# UNIT II
# DATA TYPES, EXPRESSIONS, STATEMENTS

1. Python interpreter and interactive mode,debugging;
2. Values and types: int, float, boolean, string, and list;
3. Variables
   3.1 Expressions
   3.2 Statements
   3.3 Tuple assignment
   3.4 Precedence of operators
   3.5 Comments
4. Illustrative programs:
   4.1 Exchange the values of two variables
   4.2 Circulate the values of n variables
   4.3 Distance between two points.

## 1. Python interpreter

- Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, Mac OS X, Linux, Unix
- It is free and open source.

**Starting the Interpreter**

- The Python interpreter is a program that reads and executes Python code.
- After installation, the python interpreter lives in the installed directory.
- Now there are two ways to start Python.
    1. Interactive mode
    2. Script Mode

**Interactive mode**

- Typing python in the command line will invoke the interpreter in interactive mode.

  **>>>** - is a prompt that indicates that the interpreter is ready for you to enter code.

  ```
  >>> 5 + 4
  9
  ```
- This prompt can be used as a calculator. To exit this mode type exit() or quit() and press enter.

**Script Mode**

- This mode is used to execute Python program written in a file.
- Such a file is called a script.
- Python scripts have the extension .py
- For example: helloWorld.py
- To execute this file in script mode we simply write python helloWorld.py at the command prompt.

**bug**: An error in a program.
**debugging**: The process of finding and removing any of the three kinds of programming errors.
**syntax**: The structure of a program.
**syntax error**: An error in a program that makes it impossible to parse (and therefore impossible to interpret).
**exception**: An error that is detected while the program is running.
**semantics**: The meaning of a program.

**2. Values and types**

- A value is one of the basic things a program.
- There are different values integers, float and strings.
- The numbers with a decimal point belong to a type called float.
- The values written in quotes will be considered as string, even it's an integer.
- If type of value is not known it can be interpreted as
  Eg: >>> type('Hello, World!')
  <type 'str'>
  >>> type(17)
  <type 'int'>
  >>> type('17')
  <type 'str'>
  >>> type('3.2')
  <type 'str'>

**Standard Data Types**

- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types −
  - ❖ Numbers
  - ❖ String
  - ❖ List
  - ❖ Tuple
  - ❖ Dictionary

**Python Numbers**

- Number data types store numeric values.
- Number objects are created when you assign a value to them.
- For example –
  var1 =1
  var2 =10
- You can also delete the reference to a number object by using the del statement.
- The syntax of the del statement is −
  del var1[,var2[,var3[....,varN]]]]

**Python Strings**

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.

- Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.
- For example –

```
str = 'Python Programming' print str              # Prints complete string
print str[0]          # Prints first character of the string
print str[-1]         # Prints last character of the string
print str[2:5]        # Prints characters starting from 3rd to 5th
print str[2:]         # Prints string starting from 3rd character
print str * 2         # Prints string two times
print str + " Course"     # Prints concatenated string
```

This will produce the following result –

```
Python Programming
P
g
tho
thon Programmin
Python ProgrammingPython Programming
Python Programming Course
```

**Python Lists**

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C.
- One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.
- For example −

```
list = [ 'Hai', 123 , 1.75, 'vinu', 100.25 ]
smalllist = [251, 'vinu']
print list
print list[0]
print list[-1]
print list[1:3]
print list[2:]
print smalllist * 2
print list + smalllist
```

This produces the following result –

```
['Hai', 123, 1.75, 'vinu', 100.25]
Hai
100.25
```

[123, 1.75]
[1.75, 'vinu', 100.25]
[251, 'vinu', 251, 'vinu']
['Hai', 123, 1.75, 'vinu', 100.25, 251, 'vinu']

**Python Boolean**

- A Boolean type was added to Python 2.3.
- Two new constants were added to the __builtin__ module,
- True and False.
- True and False are simply set to integer values of 1 and 0 and aren't a different type.

```
>>>bool(1)
True
>>>bool(0)
False
>>> False + 1
1
>>> False * 85
0
>>> True * 85
85
>>>True+True
2
>>>False+False
0
```

**3 VARIABLES**

- A variable is a name that refers to a value.
- Variable reserved memory locations to store values.
- This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory

**Assignment Statements**

- An assignment statement creates a new variable and gives it a value:

```
>>>message = 'Introducing Python Variable'
>>>num = 15
>>>radius = 5.4
```

- This example makes three assignments.
- The first assigns a string to a new variable named message;
- the second gives the integer 15 to num;
- the third assigns floating point value 5.4 to variable radius.

**Variable Names**

- Programmers generally choose names for their variables that are meaningful
- **The Rules**
  - ❖ Variables names must start with a letter or an underscore,such as:

    _mark

    mark_

- ❖ The remainder of your variable name may consist of letters, numbers and underscores.
  subject1
  my2ndsubject
  un_der_scores
- ❖ Names are case sensitive.
  case_sensitive, CASE_SENSITIVE, and Case_Sensitive are each a different variable.
- ❖ Can be any (reasonable) length
- ❖ There are some reserved (KeyWords)words which you cannot use as a variable name

- If you give a variable an illegal name, you get a syntax error:
  >>>1book = 'python'
  SyntaxError: invalid syntax
  >>>more@ = 1000000
  SyntaxError: invalid syntax
  >>>class = 'Fundamentals of programming'
  SyntaxError: invalid syntax
- 1book is illegal because it begins with a number.
- more@ is illegal because it contains an illegal character,
- class is illegal because it is a keyword.

**Good Variable Name**

- Choose meaningful name instead of short name.
- roll_no is better than rn.
- Maintain the length of a variable name.
- Roll_no_of_a_student is too long? Be consistent; roll_no or orRollNo
- Begin a variable name with an underscore(_) character for a special case.

**3.1 EXPRESSIONS AND STATEMENTS**

- An expression is a combination of values, variables, and operators.
- A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions:
  >>> 50
  50
  >>> 10<5
  False
  >>> 50+20
  70
- When you type an expression at the prompt, the interpreter evaluates it, which means that it finds the value of the expression.

**3.2 STATEMENT**

- A statement is a unit of code that has an effect, like creating a variable or displaying a value.
  >>> n = 25
  >>>print(n)
- The first line is an assignment statement that gives a value to n.

- The second line is a print statement that displays the value of n.
- When you type a statement, the interpreter executes it, which means that it does whatever the statement says.
- In general, statements don't have values.

**Difference Between a Statement and an Expression**

- A statement is a complete line of code that performs some action, while an expression is any section of the code that evaluates to a value.
- Expressions can be combined —horizontally‖ into larger expressions using operators, while statements can only be combined —vertically‖ by writing one after another, or with block constructs.
- Every expression can be used as a statement, but most statements cannot be used as expressions.

**3.3 TUPLE ASSIGNMENT**

- It is often useful to swap the values of two variables.
- With conventional assignments, you have to use a temporary variable.
- For example, to swap a and b:
  >>>temp = a
  >>> a = b
  >>> b = temp
  >>>a, b = b, a
- This is called tuple assignment
- The left side is a tuple of variables; the right side is a tuple of expressions.
- Each value is assigned to its respective variable.
- All the expressions on the right side are evaluated before any of the assignments.
- The number of variables on the left and the number of values on the right have to be the same.
  >>>a, b = 1, 2, 3
  ValueError: too many values to unpack
- More generally, the right side can be any kind of sequence (string, list or tuple).
- For example, to split an email address into a user name and a domain, you could write:
  >>>addr = 'monty@python.org'
  >>>uname, domain = addr.split('@')
  The return value from split is a list with two elements; the first element is assigned to uname, the second to domain.
  >>>uname 'monty'
  >>>domain 'python.org'

**3.4 OPERATORS**

- Operators are special symbols in Python that carry out computation.
- The value that the operator operates on is called the operand.
  For example:
  >>>10+5
  15
- Here, + is the operator that performs addition.

- 10 and 5 are the operands and 15 is the output of the operation.
- Python has a number of operators which are classified below.
    1) Arithmetic operators
    2) Comparison (Relational) operators
    3) Logical (Boolean) operators
    4) Bitwise operators
    5) Assignment operators
    6) Special operators

**i) Arithmetic Operators**

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

  Example

```
x = 7
y = 3
print('x + y =',x+y)
print('x - y =',x-y)
print('x * y =',x*y)
print('x / y =',x/y)
print('x // y =',x//y)
print('x % y =',x%y)
print('x ** y =',x**y)
```

  When you run the program, the output will be:

```
x + y = 10
x - y = 4
x * y = 21
x / y = 2.3333333333333335
x // y = 2
x % y = 1
x ** y = 343
```

| Operator | Meaning | Example |
|---|---|---|
| + | Add two operands or unary plus | x + y<br>+2 |
| - | Subtract right operand from the left or unary minus | x - y<br>-2 |
| * | Multiply two operands | x * y |
| / | Divide left operand by the right one (always results into float) | x / y |
| % | Modulus - remainder of the division of left operand by the right | x % y (remainder of x/y) |
| // | Floor division - division that results into whole number adjusted to the left in the number line | x // y |
| ** | Exponent - left operand raised to the power of right | x**y (x to the power y) |

**ii) Comparison or Relational Operators**

- Comparison operators are used to compare values. It either returns True or False according to the condition.

| Operator | Meaning | Example |
|---|---|---|
| > | Greater that - True if left operand is greater than the right | x > y |
| < | Less that - True if left operand is less than the right | x < y |
| == | Equal to - True if both operands are equal | x == y |
| != | Not equal to - True if operands are not equal | x != y |
| >= | Greater than or equal to - True if left operand is greater than or equal to the right | x >= y |
| <= | Less than or equal to - True if left operand is less than or equal to the right | x <= y |

**Example**

```
x = 5
y = 7
print('x > y is',x>y)
print('x < y is',x<y)
print('x == y is',x==y)
print('x != y is',x!=y)
print('x >= y is',x>=y)
print('x <= y is',x<=y)
```

When you run the program, the output will be:

```
x >y is False
x <y is True
x == y is False
x != y is True
x >= y is False
x <= y is True
```

**iii)Logical Operators**

- Logical operators are the and, or, not operators.

| Operator | Meaning | Example |
|---|---|---|
| and | True if both the operands are true | x and y |
| or | True if either of the operands is true | x or y |
| not | True if operand is false (complements the operand) | not x |

**Example**

```
x = True
y = False
print('x and y is',x and y)
print('x or y is',x or y)
print('not x is',not x)
```

When you run the program, the output will be:

```
x and y is False
x or y is True
not x is False
```

**iv) Bitwise Operators**

- Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.

  For example, 2 is 10 in binary and 7 is 111.

| Operator | Meaning | Example |
|---|---|---|
| & | Bitwise AND | x& y = 0 (0000 0000) |
| \| | Bitwise OR | x \| y = 14 (0000 1110) |
| ~ | Bitwise NOT | ~x = -11 (1111 0101) |
| ^ | Bitwise XOR | x ^ y = 14 (0000 1110) |
| >> | Bitwise right shift | x>> 2 = 2 (0000 0010) |
| << | Bitwise left shift | x<< 2 = 40 (0010 1000) |

**Example**

```
x=10
y=4
print('x& y=',x& y)
print('x | y=',x | y)
print('~x=',~x)
print('x ^ y=',x ^ y)
print('x>> 2=',x>> 2)
print('x<< 2=',x<< 2)
```

When you run the program, the output will be:

```
x& y= 0
x | y= 14
~x= -11
x ^ y= 14
x>> 2= 2
x<< 2= 40
```

**v) Assignment Operators**

- Assignment operators are used in Python to assign values to variables.
- a=10 assigns the value 10 on the right side to the variable a on the left.
- There are various compound operators in Python like a += 10 that adds to the variable and later assigns the same. It is equivalent to a = a + 10.

| Operator | Example | Equivatent to |
|---|---|---|
| = | x = 5 | x = 5 |
| += | x += 5 | x = x + 5 |
| -= | x -= 5 | x = x - 5 |
| *= | x *= 5 | x = x * 5 |
| /= | x /= 5 | x = x / 5 |
| %= | x %= 5 | x = x % 5 |
| //= | x //= 5 | x = x // 5 |
| **= | x **= 5 | x = x ** 5 |
| &= | x &= 5 | x = x & 5 |
| \|= | x \|= 5 | x = x \| 5 |
| ^= | x ^= 5 | x = x ^ 5 |
| >>= | x >>= 5 | x = x >> 5 |
| <<= | x <<= 5 | x = x << 5 |

**vi) Special Operators**

- Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

**a) Identity Operators**

- **is** and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory.
- Two variables that are equal does not imply that they are identical.

| Operator | Meaning | Example |
|---|---|---|
| is | True if the operands are identical (refer to the same object) | x is True |
| is not | True if the operands are not identical (do not refer to the same object) | x is not True |

**Example**

```
x1 = 7
y1 = 7
x2 = 'Welcome'
y2 = 'Welcome'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is not y1)
print(x2 is y2)
print(x3 is y3)
```

When you run the program, the output will be:

```
False
True
False
```

**b) Membership Operators**

- in and in not are the membership operator.
- They are used to test whether a value or a variable is found in a sequence ( string, list, tuple, set and dictionary)

| Operator | Meaning | Example |
|---|---|---|
| in | True if value/variable is found in the sequence | 5 in x |
| not in | True if value/variable is not found in the sequence | 5 not in x |

**Example**

```
x = 'Python Programming'
print('Program' not in x)
print('Program' in x)
print('program' in x)
```

When you run the program, the output will be:

```
False
```

True
False

- Here, ' Program ' is in x but ' program' is not present in x, since Python is case sensitive

### 3.4.1 PRECEDENCE OF PYTHON OPERATORS

- The combination of values, variables, operators and function calls is termed as an expression.
- Python interpreter can evaluate a valid expression.
- When an expression contains more than one operator, the order of evaluation depends on the Precedence of operations.

For example, multiplication has higher precedence than subtraction.

```
>>> 20 – 5*3
5
```

But we can change this order using parentheses () as it has higher precedence.

```
>>> (20 - 5) *3
45
```

- The operator precedence in Python are listed in the following table. It is in descending order, upper group has higher

| Operators | Meaning |
|---|---|
| () | Parentheses |
| ** | Exponent |
| +x, -x, ~x | Unary plus, Unary minus, Bitwise NOT |
| *, /, //, % | Multiplication, Division, Floor division, Modulus |
| +, - | Addition, Subtraction |
| <<, >> | Bitwise shift operators |
| & | Bitwise AND |
| ^ | Bitwise XOR |
| \| | Bitwise OR |
| ==, !=, >, >=, <, <=, is, is not, in, not in | Comparisions, Identity, Membership operators |
| not | Logical NOT |
| and | Logical AND |
| or | Logical OR |

### 3.4.2 ASSOCIATIVITY OF PYTHON OPERATORS

- We can see in the above table that more than one operator exists in the same group.
- These operators have the same precedence.
- When two operators have the same precedence, associativity helps to determine which the order of operations.
- Associativity is the order in which an expression is evaluated that has multiple operator of the same precedence.
- Almost all the operators have left-to-right associativity.

  For example, multiplication and floor division have the same precedence.
  Hence, if both of them are present in an expression, left one is evaluates first.

```
>>> 10 * 7 // 3
23
>>> 10 * (7//3)
20
>>> (10 * 7)//3
23
```

We can see that 10 * 7 // 3 is equivalent to (10 * 7)//3.
Exponent operator ** has right-to-left associativity in Python.

```
>>> 5 ** 2 ** 3
390625
>>> (5** 2) **3
15625
>>> 5 **(2 **3)
390625
```

We can see that 2 ** 3 ** 2 is equivalent to 2 ** (3 ** 2).

**3.5 COMMENTS**

- As programs get bigger and more complicated, they get more difficult to read.
- Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why.
- For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing.
- These notes are called comments, and they start with the # symbol:

```
# compute Area of a triangle using Base and Height
area= (base*height)/2
```

- In this case, the comment appears on a line by itself. You can also put comments at the end of a line:

```
area= (base*height)/2 # Area of a triangle using Base and Height
```

- Everything from the # to the end of the line is ignored—it has no effect on the execution of the program.
- Comments are most useful when they document non-obvious features of the code.
- If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example:

```
#This is a long comment
#and it extends
#to multiple lines
```

- Another way of doing this is to use triple quotes, either ''' or """.

```
"""This is also a
perfect example of
multi-line comments"""
```

## 4. ILLUSTRATIVE PROBLEMS

### 4.1 EXCHANGE THE VALUES OF TWO VARIABLES

- In python exchanging the values can be done in three ways
    - i) Using third variable
    - ii) Using tuple assignment method
    - iii) Using arithmetic operator
    - iv) Using bitwise operator

#### 4.1.1 Using third variable

```
1 var1 = input("Enter value of variable1: ")
2 var2 = input("Enter value of variable2: ")
3 temp = var1
4 var1 = var2
5 var2 = temp
6 print("After swapping:")
7 print("First Variable =",var1,)
8 print("Second Variable=",var2,)
```

When you run the program, the output will be:

```
Enter value of variable1: 5
Enter value of variable2: 10
After swapping:
First Variable = 10
Second Variable= 5
```

#### 4.1.2 Using tuple assignment

- In this method instead of using the line number 3,4 and 5 we can just code var1,var2=var2,var1

#### 4.1.3 Using arithmetic operator

##### 4.1.3.1 Addition and Subtraction

- In this method in the above program instead of line number 3,4,5 use the following code

```
x = x + y
y = x - y
x = x - y
```

##### 4.1.3.2 Multiplication and Division

- In this method in the above program instead of line number 3,4,5 use the following code

```
x = x * y
y = x / y
x = x / y
```

### 4.1.4 Using Bitwise Operator

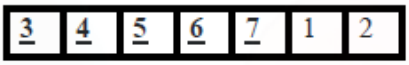
- If the variables are integers then we can perform swapping with the help of bitwise XOR operator.
- In order to do this in the above program instead of line number 3,4,5 use the following code

```
x = x ^ y
y = x ^ y
x = x ^ y
```

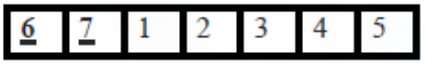
### 4.2 CIRCULATE THE VALUE OF N VARIABLES

- Problem of circulating a Python list by an arbitrary number of items to the right or left can be easily performed by List slicing operator.


- Consider the above list Figure 2.4.a; circulation of the above list by n position can be easily achieved by slicing the array into two and concatenating them.
- Slicing is done as nth element to end element + beginning element to n-1th element. Suppose n=2 means, given list is rotated 2 positions towards left side




- Suppose n= - 2 means, given list is rotated 2 position towards right side




- So the simple function to perform this circulation operation is

```
def circulate(list, n):
return list[n:] + list[:n]
>>> circulate([1,2,3,4,5,6,7], 2)
[3, 4, 5, 6, 7, 1, 2]
>>> circulate([1,2,3,4,5,6,7], -2)
[6, 7, 1, 2, 3, 4, 5]
```

### 4.3 DISTANCE BETWEEN TWO POINT

```
Import math
p1=[4,0]
p2=[6,6]
distance=math.sqrt((p1[0]-p2[0]**2)+p1[1]-p2[1]**2))
print(distance)
```

**Output**
**Distance between two points**
**6.3245532**