Univs after adding RPI', Univs)
print('Univs1 after adding RPI to Univs ', Univs1)
print('Is Univs "is" Univs1 ', Univs is Univs1, 'post adding RPI')
```

1 mark each

The correct output is:

Total 6 marks

```
Is Univs "==" Univs1  True
Is Univs "is" Univs1  False
Is Univs "is" Univs1  True post aliasing
Univs before adding RPI [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]
Univs after adding RPI [['MIT', 'Caltech', 'RPI'], ['Harvard', 'Yale', 'Brown']]
Univs1 after adding RPI to Univs [['MIT', 'Caltech', 'RPI'], ['Harvard', 'Yale', 'Brown']]
Is Univs "is" Univs1  True post adding RPI
```

3. Recall the Tower-of-Hanoi problem and the recursive function to solve it for one or more number of disks. For you benefit the function is given below. (You do not need to worry about it, if you know how the problem is solved.)

```
def ToH(n, src, dest, aux):
    if n==1:
        print("Move disk 1 from peg ", src, "to peg ", dest)
        return
    ToH(n-1, src, aux, dest)
    print("Move disk", n, "from peg ",src, "to peg ", dest)
    ToH(n-1, aux, dest, src)
```

It should be clear, that if n = 2 (for example), the number of times a disk is moved is 3. Write below the total number of times a disk is moved from a peg to another peg.
Your answer here:

The correct answers are:

total 6 marks

Value of n:	Number of disks moved
n=1:	1
n=2:	3
n=3:	7________
n=4:	15_______
n=5:	31_______
n=6:	63_______
etc. etc.	

1 mark each

2 marks

O(n) O(n^2) O(n^3) [✓ **O(2^n)**] O(n!)

4. What is the output once the following program executes completely?

```
class Point:
    def __init__(self, a, b):
        self.x = a
        self.y = b
def f(a):
    a.x = float(a.x + 1)
    print(a.x)
    return a
def h(a):
    a.y = int(a.y + 3.0)
    print(a.y)
    a = f(a)
    return a
a = Point(1, 2)
final = h(a)
print(final.x)
print(final.y)
```

The correct answer is:

```
5
2.0
2.0
5
```

$1\frac{1}{2}$ marks each

Total: 6 marks

5. I have written a function **gcd (a, b)** to compute GCD(a, b), where a>0, b>0. Here is that function/program:

```
def gcd(a, b):
    if b > a:
        return(gcd(b, a))
    if a%b == 0:
        return(b)
    return(gcd(b, a%b))
print(gcd(15, 5))
```

I have tested the function with **a=15, b=5.** But, I think I need to test this function/program with many more test cases to be relatively confident that it works correctly. Suggest an additional SIX useful but different tests that I should carry out.

The correct answer is **anything similar to the tests given below**:

a: 15	b: 5	or a: 5	b: 15
a: 24	b: 15	or a: 15	b: 24
a: 24	b: 1	or a: 1	b: 24
a: 22	b: 22		

1 mark each

total 6 marks

Part 2: There are 4 questions of 12 marks each

6. This is about defining a module and using it to solve problems related to prime numbers.

   A. Create or define a Python module, ‘`primes’`, by re-writing the following code

```
# module for working with prime numbers - named ‘primes’
"""Collection of functions related to primes, including
    Is_Prime(K)to determine whether given number K is a prime number,
    Max_Prime(N) to determine the largest prime <= N,
    Relative_Prime(P, Q)to determine whether P & Q are relatively prime."""
#
# Definition of Is_Prime(K), Max_Prime(N), Relative_Prime(P,Q) goes here
# and any other required function or variables
```

   B. Now write a main program that will use the module, ‘`primes’ ,` to

   (a) Check if number **37** is a prime number,

   (b) Compute the largest prime number **<= 5**5, and

   (c) Determine whether **24** & **35** are relatively prime.

Give your answer to part B. here:

7. Given an **unsigned** integer**, N, N >= 0**, we wish to compute its binary representation. To begin with, and as examples, the string '11001' is the binary representation of 25, string '1001110' is binary representation of 78, and the string '0' is the binary representation of 0. But these are variable length representations of unsigned integers. We are looking for fixed-size **8-bit representation** of unsigned integers, **N**, that satisfy
`0 <= N < (2**8)` or `0 <= N < 256.`

Here are more example: `37: 00100101; 1 : 00000001; 0 : 00000000; 257:` **invalid**

Write functions/statements:
`Variable_Length_rep` to compute the variable-length representation of an unsigned integer, **N**
`Fixed_Length_rep` to compute the **8-bit fixed length representation** of integer, **N,** provided **`0 <= N < 256`**
Also write statements in main program that can be used to test the above with **`N = 13.`**

```
def Variable_Length_rep(N): # N is an unsigned integer. Complete the function here
```

```
def Variable_Length_rep(n): # assumes n >= 0
    if n > 1:
        remainder = n%2
        quotient = n // 2
        return(Variable_Length_rep(quotient) + str(remainder))
    else:
        return(str(n))
```

5 marks

total 12 marks

```
def Fixed_Length_rep(N): # First check if 0 <= N < 256. Complete the function here
```

```
def Fixed_Length_rep(n):
    if n < 0 or n >= 256:
        print('invalid number')
    else:
        var_rep = Variable_Length_rep(n)
        return(str('0'*(8-len(var_rep)))+var_rep)
```

5 marks

```
# Statements that are part of Main Program
```

```
# testing above definitions
number = 13
print(Variable_Length_rep(number))
print(Fixed_Length_rep(number))
```

2 marks

8. I am interested in investing my savings in one or more mutual funds from a list of SEVEN funds suggested to me. A fund, identified by its name, such as 'HDFC', is assessed by its (**`return`**, **`risk`**). The **`return`** is specified as a percentage, 12% for example. The higher the **`return`**, the better it is. The **`risk`** is specified as a factor, between 0 to 8. The higher the risk factor the greater is the risk. The data on 7 funds suggested by my broker, together with the corresponding **`return`** & **`risk`** is stored as a **`dictionary`**, whose elements are, FOR EXAMPLE,

`MFs = {ABSL: (8, 2), HDFC: (11, 3), ICICI: (12, 4), IDBI: (22, 7), Kotak: (21, 5), Templeton: (16, 2), UTI: (14, 5)}` or

Name of fund	**Return, percentage**	**Risk, index**
ABSL	8	2
HDFC	11	3
ICICI	12	4
IDBI	22	7
Kotak	21	5
Templeton	16	2
UTI	14	5

A. Write a function, median, to compute the median of a list of numbers in a list, L.

B. In the main program:

   a. Extract returns of the 7 funds, and compute their median, **`median_return`**. Above, the median return is 14.

   b. Extract risk index of the 7 funds, and compute their median, **`median_risk`**. Above, the median return is 4.

   c. Identify those funds that have a **`return >= median_return`**, and **`risk <= median_risk`**. Above the output is 'Templeton'.

```
def median(L): # L is a list of real numbers.
# complete the function followed by the main program here
```

The correct answer is:

12 marks total

3marks

```
    def median(L):
        L.sort()
        return(L[len(L)//2])
    #
    MFs = {'ABSL': (8, 2), 'HDFC': (11, 3), 'ICICI': (12, 4), 'IDBI': (22, 7), 'Kotak': (21, 5),
    'Templeton': (16, 2), 'UTI': (14, 5)}
```

3 marks

```
    returns = []
    for k in MFs.keys():
        returns.append((MFs[k][0]))
    median_return = median(returns)
    print(median_return)
```

3 marks

```
    risks = []
    for k in MFs.keys():
        risks.append((MFs[k][1]))
    median_risk = median(risks)
    print(median_risk)
```

3 marks

```
    for k in MFs.keys():
        if MFs[k][0] >= median_return and MFs[k][1] <= median_risk:
            print(k)
```

9. Three students in my course, Ram, Shyam and Aditi, have scored the following marks in mid-term and end-term exams:

name	roll number	mid-term	end-term
Ram	1	34	37
Shyam	2	55	52
Aditi	3	75	64

I would, however, like to create a new class "**Student**" with attributes **name**, **ID**, **midTerm**, **endTerm**. Their **ID** are assigned automatically using a 'Class Variable", **stuCount** that gets incremented automatically every time an object of class **Student** is instantiated.

a. Define necessary functions & statements for a new class **Student** with attributes **name**, **ID**, **midTerm**, **endTerm**.

b. Define a function to compute the grade to be assigned to each students using the table below. Here the "Average marks" are simply **(midTerm + endTerm)/2**.

Average marks	Grade
< 30	F
< 44	D
< 66	C
< 80	B
otherwise	A

c. Instantiate three objects, s1, s2, s3 corresponding to the three students together with their marks (note: their **ID** is assigned automatically).

d. Print their **name** and **ID**, and

e. Print their **name** and **grade**.

TO ANSWER THE ABOVE, SIMPLY FILL IN THE BLANKS BELOW:

The correct answer is:

```
class Student():
    stuCount = 1
    def __init__(self, fullname, midterm_marks, endterm_marks):
        self.name = fullname
        self.ID = Student.stuCount
        Student.stuCount = Student.stuCount + 1
        self.midterm = midterm_marks
        self.endterm = endterm_marks
    def gettotal(self):
        return((self.midterm + self.endterm)/2)
    def assigngrade(self):
        if self.gettotal() < 30:
            return('F')
        if self.gettotal() < 44:
            return('D')
        if self.gettotal() < 67:
            return('C')
        if self.gettotal() < 80:
            return('B')
        else:
            return('A')
s1 = Student('Ram', 34, 37)
s2 = Student('Shyam', 55, 52)
s3 = Student('Aditi', 75, 64)
print(s1.name, s1.ID)
print(s2.name, s2.ID)
print(s3.name, s3.ID)
print(s1.name, s1.assigngrade())
print(s2.name, s2.assigngrade())
print(s3.name, s3.assigngrade())
```

1 1 1 1 1 1

2 2 2

total 12 marks

Part 2: There are 4 questions of 12 marks each

6. This is about defining a module and using it to solve problems related to prime numbers.

   A. Create or define a Python module, ‘**primes**’, by re-writing the following code

```
# module for working with prime numbers – named ‘primes’
"""Collection of functions related to primes, including
    Is_Prime(K)to determine whether given number K is a prime number,
    Max_Prime(N) to determine the largest prime <= N,
    Relative_Prime(P, Q)to determine whether P & Q are relatively prime."""
# Definition of Is_Prime(K), Max_Prime(N), Relative_Prime(P,Q) goes here
# and any other required function or variables
#
# to be saved as primes.py
```

1 mark

```
def Is_Prime(K):                    # assumes K >= 2
    if K > 3:
        composite = False
        for j in range(2, K-1):
            if K%j == 0:
                composite = True
                return(False)
    return(True)
```

2 marks

```
def Max_Prime(N):
    for m in range(N, 1, -1):
        if Is_Prime(m):
            return(m)
```

2 marks

```
def gcd(a, b):
    if b > a:
        return(gcd(b, a))
    if a%b == 0:
        return(b)
    return(gcd(b, a%b))
def Relative_Prime(P, Q):
    if gcd(P, Q) > 1:
        return(False)
    else:
        return(True)
```

2 marks

total 12 mark

   B. Now write a main program that will use the module, ‘**primes**’, to

   (a) Check if number **37** is a prime number,

   (b) Compute the largest prime number **<=** 55, and

   (c) Determine whether **24** & **35** are relatively prime.

Give your answer to part B. here:

2 marks

```
# main program
Import primes
# Check if number 37 is a prime number
print(primes.Is_Prime(37))
# Compute the largest prime number <= 55, and
print(primes.Max_Prime(55))
# Determine whether 24 & 35 are relatively prime.
print(primes.Relative_Prime(6, 15))
```

1 mark each