USN | | | | | | | | | | |

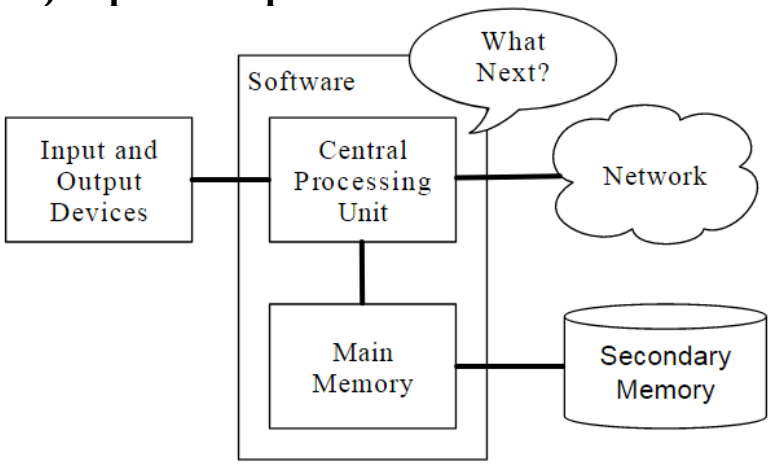
Internal Assessment Test 1 – March 2018

Sub:	Python Application Programming					Sub Code:	15CS664	Branch:	ECE / EEE
Date:	14/03/2018	Duration:	90 min's	Max Marks:	50	Sem / Sec:	ECE – C,D & EEE-A		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1 (a)	Explain computer hardware architecture with a neat diagram.	[05]	CO1	L1
(b)	Discuss different types of errors in general.	[05]	CO1	L1
2 (a)	Explain chained conditional and nested conditional statements in python.	[05]	CO1	L1
(b)	Write a program to find maximum in 3 numbers using nested if	[05]	CO1	L3
3 (a)	Why is type( ) function used ? Explain different 'types' of values in python.	[05]	CO1	L2
(b)	Write a program to convert Fahrenheit temperature to a Celsius temperature.	[05]	CO1	L3
4 (a)	Explain the usage of format operator with examples.	[05]	CO2	L2
(b)	Write a program to count the number of letters in a string using while loop.	[05]	CO2	L3
5 (a)	Explain any two functions from math and two functions from random module	[05]	CO1	L1
(b)	Explain the following built-in functions with example. i. range( ) ii. input( )	[05]	CO1	L1
6	Explain different types of operations in python. Discuss their order of execution.	[10]	CO1	L2
7	Write a program which repeatedly reads numbers until the user enters "done". Once "done" is entered, print the sum, minimum, maximum and average of the numbers. If the user enters anything other than a number, print an error message and continue the current iteration again.	[10]	CO1	L3
8	Write a program to takes score between 0.0 and 1.0 as input. If the score is out of range, print an error message. Create a function called *computegrade* which takes score as input and returns grade as a string. If the score is between 0.0 and 1.0, print a grade using the following table Score Grade >= 0.9 A >= 0.8 B >= 0.7 C >= 0.6 D < 0.6 F	[10]	CO1	L3

1 a) Explain computer hardware architecture with a neat diagram.



The high-level definitions of these parts are as follows:

- The *Central Processing Unit* (or CPU) is the part of the computer that is built to be obsessed with "what is next?" If your computer is rated at 3.0 Gigahertz, it means that the CPU will ask "What next?" three billion times per second. You are going to have to learn how to talk fast to keep up with the CPU.
- The *Main Memory* is used to store information that the CPU needs in a hurry. The main memory is nearly as fast as the CPU. But the information stored in the main memory vanishes when the computer is turned off.
- The *Secondary Memory* is also used to store information, but it is much slower than the main memory. The advantage of the secondary memory is that it can store information even when there is no power to the computer. Examples of secondary memory are disk drives or flash memory (typically found in USB sticks and portable music players).
- The *Input and Output Devices* are simply our screen, keyboard, mouse, microphone, speaker, touchpad, etc. They are all of the ways we interact with the computer.
- These days, most computers also have a *Network Connection* to retrieve information over a network. We can think of the network as a very slow place to store and retrieve data that might not always be "up". So in a sense, the network is a slower and at times unreliable form of *Secondary Memory*.

1 b) Discuss different types of errors in general.

Three general types of errors are:

Syntax errors

These are the first errors you will make and the easiest to fix. A syntax error means that you have violated the "grammar" rules of Python. Python does its best to point right at the line and character where it noticed it was confused. The only tricky bit of syntax errors is that sometimes the mistake that needs fixing is actually earlier in the program than where Python *noticed* it was confused. So the line and character that Python indicates in a syntax error may just be a starting point for your investigation.

Logic errors

A logic error is when your program has good syntax but there is a mistake in the order of the statements or perhaps a mistake in how the statements relate to one another. A good example of a logic error might be, "take a drink from your water bottle, put it in your backpack, walk to the library, and then put the top back on the bottle."

Semantic errors

A semantic error is when your description of the steps to take is syntactically perfect and in the right order, but there is simply a mistake in the program. The program is perfectly correct but it does not do what you *intended* for it to do. A simple example would be if you were giving a person directions to a restaurant and said, "...when you reach the intersection with the gas station, turn left and go one mile and the restaurant is a red building on your left." Your friend is very late and calls you to tell you that they are on a farm and walking around behind a barn, with no sign of a restaurant. Then you say "did you turn left or right at the gas station?" and they say, "I followed your directions perfectly, I have them written down, it says turn left and go one mile at the gas station." Then you say, "I am very sorry, because while my instructions were syntactically correct, they sadly contained a small but undetected semantic error."

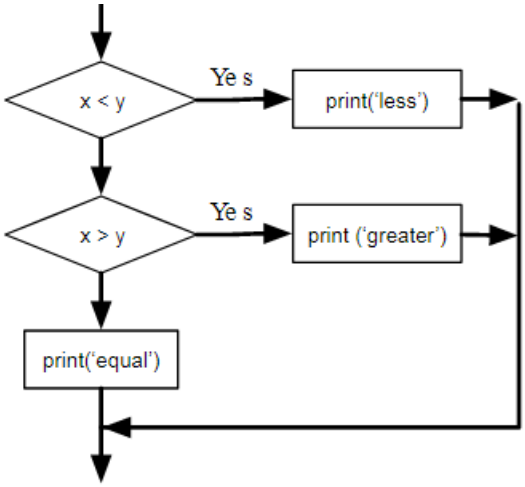
2 a) Explain chained conditional and nested conditional statements in python.

Chained Conditional statement:

Sometimes in executions there are more than two possibilities/conditions and we need more than two branches. One way to express a computation like that is a *chained conditional*:

```
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```

elif is an abbreviation of "else if." Again, exactly one branch will be executed.

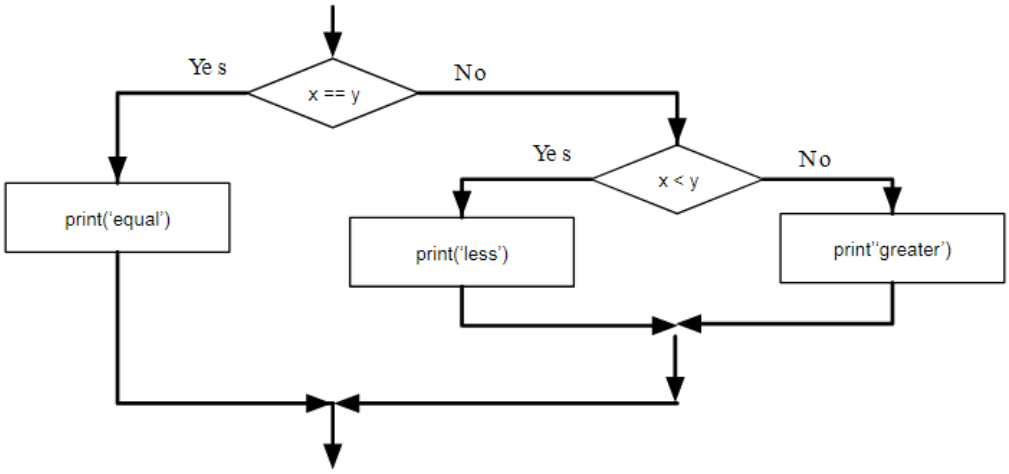


There is no limit on the number of elif statements. If there is an else clause, it has to be at the end, but there doesn't have to be one. Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

Nested Conditional statement:
One conditional can also be nested within another. We could have written the three-branch example like this:

```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

The outer conditional contains two branches. The first branch contains a simple statement. The second branch contains another if statement, which has two branches of its own. Those two branches are both simple statements, although they could have been conditional statements as well.



Logical operators often provide a way to simplify nested conditional statements. For example, we can rewrite the following code using a single conditional:

```
if 0 < x:
    if x < 10:
        print('x is a positive single-digit number.')
```

The print statement is executed only if we make it past both conditionals, so we can get the same effect with the and operator:

```
if 0 < x and x < 10:
    print('x is a positive single-digit number.')
```

2 b) Write a program to find maximum in 3 numbers using nested if

```
try:
    x=int(input("Enter 1st number"))
    y=int(input("Enter 2nd number"))
    z=int(input("Enter 3rd number"))
    if(x>=y):
        if(x>=z):
            print("Greater number is: ",x)
        elif(x<z):
            print("Greater number is: ",z)
    elif(y>=z):
        print("Greater number is: ",y)
    else:
        print("Greater number is: ",z)
except:
    print("Enter a number!!")
```

3 a) Why is type( ) function used ? Explain different ‘types’ of values in python.

- A *value* is one of the basic things a program works with, like a letter or a number. These values belong to different *types*: 2 is an integer, and “Hello, World!” is a *string*, so called because it contains a “string” of letters.
- If you are not sure what type a value has, the interpreter can tell you.

  >>> type('Hello, World!')

  <**class** 'str'>

  >>> type(17)

  <**class** 'int'>
- Less obviously, numbers with a decimal point belong to a type called float, because these numbers are represented in a format called *floating point*.

  >>> type(3.2)

  <**class** 'float'>

3 b) Write a program to convert Fahrenheit temperature to a Celsius temperature

```
inp = input('Enter Fahrenheit Temperature:')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:
    print('Please enter a number')
```

4 a) Explain the usage of format operator with examples.

- The *format operator*, % allows us to construct strings, replacing parts of the strings with the data stored in variables. When applied to integers, % is the modulus operator. But when the first operand is a string, % is the format operator.
- For example, the format sequence “%d” means that the second operand should be formatted as an integer (d stands for “decimal”):

  >>> camels = 42

  >>> '%d' % camels

  '42'

- A format sequence can appear anywhere in the string, so you can embed a value in a sentence:

  >>> camels = 42

  >>> 'I have spotted %d camels.' % camels

  'I have spotted 42 camels.'

- The following example uses “%d” to format an integer, “%g” to format a floating point number (don’t ask why), and “%s” to format a string:

  >>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')

  'In 3 years I have spotted 0.1 camels.'

4 b) Write a program to count the number of letters in a string using while loop.

```
str=input("Enter String :")
count=0
while count<len(str):
    count+=1
print("String length is : ",count)
```

5 a) Explain any two functions from math and two functions from random module

Math Module

This module is always available. It provides access to the mathematical functions defined by the C standard.

- math.ceil(*x*): Return the ceiling of *x*, the smallest integer greater than or equal to *x*.
- math.factorial(*x*): Return *x* factorial. Raises ValueError if *x* is not integral or is negative.
- math.floor(*x*): Return the floor of *x*, the largest integer less than or equal to *x*.

Random Module

- Pseudorandom numbers are not truly random because they are generated by a deterministic computation, but just by looking at the numbers it is all but impossible to distinguish them from random. The random module provides functions that generate pseudorandom numbers.
- The function random returns a random float between 0.0 and 1.0 (including 0.0 but not 1.0).

  import random

for i in range(10):

x = random.random()

print(x)

- The random function is only one of many functions that handle random numbers. The function randint takes the parameters low and high, and returns an integer between low and high (including both).

>>> random.randint(5, 10)

5

>>> random.randint(5, 10)

9

5 b) Explain the following built-in functions with example.

i)Range Function

range() constructor has two forms of definition:

1. range(stop)
2. range(start, stop[, step])

range() Parameters

range() takes mainly three arguments having the same use in both definitions:

- **start** - integer starting from which the sequence of integers is to be returned
- **stop** - integer before which the sequence of integers is to be returned.
  The range of integers end at **stop - 1**.
- **step (Optional)** - integer value which determines the increment between each integer in the sequence

>>> range(1,10)

[1, 2, 3, 4, 5, 6, 7, 8, 9]

ii) Input Function

Python provides a built-in function called input that gets input from the keyboard1. When this function is called, the program stops and waits for the user to type something. When the user presses Return or Enter, the program resumes and input returns what the user typed as a string.

>>> inp = input()

Some silly stuff

```
>>> print(inp)
Some silly stuff
```

Before getting input from the user, it is a good idea to print a prompt telling the user what to input. You can pass a string to input to be displayed to the user before pausing for input:

```
>>> name = input('What is your name?\n')
What is your name?
Chuck
>>> print(name)
Chuck
```

If you expect the user to type an integer, you can try to convert the return value to int using the int() function:

```
>>> prompt = 'What...is the airspeed velocity of an unladen swallow?\n'
>>> speed = input(prompt)
What...is the airspeed velocity of an unladen swallow?
17
>>> int(speed)
17
>>> int(speed) + 5
22
```

6) Explain different types of operations in python. Discuss their order of execution.

Operators are special symbols that represent computations like addition and multiplication.

The values the operator is applied to are called *operands*.

The operators +, -, *, /, and ** perform addition, subtraction, multiplication, division, and exponentiation, as in the following examples:

Addition operation: Use operator +

```
>>> 5+2
7
```

Multiplication operation: Use operator *

```
>>>5*9
45
```

Division Operation:

There has been a change in the division operator between Python 2.x and Python 3.x. In Python 3.x, the result of this division is a floating point result:

```
>>> minute = 59
>>> minute/60
0.9833333333333333
```

The division operator in Python 2.0 would divide two integers and truncate the result to an integer:

```
>>> minute = 59
>>> minute/60
0
```

To obtain the same answer in Python 3.0 use floored (// integer) division.

```
>>> minute = 59
>>> minute//60
0
```

In Python 3.0 integer division functions much more as you would expect if you entered the expression on a calculator.

The *modulus operator* works on integers and yields the remainder when the first operand is divided by the second. In Python, the modulus operator is a percent sign (%). The syntax is the same as for other operators:

```
>>> quotient = 7 // 3
>>> print(quotient)
2
>>> remainder = 7 % 3
>>> print(remainder)
1
```

Order of operations

When more than one operator appears in an expression, the order of evaluation depends on the *rules of precedence*. For mathematical operators, Python follows mathematical convention. The acronym *PEMDAS* is a useful way to remember the rules:

- *P*arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1) is 4, and (1+1) **(5-2) is 8. You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even if it doesn't change the result.
- *E*xponentiation has the next highest precedence, so 2**1+1 is 3, not 4, and 3*1**3 is 3, not 27.
- *M*ultiplication and *D*ivision have the same precedence, which is higher than
- *A*ddition and *S*ubtraction, which also have the same precedence. So 2*3-1 is 5, not 4, and 6+4/2 is 8.0, not 5.
- Operators with the same precedence are evaluated from left to right. So the expression 5-3-1 is 1, not 3, because the 5-3 happens first and then 1 is subtracted from 2.

7) Write a program which repeatedly reads numbers until the user enters "done". Once "done" is entered, print the sum, minimum, maximum and average of the numbers. If the user enters anything other than a number, print an error message and continue the current iteration again.

```
total=0
count=0
max=None
min=None
while True:
   print("Menu")
   print("1. Enter a number 2. Quit")
   choice=int(input("Enter your choice"))
   if(choice==1):
      try:
         num=int(input("Enter number"))
         if(min== None or num<min):
            min=num
         if(max==None or num>max):
            max=num
         total=total+num
         count=count+1
      except:
         print("Enter a valid value")
   elif(choice==2):
```

```
    txt=input("Enter done")
    if(txt=='done'):
       break;

print("Total is ",total)
print("Count is ", count)
print("Average is ",(total/count))
print("Maximum is ",max)
print("Minimum is ", min)
```

8) Write a program to takes score between 0.0 and 1.0 as input. If the score is out of range, print an error message. Create a function called *computegrade* which takes score as input and returns grade as a string. If the score is between 0.0 and 1.0, print a grade using the following table

Score	**Grade**
>= 0.9	**A**
>= 0.8	**B**
>= 0.7	**C**
>= 0.6	**D**
< 0.6	**F**

```
'''Program to compute grade'''
def computegrade(score):
    if score > 1.0 or score < 0.0:
        return 'Bad score'
    elif score >= 0.9:
        return 'A'
    elif score >= 0.8:
        return 'B'
    elif score >= 0.7:
        return 'C'
    elif score >= 0.6:
        return 'D'
    else:
        return 'F'
inp = input('Enter score: ')
try:
    score = float(inp)
    grade=computegrade(score)
    print("The grade is :",grade)
except:
    print("Enter a number!!")
```