Unit-2: Data types, I/O, Types of Errors and Conditional Constructs

Course learning objectives:

Students will get an overview of

- What is Data type and types of Data,
- How to convert from one data type to another data type,
- Mutable and Immutable types,
- Input and Output Operations and Formats,
- Types of Errors in python
- Types of conditional constructs

Module – 1

2.1 Data types, I/O, Types of Errors

A data type is a data storage format that can contain a specific type or range of values. Variables can store data of different types, and different types can do different things. Python has the following data types built-in by default, in these categories.

Numeric Types:	int, float, complex
Sequence Types:	str, list, tuple, range
Mapping Type:	Dict
Set Types:	set, frozenset

2.1.1 Numeric Types:

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them

Example

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

Int: Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

```
x = 1
y = 35656222554887711
z = -3255522
```

Float: Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

```
x = 1.10
y = 1.0
z = -35.59
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example

```
x = 35e3
y = 12E4
z = -87.7e100
```

Complex: Complex numbers are written with a "j" as the imaginary part.

Example

```
x = 3+5j
y = 5j
z = -5j
```

2.1.2 Sequence Types

In Python, **sequence** is the generic term for an ordered set. There are several types of sequences in Python, the following are the most important.

String:

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example

```
print("Hello")
print('Hello')
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

Example

```
a = """Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod empor incididunt ut labore et
dolore magna aliqua."""
print(a)
```

Or three single quotes:

```
a = '''Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua.'''
print(a)
```

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Lists:

List is an ordered sequence of items. All the items in a list do not need to be of the same type and lists are **mutable** - they can be changed. Elements can be reassigned or removed, and new elements can be inserted.

Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

```
a = [1, 2.2, 'python']
```

Tuples:

Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than lists as they cannot change dynamically.

It is defined within parentheses () where items are separated by commas.

```
t = (5,'program', 1+3j)
```

range():

The range() type returns an immutable sequence of numbers between the given start integer to the stop integer.

```
print(list(range(10)))
```

2.1.3 Mapping type:

Dictionary is an unordered collection of key-value pairs. In python there is mapping type called **dictionary**. It is mutable. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type. The values of the dictionary can be any type, but the key must be of any immutable data type such as strings, numbers, and tuples.

```
>>> d = {1:'value','key':2}
>>> type(d)
```

2.1.4 Set types:

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

```
a = {5,2,3,1,4}

# printing set variable
print("a = ", a)

# data type of variable a
print(type(a))
```

We can use the set for some mathematical operations like set union, intersection, difference etc. We can also use set to remove duplicates from a collection.

2.2. Getting the Data type

We can use the type() function to know which class a variable or a value belongs to. Similarly, the isinstance() function is used to check if an object belongs to a particular type.

```
a = 5
print(a, "is of type", type(a))
a = 2.0
print(a, "is of type", type(a))
a = 1+2j
print(a, "is complex number?", isinstance(a,complex))
```

Output

```
5 is of type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is complex number? True
```

2.2. Type Conversion

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

1. Implicit Type Conversion
2. Explicit Type Conversion

2.2.1 Implicit Type Conversion

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement. Let's see an example where Python promotes the conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.

Example 1: Converting integer to float

When we run the above program, the output will be:

```
num_int = 123
num_flo = 1.23
num_new = num_int + num_flo
print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))
print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

Output

```
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>
Value of num_new: 124.23
datatype of num_new: <class 'float'>
```

In the above program,

- We add two variables num_int and num_flo, storing the value in num_new.
- We will look at the data type of all three objects respectively.
- In the output, we can see the data type of num_int is an integer while the data type of num_flo is a float.
- Also, we can see the num_new has a float data type because Python always converts smaller data types to larger data types to avoid the loss of data.

Now, let's try adding a string and an integer, and see how Python deals with it.

Example 2: Addition of string(higher) data type and integer(lower) datatype

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))

print(num_int+num_str)
```

When we run the above program, the output will be:

```
Data type of num_int: <class 'int'>
Data type of num_str: <class 'str'>

Traceback (most recent call last):
  File "python", line 7, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In the above program,

- We add two variables num_int and num_str.
- As we can see from the output, we got TypeError. Python is not able to use Implicit Conversion in such conditions.
- However, Python has a solution for these types of situations which is known as Explicit Conversion.

2.2.2 Explicit Type Conversion

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.
This type of conversion is also called typecasting because the user casts (changes) the data type of the objects.

Syntax :

```
<required_datatype>(expression)
```

Typecasting can be done by assigning the required data type function to the expression.

Example 3: Addition of string and integer using explicit conversion

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))

num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

num_sum = num_int + num_str

print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

When we run the above program, the output will be:

```
Data type of num_int: <class 'int'>
Data type of num_str before Type Casting: <class 'str'>

Data type of num_str after Type Casting: <class 'int'>

Sum of num_int and num_str: 579
Data type of the sum: <class 'int'>
```

In the above program,

- We add num_str and num_int variable.
- We converted num_str from string(higher) to integer(lower) type using int() function to perform the addition.
- After converting num_str to an integer value, Python is able to add these two variables.
- We got the num_sum value and data type to be an integer.

Key Points to Remember

- Type Conversion is the conversion of object from one data type to another data type.
- Implicit Type Conversion is automatically performed by the Python interpreter.
- Python avoids the loss of data in Implicit Type Conversion.
- Explicit Type Conversion is also called Type Casting, the data types of objects are converted using predefined functions by the user.
- In Type Casting, loss of data may occur as we enforce the object to a specific data type.

Module-2

2.4 Mutable and Immutable types

A first fundamental **distinction** that **Python** makes on data is about whether the value changes are not. If the value can change, then is called **mutable**, while if the value cannot change, that is called **immutable**.

Mutable:

- list,
- dict
- set

Immutable:

- int,
- float,
- complex,
- string,
- tuple

2.5 Input and Output Operations and Formats

Python provides numerous built-in functions that are readily available at the Python prompt.
Some of the functions like input() and print() are widely used for standard input and output operations respectively.

2.5.1 Python Output Using print() function

We use the print() function to output data to the standard output device (screen).

Example 1:

```
print('This sentence is output to the screen')
```

Output

```
This sentence is output to the screen
```

Example 2:

```
a = 5
print('The value of a is', a)
```

Output

```
The value of a is 5
```

In the second print() statement, we can notice that space was added between the string and the value of variable a. This is by default, but we can change it.
The actual **syntax** of the print() function is:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- Here, objects are the value(s) to be printed.
- The sep(separator) is used between the values. It defaults into a space character.
- After all values are printed, end is printed. It defaults into a new line.

- The file is the object where the values are printed and its default value is sys.stdout (screen). Here is an example to illustrate this.

```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='*')
print(1, 2, 3, 4, sep='#', end='&')
```

Output

```
1 2 3 4
1*2*3*4
1#2#3#4&
```

2.5.2 Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

```
>>> x = 5; y = 10
>>> print('The value of x is {} and y is {}'.format(x,y))
The value of x is 5 and y is 10
```

Here, the curly braces {} are used as placeholders. We can specify the order in which they are printed by using numbers.

```
print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))
```

Output

```
I love bread and butter
I love butter and bread
```

We can even use keyword arguments to format the string.

```
>>> print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))
Hello John, Goodmorning
```

We can also format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

```
>>> x = 12.3456789
>>> print('The value of x is %3.2f' %x)
The value of x is 12.35
>>> print('The value of x is %3.4f' %x)
The value of x is 12.3457
```

2.5.3 Python Input

Until now, our programs were static. The value of variables was defined or hard coded into the source code.

To allow flexibility, we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is:

```
input([prompt])
```

where prompt is the string we wish to display on the screen. It is optional.

```
>>> num = input('Enter a number: ')
Enter a number: 10
>>> num
'10'
```

Here, we can see that the entered value 10 is a string, not a number. To convert this into a number we can use int() or float() functions.

```
>>> int('10')
10
>>> float('10')
10.0
```

2.6 Types of Errors in python

No matter how smart or how careful you are, errors are your constant companion. With practice, you will get slightly better at not making errors, better at finding and correcting them. There are three kinds of errors: syntax errors, runtime errors, and logic errors.

2.6.1 Syntax errors: Syntax errors are produced by Python when it is translating the source code into byte code. They usually indicate that there is something wrong with the syntax of the program.

Example: Omitting the colon at the end of an if statement yields the somewhat redundant message.

SyntaxError: invalid syntax.

Here are some ways to avoid the most common syntax errors:

1. Make sure you are not using a Python keyword for a variable name.
2. Check that you have a colon at the end of the header of every compound statement, including for, while, if, and def statements.
3. Check that indentation is consistent. You may indent with either spaces or tabs but it's better not to mix them. Each level should be nested the same amount.
4. Make sure that strings in the code have matching quotation marks.
5. If you have multiline strings with triple quotes (single or double), make sure you have terminated the string properly. An un-terminated string may cause an invalid token error at the end of your program, or it may treat the following part of the program as a string until it comes to the next string. In the second case, it might not produce an error message at all!
6. An unclosed bracket – (, {, or [– makes Python continue with the next line as part of the current statement. Generally, an error occurs almost immediately in the next line.
7. Check for the classic = instead of == inside a conditional.

2.6.2 Run Time Error: Errors that occur after the code has been executed and the program is running. The error of this type will cause your program to behave unexpectedly or even crash. An example of a runtime error is the division by zero. Consider the following example:

```
x = float(input('Enter a number: '))
y = float(input('Enter a number: '))
z = x/y
```

```
print (x,'divided by',y,'equals: ',z)
```

The program above runs fine until the user enters **0** as the second number:

```
>>>
Enter a number: 9
Enter a number: 2
9.0 divided by 2.0 equals: 4.5
>>>
Enter a number: 11
Enter a number: 3
11.0 divided by 3.0 equals: 3.6666666666666665
>>>
Enter a number: 5
Enter a number: 0
Traceback (most recent call last):
File "C:/Python34/Scripts/error1.py", line 3, in <module>
z = x/y
ZeroDivisionError: float division by zero
>>>Erors
```

2.6.3 Logical Errors:

It is also called **semantic errors**, logical errors cause the program to behave incorrectly, but they do not usually crash the program. Unlike a program with syntax errors, a program with logic errors can be run, but it does not operate as intended. Consider the following example of logical error:

```
x = float(input('Enter a number: '))
y = float(input('Enter a number: '))
z = x+y/2
print ('The average of the two numbers you have entered is:',z)
```

The example above should calculate the average of the two numbers the user enters. But, because of the order of operations in arithmetic (the division is evaluated before addition) the program will not give the right answer:

```
>>>
Enter a number: 3
Enter a number: 4
The average of the two numbers you have entered is: 5.0
>>>
```

To rectify this problem, we will simply add the parentheses: **z = (x+y)/2**

Now we will get the right result:

```
>>>

Enter a number: 3

Enter a number: 4

The average of the two numbers you have entered is: 3.5

>>>
```

Module-3:

3.7 Conditional Constructs

There come situations in real life when we need to make some decisions and based on these decisions, we decide what we should do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.

Decision making statements in programming languages decides the direction of flow of program execution. Decision making statements available in python are:

- if statement
- if..else statements
- nested if statements
- if-elif ladder
- Short Hand if statement
- Short Hand if-else statement

3.7.1 if statement: if statement is the simplest decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if condition:
   # Statements to execute if
   # condition is true
```

Here, condition after evaluation will be either true or false. if statement accepts boolean values – if the value is true then it will execute the block of statements below it otherwise not. We can use *condition* with bracket '(' ')' also.

As we know, python uses indentation to identify a block. So the block under an if statement will be identified as shown in the below example:

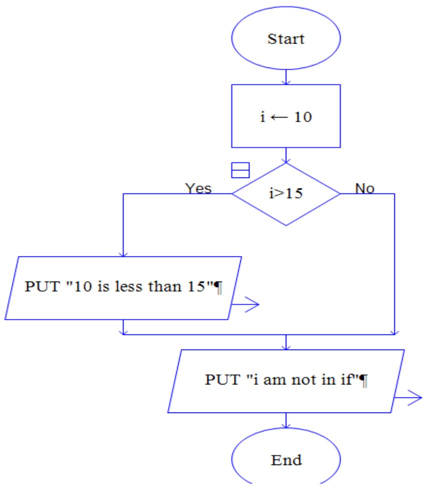
```
if condition:
   statement1
statement2

# Here if the condition is true, if block
# will consider only statement1 to be inside
# its block.
```

Flowchart:-

```
# python program to illustrate If statement

i = 10
if (i > 15):
   print ("10 is less than 15")
print ("I am Not in if")
```



Output:

```
I am Not in if
```

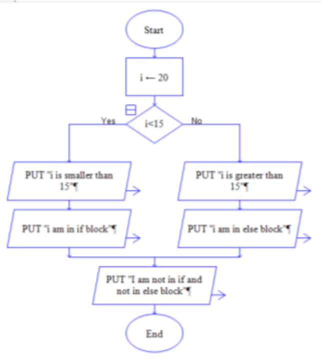
As the condition present in the if statement is false. So, the block below the if statement is not executed.

3.7.2 if- else: The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the *else* statement. We can use the *else* statement with *if* statement to execute a block of code when the condition is false.

Syntax:

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

Flow Chart:-


Start
i ← 20
i<15
Yes
No
PUT "i is smaller than 15"¶
PUT "i is greater than 15"¶
PUT "i am in if block"¶
PUT "i am in else block"¶
PUT "I am not in if and not in else block"¶
End

```
# python program to illustrate If else statement
#!/usr/bin/python

i = 20;
if (i < 15):
    print ("i is smaller than 15")
    print ("i'm in if Block")
else:
    print ("i is greater than 15")
    print ("i'm in else Block")
print ("i'm not in if and not in else Block")
```

Output:

```
i is greater than 15
i'm in else Block
i'm not in if and not in else Block
```

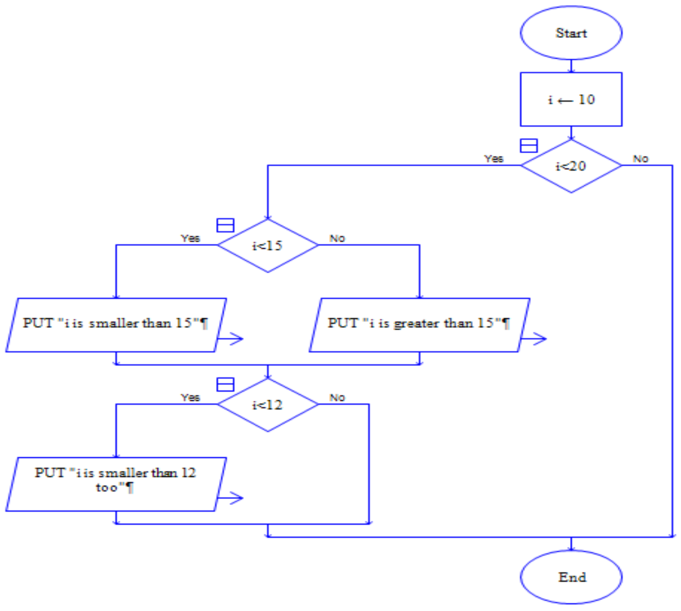
The block of code following the else statement is executed as the condition present in the if statement is false after call the statement which is not in block(without spaces).

3.7.3 nested-if: A nested if is an if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement. Yes, Python allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```

Flow chart:


Start
i ← 10
i<20
Yes
No
i<15
Yes
No
PUT "i is smaller than 15"¶
PUT "i is greater than 15"¶
i<12
Yes
No
PUT "i is smaller than 12 too"¶
End

```
# python program to illustrate nested If statement
#!/usr/bin/python
i = 10
if (i <20):
    #  First if statement
    if (i < 15):
        print ("i is smaller than 15")
    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    else:
        print ("i is greater than 15")
    if (i < 12):
        print ("i is smaller than 12 too")
```

Output:

```
i is smaller than 15
i is smaller than 12 too
```

3.7.4 if-elif-else ladder: Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:-

```
if (condition):
    statement
elif (condition):
    statement
.
```

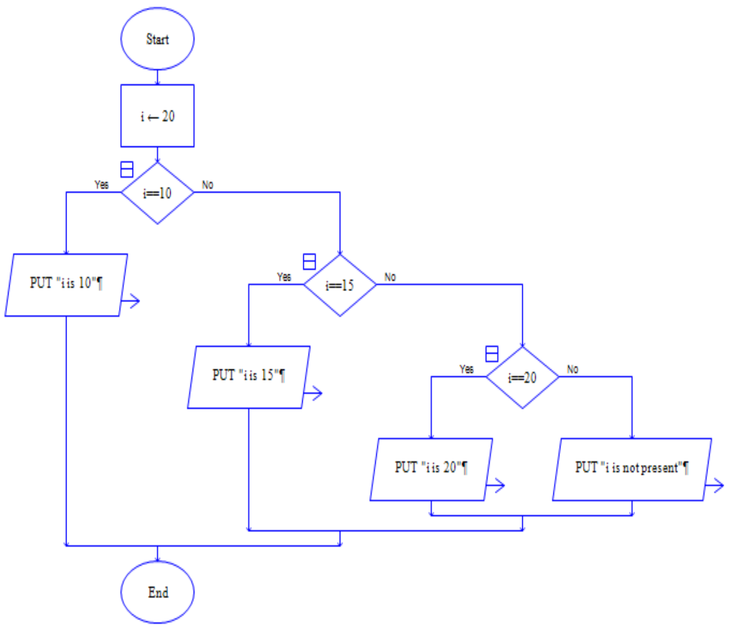
```
.
else:
    statement
```

Flow Chart:-

Example:-

```
# Python program to illustrate if-elif-else ladder
#!/usr/bin/python

i = 20
if (i == 10):
    print ("i is 10")
elif (i == 15):
    print ("i is 15")
elif (i == 20):
    print ("i is 20")
else:
    print ("i is not present")
```



Output:

```
i is 20
```

3.8 Short Hand if statement

Whenever there is only a single statement to be executed inside the if block then shorthand if can be used. The statement can be put on the same line as the if statement.

Syntax:

```
if condition: statement
```

Example:

```
# Python program to illustrate short hand if
i = 10
if i < 15: print("i is less than 15")
```

Output:

```
i is less than 15
```

Short Hand if-else statement: This can be used to write the if-else statements in a single line where there is only one statement to be executed in both if and else block.

Syntax:

```
statement_when_True if condition else statement_when_False
```

Example:

```
# Python program to illustrate short hand if-else
i = 10
print(True) if i < 15 else print(False)
```

Output:

```
True
```

MCQ on Data Types and Conditional Constructs

1. Which statement is correct?
   a. List is immutable && Tuple is mutable
   [✓ b. List is mutable && Tuple is immutable]
   c. Both are Mutable.
   d. Both are Immutable.
2. Which one of the following is mutable data type?
   [✓ a. Set]
   b. Int
   c. Str
   d. tupl
3. Which one of the following is immutable data type?
   a. List
   b. Set
   [✓ c. Int]
   d. dict
4. Which of these is not a core data type?
   a. List
   b. Tuple
   c. Dictionary
   [✓ d. Class]
5. Which one of the following is False regarding data types in Python?
   a. In python, explicit data type conversion is possible
   b. Mutable data types are those that can be changed.
   c. Immutable data types are those that cannot be changed.
   [✓ d. None of the above]
6. Which of the following function is used to know the data type of a variable in Python?
   a. datatype()
   b. typeof()
   [✓ c. type()]
   d. vartype()
7. Which one of the following is a valid Python if statement :
   [✓ a. if (a>=2):]
   b. if (a >= 2)
   c. if (a => 22)
   d. if a >= 22
8. What keyword would you use to add an alternative condition to an if statement?
   a. else if
   b. elseif
   c. elif
   d. None of the above
9. Can we write if/else into one line in python?
   a. Yes
   b. No
   c. if/else not used in python
   d. None of the above
10. Which statement will check if a is equal to b?
    a. if a = b:
    [✓ b. if a == b:]
    c. if a === c:
    d. if a == b

Solved Problem Set

1. Write a Python program to find whether the given number is divisible by 7 and multiple of 5?

```
num=int(input("Enter a number: "))
if num%7==0 and num%5==0:
    print(num, "is divisible by 7 and multiple of 5")
else:
    print("the given number is either divisible by 7 or multiple of 5")
```

2. Write a program to input any number and check whether it is even or odd

```
num=int(input("Enter a number: "))
if num%2==0:
    print("Even number")
else:
    print("Odd number")
```

3. Write a program to input any number and check whether it is negative, positive or zero

```
num=int(input("Enter a number: "))
if num>0:
    print("Positive number")
else if num<=0:
    print("Negative number")
else:
    print("Zero")
```

4. A student will not be allowed to sit in exam if his/her attendence is less than 75%.
   Take following input from user
   Number of classes held
   Number of classes attended.
   And print
   percentage of class attended
   Is student is allowed to sit in exam or not.

```
print "Number of classes held"
noh = input()
print "Number of classes attended"
noa = input()
atten = (noa/float(noh))*100
print "Attendence is",atten
if atten >= 75:
  print "You are allowed to sit in exam"
else:
  print "Sorry, you are not allowed. Attend more classes from next time."
```

Un-Solved Problem Set

1. Take values of length and breadth of a rectangle from user and check if it is square or not.
2. Take two int values from user and print greatest among them
3. A shop will give discount of 10%, if the cost of purchased quantity is more than 1000.
   Ask user for quantity
   Suppose, one unit will cost 100.
   Ex: quantity=11
   cost=11*100=1100>1000
   so, he will get 10% discount, that is 110.
   Final cost is 1100-110=990
4. A company decided to give bonus of 5% to employee if his/her year of service is more than 5 years.
   Ask user for their salary and year of service and print the net bonus amount.
5. A school has following rules for grading system:
   a. Below 25 - F
   b. 25 to 45 - E
   c. 45 to 50 - D
   d. 50 to 60 - C
   e. 60 to 80 - B
   f. Above 80 - A
   Ask user to enter marks and print the corresponding grade.
6. Take input of age of 3 people by user and determine oldest and youngest among them.
7. Write a program to print absolute vlaue of a number entered by user. E.g.-
   INPUT: 1   OUTPUT: 1
   INPUT: -1   OUTPUT: 1
8. Modify the 4th question (in solved problem set) to allow student to sit if he/she has medical cause. Ask user if he/she has medical cause or not ('Y' or 'N') and print accordingly.
9. Ask user to enter age, sex (M or F), marital status (Y or N) and then using following rules print their place of service.
   if employee is female, then she will work only in urban areas.
   if employee is a male and age is in between 20 to 40 then he may work in anywhere
   if employee is male and age is in between 40 t0 60 then he will work in urban areas only.
   And any other input of age should print "ERROR".

Descriptive Questions

1. What is data type? What are the different types of data?
2. What are the numeric data types? Explain with examples.
3. What are the sequence types of data?
4. Explain about type conversions.
5. What is mutable and immutable? What are mutable and immutable data types?
6. Explain about python output formatting.
7. Describe types of errors.
8. Explain about types of conditional constructs.
9. Give examples for all conditional constructs.
10. Write shorthand if and shorthand if-else statement.