Unit 2nd Data Types & Control statement

Python Variables:

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value. In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type. Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for variable name. Rahul and rahul both are two different variables.

Declaring Variable and Assigning Values:

Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Ex:

```
a=10

b=10.5
```

Multiple Assignments:

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignments.

We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Lets see given examples,

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
print (x )
print (y)
print (z)
```

2. Assigning multiple values to multiple variables:

Eg:

```
a,b,c=5,10,15
print (a)
print (b)
print (c)
```

Python Operators:

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a particular programming language. Python provides a variety of operators described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic operators:

Arithmetic operators are used to perform arithmetic operations between two operands. It includes +(addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**).

Consider the following table for a detailed explanation of arithmetic operators.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value result negative. For example, if a = 20, b = 10 => a - b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

Comparison operator

Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Python assignment operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assign the reminder back to left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

Logical Operators

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
not	If an expression **a** is true then not (a) will be false and vice versa.

Membership Operators:- Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Identity Operators

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.

is not	It is evaluated to be true if the reference present at both side do not point to the same object.

Python Data Types:

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

```
A=20
b="Hi Python developers…"
c = 10.45
print(type(a));
print(type(b));
print(type(c));
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

In this section of the tutorial, we will give a brief introduction of the above data types. We will discuss each one of them in detail later in this tutorial.

Numbers

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

a = 3 , b = 5 #a and b are number objects

Python supports 4 types of numeric data.

1. int (signed integers like 10, 2, 29, etc.)
2. long (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
3. float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
4. complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.

A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts respectively).

String

The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single, double, or triple quotes to define a string.

String handling in python is a straightforward task since there are various inbuilt functions and operators provided.

In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.

The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

The following example illustrates the string handling in python.

```
str1 = 'hello java' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

Output:

```
he
o
hello javahello java
hello java how are you
```

List

Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Consider the following example.

```
l  = [1, "hi", "python", 2]
print (l[3:]);
print (l[0:2]);
print (l);
print (l + l);
print (l * 3);
```

Output:

```
[2]
[1, 'hi']
[1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```

Tuple

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Let's see a simple example of the tuple.

```
t  = ("hi", "python", 2)
print (t[1:]);
print (t[0:1]);
print (t);
print (t + t);
print (t * 3);
print (type(t))
t[2] = "hi";
```

Output:

```
('python', 2)
('hi',)
('hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)
<type 'tuple'>
Traceback (most recent call last):
  File "main.py", line 8, in <module>
    t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary

Dictionary is an ordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

Consider the following example.

```
d = {1:'Amol', 2:'Arun', 3:'Shaym', 4:'Mahesh'};
print("1st name is "+d[1]);
print("2nd name is "+ d[4]);
print (d);
print (d.keys());
print (d.values());
```

Output:

```
1st name is Amol
2nd name is Mahesh
{1: 'Amol', 2: 'Arun', 3: 'Shaym', 4: 'Mahesh'}
[1, 2, 3, 4]
['Amol', 'Arun', 'Shaym', 'Mahesh']
```

Python String:-

In python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

Consider the following example in python to create a string.

str = "Hi Python !"

Here, if we check the type of the variable str using a python script

print(type(str)), then it will **print** string (str).

In python, strings are treated as the sequence of strings which means that python doesn't support the character data type instead a single character written as 'p' is treated as the string of length 1.

String Operators:

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

Built-in String functions:

1. ***capitalize () Method:-*** Python **capitalize()** method converts first character of the string into uppercase without altering the whole string. It changes the first character only and skips rest of the string unchanged.

Ex:

```
str = "python"
# Calling function
str2 = str.capitalize()
# Displaying result
print("Old value:", str)
print("New value:", str2)
```

2. ***Center() Method:-*** Python **center()** method alligns string to the center by filling paddings left and right of the string. This method takes two parameters, first is a width and second is a fillchar which is optional. The fillchar is a character which is used to fill left and right padding of the string.

Ex:

```
# Python center () function example
# Variable declaration
str = "Hello Python"
# Calling function
str2 = str.center(20,'#')
# Displaying result
print("Old value:", str)
print("New value:", str2)
```

3. ***Count() Method:-*** It returns the number of occurrences of substring in the specified range. It takes three parameters, first is a substring, second a start index and third is last index of the range. Start and end both are optional whereas substring is required.

Ex:

```
str = "Hello Javatpoint"
str2 = str.count('t')
# Displaying result
print("occurences:", str2)
```

4. *endswith() Method:-*

Python **endswith()** method returns true of the string ends with the specified substring, otherwise returns false.

Signature

endswith(suffix[, start[, end]])

Parameters

- **suffix** : a substring
- **start** : start index of a range
- end : last index of the range

Ex:-

```
str = "Hello this is javatpoint."
isends = str.endswith(".")
# displaying result
print(isends)
```

5. ***format() Method:-*** Python format() method is used to perform format operations on string. While formatting string a delimiter {} (braces) is used to replace it with the value. This delimiter either can contain index or positional argument.

Ex:

```
str = "Java"
str2 = "C#"
# Calling function
str3 = "{} and {} both are programming languages".format(str,str2)
# displaying result
print(str3)
```

Output:

```
Java and C# both are programming languages
```

6. ***find() Method:-*** Python **find()** method finds substring in the whole string and returns index of the first match. It returns -1 if substring does not match**.**

```
str = "Welcome to the python."
# Calling function
str2 = str.find("the")
# Displaying result
print(str2)
```

7. ***istitle() Method:*** Python istitle() method returns True if the string is a titlecased string. Otherwise returns False.

```
str = "Welcome To Python."
# Calling function
str2 = str.istitle()
# Displaying result
print(str2)
```

Python Loops:-

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, the programming languages provide various types of loops which are capable of repeating some specific code several numbers of times.

Why we use loops in python?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantages of loops

There are the following advantages of loops in Python.

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

Python for loop

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

```
for iterating_var in sequence:
    statement(s)
```

Example

```
i=1
n=int(input("Enter the number up to which you want to print the natural numbers?"))
for i in range(0,10):
    print(i,end = ' ')
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Python for loop example: printing the table of the given number

```
i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
   print("%d X %d = %d"%(num,i,num*i));
```

Output:

```
Enter a number:10
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

Nested for loop in python

Python allows us to nest any number of for loops inside a for loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax of the nested for loop in python is given below.

```
for iterating_var1 in sequence:
    for iterating_var2 in sequence:
      #block of statements
#Other statements
```

Example 1

```
n = int(input("Enter the number of rows you want to print?"))
i,j=0,0
for i in range(0,n):
   print()
   for j in range(0,i+1):
     print("*",end="")
```

Output:

```
Enter the number of rows you want to print?5
*
**
***
****
*****
```

Python while loop

The while loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed as long as the given condition is true.

It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

The syntax is given below.

```
while expression:
    statements
```

Here, the statements can be a single statement or the group of statements. The expression should be any valid python expression resulting into true or false. The true is any non-zero value.

Example 1

```
i=1;
while i<=10:
    print(i);
    i=i+1;
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Python If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

Indentation in Python

For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation. We will see how the actual indentation takes place in decision making and other stuff in python.

The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

The syntax of the if-statement is given below.

```
if expression:
    statement
```

Example 1

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even")
```

Output

```
enter the number?10
Number is even
```

Example 2: Program to print the largest of the three numbers.

```
a = int(input("Enter a? "));
b = int(input("Enter b? "));
c = int(input("Enter c? "));
if a>b and a>c:
    print("a is largest");
if b>a and b>c:
    print("b is largest");
if c>a and c>b:
    print("c is largest");
```

Output:

```
Enter a? 100
Enter b? 120
Enter c? 130
c is largest
```

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

The syntax of the if-else statement is given below.

```
if condition:
    #block of statements
else:
    #another block of statements (else-block)
```

Example 1: Program to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age? "))
if age>=18:
   print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
```

Output:

```
Enter your age? 90
You are eligible to vote !!
```

Example 2: Program to check whether a number is even or not.

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even...")
else:
    print("Number is odd...")
```

Output:

```
enter the number?10
Number is even
```

The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

```
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
    # block of statements

else:
    # block of statements
```

Example 1

```
number = int(input("Enter the number?"))
if number==10:
    print("number is equals to 10")
elif number==50:
    print("number is equal to 50");
elif number==100:
    print("number is equal to 100");
else:
    print("number is not equal to 10, 50 or 100");
```

Output:

```
Enter the number?15
number is not equal to 10, 50 or 100
```