# UNIT–I

**Objective:**

To explore basic knowledge on Python language basics features.

**Syllabus:**

**Basics of Python Programming and Control Statements:**

Features and history of python, literal constants, data types, variables, operators, operator precedence, expressions, type conversion, command line arguments, input and output operation.

**Conditional Statements:** simple-if, if-else, nested-if and if-elif-else.

**Iterative Statements:** while, for and else with for and while.

**un-conditional branching:** break, continue and pass statement.

**Learning Outcomes:**

At the end of the unit student will be able to

- understand the features and history of python.
- describe python tokens.
- solve simple formula based problems on computer using Python.
- understand the conditional and unconditional Statement.
- describe conditional and unconditional Statements, Decision Control Statements .
- adapt and combine standard algorithms to solve a given problem using decision control and looping statements.
- implement programs using python control Structure.

**LEARNING MATERIAL**

1. **Features and History of Python:**

   **1.1Features of Python**

- **Simple:** Reading a program written in Python feels almost like reading english. The main strength of Python which allows programmer to concentrate on the solution to the problem rather than language itself.
- **Easy to Learn:** Python program is clearly defined and easily readable. The structure of the program is simple. It uses few keywords and clearly defined syntax.
- **Versatile:** Python supports development of wide range of applications such as simple text processing, WWW browsers and  games etc..
- **Free and Open Source:** It is a Open Source Software. So, anyone can freely distribute it, read the source code, edit it, and even use the code to write new (free) programs.
- **High-level Language:** While writing programs in Python we do not worry about the low-level details like managing memory used by the program.
- **Interactive:** Programs in Python work in interactive mode which allows interactive testing and debugging of pieces of code. Programmer can easily interact with the interpreter directly at the python prompt to write their programs.
- **Portable**: It is a portable language and hence the programs behave the same on wide variety of hardware platforms with different operating systems**.**
- **Object Oriented:** Python supports object-oriented as well as procedure-oriented style of programming .While object-oriented technique encapsulates data and functionality with in objects, Procedure oriented at other hand, builds programs around procedure or functions.
- **Interpreted:** Python is processed at runtime by interpreter. So, there is no need to compile a program before executing it. You can simply run the program. Basically python converts source program into intermediate form called byte code.
- **Dynamic and strongly typed language:** Python is strongly typed as the interpreter keeps track of all variables types. It's also very dynamic as it rarely uses what it knows to limit variable usage.

- **Extensible:** Since Python is an open source software, anyone can add low-level modules to the python interpreter. These modules enable programmers to add to or customize their tools to work more efficiently.
- **Embeddable:** Programmers can embed Python within their C, C++, COM, ActiveX, CORBA and Java Programs to give 'scripting 'capability for users.
- **Extensive Libraries:** Python has huge set of libraries that is easily portable across different platforms with different operating systems.
- **Easy maintenance:** Code Written in Python is easy to maintain.
- **Secure:** This Programming language is secure for tampering. Modules can be distributed to prevent altering of source code. Additionally, Security checks can be easily added to implement additional security features.
- **Robust:** Python Programmers cannot manipulate memory directly, errors are raised as exceptions that can be catch and handled by the program code. For every syntactical mistake, a simple and easy to interpret message is displayed. All these make python robust.
- **Multi-threaded:** Python supports executing more than one process of a program simultaneously with the help of Multi Threading.
- **Garbage Collection:** The Python run-time environment handles garbage collection of all python objects. For this, a reference counter is maintained to assure that no object that is currently in use is deleted.

## 1.2 History of Python.

- Python was first developed by ***Guido van Rossum*** in the late 80's and early 90's at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- It has been derived from many languages such as ABC, Modula-3, C, C++, Algol-68, SmallTalk, UNIX shell and other scripting languages.
- Since early 90's Python has been improved tremendously. Its version 1.0 was released in 1991, which introduced several new functional programming tools.
- While version 2.0included list comprehension was released in 2000 by the Be Open Python Labs team.
- Python 2.7 which is still used today will be supported till 2020.

- Currently Python 3.6.4 is already available. The newer versions have better features like flexible string representation e.t.c,
- Although Python is copyrighted, its source code is available under GNU General Public License (GPL) like that Perl.
- Python is currently maintained by a core development team at the institute which is directed by Guido Van Rossum.
- These days, from data to web development, Python has emerged as very powerful and popular language. It would be surprising to know that python is actually older than Java, R and JavaScript.

## 1.3 Applications of Python:

- **Embedded scripting language**: Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.
- **3D Software**: 3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.
- **Web development**: Python is an easily extensible language that provides good integration with database and other web standards.
- **GUI-based desktop applications***:* Simple syntax, modular architecture, rich text processing tools and the ability to work on multiple operating systems makes Python a preferred choice for developing desktop-based applications.
- **Image processing and graphic design applications***:* Python is used to make 2D imaging software such as Inkscape, GIMP, Paint Shop Pro and Scribus. It is also used to make 3D animation packages, like Blender, 3ds Max, Cinema 4D, Houdini, Light wave and Maya.
- **Scientific and Computational applications***:* Features like high speed, productivity and availability of tools, such as Scientific Python and Numeric Python, have made Python a preferred language to perform computation and processing of scientific data. 3D modeling software, such as FreeCAD, and finite element method software, like Abaqus, are coded in Python.
- **Games:** Python has various modules, libraries, and platforms that support development of games. Games like Civilization-IV, Disney's Toontown Online, Vega Strike, etc. are coded using Python.

- **Enterprise and Business applications:** Simple and reliable syntax, modules and libraries, extensibility, scalability together make Python a suitable coding language for customizing larger applications. For example, Reddit which was originally written in Common Lips, was rewritten in Python in 2005. A large part of Youtube code is also written in Python.
- **Operating Systems:** Python forms an integral part of Linux distributions.

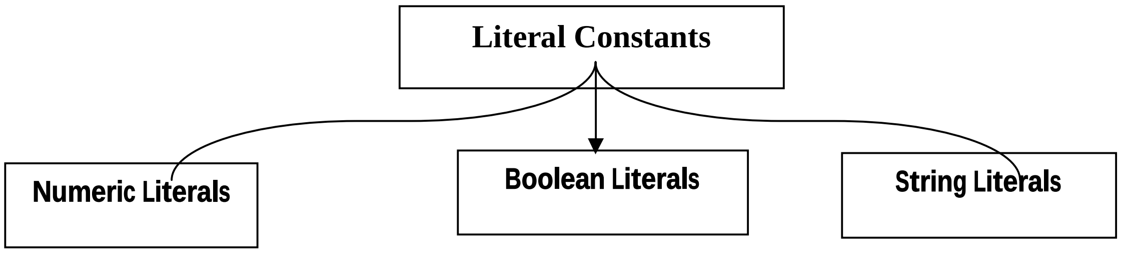
## 1.4 Keyword in Python:

- Keywords are the reserved words in Python. We cannot use a keyword as variable name, function name or any other identifier.
- Here's a list of all keywords in Python Programming.
- There are 33 keywords in Python 3.3. This number can vary slightly in course of time.
- All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

| **Keywords in Python Programming Language** | | | | |
|---|---|---|---|---|
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass | |
| break | except | in | raise | |

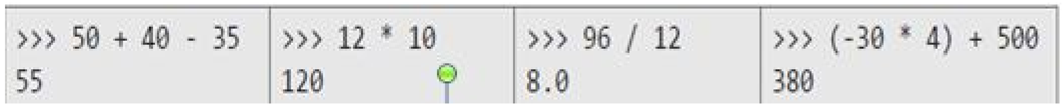
## 2. Literal Constants

- In programming constants are referred to variables that cannot be changed.
- Generally Literal constants are classified in to three types.

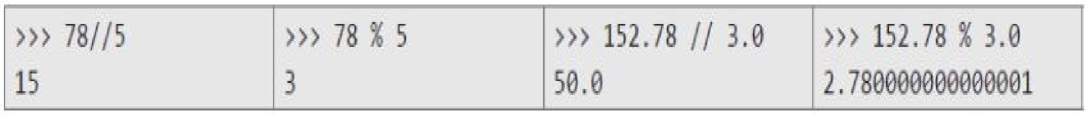


### 2.1 Numeric Literals

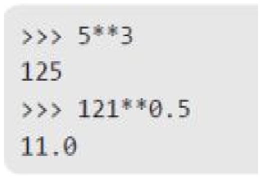
- The value of a literal constant can be used directly in programs. For example, 7, 3.9, 'A', and "Hello" are literal constants.
- Numbers refers to a numeric value. You can use four types of numbers in Python program- **integers, long integers, floating point and complex numbers.**
- Numbers like 5 or other whole numbers are referred to as ***integers*.** Bigger whole numbers are called ***long integers***. For example, 535633629843L is a long integer.
- Numbers like are 3.23 and 91.5E-2 are termed as ***floating point numbers***.
- Numbers of a + bj form (like -3 + 7j) **are *complex numbers***.



```
>>> 50 + 40 - 35
55
>>> 12 * 10
120
>>> 96 / 12
8.0
>>> (-30 * 4) + 500
380
```



```
>>> 78//5
15
>>> 78 % 5
3
>>> 152.78 // 3.0
50.0
>>> 152.78 % 3.0
2.780000000000001
```

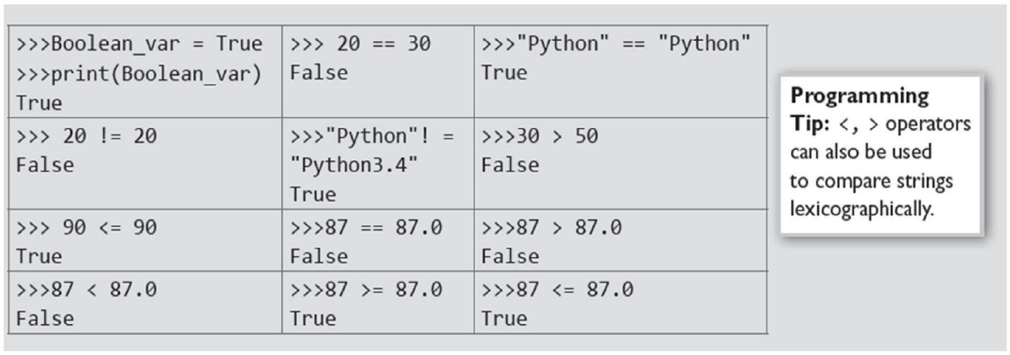


```
>>> 5**3
125
>>> 121**0.5
11.0
```

### 2.2 Boolean Literals

- A Literals Boolean type can have one of the two values- True or False.

Examples:



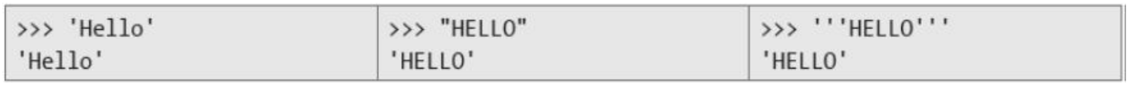
| | | |
|---|---|---|
| `>>>Boolean_var = True`<br>`>>>print(Boolean_var)`<br>`True` | `>>> 20 == 30`<br>`False` | `>>>"Python" == "Python"`<br>`True` |
| `>>> 20 != 20`<br>`False` | `>>>"Python"! =`<br>`"Python3.4"`<br>`True` | `>>>30 > 50`<br>`False` |
| `>>> 90 <= 90`<br>`True` | `>>>87 == 87.0`<br>`False` | `>>>87 > 87.0`<br>`False` |
| `>>>87 < 87.0`<br>`False` | `>>>87 >= 87.0`<br>`True` | `>>>87 <= 87.0`<br>`True` |

**Programming Tip:** <, > operators can also be used to compare strings lexicographically.

**2.3 String Literals**

- A *string* is a group of characters.
- *Using Single Quotes ('):* For example, a string can be written as 'HELLO'.
- *Using Double Quotes ("):* Strings in double quotes are exactly same as those in single quotes. Therefore, 'HELLO' is same as "HELLO".
- *Using Triple Quotes (''' '''):* You can specify multi-line strings using triple quotes. You can use as many single quotes and double quotes as you want in a string within triple quotes.

**Examples:**



| | | |
|---|---|---|
| `>>> 'Hello'`<br>`'Hello'` | `>>> "HELLO"`<br>`'HELLO'` | `>>> '''HELLO'''`<br>`'HELLO'` |

**2.3.1Unicode Strings**

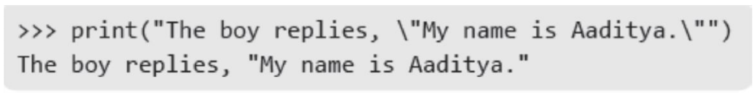
- Unicode is a standard way of writing international text. That is,if you want to write some text in your native language like hindi,then you need to have a Unicode-enable text editor.
- Python allows you to specify Unicode text by prefixing the string with a u or U.
- For Example: u”Sample Unicode string”

  **Note** :The ‘U’ prefix specifies that the file contains text written in language other than English

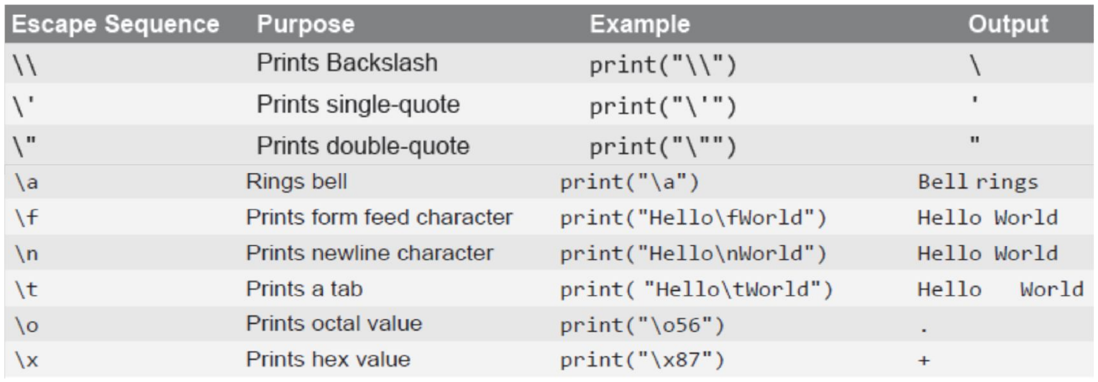
**2.3.2Escape Sequences**

- Some characters (like ", \) cannot be directly included in a string. Such characters must be escaped by placing a backslash before them.

Example:



```
>>> print("The boy replies, \"My name is Aaditya.\"")
The boy replies, "My name is Aaditya."
```

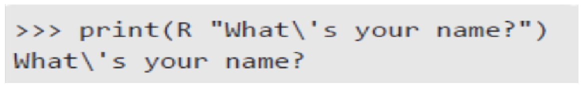


| Escape Sequence | Purpose | Example | Output |
|---|---|---|---|
| `\\` | Prints Backslash | `print("\\")` | `\` |
| `\'` | Prints single-quote | `print("\'")` | `'` |
| `\"` | Prints double-quote | `print("\"")` | `"` |
| `\a` | Rings bell | `print("\a")` | Bell rings |
| `\f` | Prints form feed character | `print("Hello\fWorld")` | Hello World |
| `\n` | Prints newline character | `print("Hello\nWorld")` | Hello World |
| `\t` | Prints a tab | `print( "Hello\tWorld")` | Hello   World |
| `\o` | Prints octal value | `print("\o56")` | . |
| `\x` | Prints hex value | `print("\x87")` | + |

### 2.3.2Raw Strings

- If you want to specify a string that should not handle any escape sequences and want to display exactly as specified then you need to specify that string as a *raw string*. A raw string is specified by prefixing r or R to the string.

Example:



```
>>> print(R "What\'s your name?")
What\'s your name?
```

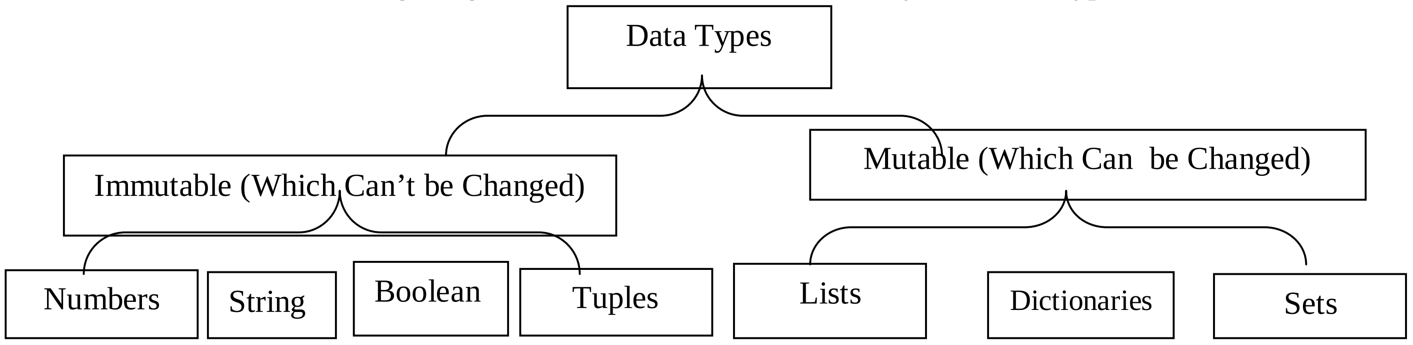
## 3.Data types

- The variables can hold values of different type called ***Data Type***.
- Data type is a set of values and the allowable operations on those values.
- Python has a great set of useful data types. Python's data types are built in the core of the language. They are easy to use and straightforward.
- Example a person age is stored in a number ,his name is made only with characters, and his address is made with mixture of numbers and characters.
- Python ha various standard data types to define the operations possible on them and storage method for each of them.
- Python supports the following five standard data types

  1.Numbers
  2.Strings
  3.Lists
  4.Tuple
  5.Dictionary

**Note:** Python is pure object oriented programming language.it refers to everything as an object including numbers and strings.

➢ The Following Diagram shows the classification of Python Data Types.


Data Types
Immutable (Which Can't be Changed)
Mutable (Which Can be Changed)
Numbers
String
Boolean
Tuples
Lists
Dictionaries
Sets

### 3.1.1. Assigning or Initializing Values to Variables

➢ In Python, programmers need not explicitly declare variables to reserve memory space. The declaration is done automatically when a value is assigned to the variable using the equal sign (=). The operand on the left side of equal sign is the name of the variable and the operand on its right side is the value to be stored in that variable. **Example:** Program to display data of different types using variables and literal constants.

```
num = 7
amt = 123.45
code = 'A'
pi = 3.1415926536
population_of_India = 10000000000
msg = "Hi"

print("NUM = "+str(num))
print("\n AMT = "+ str(amt))
print("\n CODE = " + str(code))
print("\n POPULATION OF INDIA = " + str(population_of_India))
print("\n MESSAGE = "+str(msg))
```

**OUTPUT**

```
NUM = 7
AMT = 123.45
CODE = A
POPULATION OF INDIA = 10000000000
MESSAGE = Hi
```

➢ In Python , you can reassign variables as many times as you want to change the value stored in them. You may even store value of one data type in a statement and other data type in subsequent statement. This is possible because Python variables do not have specific types, i.e., we can assign integer to the variable, later we assign string to the same variable.

**Example:**Program to reassign value to a variable

val = 'Hello'

print(val)

```
val = 100
print(val)
val=10.32
print(val)
```

**Output**

```
Hello
100
10.32
```

### 3.1.2Multiple Assignments

- Python allows programmers to assign single value to more than one variable simultaneously.
- For example

  >>>sum = flag = a = b = 0
- In the above statement, all four integer variables are assigned a value 0.You can also assign different values to multiple variables simultaneously as shown below
- For example

  >>>sum, a, b, mesg = 0, 3, 5, “Result”

  Here, variable sum,a and b are integers(numbers) and mesg is assigned “Result”.

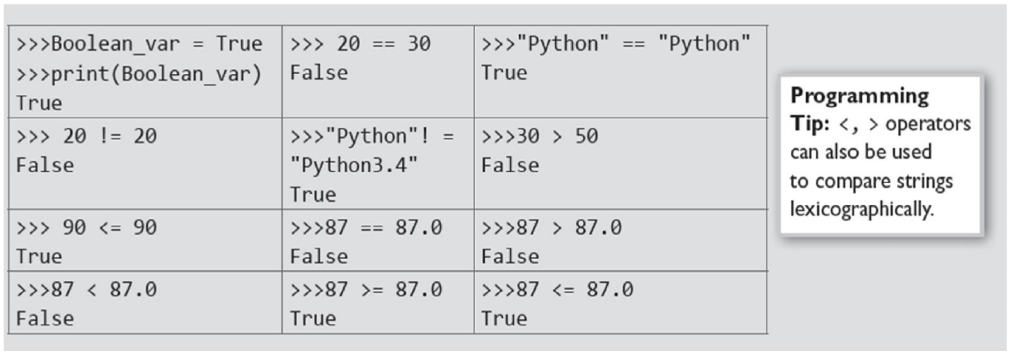
**Note**: Removing a variable means that the reference from the name to the value has been deleted.However, deleted variables can be used again in the code if and only if you reassign them some value.

## 3.2 Boolean Type

A variable of Boolean type can have one of the two values- True or False.

Similar to other variables, the Boolean variables are also created while we assign a value to them or when we use a relational operator on them**.**

Examples:



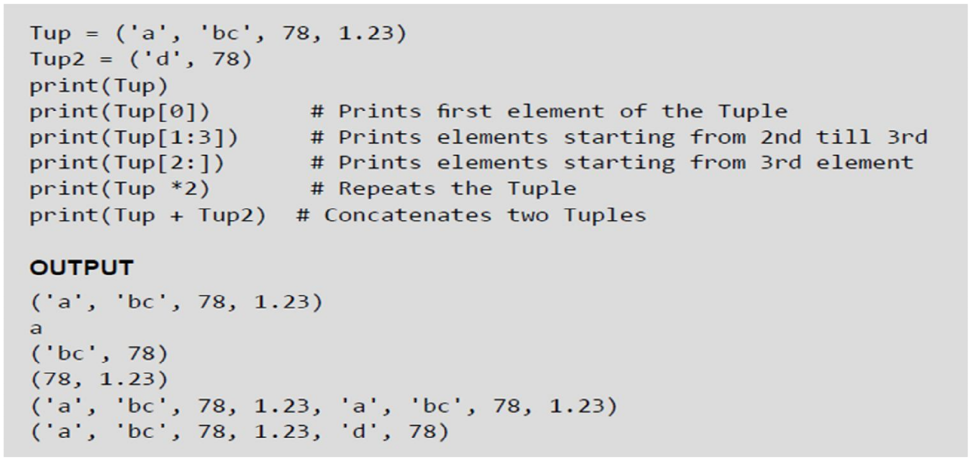
| | | |
|---|---|---|
| >>>Boolean_var = True<br>>>>print(Boolean_var)<br>True | >>> 20 == 30<br>False | >>>"Python" == "Python"<br>True |
| >>> 20 != 20<br>False | >>>"Python"! =<br>"Python3.4"<br>True | >>>30 > 50<br>False |
| >>> 90 <= 90<br>True | >>>87 == 87.0<br>False | >>>87 > 87.0<br>False |
| >>>87 < 87.0<br>False | >>>87 >= 87.0<br>True | >>>87 <= 87.0<br>True |

**Programming Tip:** <, > operators can also be used to compare strings lexicographically.

## 3.3 Tuples

- A tuple is similar to the list as it also consists of a number of values separated by commas and enclosed within parentheses.
- The main difference between lists and tuples is that you can change the values in a list but not in a tuple. This means that while tuple is a read only data type, the list is not.

Examples:



```
Tup = ('a', 'bc', 78, 1.23)
Tup2 = ('d', 78)
print(Tup)
print(Tup[0])       # Prints first element of the Tuple
print(Tup[1:3])     # Prints elements starting from 2nd till 3rd
print(Tup[2:])      # Prints elements starting from 3rd element
print(Tup *2)       # Repeats the Tuple
print(Tup + Tup2)  # Concatenates two Tuples
```

**OUTPUT**

```
('a', 'bc', 78, 1.23)
a
('bc', 78)
(78, 1.23)
('a', 'bc', 78, 1.23, 'a', 'bc', 78, 1.23)
('a', 'bc', 78, 1.23, 'd', 78)
```

## 3.4 Lists

- Lists are the most versatile data type of Python language.
- A list consist of items separated by commas and enclosed within square brackets The values stored in a list are accessed using indexes.
- The index of the first element being 0 and n-1 as that of the last element, where n is the total number of elements in the list. Like strings, you can also use the slice, concatenation and repetition operations on lists.
- Example program to demonstrate operations on lists

```
list = ['a', 'bc', 78, 1.23]
list1 = ['d', 78]
print(list)
print(list[0])
print(list[1:3])
print(list[2:])
print(list * 2)
print(list + list1)
```

**Output:**

['a', 'bc', 78, 1.23]

a

['bc', 78]

[78, 1.23]

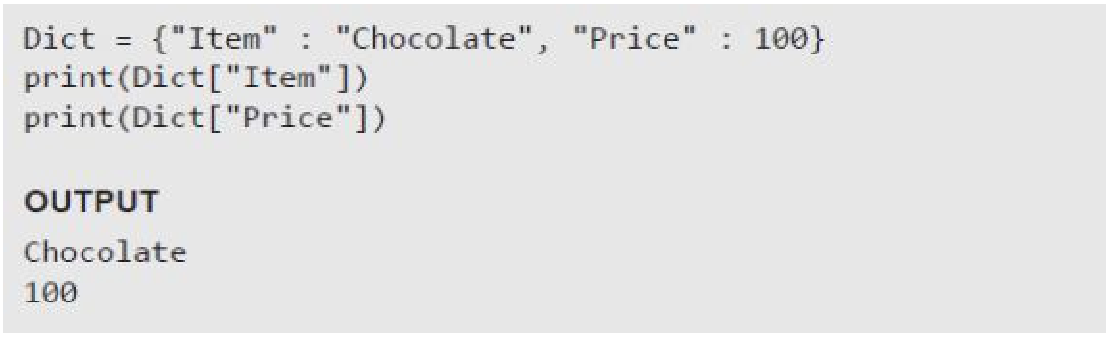
['a', 'bc', 78, 1.23, 'a', 'bc', 78, 1.23]

['a', 'bc', 78, 1.23, 'd', 78]

**3.5 Dictionary**

- Python's dictionaries stores data in key-value pairs.
- The key values are usually strings and value can be of any data type. The key value pairs are enclosed with curly braces ({ }).
- Each key value pair separated from the other using a colon (:). To access any value in the dictionary, you just need to specify its key in square braces ([]).Basically dictionaries are used for fast retrieval of data.

**Example**



```
Dict = {"Item" : "Chocolate", "Price" : 100}
print(Dict["Item"])
print(Dict["Price"])
```

**OUTPUT**

```
Chocolate
100
```

**4. Variables and Identifiers**

**4.1 Variables**

- Variable means its value can vary. You can store any piece of information in a variable.
- Variables are nothing but just parts of your computer's memory where information is stored. To identify a variable easily, each variable is given an appropriate name.

**4.2Identifiers**

Identifiers are names given to identify something. This something can be a variable, function, class, module or other object. For naming any identifier, there are some basic rules like:

- The first character of an identifier must be an underscore ('_') or a letter (upper or lowercase).
- The rest of the identifier name can be underscores ('_'), letters (upper or lowercase), or digits (0-9).
- Identifier names are case-sensitive. For example, myvar and myVar are not the same.
- Punctuation characters such as @, $, and % are not allowed within identifiers.
- *Examples of valid identifier names* are sum, __my_var, num1, r, var_20, First, etc.
- *Examples of invalid identifier names* are 1num, my-var, %check, Basic Sal, H#R&A, etc.,

## 5. Operators

- Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.
- For example:

```
>>> 2+3
5
```

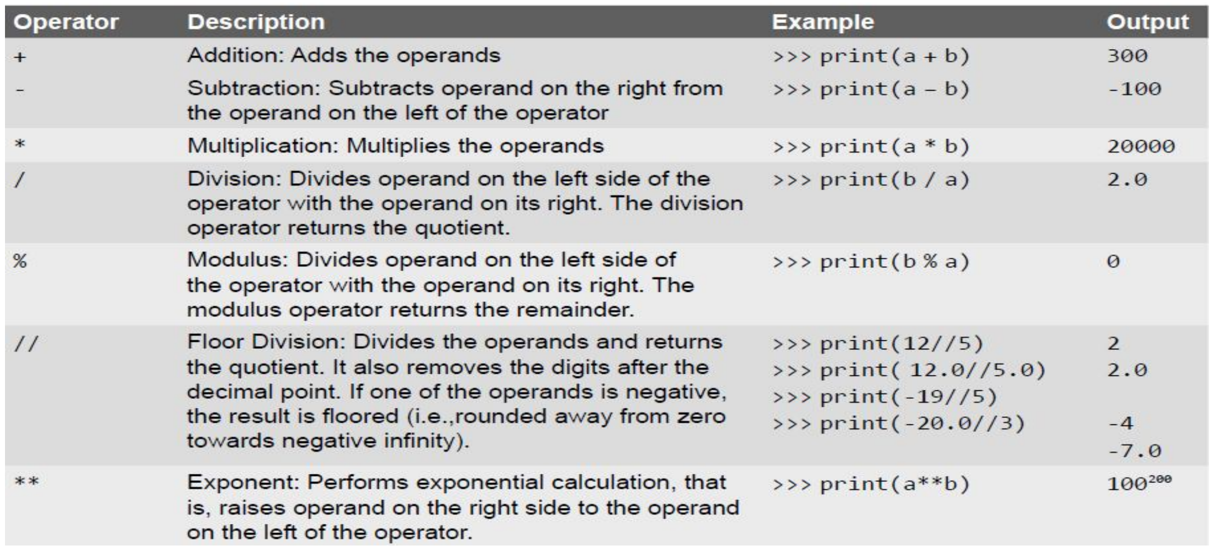
  Here, + is the operator that performs addition. 2 and 3 are the operands and 5 is the output of the operation.

- Python supports the following operators
  1. Arithmetic operators
  2. Comparison (Relational) operators
  3. Unary Operators
  4. Bitwise operators
  5. Shift Operators
  6. Logical Operators
  7. Membership and Identity Operators
  8. Assignment operators
  9. Special operators

### 5.1Arithmetic Operators

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.
- This operator will work on two operands.

➢ Example: If a=100 and b=200 then look at the table below, to see the result of arithmetic operations.

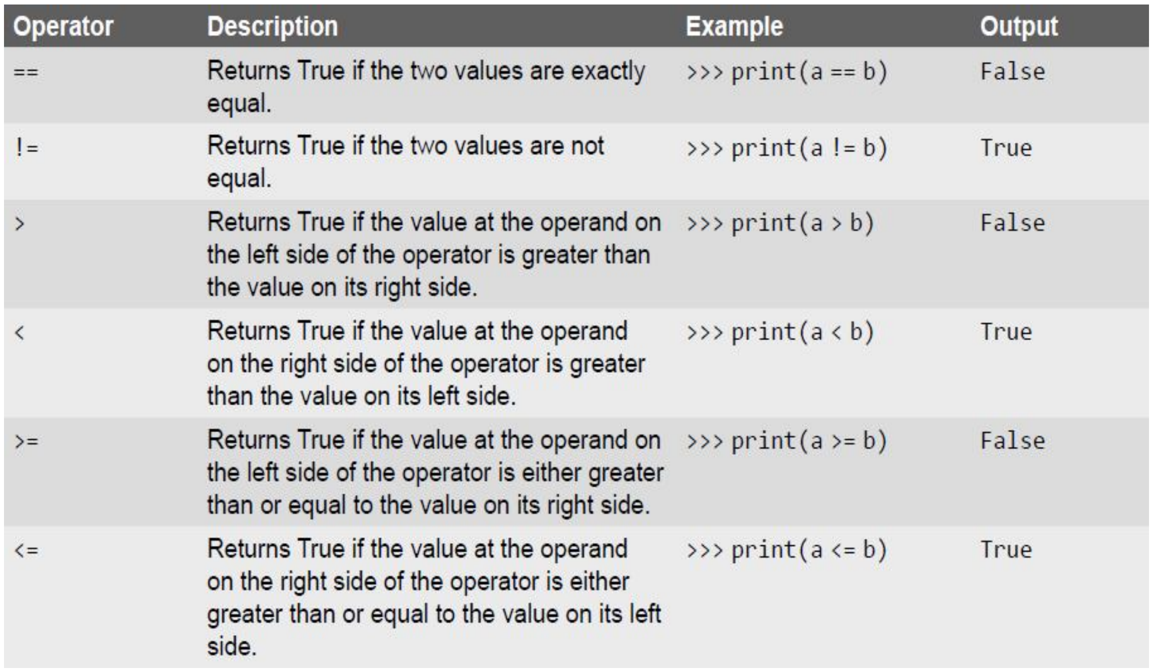


| Operator | Description | Example | Output |
|---|---|---|---|
| + | Addition: Adds the operands | `>>> print(a + b)` | 300 |
| - | Subtraction: Subtracts operand on the right from the operand on the left of the operator | `>>> print(a – b)` | -100 |
| * | Multiplication: Multiplies the operands | `>>> print(a * b)` | 20000 |
| / | Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient. | `>>> print(b / a)` | 2.0 |
| % | Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder. | `>>> print(b % a)` | 0 |
| // | Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity). | `>>> print(12//5)`<br>`>>> print( 12.0//5.0)`<br>`>>> print(-19//5)`<br>`>>> print(-20.0//3)` | 2<br>2.0<br>-4<br>-7.0 |
| ** | Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator. | `>>> print(a**b)` | $100^{200}$ |

## 5.2Comparision (Relational) Operators

➢ A Relational or Comparison operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0

➢ For Example assuming a=100 and b=2000,we can use the comparison operators on them as specified in the following table.



| Operator | Description | Example | Output |
|---|---|---|---|
| == | Returns True if the two values are exactly equal. | `>>> print(a == b)` | False |
| != | Returns True if the two values are not equal. | `>>> print(a != b)` | True |
| > | Returns True if the value at the operand on the left side of the operator is greater than the value on its right side. | `>>> print(a > b)` | False |
| < | Returns True if the value at the operand on the right side of the operator is greater than the value on its left side. | `>>> print(a < b)` | True |
| >= | Returns True if the value at the operand on the left side of the operator is either greater than or equal to the value on its right side. | `>>> print(a >= b)` | False |
| <= | Returns True if the value at the operand on the right side of the operator is either greater than or equal to the value on its left side. | `>>> print(a <= b)` | True |

## 5.3Unary Operator

➢ Unary operators act on single operands. Python supports unary minus operator.

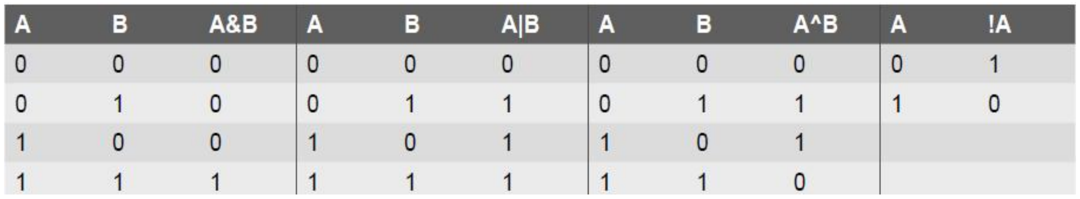
- Unary minus operator is strikingly different from the arithmetic operator that operates on two operands and subtracts the second operand from the first operand.
- When an operand is preceded by a minus sign, the unary operator negates its value.
- For example, if a number is positive, it becomes negative when preceded with a unary minus operator. Similarly, if the number is negative, it becomes positive after applying the unary minus operator. Consider the given example.

b = 10 a = -(b)

- The result of this expression, is a = -10, because variable b has a positive value. After applying unary minus operator (-) on the operand b, the value becomes -10, which indicates it as a negative value.

## 5.4 Bitwise Operators

- As the name suggests, bitwise operators perform operations at the bit level.
- These operators include bitwise AND, bitwise OR, bitwise XOR, and shift operators.
- Bitwise operators expect their operands to be of integers and treat them as a sequence of bits.
- The truth tables of these bitwise operators are given below.



| A | B | A&B | A | B | A\|B | A | B | A^B | A | !A |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | | |

- Example: If a=60 and b=13 then look at the table below, to see the result of Bitwise operations.

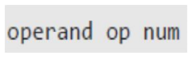
| Operator | Description | Example |
|---|---|---|
| & <br><br> Binary AND | Operator copies a bit to the result if it exists in both operands | (a & b) =12 <br> (means 0000 1100) |
| \| | It copies a bit if it exists in either operand. | (a \| b) = 61 |

| | | |
|---|---|---|
| Binary OR | | (means 0011 1101) |
| ^<br><br>Binary XOR | It copies the bit if it is set in one operand but not both. | (a ^ b) = 49<br>(means 0011 0001) |
| ~<br><br>Binary Ones Complement | It is unary and has the effect of 'flipping' bits. | (~a ) = -61<br>(means 1100 0011 in 2's complement form due to a signed binary number. |
| <<<br><br>Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | a << 2 = 240<br>(means 1111 0000) |
| >><br><br>Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 = 15<br>(means 0000 1111) |

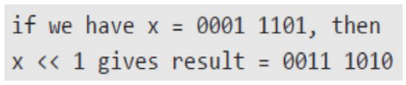
### 5.5 Shift Operators

- Python supports two bitwise shift operators. They are shift left (<<) and shift right (>>).
- These operations are used to shift bits to the left or to the right. The syntax for a shift operation can be given as follows:

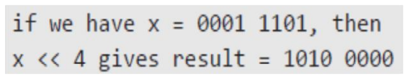
Examples:



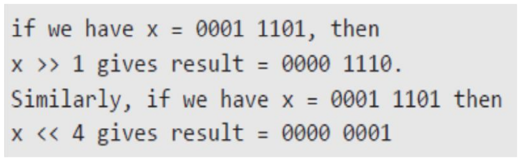
```
operand op num
```



```
if we have x = 0001 1101, then
x << 1 gives result = 0011 1010
```



```
if we have x = 0001 1101, then
x << 4 gives result = 1010 0000
```



```
if we have x = 0001 1101, then
x >> 1 gives result = 0000 1110.
Similarly, if we have x = 0001 1101 then
x << 4 gives result = 0000 0001
```

### 5.6 Logical Operators

- Logical operators are used to simultaneously evaluate two conditions or expressions with relational operators.

- ***Logical AND (and)*** If expressions on both the sides (left and right side) of the logical operator are true, then the whole expression is true.

  For example, If we have an expression (a>b) and (b>c), then the whole expression is true only if both expressions are true. That is, if b is greater than a and c.
- ***Logical OR (or)*** operator is used to simultaneously evaluate two conditions or expressions with relational operators. If one or both the expressions of the logical operator is true, then the whole expression is true.

  For example, If we have an expression (a>b) or (b>c), then the whole expression is true if either b is greater than a or b is greater than c.
- ***Logical NOT (not)*** operator takes a single expression and negates the value of the expression. Logical NOT produces a zero if the expression evaluates to a non-zero value and produces a 1 if the expression produces a zero. In other words, it just reverses the value of the expression.

  For example, a = 10; b = not a; Now, the value of b = 0.

## 5.7 Membership and Identity Operators

### 5.7.1. Membership Operator

Python supports two types of membership operators–in and not in. These operators, test for membership in a sequence such as strings, lists, or tuples.

- **in Operator:** The operator returns true if a variable is found in the specified sequence and false otherwise. For example, a in nums returns 1, if a is a member of nums.
- **not in Operator:** The operator returns true if a variable is not found in the specified sequence and false otherwise. For example, a not in nums returns 1, if a is not a member of nums.

### 5.7.2. Identity Operators

- **is Operator:** Returns true if operands or values on both sides of the operator point to the same object and false otherwise. For example, if a is b returns 1, if id(a) is same as id(b).
- **is not Operator:** Returns true if operands or values on both sides of the operator does not point to the same object and false otherwise. For example, if a is not b returns 1, if id(a) is not same as id(b).

## 5.8 Assignment Operators

- Assignment operators are used in Python to assign values to variables.

- a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

| **Assignment operators in Python** | | |
|---|---|---|
| **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 |

## 5.9 Operator Precedence

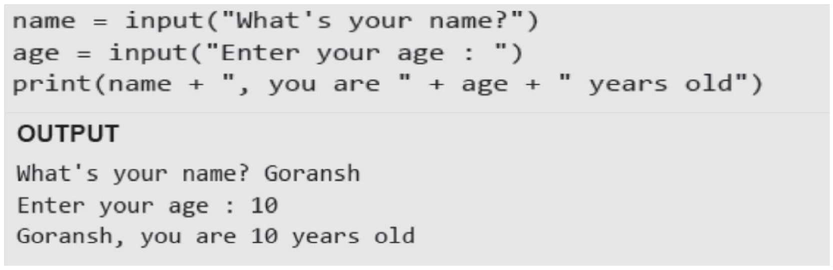
- The operator precedence in Python are listed in the following table. It is in descending order, upper group has higher precedence than the lower ones.

| Operator precedence rule in Python | |
|---|---|
| **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 | Comparison, Identity, Membership operators |
| not | Logical NOT |
| and | Logical AND |
| or | Logical OR |

## 6. Input Operation

- To take input from the users, Python makes use of the input( ) function. The input( ) function prompts the user to provide some information on which the program can work and give the result.
- However, we must always remember that the input function takes user's input as a string.

Example:



```
name = input("What's your name?")
age = input("Enter your age : ")
print(name + ", you are " + age + " years old")
```

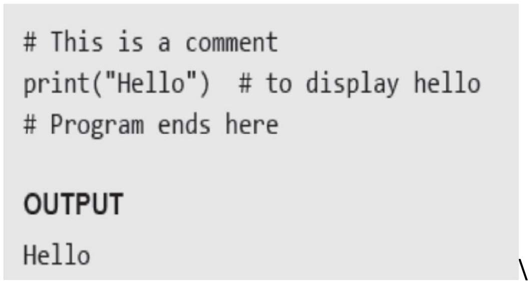
**OUTPUT**

```
What's your name? Goransh
Enter your age : 10
Goransh, you are 10 years old
```

**7. Comments**

- Comments are the non-executable statements in a program. They are just added to describe the statements in the program code.
- Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simply ignores the comments.
- In Python, a hash sign (#) that is not inside a string literal begins a comment. All characters following the # and up to the end of the line are part of the comment

Example:



```
# This is a comment
print("Hello")  # to display hello
# Program ends here
```

**OUTPUT**

```
Hello
```

\

- **Note:** For writing *Multi line comments*. Make sure to indent the leading ' ' ' appropriately to avoid an Indentation Error

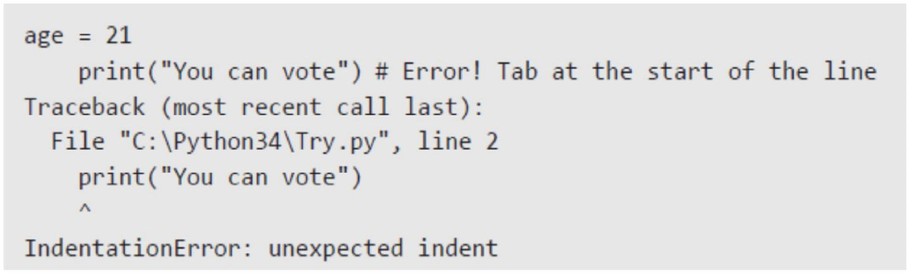
```
' ' '
This is a multiline
comment.
' ' '
```

**8. Indentation**

- Whitespace at the beginning of the line is called *indentation*. These *whitespaces or the indentation* are very important in Python.

- In a Python program, the leading whitespace including spaces and tabs at the beginning of the logical line determines the indentation level of that logical line.

Example:



```
age = 21
    print("You can vote") # Error! Tab at the start of the line
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2
    print("You can vote")
    ^
IndentationError: unexpected indent
```

**9. Expressions**

- An expression is any legal combination of symbols (like variables, constants and operators) that represents a value.
- In Python, an expression must have at least one operand (variable or constant) and can have one or more operators. On evaluating an expression, we get a value. *Operan*d is the value on which operator is applied.
- Generally Expressions are divided into the following types

1. *Constant Expressions*: One that involves only constants.
   Example: 8 + 9 – 2
2. *Integral Expressions:* One that produces an integer result after evaluating the expression.
   Example:a = 10
3. *Floating Point Expressions:* One that produces floating point results.
   Example: a * b / 2.0
4. *Relational Expressions: One that returns either true or false value.*
   Example: c = a>b
5. *Logical Expressions:* One that combines two or more relational expressions and returns a value as *True* or *False*.
   Example: a>b and y! = 0
6. *Bitwise Expressions:* One that manipulates data at bit level.
   Example: x = y&z
7. *Assignment Expressions:* One that assigns a value to a variable.
   Example: c = a + b or c = 10

   **Example Program**:

   Give the output for the following statements.( ***April 2018 Regular***)

```
a = 20
b = 10
c = 15
d = 5
print ("a:%d b:%d c:%d d:%d" % (a,b,c,d ))
e = (a + b) * c / d
print ("Value of (a + b) * c / d is ", e)
e = ((a + b) * c) / d
print ("Value of ((a + b) * c) / d is ", e)
e = (a + b) * (c / d)
print ("Value of (a + b) * (c / d) is ", e)
e = a + (b * c) / d
print ("Value of a + (b * c) / d is ", e)
```

**Output:**

```
a:20 b:10 c:15 d:5
Value of (a + b) * c / d is  90.0
Value of ((a + b) * c) / d is  90.0
Value of (a + b) * (c / d) is  90.0
Value of a + (b * c) / d is  50.0
```

## 10. Operations on Strings

- Like numbers we can also manipulate strings by performing operations on them.
- Basically there are three operations on strings

    1.String concatenation.

    2.String repetition or multiplication.

    3.String slicing.

### 10.1String Concatenation

- Like numbers we can add two strings in Python. The process of combining two strings is called concatenation.
- Two strings whether created using single or double quote are concatenated in the same way.Look at the codes given in the following example.
- Example: Codes to demonstrate how easily two strings are concatenated.

```
>>>print(“hello”  +  “-world”)
```

**output**

```
hello-world
>>>print(“hello”+’-world’)
```

**output**

```
hello-world
>>print(‘hello’+’-world’)
```

**output**

```
hello-world.
```

***Note:*** we cannot add string to number which generates an error.

## 10.2 String Repetition or Multiplication

- You cannot add string and number but we can multiply string and number with the help of this string repetition.
- When an string is multiplied with an integer n, then the string is repeted n times.
- Thus, the * operator is also called as string repetition operator.
- The order of string and integer is not important.

  Example

  ```
  >>print(“hello” * 5)
  ```

  **Output**

  ```
  hello hello hello hello hello
  >>print(5 * “hello”)
  ```

  **Output**

  ```
  hello hello hello hello hello
  ```

  ***Note:*** We cannot multiply two strings and cannot multiply string with floating point number.

## 10.3 Slice Operations on Strings

- You can extract subsets of strings by using the slice operator ([ ] and [:]). You need to specify index or the range of index of characters to be extracted.
- The index of the first character is 0 and the index of the last character is n-1, where n is the number of characters in the string.
- If you want to extract characters starting from the end of the string, then you must specify the index as a negative number. For example, the index of the last character is -1.
- Program to perform slice operation on strings

  ```
  #string operations
  ```

```
str = ‘Python is easy !!!’
print(str)
print(str[0])
print(str[3:9])
print(str[4:])
print(str[-1])
print(str[:5])
print(str * 2)
print(str + ‘Isn’t gec’)
```

**Output**

```
Python is easy !!!
P
hon is
on is easy
!
Pytho
Python is easy !!! Python is easy !!!
Python is easy !!! Isn’t gec
```

**11.Type Conversion**

- In Python, it is just not possible to complete certain operations that involves different types of data.
- For example, it is not possible to perform "2" + 4 since one operand is an integer and the other is of string type.

```
>>>”20” + “30”
```

**Output**

**‘2030’**

```
>>> int(“2”) + int(“3”)
```

**Output**

**5**

- Another situation in which type conversion is must when we want to accept a non string value(integer or float) as an input.we know that input function returns string,so we must typecast the input to numbers to perform calculations on them.
- **Example1:**

```
x=input(“Enter the first number”)
y=input(“Enter the second number”)
print(x+y)
```

**output**

```
Enter the first number 6
Enter the second number 7
67
```

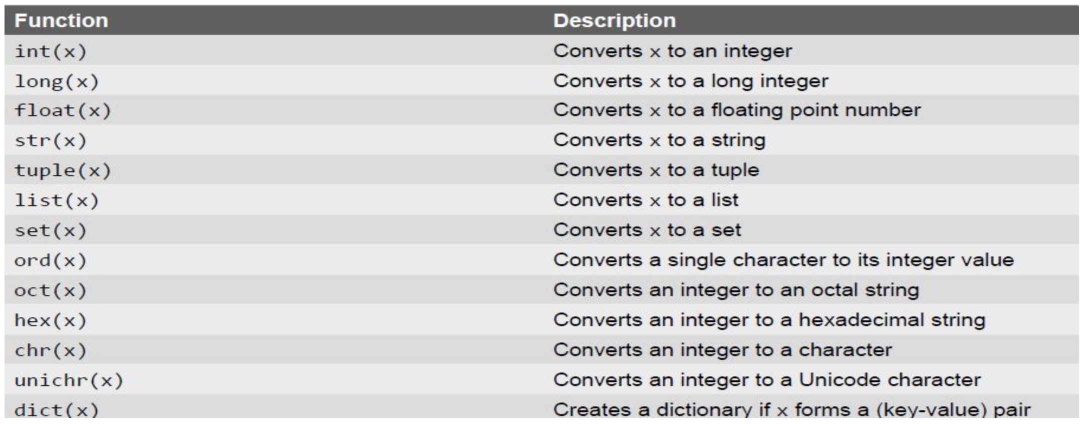
- **Example2:**

```
x=int(input("Enterthefirstnumber))
y=int(input("Enterthesecondnumber))
print(x+y)
```

**Output**

```
Enterthefirstnumber6
Enterthesecondnumber7
13
```

- Python provides various built-in functions to convert value from one data type to another datatype.The following are the functions return new object representing the coverted value. Some of them are given in the following table.



| Function | Description |
|---|---|
| int(x) | Converts x to an integer |
| long(x) | Converts x to a long integer |
| float(x) | Converts x to a floating point number |
| str(x) | Converts x to a string |
| tuple(x) | Converts x to a tuple |
| list(x) | Converts x to a list |
| set(x) | Converts x to a set |
| ord(x) | Converts a single character to its integer value |
| oct(x) | Converts an integer to an octal string |
| hex(x) | Converts an integer to a hexadecimal string |
| chr(x) | Converts an integer to a character |
| unichr(x) | Converts an integer to a Unicode character |
| dict(x) | Creates a dictionary if x forms a (key-value) pair |

## 12Type Casting vs Type Coercion

- we have done explicit conversion of a value from one data type to another. This is known as *type casting*.
- However, in most of the programming languages including Python, there is an implicit conversion of data types either during compilation or during run-time. This is also known *type coercion*.
- For example, in an expression that has integer and floating point numbers (like 21 + 2.1 gives 23.1), the compiler will automatically convert the integer into floating point number so that fractional part is not lost.

## 13 Limitations of Python:

- Parallel processing can be done in Python but not as elegantly as done in some other languages (like JavaScript and Go Lang).
- Being an interpreted language, Python is slow as compared to C/C++. Python is not a very good choice for those developing a high-graphic 3d game that takes up a lot of CPU.

- As compared to other languages, Python is evolving continuously and there is little substantial documentation available for the language.
- As of now, there are few users of Python as compared to those using C, C++ or Java.
- It lacks true multiprocessor support.
- It has very limited commercial support point.
- Python is slower than C or C++ when it comes to computation heavy tasks and desktop applications.
- It is difficult to pack up a big Python application into a single executable file. This makes it difficult to distribute Python to non-technical.

**Procedure for Executing your First Python Program**

**Step1:** Type the program in notepad and save your python program with the following extension

Filename.py

**Step2:** To Open Python Shell goto

Start->All Programs->Python 3.4->IDLE(Python GUI)

**Step3:** To Load your python file in the shell goto

Select->open->Select the path of the python file where it actually Store.

*Output:*A Separate window will be opened

**Step4:** For running your Python program click on run tab on the top of the separate window in the previous step and select run module or simply press F5.

**1.simplemsg.py**

```
# Write a Python program to display the simple message "Hello World"
print("Hello World")
```

**Output**

```
Hello World
```

**2.sumoftwonumbers.py**

```
# Write a Python program to add two numbers
num1=int(input("Enter first number"))
num2=int(input("Enter second number"))
res=num1+num2
print(str("The sum of two numbers is"))
print(str(num1)+"+"+str(num2)+"="+str(res))
```

**Output:**

```
Enter first number 10
Enter second number 20
The sum of two numbers is
10+20=30
```

**<u>3.distancebetweentwopoints.py</u>**

**#Write a Python Program to find distance between two points using Pythagoras Theorem**

```
x1=(int(input("Enter x1")))
x2=(int(input("Enter y1")))
y1=(int(input("Enter x2")))
y2=(int(input("Enter y2")))
distance=((x2-x1)**2+(y2-y1)**2)**0.5
print("Distance Between two points")
print(distance)
```

**Output:**

```
Enter x1   8
Enter y1   9
Enter x2   10
Enter y2  12
Distance Between two points
2.2360679775
```

**Syllabus: Decision Control and Looping Statements**

Conditional and un-conditional branching, iterative statements, nesting of decision Control Statements and loops.

**Programs:** Write a python program to

1. Test whether a given number is even or odd.
2. Print out the decimal equivalents of ½, 1/3 ,1/4........1/10,using for a loop.
3. Print a count down from the given number to zero using a while loop.
4. Find the sum of all the primes below hundred.
5. Find the factorial of a given number.

**14.Control Statements:**

- A *control statement* is a statement that determines the control flow of a set of instructions, i.e., it decides the sequence in which the instructions in a program are to be executed.
- Types of Control Statements —
  - **Selection/Conditional Control:** To execute only a selected set of statements.
  - **Iterative Control:** To execute a set of statements repeatedly.
  - **Un-conditional Control:**

**15. Selection /Conditional Branching Statements:**

- Python language supports different types of conditional branching statements which are as follows:
  - if Statement
  - if-else Statement
  - Nested if statement
  - if-elif-else statement.

**15. 1 if Statement:**

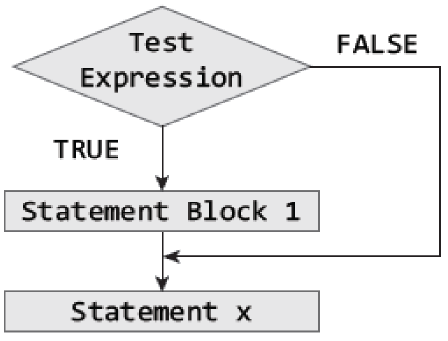
- An if statement is a selection control statement which is based on the value of a given Boolean Expression.

  **Syntax:**

```
if test_expression:
        statement 1
        .....
        statement n
statement x
```

- if structure may include 1 or n statements enclosed within if block.
- First, test expression is evaluated. If the test expression is true, the statement of if block (statement 1 to n) are executed, otherwise these statements will be skipped and the execution will jump to statement x.

  **Flow chart:**



**Example:**

```
x = 10        #Initialize the value of x
if(x>0):      #test the value of x
   x = x+1      #Increment the value of x if it is > 0
print(x)       #Print the value of x

OUTPUT
x = 11
```

## 15.2 if else Statement:

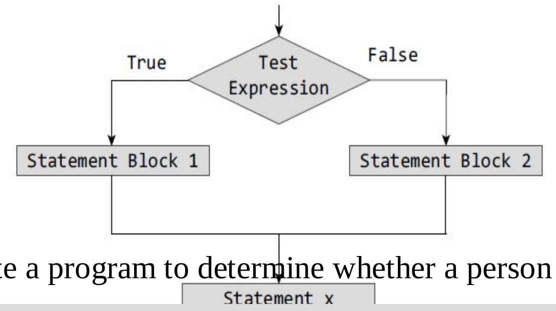
➢ The if .... else statement executes a group of statements when a test expression is true; otherwise, it will execute another group of statements.

**Syntax:**

```
if (test expression):
        statement_block 1
else:
        statement_block 2
statement x
```

➢ If the condition is **true,** then it will execute statement block 1 and if the condition is **false** then it will execute statement block 2.

**Flowchart:**



**Example:** Write a program to determine whether a person is eligible to vote:

```
age = int(input("Enter the age : "))
if(age>=18):
   print("You are eligible to vote")
else:
       yrs = 18 - age
       print("You have to wait for another " + str(yrs) +" years to cast your vote")

OUTPUT
Enter the age : 10
You have to wait for another 8 years to cast your vote
```

## 15.3 Nested if Statements :

- A statement that contains other statements is called a compound statement.
- To perform more complex checks, if statements can be nested, that is, can be placed one inside the other.
- In such a case, the inner if statement is the statement part of the outer one.
- Nested if statements are used to check if more than one conditions are satisfied.
- if statements can be nested resulting in multi-way selection.

```
var = 100
if var < 200:
  print( "Expression value is less than 200")
  if var == 150:
    print ("Which is 150”)
  elif var == 100:
    print ("Which is 100")
  elif var == 50:
    print ("Which is 50")
  elif var < 50:
    print ("Expression value is less than 50")
else:
  print ("Could not find true expression")
print ("Good bye!")
```

**Output:-**

```
Expression value is less than 200
Which is 100
Good bye!
```

## 15.4 if-elif-else Statement :

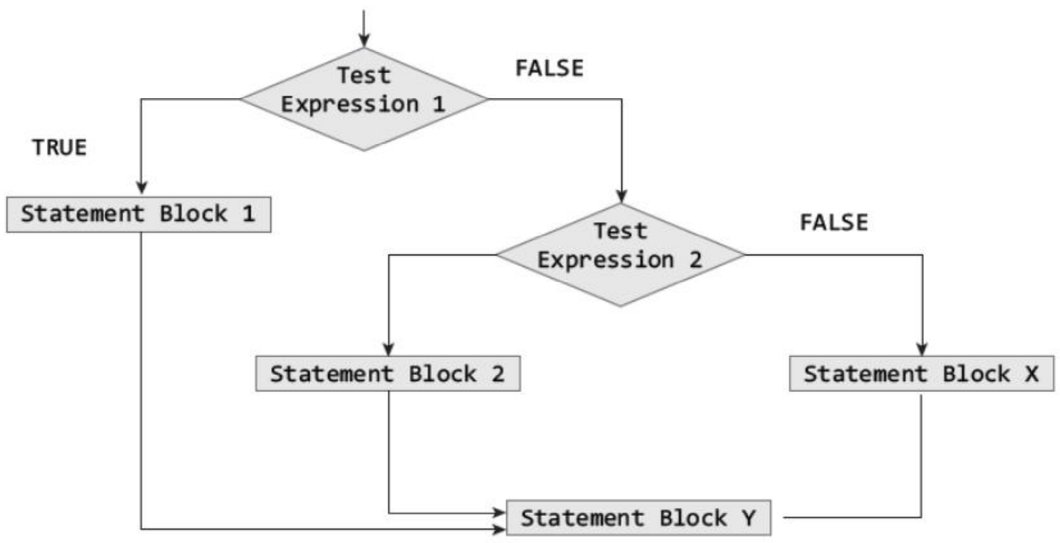
- Python supports if-elif-else statements to test additional conditions apart from the initial test expression.
- The if-elif-else construct works in the same way as a usual if-else statement.
- If-elif-else construct is also known as nested-if construct.
- A series of if and elif statements have a final else block, which is executed if none of the if or elif expressions is True.

**Syntax:**

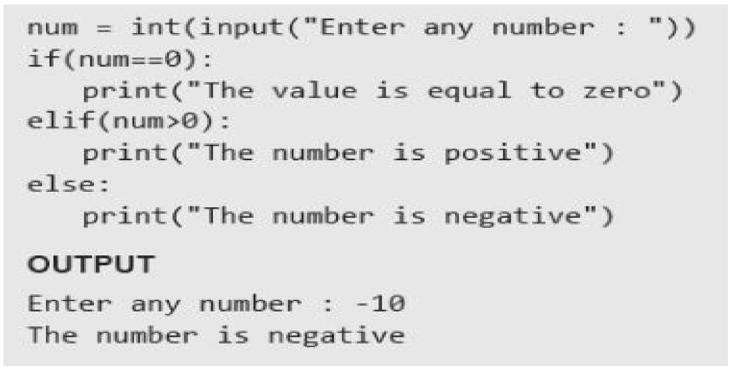
```
if (test expression 1):
        statement block1
```

```
elif (test expression 2):
        statement block2
. . . . . . . . . . . . . . . ..
elif( test expression N):
        statement block N
else:
        statement block X
```

**Flowchart:**



**Program:** To test whether a number entered by the user is negative, positive, or zero



```
num = int(input("Enter any number : "))
if(num==0):
    print("The value is equal to zero")
elif(num>0):
    print("The number is positive")
else:
    print("The number is negative")
```

**OUTPUT**

```
Enter any number : -10
The number is negative
```

## 16. Looping

**Statements/Iterative Structure:**

- ***Iterative statements*** are decision control statements that are used to repeat the execution of a list of statements.
- Python supports 2 types of iterative statements-*while* loop and *for* loop.

### 2.1 while Loop :

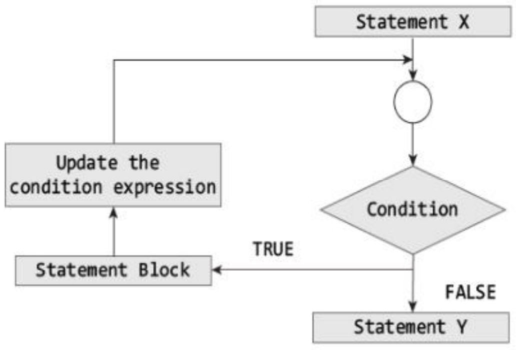
➢ The While loop provides a mechanism to repeat one or more statements while a particular condition is **TRUE.**

**Syntax:**

```
Statement x
while (condition):
    Statement block
Statement y
```

- In while loop, the condition is tested before any of the statements in the statement block is executed.
- If the condition is **TRUE,** only then the statements will be executed otherwise if the condition is **False,** the control will jump to statement y, that is the immediate statement outside the *while* loop block.

**Flowchart:**



**Example:** Program to print first 10 numbers using a while loop

```
i=0
while(i<=10):
    print(i, end=” “)
    i=i+1
```

**Output:** 0 1 2 3 4 5 6 7 8 9 10

**16.2. for Loop:**

- For loop provides a mechanism to repeat a task until a particular condition is True. It is usually known as a determinate or definite loop because the programmer knows exactly how many times the loop will repeat.
- The for...in statement is a looping statement used in Python to iterate over a sequence of objects.

**Syntax:**

```
for  loop_control_var  in sequence:
    statement block
```

**Flowchart:**

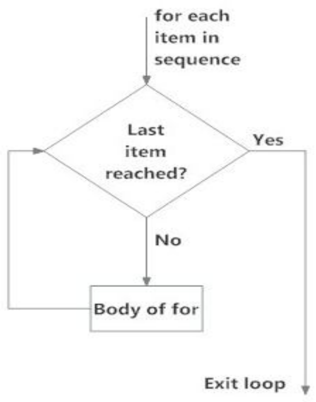


Fig: operation of for loop

### 16.2.1 for Loop and range() Function :

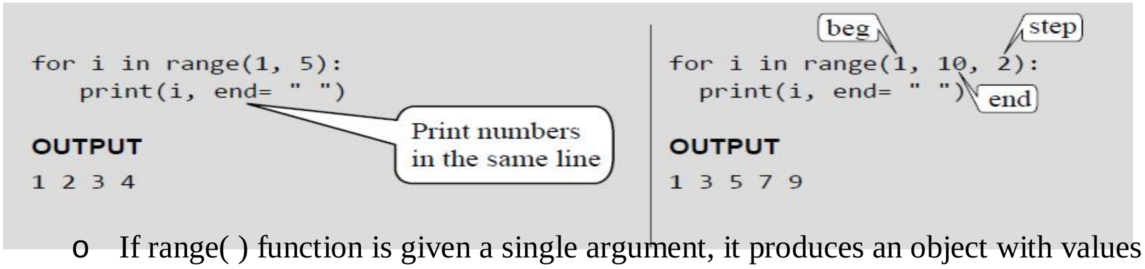
- The range( ) function is a built-in function in Python that is used to iterate over a sequence of numbers.

  **Syntax:**

  **range(beg, end, [step])**

- The range( ) produces a sequence of numbers starting with beg (inclusive) and ending with one less than the number end.
- The step argument is option (that is why it is placed in brackets). By default, every number in the range is incremented by 1 but we can specify a different increment using step. It can be both negative and positive, but not zero.

**Example:** Program to print first n numbers using the range() in a for loop

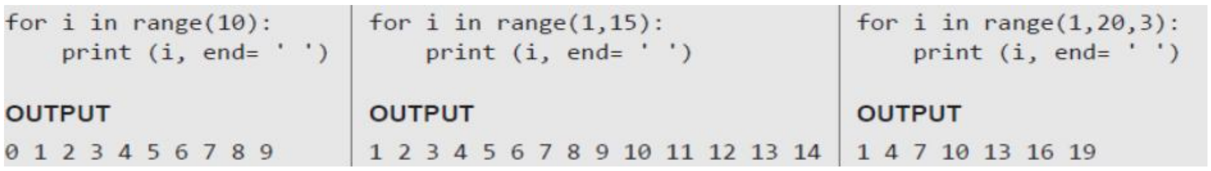


```
for i in range(1, 5):
   print(i, end= " ")

OUTPUT
1 2 3 4
```

```
for i in range(1, 10, 2):
   print(i, end= " ")

OUTPUT
1 3 5 7 9
```

- If range( ) function is given a single argument, it produces an object with values from 0 to argument-1. **For example**: range(10) is equal to writing range(0, 10).
- If range( ) is called with two arguments, it produces values from the first to the second. **For example**, range(0, 10) gives 0-9.
- If range( ) has three arguments then the third argument specifies the interval of the sequence produced. In this case, the third argument must be an integer. **For example**, range(1, 20, 3) gives 1, 4, 7, 10, 13, 16, 19.

**Example:**



```
for i in range(10):
    print (i, end= ' ')

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

```
for i in range(1,15):
    print (i, end= ' ')

OUTPUT
1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
for i in range(1,20,3):
    print (i, end= ' ')

OUTPUT
1 4 7 10 13 16 19
```

1. Program that accepts an integer (n) and computes the value of n+nn+nnn. (Eg. If n=5, find 5+55+555).

```
n = int(input("Enter a number: "))
str_n = str(n)
sum = n
sum_str = str(n)
for i in range(1, 3):
        sum_str = sum_str + str_n
        sum = sum + int(sum_str)
print(sum)
```

2. Program that accepts a word from the user and reverse it

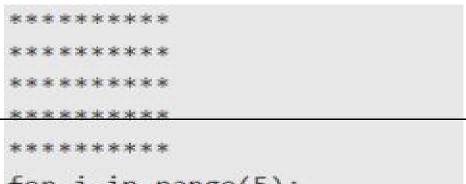
```
s = input("Enter a word: ")
str = ""
for i in s:
str = i + str
print("Reverse of", s, "is:", str)
```

**16.3. Nested Loops :**

- Python allows its users to have nested loops, that is, loops that can be placed inside other loops.
- Although this feature will work with any loop like while loop as well as for loop.
- A for loop can be used to control the number of times a particular set of statements will be executed.
- Another outer loop could be used to control the number of times that a whole loop is repeated.
- Loops should be properly indented to identify which statements are contained within each for statement.

**Example:**

1. Program to print the following pattern

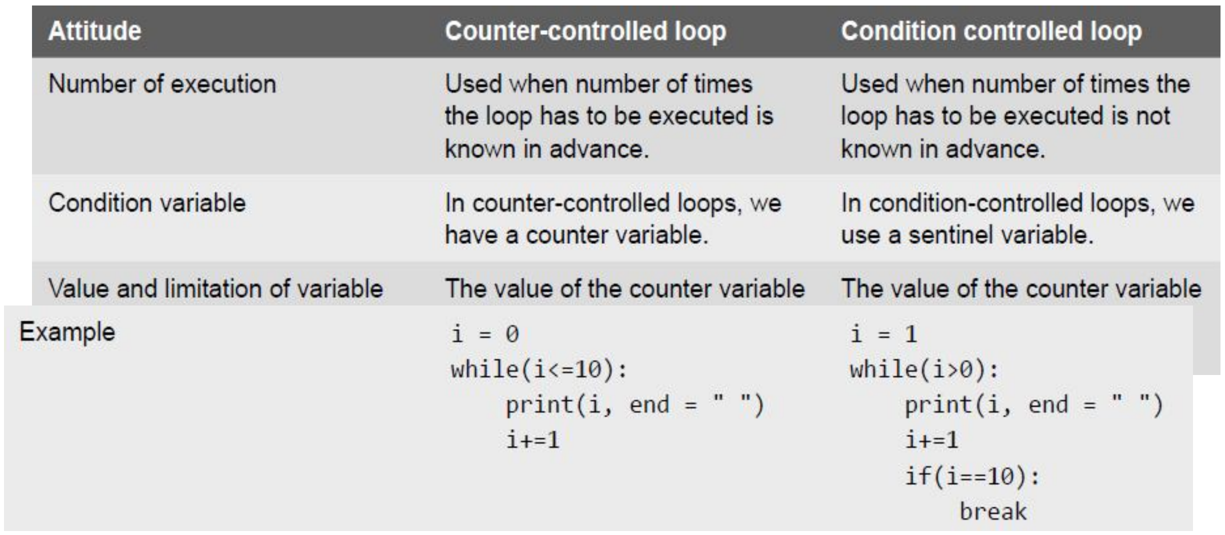


```
**********
**********
**********
**********
**********
for i in range(5):
```

2. Program to display multiplication tables from 1 to 10

```
for i in range(1, 11):
        for j in range(1, 11):
                print(i, '*', j, '=', i*j)
```

**16.3. Condition-controlled and Counter-controlled Loops :**



| Attitude | Counter-controlled loop | Condition controlled loop |
|---|---|---|
| Number of execution | Used when number of times the loop has to be executed is known in advance. | Used when number of times the loop has to be executed is not known in advance. |
| Condition variable | In counter-controlled loops, we have a counter variable. | In condition-controlled loops, we use a sentinel variable. |
| Value and limitation of variable | The value of the counter variable | The value of the counter variable |
| Example | `i = 0`<br>`while(i<=10):`<br>`    print(i, end = " ")`<br>`    i+=1` | `i = 1`<br>`while(i>0):`<br>`    print(i, end = " ")`<br>`    i+=1`<br>`    if(i==10):`<br>`        break` |

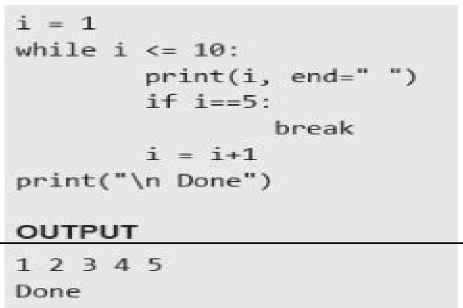
**16.4. The Break Statement:**

➢ The ***break*** statement is used to terminate the execution of the nearest enclosing loop in which it appears.
➢ The break statement is widely used with for loop and while loop.
➢ When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears.

**Syntax:**

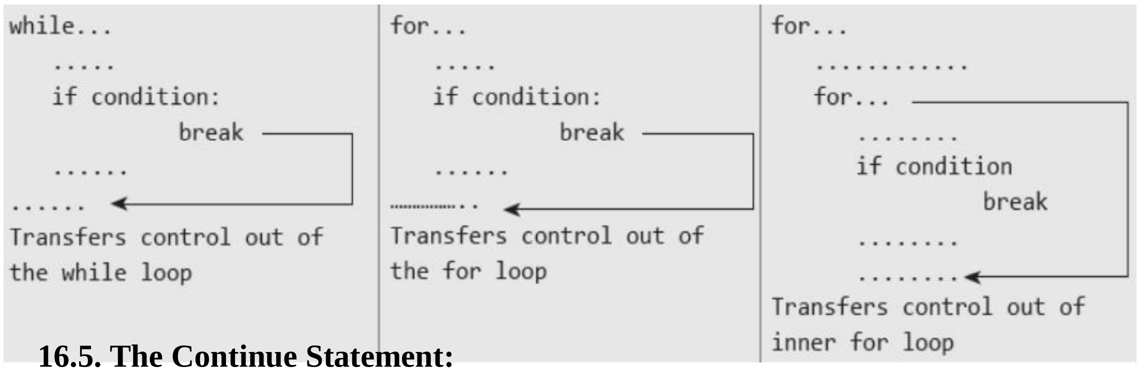
***break***

**Example:** Program to demonstrate the break statement



```
i = 1
while i <= 10:
        print(i, end=" ")
        if i==5:
                break
        i = i+1
print("\n Done")

OUTPUT
1 2 3 4 5
Done
```

- Above code is meant to print first 10 numbers using a ***while*** loop, but it will actually print only numbers from 0 to 4. As soon as i becomes equal to 5, the **break** statement is executed and the control jumps to the following while loop.
- Hence, the break statement is used to exit a loop from any point with in its body, by passing its normal termination expression. Below, Figure shows the transfer of control when the ***break*** statement is encountered.



```
while...
   .....
   if condition:
            break
   ......
......
Transfers control out of
the while loop
```

```
for...
   .....
   if condition:
            break
   ......
..........
Transfers control out of
the for loop
```

```
for...
   ...........
   for...
      ........
      if condition
               break
      ........
      ........
Transfers control out of
inner for loop
```

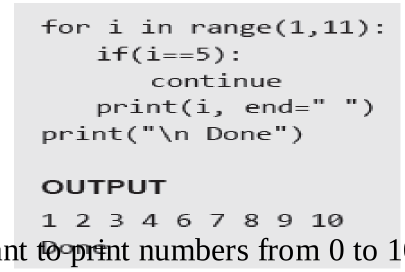
### 16.5. The Continue Statement:

- Like the break statement, the continue statement can only appear in the body of a loop.
- When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

  **Syntax:**

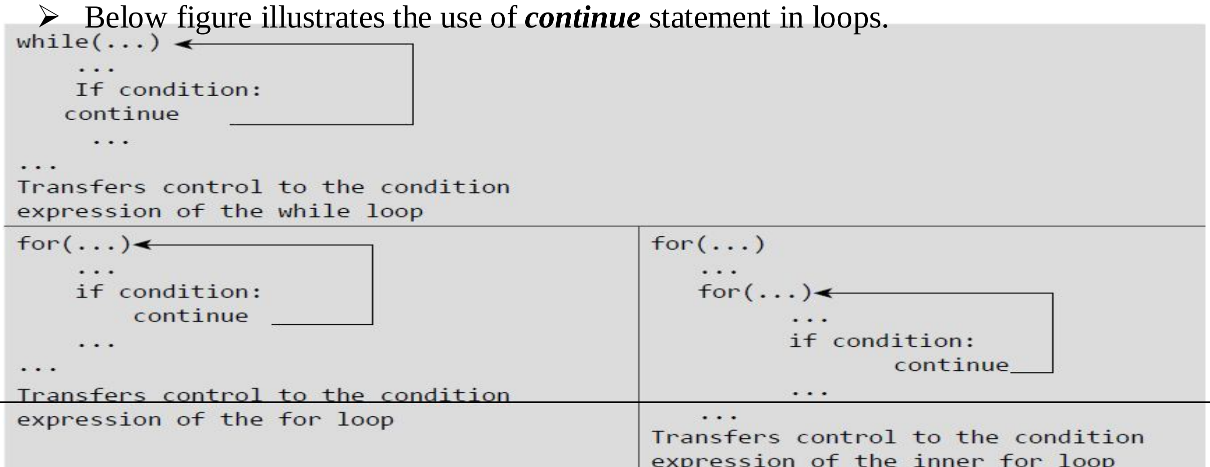
  ***Continue***

  **Example:** Program to demonstrate continue statement



```
for i in range(1,11):
    if(i==5):
        continue
    print(i, end=" ")
print("\n Done")

OUTPUT
1 2 3 4 6 7 8 9 10
Done
```

- Note that the code is meant to print numbers from 0 to 10.But as soon as i becomes equal to 5, the continue statement is encountered, so rest of the statements in the loop are skipped. In the output, 5 is missing as continue caused early increment of i and skipping of statement that printed the value of i on screen.
- Below figure illustrates the use of ***continue*** statement in loops.



```
while(...)
    ...
    If condition:
   continue
    ...
...
Transfers control to the condition
expression of the while loop
```

```
for(...)
    ...
    if condition:
        continue
    ...
...
Transfers control to the condition
expression of the for loop
```

```
for(...)
   ...
   for(...)
      ...
      if condition:
             continue
      ...
   ...
Transfers control to the condition
expression of the inner for loop
```

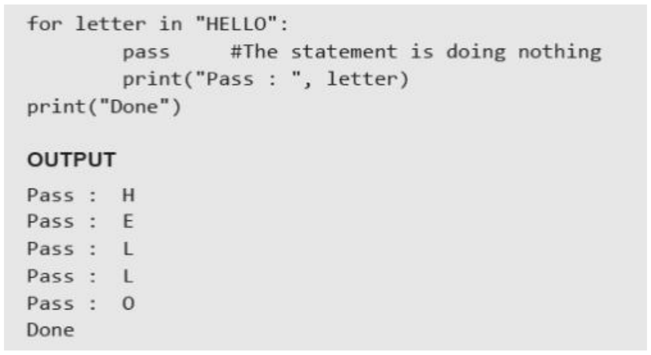
- It can be concluded that the continue statement is somewhat the opposite of the break statement. It forces the next iteration of the loop to take place, skipping any code in between itself and the test condition of the loop.
- The continue statement is usually used to restart a statement sequence when an error occurs.

### 16.6. The Pass Statement:

- Pass statement is used when a statement is required syntactically but no command or code has to be executed.
- It specified a null operation or simply No Operation (NOP) statement. Nothing happens when the pass statement is executed.
- The difference between a comment and pass statement is that while the interpreter ignores a comment entirely, pass is not ignored.
- Comment is not executed but pass statement is executed but nothing happens.
- Pass is a null statement.

**Example:**

1. Program to demonstrate **pass** statement



```
for letter in "HELLO":
        pass     #The statement is doing nothing
        print("Pass : ", letter)
print("Done")
```

**OUTPUT**

```
Pass :  H
Pass :  E
Pass :  L
Pass :  L
Pass :  O
Done
```

### 2.2 Difference between break, continue and pass

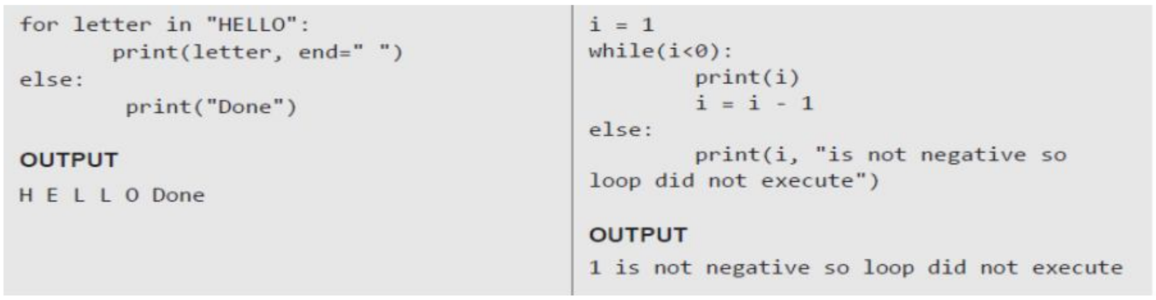
- The **break** statement terminates the execution of the nearest enclosing loop in which it appears.
- The **continue** statement skips the rest of the statements in the loop transfer the control un-conditionally to the loop-continuation portion of the nearest enclosing loop.
- The **pass** statement is a do-nothing statement in a loop. It is just added to make the loop syntactically correct. i.e, a pass statement is written as we can not have an empty body of the loop.

### 2.3 The Else Statement Used With Loops:

- In Python you can have the ***else*** statement associated with a loop statements.

- If the else statement is used with a ***for*** loop, the else statement is executed when the loop has completed iterating.
- But when used with the ***while*** loop, the else statement is executed when the condition becomes false.

**Examples:**



```
for letter in "HELLO":
       print(letter, end=" ")
else:
        print("Done")

OUTPUT
H E L L O Done
```

```
i = 1
while(i<0):
        print(i)
        i = i - 1
else:
        print(i, "is not negative so
loop did not execute")

OUTPUT
1 is not negative so loop did not execute
```

**17. Programs:**

3. 1 Write a python program to Test whether a given number is even or odd.

```
num = int(input("Enter a number: "))
if (num % 2==0):
        print(num, "is an even number.")
else:
        print(num, "is an odd number.")
```

**Output:**

```
Enter a number: 5
5 is an odd number.
```

3. 2 Write a python program to Print out the decimal equivalents of 1/1, 1/2, 1/3, 1/4........1/10 using for loop.

```
i=1
for i in range(1,11):
        value=1.0/i
        print("1/", i, "=", value)
```

**Output:**

```
1/ 1 = 1.0
1/ 2 = 0.5
```

```
1/ 3 = 0.333333333333
1/ 4 = 0.25
1/ 5 = 0.2
1/ 6 = 0.166666666667
1/ 7 = 0.142857142857
1/ 8 = 0.125
1/ 9 = 0.111111111111
1/ 10 = 0.1
```

3. 3. Write a python program to  Print a count down from the given number to zero using a while loop.

```
num=int(input("Enter a number: "))
print("count down from ", num, "to 0 :")
while (num >= 0):
    print(num)
    num = num - 1
```

**Output:**

```
Enter a number: 6
count down from  6 to 0:
    6
    5
    4
    3
    2
    1
    0
```

3.4. Write a python program to Find the sum of all the primes below hundred.

```
sum=0
for j in range(1,100):
    for i in range(2,j):
        if (j% i) == 0:
                break
            else:
                sum=sum+j     #where j is a prime number
print("Sum of prime numbers up to 100 is", sum)
```

**Output:**

```
Sum of prime numbers up to 100 is 1061
```

3. 5. Write a python program to find the factorial of a given number.

```
num=int(input("Enter a number: "))
fact=1
while (num>0):
    fact=fact*num
    num=num-1
print("Factorial of", num, "is",fact)
```

**Output:**

```
Enter a number: 6
Factorial of 6 is 720
```

**Example Programs :**

# **Write a Python program to display the simple message “Hello World”**

```
print(“Hello World”)
```

**Output**

```
Hello World
```

# **Write a Python program to add two numbers**

```
num1=int(input("Enter first number"))
num2=int(input("Enter second number"))
res=num1+num2
print(str("The sum of two numbers is"))
print(num1,"+","num2","=",res)
```

**Output:**

```
Enter first number 10
Enter second number 20
The sum of two numbers is
10+20=30
```

**#Write a Python Program to find distance between two points using Pythagoras Theorem**

```
x1=(int(input("Enter x1")))
```

```
x2=(int(input("Enter y1")))
y1=(int(input("Enter x2")))
y2=(int(input("Enter y2")))
distance=((x2-x1)**2+(y2-y1)**2)**0.5
print("Distance Between two points")
print(distance)
```
**Output:**
```
Enter x1   8
Enter y1   9
Enter x2   10
Enter y2  12
Distance Between two points
2.2360679775
```
**#program to check the person is eligible for vote using simple-if statement**
```
age=int(input("Enter the age of the person"))
if age>18:
   print("The person is eligible for vote")
   print("You can cast your vote")
print("Goodbye!")
```
**Output1:-**
```
Enter the age of the person23
The person is eligible for vote
You can cast your vote
Goodbye!
```
**Output2:-**
```
Enter the age of the person17
Goodbye!
```
**#program to demonstrate the if-else statement**
```
age=int(input("Enter the age of the person"))
if(age>=18):
   print("This person is eligible for vote")
   print("Cast your vote")
else:
   yrs=18-age
   print("You have to wait",yrs,"more year to cast your vote")
print("Goodbye!")
```
**Output1:-**
```
Enter the age of the person18
This person is eligible for vote
Cast your vote
Goodbye!
```
**Output2:**
```
Enter the age of the person15
You have to wait 3 more year to cast your vote
Goodbye!
```
**#week2a python program to check whether the given number is even or not**
```
num=int(input("Enter the number"))
if num%2==0:
   print("The num",num ,"is even")
```

```
else:
    print("The num",num," is odd")
```
**Output:**
```
Enter the number24
The num 24 is even
>>>
==================== RESTART: D:/Pythonprogramscseb/Week2a.py
====================
Enter the number13
The num 13  is odd
```
**#program to print max of three numbers using nested if**
```
a=int(input("Enter the first number"))
b=int(input("Enter the second number"))
c=int(input("Enter the third number"))
if a>b:
    if a>c:
        print("a is max")
    else:
        print("c is max")
else:
    if b>c:
        print("b is max")
    else:
        print("c is max")
```
**Output:-**
```
Enter the first number-3
Enter the second number5
Enter the third number9
c is max
>>>
==================== RESTART: D:/Pythonprogramscseb/nestedif.py
===================
Enter the first number3
Enter the second number-2
Enter the third number8
c is max
>>>
==================== RESTART: D:/Pythonprogramscseb/nestedif.py
===================
Enter the first number0
Enter the second number-1
Enter the third number-5
a is max
```
**#python program to solve max of three numbers using if-elif-else**
```
a=int(input("Enter the first number"))
b=int(input("Enter the second number"))
c=int(input("Enter the third number"))
if a>b and a>c:
    print("a is max")
```

```
elif b>c:
    print("b is max")
else:
    print("C is max")
```

**Output:-**

```
Enter the first number0
Enter the second number-1
Enter the third number-5
a is max
```

**#Week 2b program to display the grade of the student based on average**

```
sub1=int(input("Enter the sub1 marks"))
sub2=int(input("Enter the sub2 marks"))
sub3=int(input("Enter the sub3 marks"))
sub4=int(input("Enter the sub4 marks"))
sub5=int(input("Enter the sub5 marks"))
sum1=sub1+sub2+sub3+sub4+sub5;
avg=float(sum1/5)
print("The total marks obtained by the student is",sum1)
print("The avg marks obtained by student is",avg)
if avg>=90:
    print("Congratulations O grade")
elif avg>=80 and avg<90:
    print("Congratulations A+ Grade")
elif avg>=70 and avg<80:
    print("Congratulations A Grade")
elif avg>=60 and avg<70:
    print("Congratulations B+ Grade")
elif avg>=50 and avg<60:
    print("Congratulations B Grade")
elif avg>=40 and avg<50:
    print("Congratulations C Grade")
elif avg<40:
    print("Sorry Fail")
```

**Output:-**

```
Enter the sub1 marks92
Enter the sub2 marks94
Enter the sub3 marks95
Enter the sub4 marks56
Enter the sub5 marks45
The total marks obtained by the student is 382
The avg marks obtained by student is 76.4
Congratulations A Grade
```

**#program to display the decimal equivalents of 1/2,1/3,1/4,1/5,...1/10**

```
i=2
while i<=10:
    print("1/",i,"=",float(1/i))
    i=i+1
```

Output:

```
1/ 2 = 0.5
```

```
1/ 3 = 0.3333333333333333
1/ 4 = 0.25
1/ 5 = 0.2
1/ 6 = 0.16666666666666666
1/ 7 = 0.14285714285714285
1/ 8 = 0.125
1/ 9 = 0.1111111111111111
1/ 10 = 0.1
```

**#Example program for while demo**

```
n=int(input("Enter the number"))
i=0
while(i<=n):
    print(i,end='\t')
    i=i+1
```

Output:

```
Enter the number10
0	1	2	3	4	5	6	7	8	9	10
```

**Write a python program to find the factorial of a given number.**

```
num=int(input("Enter a number: "))
fact=1
while (num>0):
      fact=fact*num
       num=num-1
print("Factorial of", num, "is",fact)
```

**Output:**

```
Enter a number: 6
Factorial of 6 is 720
```

**#Write a python program to  Print a count down from the given number to zero using a while loop.**

```
num=int(input("Enter a number: "))
print("count down from ", num, "to 0 :")
while (num >= 0):
       print(num)
       num = num - 1
```

**Output:**

```
 Enter a number: 6
count down from  6 to 0:
6
5
4
3
2
1
0
```

**#python program to solve factorial of given number using for loop**

```
n=int(input("Enter the  number"))
fact=1
for i in range(n,0,-1):
    fact=fact*i
print("The factorial of given of given number is",fact)
```

Output:

```
Enter the  number7
The factorial of given of given number is 5040
```

**#python program to print fib sequence upton**

```
n=int(input("Enter the number"))
prev=0
next1=1
print("The fib sequence is")
print(prev,end=' ')
print(next1,end=' ')
i=1
while i<=n-2:
    sum1=prev+next1
    print(sum1,end=' ')
    prev=next1
    next1=sum1
    i=i+1
```

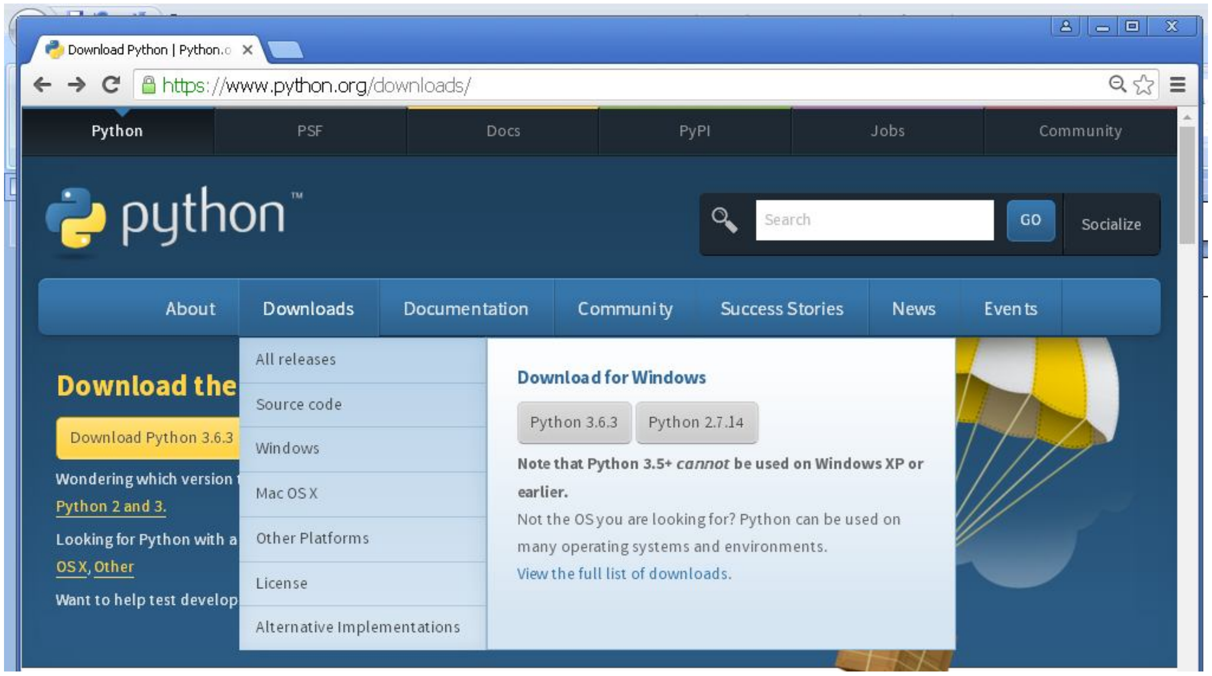
**Output:**

```
Enter the number6
The fib sequence is
0 1 1 2 3 5
```

## HOW TO INSTALL PYTHON

**Step1:**In order to use Python, it must first be installed on your computer. Follow these steps.

Go to the python website **www.python.org** and click on the 'Download' menu choice.



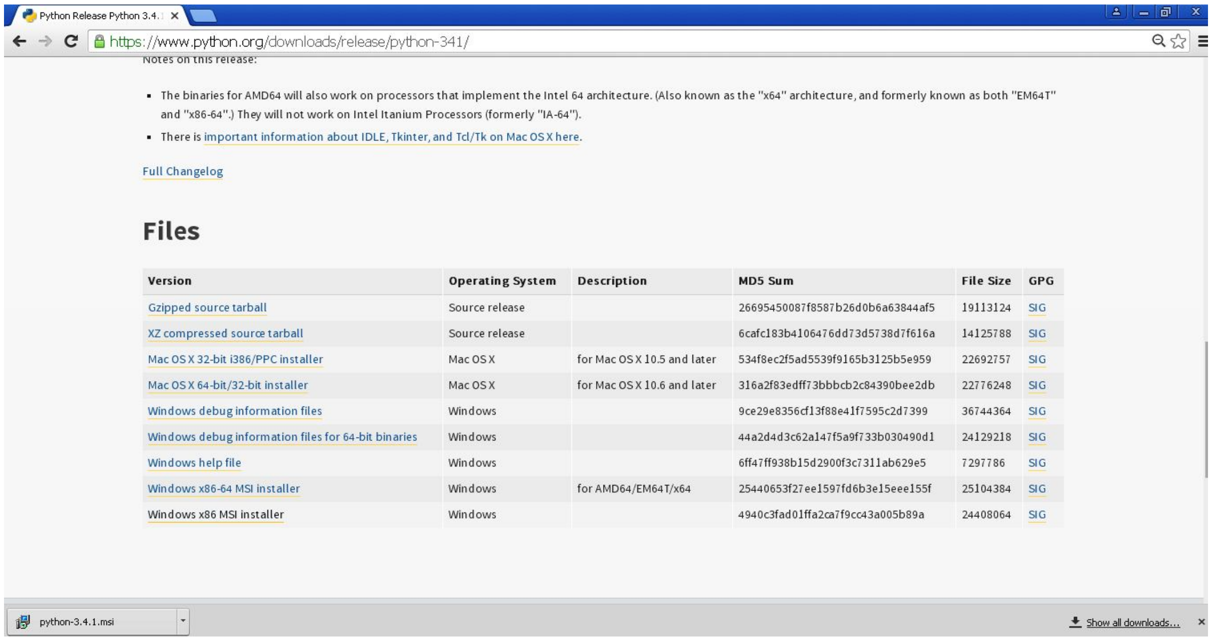
Wondering which version
Python 2 and 3.
Looking for Python with a
OS X, Other
Want to help test develop

**Note that Python 3.5+ *cannot* be used on Windows XP or earlier.**

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

**Step2:** Next click on the Python 3.4.1 (note the version number may change) Windows Installer to download the installer. If you know you're running a 32-bit os, you can choose the <u>**Windows x86 MSI installer**</u>


Python Release Python 3.4.
https://www.python.org/downloads/release/python-341/
python-3.4.1.msi
Show all downloads...

Notes on this release:

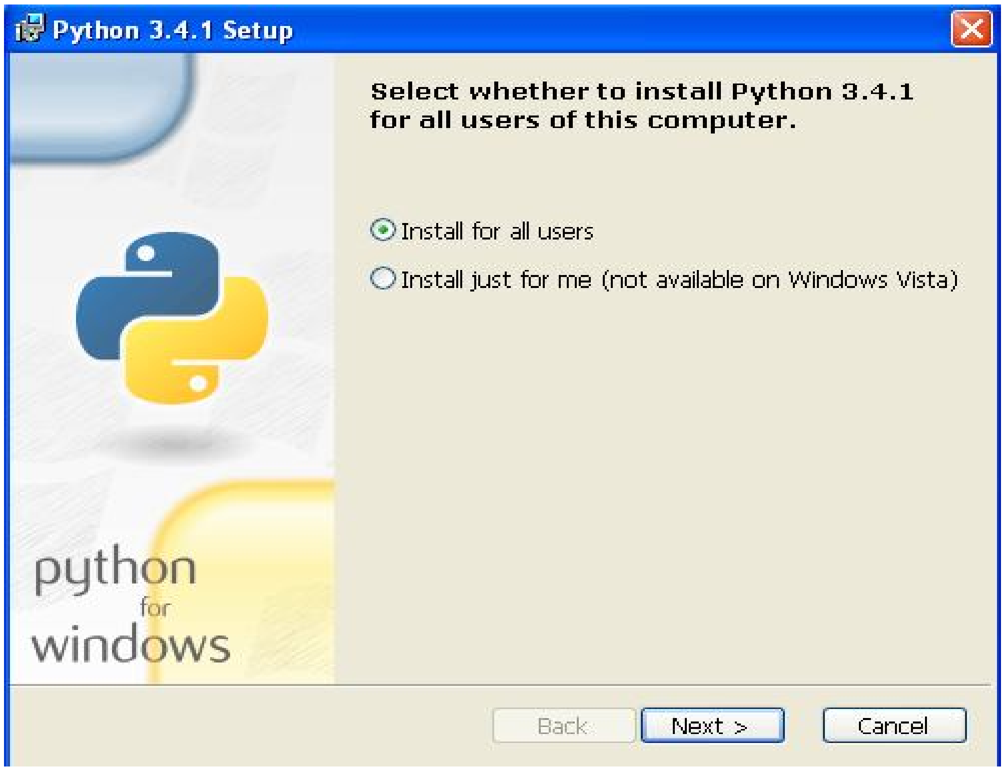
- The binaries for AMD64 will also work on processors that implement the Intel 64 architecture. (Also known as the "x64" architecture, and formerly known as both "EM64T" and "x86-64".) They will not work on Intel Itanium Processors (formerly "IA-64").
- There is important information about IDLE, Tkinter, and Tcl/Tk on Mac OS X here.

Full Changelog

Files

| Version | Operating System | Description | MD5 Sum | File Size | GPG |
|---|---|---|---|---|---|
| Gzipped source tarball | Source release | | 26695450087f8587b26d0b6a63844af5 | 19113124 | SIG |
| XZ compressed source tarball | Source release | | 6cafc183b4106476dd73d5738d7f616a | 14125788 | SIG |
| Mac OS X 32-bit i386/PPC installer | Mac OS X | for Mac OS X 10.5 and later | 534f8ec2f5ad5539f9165b3125b5e959 | 22692757 | SIG |
| Mac OS X 64-bit/32-bit installer | Mac OS X | for Mac OS X 10.6 and later | 316a2f83edff73bbbcb2c84390bee2db | 22776248 | SIG |
| Windows debug information files | Windows | | 9ce29e8356cf13f88e41f7595c2d7399 | 36744364 | SIG |
| Windows debug information files for 64-bit binaries | Windows | | 44a2d4d3c62a147f5a9f733b030490d1 | 24129218 | SIG |
| Windows help file | Windows | | 6ff47ff938b15d2900f3c7311ab629e5 | 7297786 | SIG |
| Windows x86-64 MSI installer | Windows | for AMD64/EM64T/x64 | 25440653f27ee1597fd6b3e15eee155f | 25104384 | SIG |
| Windows x86 MSI installer | Windows | | 4940c3fad01ffa2ca7f9cc43a005b89a | 24408064 | SIG |

**Step3:** Once the installer starts, it will ask who to install the program for. Usually installing for all users is the best choice.

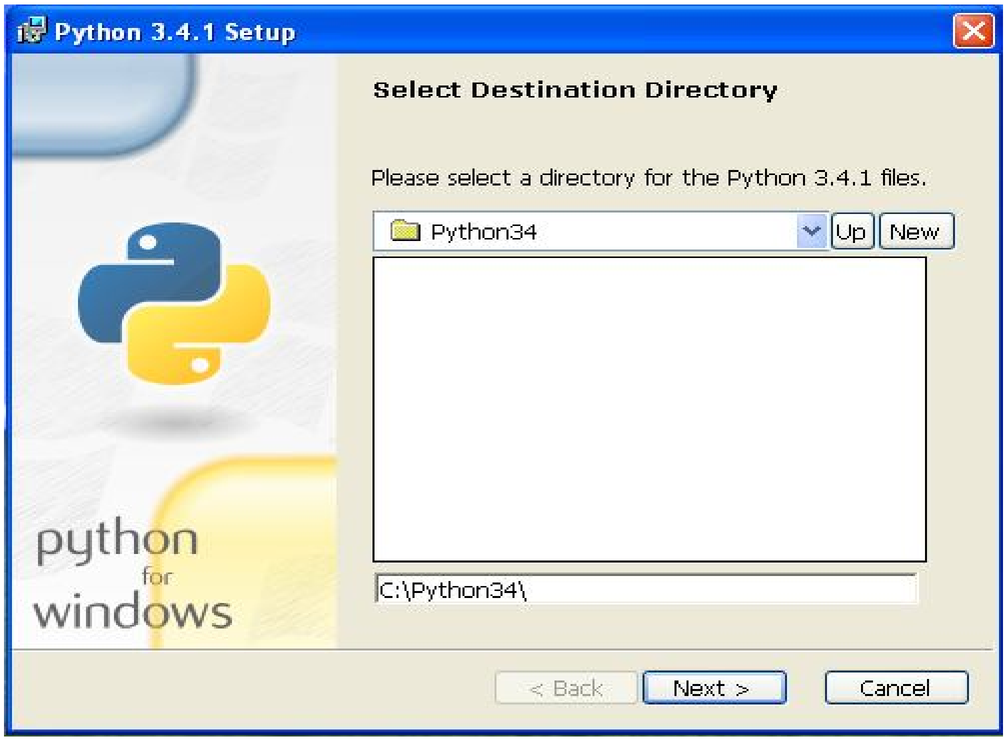


Select whether to install Python 3.4.1 for all users of this computer.

[✓ Install for all users]

Install just for me (not available on Windows Vista)

**Step4:** Next, it needs to know where to install the file. The default choice is fine.



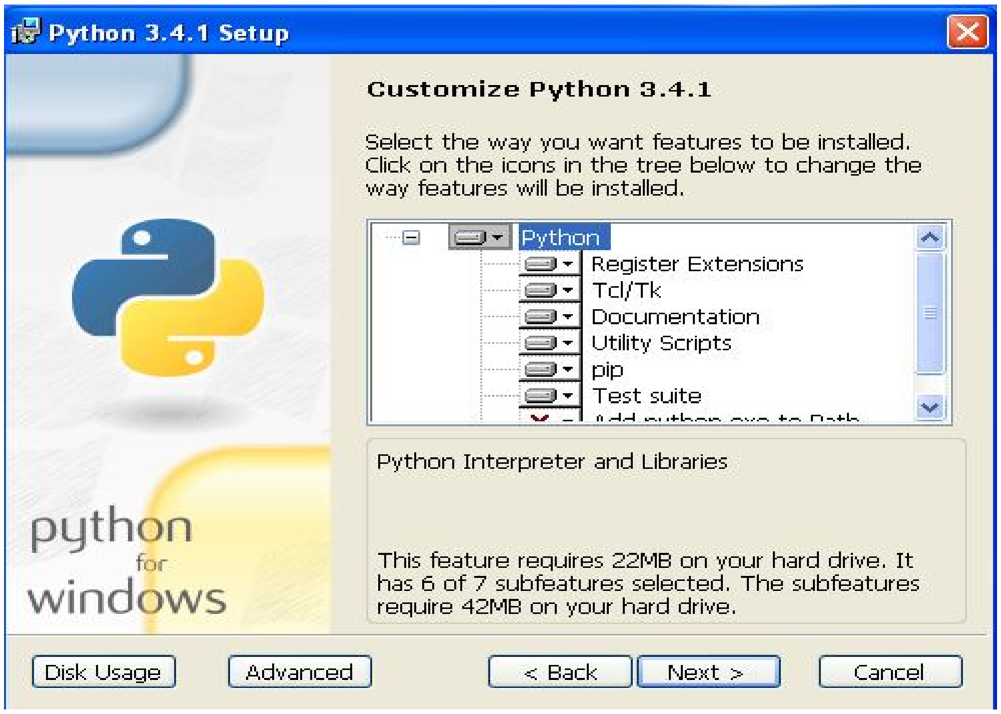
Select Destination Directory

Please select a directory for the Python 3.4.1 files.

Python34

C:\Python34\

**Step5**: You don't need to install the entire package, but we did.


Python 3.4.1 Setup

**Customize Python 3.4.1**

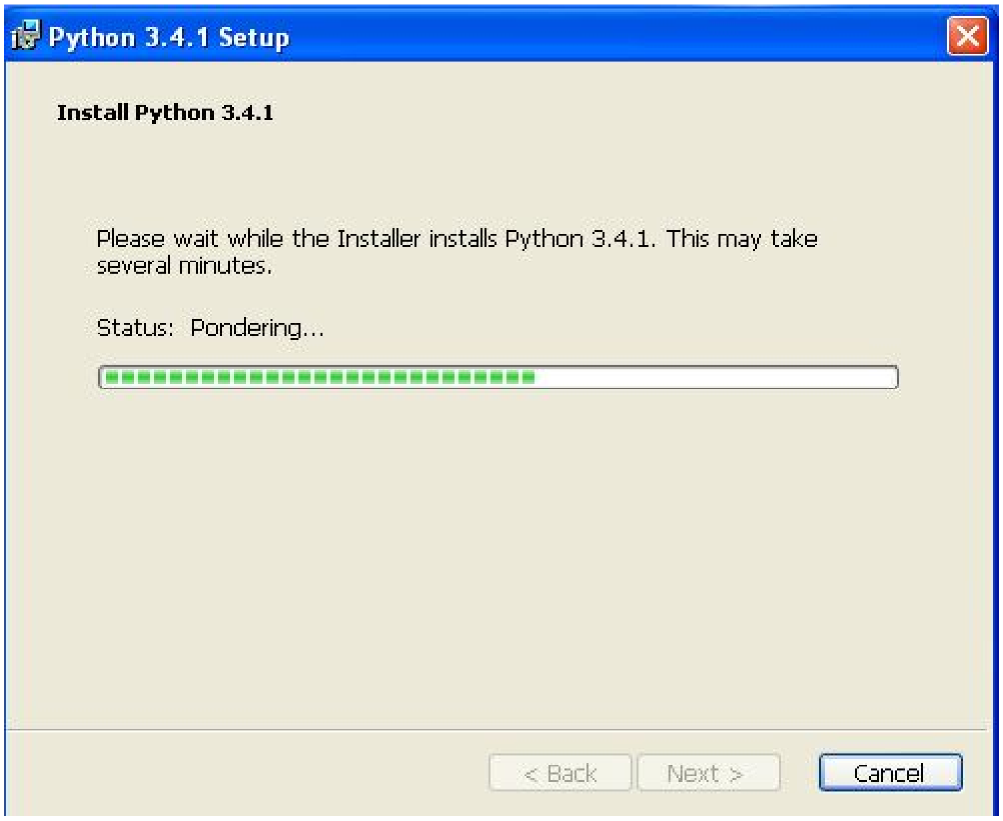
Select the way you want features to be installed.
Click on the icons in the tree below to change the way features will be installed.

Python
- Register Extensions
- Tcl/Tk
- Documentation
- Utility Scripts
- pip
- Test suite

Python Interpreter and Libraries

This feature requires 22MB on your hard drive. It has 6 of 7 subfeatures selected. The subfeatures require 42MB on your hard drive.

**Step6**: It will take a while to install.

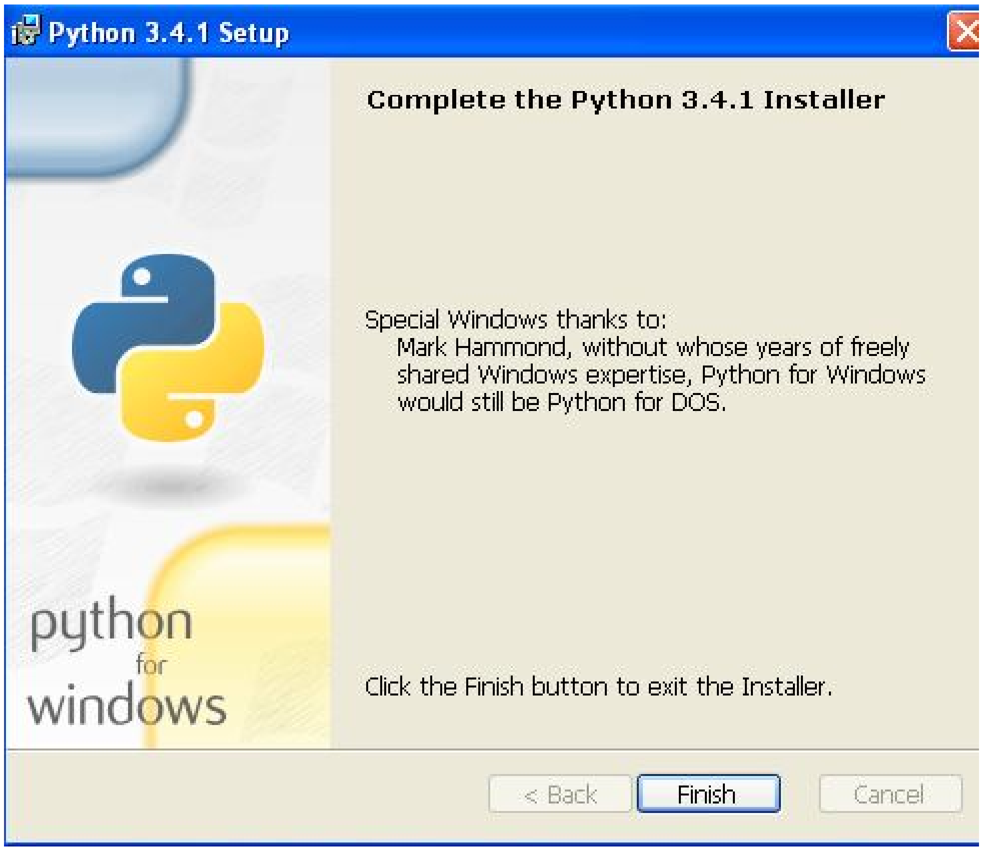

Python 3.4.1 Setup

**Install Python 3.4.1**

Please wait while the Installer installs Python 3.4.1. This may take several minutes.

Status: Pondering...

**Step7:** Click 'Finish' to exit the installer

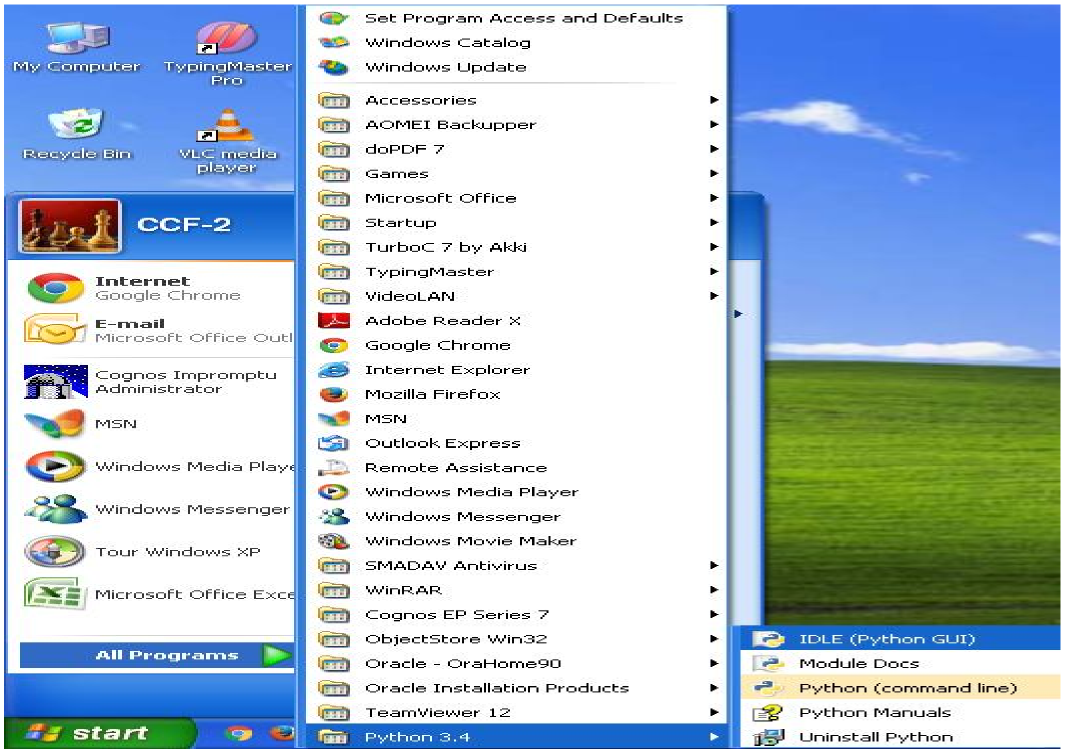

Python 3.4.1 Setup

Complete the Python 3.4.1 Installer

Special Windows thanks to:
Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

Click the Finish button to exit the Installer.

**Step8:** After installed, you should now have a Python menu choice. Start the program by choosing IDLE (Python GUI)



## Assignment-Cum-Tutorial Questions

**A. Objective Questions**

1. Literal is of the form a+bj is called__________
2. Identify the words which describes Python [ ]

   a)Interpreted b)simple c)reliable d)all of these
3. Python allows you to specify Unicode Text by prefixing the string with which character [ ]

   a)U b)R c)S d)A
4. Which of the following is a valid string literal [ ]

   a)”computer” b)’computer’ c)’’’computer’’’ d) all of these
5. Which of this is valid variable name in Python [ ]

   a) This is a variable b)This_is_a_variable c)This-is-a-variable d)^var
6. A Comments in python start with which symbol______________
7. All spaces and tabs with in a string are preserved in quotes [True/False]
8. Bitwise Operator can be applied on which datatype [ ]

   a)integer b)float c)string d)list
9. Identify valid assignment statements [ ]

   a)=b+1 b)a=a+1 c)a+b=10 d)a+1=1
10. __________operator perform logical negation on each bit of the operand.
11. What should be written in the blank to generate ZeroDivisionError in the case of (25+36)/(-8+__________)
12. Predict the output of the following program [ ]

    >>spam=”eggs”

    >>print(spam*3)

    a)spamspamspam b)eggseggseggs c)”spamspamspam” d)spam*3
13. Which of the following returns true [ ]

    a)>>>9=9 and 1==1

    b)>>>3==5 and 7==3

    c)>>>7!=1 and 5==5

    d)>>>4<1 and 1>6
14. Identify the valid numeric literals in Python [ ]

    a)5678 b)5,678 c)5678.0 d)0.5678 e)0.56+10
15. You can print string without using print function [True/False]

16. Predict the output of the following program [ ]

```
 >>>print (format(56.78901,’.3f’))
```

a)56.789 b)5.6789 c)0.56789 d)56789

17. The following statement will produce ___lines of output [ ]

```
>>print(‘Good\nMorning\nWorld\n---Bye’)
```

a)1 b)2 c)3 d)4

18. Identify the correct arithmetic expression in python [ ]

a)11(12+13) b)(5*6)(7+8) c)4*(3-2) d)5***3

19. Which line of code produce error [ ]

a)”one”+”2” b)’5’+6 c)3+4 d)”7”+’eight’

20. Predict the output of the following program [ ]

```
>>>print(abs(10-20)*3)
```

a)-30 b)30 c)-50 d)none of these

21.How many asterisks will be printed when the following code executes?

```
for x in [0, 1, 2, 3]:
    for y in [0, 1, 2, 3, 4]:
        print('*')
```

a.4 b.20 c.15 d)2

22.What is the output of following code?

```
str=Walking"
out = ""
for char in str:
        if char == "i":
            break
        if char == 'a':
            continue
            out += char
print(out)
```

23.What will the following code print?

```
for i in range(1,4):
      for j in range(1,4):
          print(i, j, end=' ')
```

Write the output of the following Python code:

```
for i in range(2,7,2):
        print(i * ‘$’, end="")
```

24.Find the output of the following program segments:

```
for x in range(1,4):
        for y in range(2,5):
              if x * y > 6:
              break
              print (x * y,end=' ')
```

25.What is the output of following code?

```
i=5
while True
          if i % o011==0:
                break
          i+=1
          print(i,end=" ")
```

26.What is the result of executing the following code?

```
number = 5
while number <= 5:
             if number < 5:
                 number = number + 1
             print(number,end=' ')
```

27.What is the output of the following loop______________.

```
for l in 'Jhon':
           if l == 'o':
                    pass
           print(l, end=", ")
```

28.What is the value of the var after the for loop completes its execution________.

```
var = 10
for i in range(10):
      for j in range(2, 10, 1):
                     if var % 2 == 0:
                                continue
                                var += 1
      var+=1
else:
     var+=1
 print(var)
```

29.Given the nested if-else below, what will be the value x when the code executed successfully_.

```
x = 0
a = 5
b = 5
if a > 0:
    if b < 0:
        x = x + 5
    elif a > 5:
        x = x + 4
    else:
        x = x + 3
else:
    x = x + 2
print(x)
```

30. Python uses ____________ to form a block of code.
31. Which part of if statement should be indented [ ]
    a) The first statement b) All the statements
    c) Statements within the if block d)None of these
32. Which of the following is placed after the **if** condition [ ]
    a) **;** b) **.** c) **:** d) **,**
33. elif and else blocks are optional [True/False]
34. How many lines will be printed by this code?

```
while False:
    print("hello")
```
[ ]

    a) 1 b) 0 c) 10 d) countless
35. __________ is a built-in function that is used to over a sequence of numbers.
36. Which statement is used to stop the current iteration of the loop and continue with the next one? [ ]
    a) pass b) break c)continue d)jump
37. Which statement is used to terminate the execution of the nearest enclosing loop in which it appears? [ ]
    a) pass b)break c)continue d)jump
38. Which statements indicates a NOP [ ]
    a) pass b)break c)continue d)jump

**39.** It is possible to use 'else suite' along with loops. [True/False]

**40.** **[ ]**

```
x=100
y=200
______ x>y _____
    print ("in if")
______
    print ("in else")
```

    a) if , else b) if ; else c)if : else : d) if | else
41. How many numbers will be printed? [ ]

```
i=5
while i>=0:
        print(i)
    i=i-1
```

a) 5 b) 6 c) 4 d)0

42. What is the output of the following code? [ ]

```
i = 1
while true:
    if i%3 == 0:
        break
    print(i)
    i + = 1
```

a) 1 2 b) 1 2 3 c) error d) none of the mentioned

43. What is the output of the following code? [ ]

```
for i in range(2.0):
    print(i)
```

a) 0.0 1.0 b) 0 1

c) error d) none of the mentioned

44. What is the output of the following code? [ ]

```
for i in range(10):
    if i == 5:
        break
    else:
        print(i)
else:
    print("here")
```

a) 0 1 2 3 4 here b) 0 1 2 3 4 5 here c) 0 1 2 3 4 d) 1 2 3 4 5

**B. Subjective Questions**

1. Describe the features of Python [BL1]
2. Differentiate between literals and variables in python.[BL2] (***November 2018*** **Supplementary**)
3. What are literals? Explain with the help of suitable examples? [BL1]
4. Explain the significance of Escape sequences with relevant examples[BL1]
5. Write briefly about Data types in Python[BL1]
6. Explain in detail about Membership and Identity Operators. [BL1]
7. How can the ternary operator used in python? [BL1]

(***April 2018 Regular***)

8. Give the operator precedence in python. [BL1] (***November 2018* Supplementary**)
9. Define Expression? Explain different types of Expressions supported by Python? [BL1]
10. Differentiate string with slicing operator. [BL1]
11. What is tuple? What are the different operations performed on tuple? Explain with an example? [BL1] (***November 2018* Supplementary**)
12. Write briefly about Type Conversion process in Python.Write the meaning for the following. str(x), chr(x), float(x), ord(x)[BL:2] (***November 2018* Supplementary**)
13. Momentum is calculated as, e=mc$^2$, where m is the mass of the object and c is the velocity. Write a Python program that accepts object's mass (in kilograms) and velocity (in meters per second) and displays its momentum. [BL:2]
14. a)Write a Python Program to convert temperature in Celsius to Fahrenheit[BL1]

    b) Write a Python Program to convert Fahrenheit to Celsius.[BL1]
15. Write a Python program to calculate the area of triangle using Heron's formula[BL1]

    ***Hint***: $\sqrt{s(s-a)(s-b)(s-c)}$
16. Evaluate the following Expression[BL2]

    a) True and False

    b) (100<0) and (100>20)

    c) not(true) and false

    d) not true and false or true

    e) not(100<0 or 100>20)

    f)100<0 and not 100>20
17. Give an appropriate boolean expression for the each of the following

    a)check if variable v is greater than or equal to 0,and less than 10[BL2]

    b)check if variable v is less than 10 and greater than or equal to 0,or it is equal to 20.[BL1]

    c)check if either the name 'cse' or 'it' appears in the list of names assigned to variable last_names.[BL2]

    d)check if the name 'cse' appears and the name 'it'does not appear in the list of last name assigned to variable last_names.[BL2]
18. Identify the datatype is best suitable to represent the following data values[BL2]

    a)Number of days in the year

b)The circumference of a rectangle

c)Yours father salary

d)Distance between moon and earth

e)Name of your best friend

f)Whether you go for the party

19. Explain Conditional Statements in Python with examples. [BL2]
20. Write syntax and logical flow for if-elif-else.[BL1]
21. Explain the significance of for loop with else using an example.[BL2]
22. Differentiate between counter-controlled loops and sentinel-controlled loops.[BL2]
23. Write the differences between iteration and recursion.[BL2]
24. Explain the utility of break and continue statements with the help of an example.[BL2]
25. What is pass statement in python?[BL1]
26. Explain with an example, how continue statement is used in python.[BL2]
27. Write a program to display multiplication tables from 1 to 10.[BL2]
28. Write a Python program that accepts a word from the user and reverse it[BL2]
29. Write a Python program that accepts an integer (n) and computes the value of n+nn+nnn. (Eg. If n=5, find 5+55+555).[BL2]
30. Write a program to find the factorial of a given number.[BL2]

-------------ooo----------------