UNIT – I

Python:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas other language use punctuation and it has fewer syntactical construction than other languages.

- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile the program before executing it. This is similar to PERL and PHP.
- Python is Interactive! – The user can directly interact with the interpreter to write a program.
- Python is object oriented: – Python supports object-oriented style or technique of programming that encapsulate code within objects.
- Python is a Beginner's language! – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python Features

Python's features include! –

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the user to pickup the

language quickly.

- Easy-to-read:- Python code is more clearly defined.
- Easy-to-maintain:- Python's source code is fairly easy-to-maintain.
- A broad standard library:- Python's library is very portable and cross-platform compatible on UNIX, Windows and Macintosh.
- Interactive Mode:- Python has support for an interactive mode which allows interactive testing and debugging of code.
- Portable:- Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable:- The user can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases:- Python provides interfaces to all major commercial databases.
- GUI Programming:- Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems such as windows, Macintosh and the X windows system of UNIX.
- Scalable: Python provides a better structure and support for large programs than shell scripting.

Need of Python Programming

- Software Quality

Python code is designed to be readable and hence reusable and maintainable. The uniformity of Python code makes it easy to understand, even if you did not write it. Python also has deep support for more advanced software reuse mechanisms, such as object-oriented and function programming.

- Developer Productivity

Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++ and Java. Python code is typically one-third to less to debug and less to maintain. Python programs also run immediately, without the lengthy compile and link steps required by some other tools, boosting programmer speed. Most Python programs run unchanged on all major computer platforms.

- Support Libraries

Python comes with a large collection of prebuilt and portable functionality, known as the standard library. This library supports an array of application-level programming tasks, from text pattern matching to network scripting.

- Component Integration:

Python scripts can easily communicate with other parts of an application using a variety of integration mechanism. Such integrations allow Python.

to be used as a product customization and extension tool.

- It's object oriented! - Python is an object oriented language. Its class model supports advanced notions such as polymorphism, operator overloading and multiple inheritance.

- It's free! -
  Python is freeware - something which has lately been come to be called open source software. The user can get the entire system for free over the Internet. There are no restrictions on copying it, embedding it in your systems.

- It's Portable! -
  Python is written in portable ANSI C and compiles and runs on virtually every major platform in use today. For example, it runs on UNIX Systems, MS-DOS, MS-Windows (95, 98, NT), Macintosh, and more.

- It's Powerful:
  From a feature perspective, Python is something of a hybrid. Its tool set place it between traditional scripting language (such as Tel, Scheme and Perl) and systems languages (such as C, C++ and Java). Python provides all the simplicity and ease of use of a scripting language, along with more advanced programming tools typically found in systems development language.

- Automatic Memory Management:
  Python automatically allocates and reclaims ("garbage collects") objects when no longer used and must grow and shrink on demand.

- Programming-in the large support
  Finally, for building large systems, Python includes tools such as modules classes and exceptions; they allow to organize systems into components.

- Database Programming
  Python's standard pickle module provides a simple object-persistence system: it allows programs to easily save and restore entire Python objects to files. There is even a portable SQL database API for Python that runs the same on a variety of underlying database systems and a system named gadfly that implement an SQL database for Python programs.

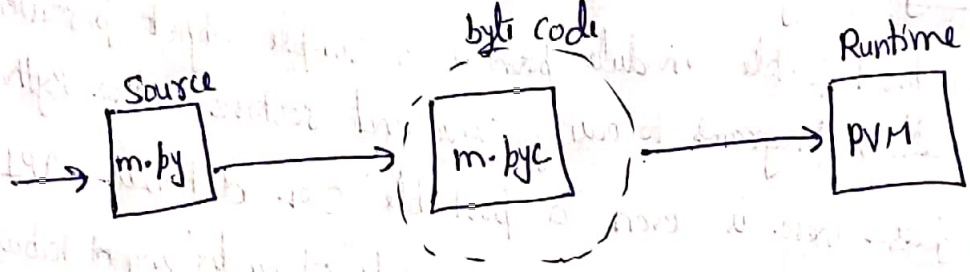
Byte Code Compilation:-

Python first compiles your source code into a format known as byte code. Compilation is simply a translation step and byte code is a lower level, platform-independent representation of source code. Python translates each of source statements into a group of byte code instructions by decomposing them into

individual steps. This byte code translation is performed to speed execution - byte code can be run much more quickly than the original source code statements.

The Python Virtual Machine.

Once the program has been compiled to byte code (or the byte code has been loaded from existing .pyc file), it is shipped off for execution to something generally known as the python virtual machine (PVM).



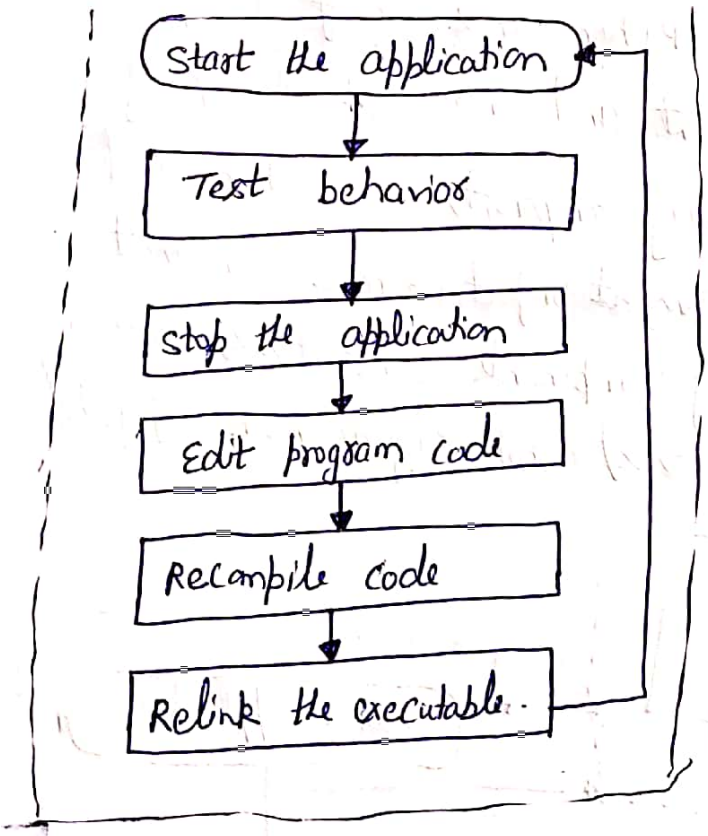
Applications of Python:-

1. Systems Programming
2. GUIs
3. Internet Scripting
4. Component Integration
5. Database Programming
6. Rapid prototyping.
7. Numeric and scientific Programming.

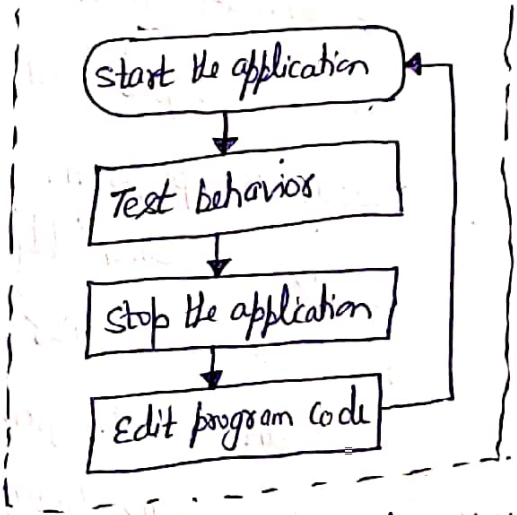
Python Development Cycle! -

Python's development cycle is dramatically shorter than that of traditional tools. In Python, there are no compile or link steps. Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made. And in cases where dynamic module reloading can be used, it's even possible to change and reload parts of a running program without stopping it at all. Figure shows Python's impact on the development cycle.

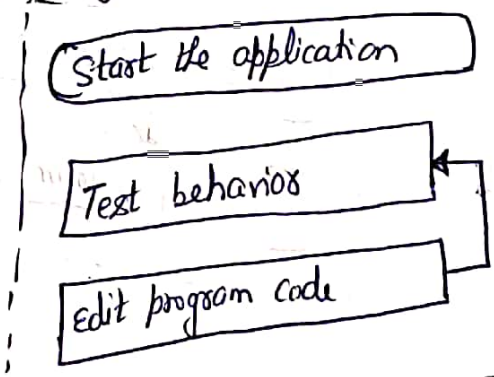
1. Traditional Development cycle -



2. Python's Development Cycle -



3. Python's Development Cycle with Module Reloading -



Because Python is interpreted, there's a rapid turnaround after program changes. And because Python's parser is embedded in Python-based systems, it's easy to modify programs at runtime. For example, we saw how GUI programs developed with Python allow developers to change the code that handles a button press while the GUI remains active; the effect of the code change may be observed immediately when the button is pressed again. There is no need to stop and rebuild.

More generally, the entire development process in Python is an exercise in rapid prototyping. Python lends itself to experimental, interactive program development and encourages developing systems incrementally by testing components in isolation and putting them together later. ~~anyfact~~ In fact, we have seen that we can switch from testing components (unit tests) to testing whole systems (integration tests) arbitrarily as shown in fig. below: -

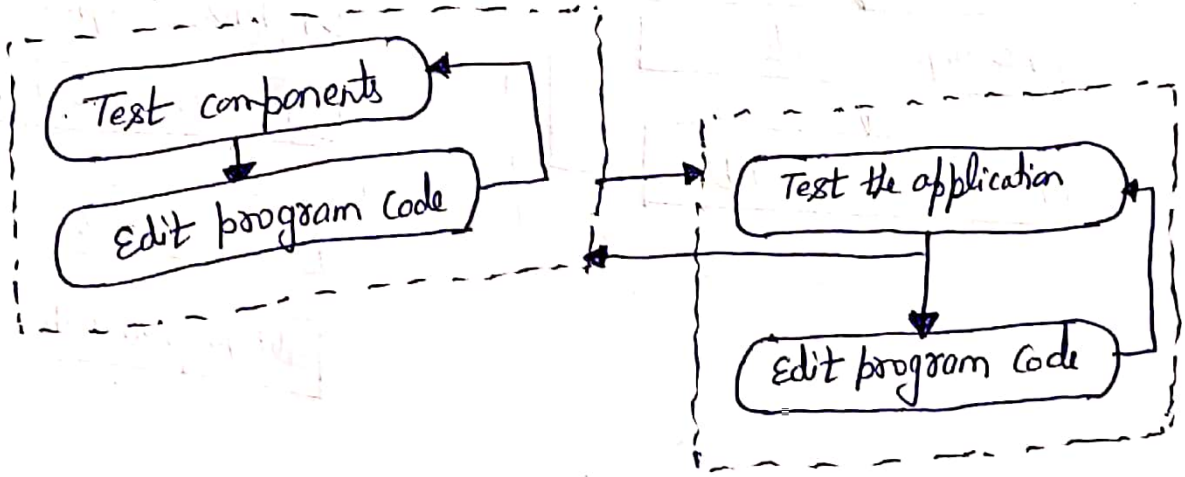


Fig:- Incremental development.

PYTHON IDE:-

An IDE (or integrated Development Environment) is a program dedicated to software development. As the name implies, IDEs integrate several tools specifically designed for software development.

These tools usually include:-

- An editor designed to handle code (with, for example, syntax highlighting and auto-completion).
- Build, execution and debugging tools.
- Some form of source control.

There are several IDEs available to create and execute Python programs. Using IDEs to develop programs makes very simple. The following are the popular IDEs specifically developed for Python Programming.

- PyCharm
- Spyder
- Jupyter
- Thonny
- Atom.

A program is written as a set of instructions. The instructions written in programs are termed as statements.

* **Statements:-**

A python statement is the smallest program unit. It is a complete instruction executed by the Python interpreter. A program may consist of more than one statement.

The various types of statement in Python are as:-

1. **Simple Statements:-**

A simple statement is the basic unit of execution that ends within a single line. Every simple statement starts with a new line and ends within that same line. There are different types of simple statements such as the expression statement, assignment statement, print statement, input statement, import statement and comment statement which are explained as follows:-

- **Expression Statement:-**

Expression statement are used to compute and write values to some variables or can be used to call a function. The expression may basically perform some action which consists of a combination of an operator and operand. Expression statements include assignment statements, methods or function calling statements and so on.

For example, a=10 and a=c+d are few examples of expression statements in Python.

- **Output Statement:-**

The output statement is used to print the results generated by the program on the screen. In Python, the print statement was used in version 2.x to display the output. This print statement is replaced by the print function in the 3.x version. The print() function is a function used to display the output on the screen by taking parameters. It always shift the cursor to the next line after printing results on the screen. The print() function can be used in many ways, as follows:

(i) print("string"):- This function takes the string parameter in double quotes. This statement directly prints the string enclosed in double quotes on the screen. For eg:-

```
>>> print("Hello world")
Hello world.  — o/p.
```

The preceding line prints the message Hello world on the screen and takes the cursor to the next line after printing the message.

(ii) print(variable list):- This function can also be used to print the values stored in the variables. For example,

```
>>> a = 10
>>> print(a)
10  —  o/p.
```

If we want to print the value of the variable with some message, we need to pass the message to be print in double quotes or single quotes and also the name of the variable. For example.

```
>>> a = 10
>>> print ("value of a =", a)
```

o/p → value of a = 10

(iii) print(object) :- we can also pass objects such as list, tuple and dictionary to the print function. The print ~~statement~~ function directly prints all the values stored in the object without the need of writing the loop statement. For example,

```
>>> A = [10, 15, 20, 25, 30]
>>> print ("Values in the list are: ", A)
```

Values in the list are: [10, 15, 20, 25, 30].

(iv) print (formatting string) :- The function is used to display the string in a proper formatted manner. For example.

```
>>> a = 10
>>> b = 15
>>> print ("value of a is: %d and the value of b is: %d" %(a,b))
```

o/p → value of a is 10 and value of b is 15.

The table given below shows the different format specifiers used in Python:

Format Specifier	Meaning
%d or %i	signed integer decimal
%o	unsigned octal
%x or %X	unsigned hexa
%e or %E	Floating point exponential format
%f or %F	Floating point decimal format

For Example:-

```
>>> print("Exponential form is: %e" % (3.122342))
```

o/p → Exponential form is: 3.122342e+00

```
>>> print("Floating point value is: %f" % (23424.23423))
```

o/p → Floating point value is: 23424.234230.

* we can also use escape sequences in Python to format the string. The Python interpreter converts the code of an escape sequence while execution.

For example:-

```
>>> print("Hello \n world")
```

o/p →
```
Hello
world.
```

- **Input Statement**

This input() function is used in the 3.x version and takes input from the outside world and passes it to the Python program. This function takes the value from the keyboard and returns the inputted value in the string form. For example:

```
>>> a = input("Enter the value of the variable :")
```

o/p → Enter the value of the variable: xyz.

The input function prompts a message to enter the value and when the user enters the value and presses the Enter key only then the input function takes the value and assigns that value to the variable on the left hand side. The inputted value is xyz.

* The input function only takes the string values, but if want to take values in int, float, list, tuple or any other type of values, we need to convert the inputted string to the appropriate data type by applying the function of the type accordingly. For example, to convert the string to integer int() function is used, for floating point float() function, for list conversion list() function is used. For example:-

```
>>> a = int(input("Enter the integer value :"))
    Enter the integer value : 23.
>>>
```

- **Import statement:**

In python programming language, the programs are written in modules. The python code written in one module gain access to the code in another module by the process of importing it. The import statement is used to import the code of another module to the current module. This statement is required to use the functions or variables defined in the other modules.

For example, to use the square root function of the Math module, need to import the Math module in program; otherwise, it will return an error.

```
>>> import math
>>> math.sqrt(234)
```

o/p → 15.297058540778355

- **Comment statement:**

The comments are the non-executable statements that are written in the program for the understanding of the programmer, but they do not display the interpreter in the output window. The comments in the programs are created using the '#' symbol and the comment statement completes within the single line. For example:

```
# Program to add two numbers.
```

2. Control Statements

Control statements are a set of statements that are responsible to change the flow of execution of the program. There are mainly three types of control statements :

(i) Decision Control Statement

Decision control statements are those block of statements that execute a particular block according to the Boolean expression evaluation. The if, if....else, elif are decision control statements.

(ii) Iterative control statement: -

Iterative control statements are loop statements that are repeatedly executed a number of times until the stated condition becomes true. For and while are the iterative control statements in Python.

(iii) Branching statement :

Branching statement are the jumping statements that transfer the control to another block of statements according to a particular condition. The break, continue, pass and assert are the branching statements.

Expressions:-

An expression is a statement that consists of an operand and operator. The operand in the expression can be a variable, literal or a call to the function that returns the value etc. and the operator can be any operator such as arithmetic, relational, assignment and so on. The simplest expression is the assignment expression in which the value on the right-hand side is assigned to the variable on the left-hand side of the = operator. For example,

```
>>> A = 10
>>> B = 20.3
```

The above two statements are assignment expressions that assign the value 10 to the variable A and 20.3 to the variable B.

Some more examples are

```
d = a+b/c*a.
d = e+f
```

Assignment Statement

Python assignment statement for a while to assign objects to names. In its basic form, write the target of an assignment on

the left of an equal sign and the object to be assigned on the right. The target on the left may be a name or object component and the object on the right can be an arbitrary expression that computes an object.

The different assignment statement forms in Python and their syntax are! —

Operation	Interpretation
• Spam = 'Spam'	Basic form
• Spam, ham = 'yum', 'YUM'	Tuple assignment (positional)
• [spam, ham] = ['yum', 'YUM']	List assignment (positional).
• Spam += 42	Augmented assignment (equivalent to Spam = Spam + 42)

OPERATORS -

An operator is a special symbol that tells the interpreter to perform a specific operation on the operands. The operands can be literals, variables or expressions which are required to be calculated. The operator and operand when combined to perform a certain operation, it becomes an expression.

For example, in expression a * b, a and b are the variable and the '*' sign is the operator that specifies the type of operation performed on the variables.

Python provides a rich set of operators which are as follows:-

1. Arithmetic operators:-

The arithmetic operators perform basic arithmetic operations. All arithmetic operators are binary operators because they can perform operations on two operands.

- '+' operator: The plus operator is used to add two value and two concatinates two object.

  For example 10+20 o/p → 30

  Hello+world o/p → Hello world.

- '−' operator: - It is used to subtract one value from another

  For example 20 − 10 o/p → 10.

- '*' operator: It is used to multiply one value with the other, but when it applies to objects like string, list, it repeats the object number of times according to the number mentioned after the '*' operator.

  For example 10 * 20 o/p → 200

  Hello*2 o/p → Hello Hello.

- '/' operator :- The division operator performs the division operation on two integers or floating point values.

For example

5/2 o/p → 2.5.

- '**' operator : The exponential operator is the most powerful operator in Python that is used to find the power of a literal written on the left hand side of the operator.

For example :-

5**3 o/p → 125.

- '//' operator : The floor division operator is used to return the integer values obtained after applying the division operator. It is used when we want to skip the values after the decimal point.

For example

5//2 o/p → 2.

- % operator :- The modulus operator is used to find the remainder of the numbers after division ~~when it~~ and it formats the strings in the print statement when it is applied to the strings.

For example:

5%2 o/p → 1.

2. Relational operators:-

The relational operators are used to compare the values stored in two variables. These operators are also termed as comparision operators. Relational operators are binary operator that compare the value of the operands and return a Boolean result that is, either True or false.

- '<' operator: The less than operator applies to two operands and evaluates to true if the first operand is less than the second operand; otherwise, it evaluates to false.

For eg:- $10 < 20$ o/p → True.

- '>' operator: The greater than operator applies to two operands and and evaluates to true if the first operand is greater than the second operand; otherwise, it evaluates to false.

for eg:- $10 > 20$ o/p → False.

- '<=' operator:- The less than equal to operator applies to two operands and evaluates to true if the first operand is less than or equal to the second operand; otherwise, it evaluates to false.

for eg:- $20 <= 20$ o/p → True.

- '>=' operator: The greater than equal to operator applies to two operands and evaluates to true if the first operand is greater than or equal to the second operand; otherwise, it evaluates to false.

- '==' operator: The equality operator applies to two operands and evaluates to true if first operand value is equal to second operand value; otherwise, it evaluates to false.

For eg:-

```
A = 5
B = 10
A==B        o/p → False.
```

- '!=' operator:- The not equal operator applies to two operands and evaluates to true if the first operand is not equal to the second operand; otherwise, it evaluates to false.

* These all operators operates on Integer, floating point, string, tuple, list, dictionary, boolean.

3. Logical operators.

The logical operators are used to combine one or more relational expressions that result in complex relational operations. The result of the logical operator is evaluated in terms of True or

or false according to the result of the logical expression. The logical operators are Boolean operators that operate on two relational expressions. There are three logical operators provided by the Python which are as follows! -

- 'and' operator :- The logical and operator is the binary operator that operates on two relational expressions and result in TRUE if both the relational evaluate to TRUE; otherwise, it evaluates to false.

For example 10 < 20 and 20 < 30 o/p TRUE.

- 'or' operator :- The logical or operator is the binary operator that operates on two relational expressions and result in TRUE if one of relational expression evaluates to True; otherwise, it evaluates to false.

For example :- 10 > 20 or 20 > 30 o/p → False.

- 'not' operator :- The logical not operator is the unary operator that operates on a single relational expression. and reverse the result obtained after evaluation of of the relational expression.

For example :- Not 20 > 30 o/p → TRUE.

* These all operators operate on Integer, floating point, string, list, tuple, dictionary, boolean.

4. Bitwise operators:-

Python provides the bit manipulation operators to directly operate on the bits or binary numbers directly. When we use bitwise operators on the operands, the operands are firstly converted to bits and then the operation is performed on the bit directly. Bitwise operators are binary operators and unary operators that can be operated on two operands or one operand.

The bitwise operators are as follows:-

- Bitwise And (&) operator:- Bitwise AND operators combine two operands by performing the logical and operation on individual bits. It result in bit 1 if both the corresponding bits are 1 and 0 if both or one of the corresponding bits is 0.

For example:- To perform bitwise AND operation between 25 and 45.

Binary of 25 is 011001

Binary of 45 is 101101

The result is

25	011001
45	101101
25 and 45	001001

- **Bitwise OR (|) operator:-**

Bitwise OR operator combine two operands by performing the logical or operation on individual bits. It result in bit 1 if both the corresponding bits are 1 or either one of bit is 1 and 0 if both the corresponding bits are 0.

For example:- 25 | 45

25	011001
45	101101
25 | 45	111101

- **Bitwise XOR (^) operator:-**

Bitwise XOR operators combines two operand by performing the XOR operation on individual bits. It results in bit 1 if one of the corresponding bit is 1 and 0 if both the bits are same.

For example:- 25 ^ 45

25	011001
45	101101
25^45	110100

- **Bitwise complement (~) operator:-** The bitwise ~ operator is the unary operator that operates by reversing all the bits in the operand by performing the logical not operation on individual bits. It results

in bit 1 if the corresponding bit is 0 and the result is 0 if the corresponding bit is 1.

For example ~25

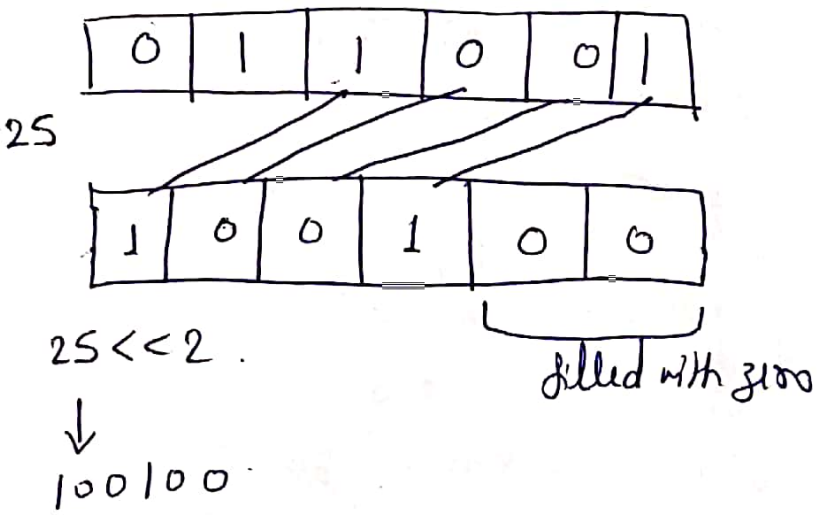
25	011001
~25	100110

- Bitwise left shift (<<) operator:- It is the unary operator that shifts the number of bits to its left position. The number of bits is shifted according to the values specified on the right-hand side of the operator. After performing the left shift operation on the bits, the rightmost bits are filled with 0.

For example:- to perform the 2-bit left operation using the bitwise left shift operator on 25, we first need to convert 25 to bits and then perform the left shift operation on the bits.

Binary of 25 is 011001.

25<<2



↓

100100

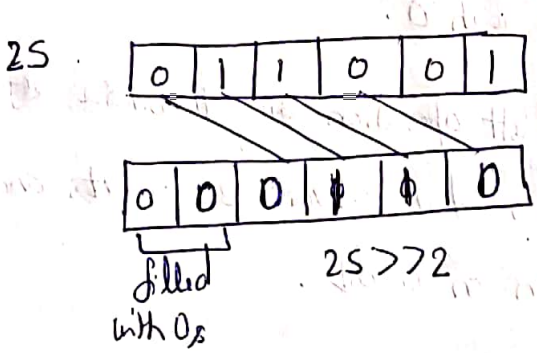
- **Bitwise right shift (>>) operator :-**

The bitwise >> operator is the unary operator that shifts the number of bits to its right position. The number of bits is shifted according to the values specified on the right-hand side of the operator. After performing the right shift operation on the bits, the leftmost bits are filled with 0.

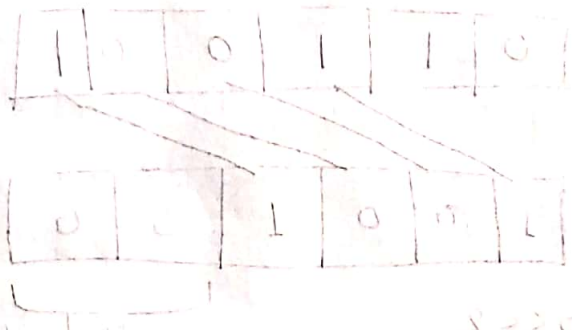
For example:-

25>>2

25 — 011001



25>>2 — 000110



5. Assignment operator:

The assignment operator is used to store the value on the right hand side of the expression on the left hand side variable in the expression. The operand on the right hand side can be a variable, literal or the function call. The assignment operator can also be used to assign the value to the left hand side variable by performing arithmetic operations.

Following are the assignment operator provided by the Python:-

- **'=' operator:-** Assign the value on the right-hand side of the = sign to the variable at the left hand side of the = operator.

For example, `a = 10` Literal 10 is assigned to the variable a.

- **'+=' operator:** Perform the addition operation between the variable on the left hand side of the = operator and the variable on the right hand side of the += operator and assign the result to the left hand side of the = operator.

For example:- `a+=10`

The value of a is added to 1 and the result is assigned to a.

- **'-=' operator:** Perform the subtraction operation between the variable on the left hand side of the = operator and the variable.

on the right hand side of the = operator and assign the result to the left hand side of the = operator.

For example a -= 10. The value of a is subtracted by 10 and the result is assigned to a.

- '*=' operator:- Perform the multiplication operation between the variable on the left hand side of the = operator and the variable on the right hand side of the = operator and assigns the result to the left hand side of the = operator.

For example a*=10 The value of a is multiplied by 10 and the result is assigned to the variable a.

- '/=' operator:- Perform the division operation between the variable on the left hand side of the = operator and the variable on the right hand side of the = operator and assigns the result to the left hand side of the = operator.

For example:- a/=10 The value of a is divided by 10 and the result is assigned to the variable a.

- '**=' operator:- Perform the exponential operation between the variable on the left hand side of the = operator and the variable on the right hand side of the = operator and assigns the

result to the left hand side of the = operator.

For example $a^{**}= 2$ The value of a is powered by 2 and the result is assigned to the variable a.

- '//=' operator Perform the floor division operation between the variable on the left hand side of the = operator and the variable on the right hand side of the = operator and assign the result to the left hand side of the = operator.

For example :- $a//=10$ The value of a is floor divided by 10 and the result is assigned to the variable a.

- '%=' operator :- Performs the modulus operation between the variable on the left hand side of the = operator and the variable on the right hand side of the = operator and assign the result to the left hand side of the = operator.

For example :- $a\%=10$ Returns the remainder value after dividing a by 10 and result is assigned to the variable a.

Operator Precedence :-

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

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order one want. Since expressions in parentheses are evaluated first, 2*(3-1) is 4. We can also use paranthesis to make an expression easier to read.
- Exponentiation has the next highest precedence, so 1 + 2**3 is 9
- Multiplication and Division have the higher precedence than Addition and subtraction. So 2*3-1 is 5 not 4.
- operators with the same precedence are evaluated from left to right. So in the expression degrees / 2 * pi, the division happens first and the result is multiplied by pi. To divide by 2π, can use paranthesis or write degrees / 2 / pi.