UNIT - I

1) Python and its Features
2) Various IDEs of Python
3) Variables and its Scope
4) Input and Output Statement
5) Operators
6) Operator Precedence
7) Selective and Iterative Statements
8) Strings

1. The Python Programming Language

- Python is a **high-level, interpreted, and general-purpose programming language** that was first released in 1991 by **Guido van Rossum.**
- It is designed to emphasize **code readability**, with a syntax that allows programmers to express concepts in fewer lines of code than would be possible in languages like C++ or Java.
- Python is used for a **wide range of applications, including web development, data analysis, artificial intelligence, machine learning, and scientific computing.**
- It is known for its **simplicity, flexibility, and ease of use, making it a popular choice among developers of all skill levels.**
- Python is an **open-source language**, which means that the source code is available to the public, and anyone can contribute to its development.
- The creator of Python, **Guido van Rossum, was a big fan of Monty Python's Flying Circus, a popular British television comedy show that aired in the 1970s.**

1. Python Features

1. Easy to Code

- Python is a very high-level programming language, yet it is effortless to learn because Python syntax is very easy, as compared to other popular languages like C, C++, and Java.

2. Easy to Read

- Python code looks like simple English words.
- There is no use of semicolons or brackets, and the indentations define the code block.

3. Free and Open-Source

- Python is developed under an OSI-approved open source license.
- Hence, it is completely free to use, even for commercial purposes.
- It doesn't cost anything to download Python or to include it in your application.
- It can also be freely modified and re-distributed. Python can be downloaded from the official Python website.

4. Robust Standard Library

- Python has an extensive standard library available for anyone to use.
- This means that programmers don’t have to write their code for every single thing unlike other programming languages.

1. Python Features

5. Interpreted

- When a programming language is interpreted, it means that the source code is executed line by line, and not all at once.

6. Portable

- Python is portable in the sense that the same code can be used on different machines.
- Suppose you write a Python code on a Mac.
- If you want to run it on Windows or Linux later, you don't have to make any changes to it.

7. Object-Oriented and Procedure-Oriented

- A programming language is object-oriented if it focuses design around data and objects, rather than functions and logic.
- On the contrary, a programming language is procedure-oriented if it focuses more on functions (code that can be reused).
- One of the critical Python features is that it supports both object-oriented and procedure-oriented programming.

1. Python Features

8. Expressive

- Python needs to use only a few lines of code to perform complex tasks.
- For example, to display Hello World, you simply need to type one line - print(“Hello World”).
- Other languages like Java or C would take up multiple lines to execute this.

9. Support for GUI

- One of the key aspects of any programming language is support for GUI .
- Python offers various toolkits, such as Tkinter, wxPython and JPython, which allows for fast and easy GUI's development.

10. Dynamically Typed

- Many programming languages need to declare the type of the variable before using it .
- With Python, the type of the variable can be decided during runtime. This makes Python a dynamically typed language.

1. Python Features

11. Simplify Complex Software Development

- Python can be used to develop both desktop and web apps and complex scientific and numerical applications.
- Python's data analysis features help to create custom big data solutions without so much time and effort.
- Python also helps to use the Python data visualization libraries and APIs to present data in a more appealing way.
- Several advanced software developers use Python to accomplish high-end AI and natural language processing tasks.

2. Python IDE's

- IDE stands for Integrated Development Environment . It is a programming environment that contains a lot of things in a single package i.e. **code editor, compiler, debugger**.
- It has a user-friendly interface consisting of an editor and a compiler. We can write the code in the editor window and compile it using the compiler.
- Consequently, we can run it to check the output of the program on the output terminal.
- IDEs increase programmer productivity by introducing features **like editing source code, building executables, and debugging.**
- IDEs and code editors are tools that software developers use to write and edit code.
  - ✓ IDEs, or Integrated Development Environments, are usually more feature-rich and include tools for debugging, building and deploying code.
  - ✓ Code editors are generally more straightforward and focused on code editing. Many developers use IDEs and code editors, depending on the task

2. Features of IDE's

1. **Code editor**: IDEs provide a code editor with advanced features such as syntax highlighting, code completion, and code refactoring. These features help developers write code more efficiently and accurately.
2. **Integrated debugger**: IDEs have a built-in debugger that allows developers to identify and fix errors in their code. This feature saves time by reducing the need for manual debugging.
3. **Version control integration**: IDEs can be integrated with version control systems such as Git or SVN, allowing developers to manage changes to their code and collaborate with others.
4. **Project management**: IDEs provide tools for managing projects, such as file organization, project templates, and code generation.
5. **Build automation**: IDEs can automate the build process, allowing developers to compile and run their code without having to use command-line tools.
6. **Testing tools**: IDEs provide testing tools that help developers test their code and identify defects early in the development cycle.
7. **Plugins and extensions**: IDEs often support plugins and extensions that can be used to extend their functionality, adding new features or integrating with other tools.

Overall, IDEs provide a complete environment for software development, making it easier and more efficient for developers to write, test, and debug their code.

LIST OF PYTHON IDE'S

IDLE	**PyCharm**	**Spyder**	**Thonny**
Jupyter	PyDev	Vim	GNU Emacs
Wing	Geany	Sublime Text	Eric Python

2. IDLE

- Python IDLE is an integrated development environment **(IDE) for Python** that provides an **interactive mode and a script mode** for writing and running Python code.
- Here are the differences between **interactive mode and script mode in Python IDLE**:

1) **Interactive mode**: In interactive mode, you can type Python code directly into the IDLE shell, and the code is executed immediately after you press the Enter key. This allows you to experiment with Python code, test out ideas, and get immediate feedback.
2) **Script mode:** In script mode, you write Python code in a file with a .py extension, and then run the file from within IDLE or from the command line. This mode is used for writing larger programs that require multiple lines of code and need to be saved for later use.

3. Variables and its Scope

- A variable is a named identifier that represents a memory location used to store a value.
- It is a fundamental concept in programming and allows you to store and manipulate data.
- Variables are used to store different types of values, such as numbers, text, or objects.
- To create a variable in Python, you simply assign a value to a name using the assignment operator (**=**). Here's an example:

```python
message = "Hello, World!"
```

- In the above example, the variable **message** is created and assigned the value **"Hello, World!"**.
- Now, you can use the variable **message** to access or modify the stored value.
- Python is a dynamically typed language, which means no need to explicitly declare the type of a variable.
- The type of the variable is inferred based on the value assigned to it.
- For example, in the previous code snippet, the variable **message** is inferred to be of type **str** (string) because it was assigned a string value.
- Variables can be reassigned with s a different value or change its type later in the code:

3. Variables and its Scope

```python
message = "Hello, World!"
print(message)  # Output: Hello, World!

message = "Welcome to Python"
print(message)  # Output: Welcome to Python

message = 42
print(message)  # Output: 42
```

- In this example, the variable **message** is initially assigned a string value, then reassigned to a different string value, and finally reassigned to an integer value.
- Variables provide a way to store and manipulate data throughout your program, making your code more flexible and dynamic. They play a crucial role in programming and are widely used to store and retrieve information.

4. Input and Output Statements

input() function:

- This function prompts the user for input from the console.
- It returns the user's input as a string.
- It takes prompt message as an argument to the function.

```
name = input("Enter your name: ")
print("Hello,", name, "! Welcome!")
```

Output

```
Enter your name: GeeksforGeeksHello, GeeksforGeeks ! Welcome!
```

4. Input and Output Statements

To get Integer Input

```
# Prompting the user for input
age_input = input("Enter your age: ")

# Converting the input to an integer
age = int(age_input)

# Checking conditions based on user input
if age < 0:
    print("Please enter a valid age.")
elif age < 18:
    print("You are a minor.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

Output

```
Enter your age: 22
You are an adult.
```

To get Floating Point Input

```
# Taking input as float
# Typecasting to float
price = float(input("Price of each rose?: "))
print(price)
```

Output

```
Price of each rose?: 50.3050.3
```

4. Input and Output Statements

To get Multiple Input

To take multiple inputs from the user in a single line, the values entered by the user are split into separate variables for each value using the split() method.

```
# taking two inputs at a time
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)

# taking three inputs at a time
x, y, z = input("Enter three values: ").split()
print("Total number of students: ", x)
print("Number of boys is : ", y)
print("Number of girls is : ", z)
```

Output

```
Enter two values: 5 10
Number of boys:  5
Number of girls:  10
Enter three values: 5 10 15
Total number of students:  5
Number of boys is :  10
Number of girls is :  15
```

4. Input and Output Statements

print() function:

- This function displays output to the console.
- multiple arguments can be passed to print(), separated by commas.
- print() automatically adds a newline character at the end of the output.

```
print("Hello, World!")
```

Output

```
Hello, World!
```

4. Input and Output Statements

print single and multiple variables

Print statement can be used to print single and multiple variables

```
# Single variable
s = "Bob"
print(s)

# Multiple Variables
s = "Alice"
age = 25
city = "New York"
print(s, age, city)
```

Output

```
Bob
Alice 25 New York
```

String Concatenation

You can use the + operator to concatenate strings, but this can be less readable for complex formatting.

```
name = "Smith"
age = 25
print("My name is " + name + " and I am " + str(age) + " years old.")
```

Output

My name is Smith and I am 25 years old.

4. Input and Output Statements

Formatted Output

- ✓ f-strings(Formatted string literals)
- ✓ These provide a concise and readable way to embed expressions within a string
- ✓ Prefix the string with f and enclose expressions in curly braces {}

Python

```
name = "Alice"
age = 30
print(f"My name is {name} and I am {age} years old.")
```

Output

```
My name is Alice and I am 30 years old.
```

Controlling output behavior

- `sep` argument: Specifies the separator between arguments (default is a space).
- `end` argument: Specifies what to print at the end of the line (default is a newline `\n`).

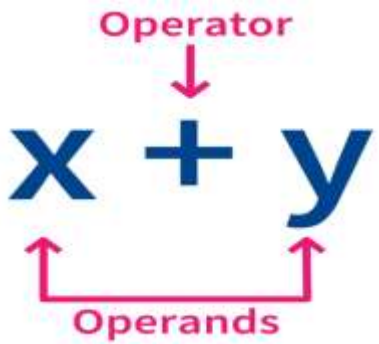
```
# end Parameter with '@'
print("Python", end='@')
print("GeeksforGeeks")


# Seprating with Comma
print('G', 'F', 'G', sep='')

# for formatting a date
print('09', '12', '2016', sep='-')

# another example
print('pratik', 'geeksforgeeks', sep='@')
```

Output

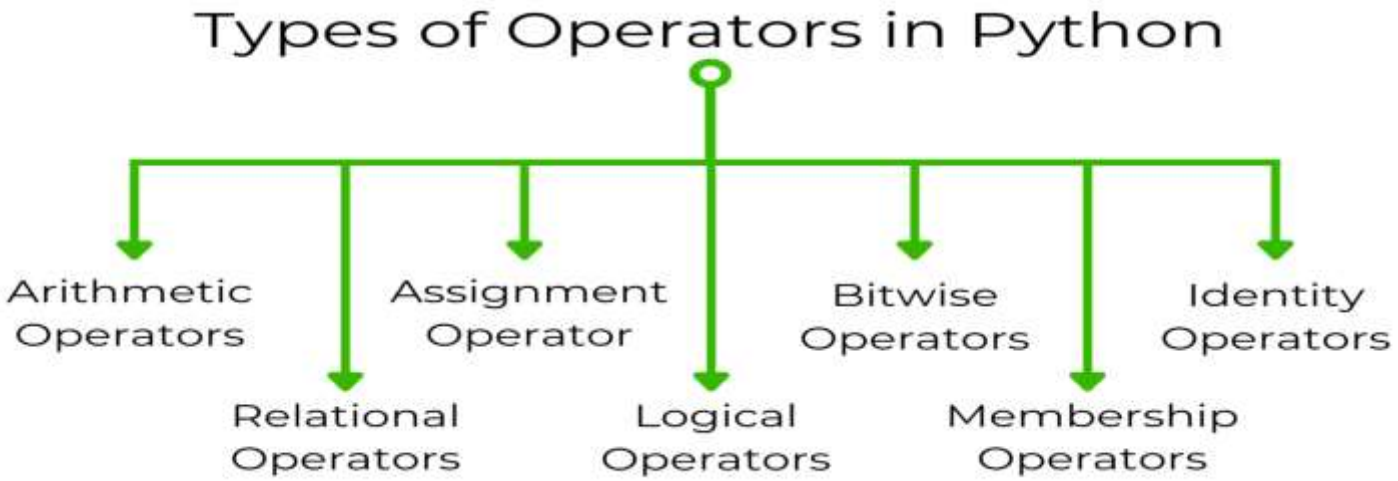
```
Python@GeeksforGeeks
GFG
09-12-2016
pratik@geeksforgeeks
```

5. Operators

- ✓ Operators In General Are Used To Perform Operations On Values And Variables.
- ✓ **Operators:** These Are The Special Symbols. eg- + , * , /,
- ✓ **Operand:** It Is The Value or Variable On Which The Operator Is Applied. Eg :



5. Operators



Operators in Python

Operators	Type
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
AND, OR, NOT	Logical operator
&, |, <<, >>, -, ^	Bitwise operator
=, +=, -=, *=, %=	Assignment operator

Arithmetic Operators

Python Arithmetic operators are used to perform basic mathematical operations like **addition, subtraction, multiplication** and **division**.

5. Operators

Arithmetic Operators

Python Arithmetic operators are used to perform basic mathematical operations like **addition, subtraction, multiplication** and **division**.

```
# Variables
a = 15
b = 4

# Addition
print("Addition:", a + b)

# Subtraction
print("Subtraction:", a - b)

# Multiplication
print("Multiplication:", a * b)

# Division
print("Division:", a / b)

# Floor Division
print("Floor Division:", a // b)

# Modulus
print("Modulus:", a % b)

# Exponentiation
print("Exponentiation:", a ** b)
```

Output

```
Addition: 19
Subtraction: 11
Multiplication: 60
Division: 3.75
Floor Division: 3
Modulus: 3
Exponentiation: 50625
```

5. Operators

Comparison Operators

In Python Comparison of Relational operators compares the values. It either returns **True** or **False** according to the condition.

```
a = 13
b = 33

print(a > b)
print(a < b)
print(a == b)
print(a != b)
print(a >= b)
print(a <= b)
```

Output

```
False
True
False
True
False
True
```

5. Operators

Logical Operators

Python Logical operators perform **Logical AND**, **Logical OR** and **Logical NOT** operations. It is used to combine conditional statements.

```
a = True
b = False
print(a and b)
print(a or b)
print(not a
```

Output

```
False
True
False
```

5. Operators

Bitwise Operators

Python Bitwise operators act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

Bitwise Operators in Python are as follows:

1.Bitwise NOT

2.Bitwise Shift

3.Bitwise AND

4.Bitwise XOR

5.Bitwise OR

```
a = 10
b = 4

print(a & b)
print(a | b)
print(~a)
print(a ^ b)
print(a >> 2)
print(a << 2)
```

Output

```
0
14
-11
14
2
40
```

5. Operators

Assignment Operators

Python Assignment operators are used to assign values to the variables. This operator is used to assign the value of the right side of the expression to the left side operand.

```
a = 10
b = a
print(b)
b += a
print(b)
b -= a
print(b)
b *= a
print(b)
b <<= a
print(b)
```

Output

```
10
20
10
100
102400
```

5. Operators

Identity Operators

- ✓ In Python, **is** and **is not** are the identity operators both are used to check if two values are located on the same part of the memory.
- ✓ Two variables that are equal do not imply that they are identical.
- ✓ **is** True if the operands are identical **is not** True if the operands are not identical

```
a = 10
b = 20
c = a

print(a is not b)
print(a is c)
```

Output

```
True
True
```

5. Operators

Membership Operators

- ✓ In Python, **in** and **not in** are the membership operators that are used to test whether a value or variable is in a sequence.
- ✓ **in** True if value is found in the sequence **not in** True if value is not found in the sequence

```
x = 24
y = 20
list = [10, 20, 30, 40, 50]

if (x not in list):
    print("x is NOT present in given list")
else:
    print("x is present in given list")

if (y in list):
    print("y is present in given list")
else:
    print("y is NOT present in given list")
```

Output

```
x is NOT present in given list
y is present in given list
```

6. Operators Precedence

Operator	Symbol	Description
Parentheses	( )	Parentheses
Exponent	**	Exponentiation
	~X	Bitwise Nor
Arithmetic	*, /, //, %	Multiplication, Divide, Floor Division, Mod
	+ , -	Addition, Subtraction
Bitwise	&	Bitwise And
	^	Bitwise Not
	|	Bitwise Or
Relational, Identity	<, <=, >, >=, !=, ==, is, is not	Comparison, Identity Operators
Logical Operator	not	Not (Logical Operator)
	and	And (Logical Operator)
	or	Or (Logical Operator)

Highest

Lowest

6. Operators Precedence

1. Parentheses:

```python
result = (2 + 3) * 4  # Parentheses force addition first
print(result)  # Output: 20
```

Without parentheses:

```python
result = 2 + 3 * 4  # Multiplication happens before addition
print(result)  # Output: 14
```

2. Exponentiation:

```python
result = 2 ** 3 * 2  # Exponentiation happens before multiplication
print(result)  # Output: 16
```

3. Multiplication and Division vs. Addition and Subtraction:

```python
result = 10 + 5 * 2 - 8 / 4  # Multiplication and division before addition and
print(result)  # Output: 18.0
```

Calculation breakdown:

- `5 * 2 = 10`
- `8 / 4 = 2.0`
- `10 + 10 - 2.0 = 18.0`

4. Logical Operators:

```python
result = True and not False or False  # 'not' before 'and' before 'or'
print(result)  # Output: True
```

Calculation breakdown:

- `not False = True`
- `True and True = True`
- `True or False = True`

6. Operators Precedence

5. Comparison Operators:

Python

```
result = 5 > 3 and 2 < 4
print(result) #output: True
```

Comparisons are done before the and operator.

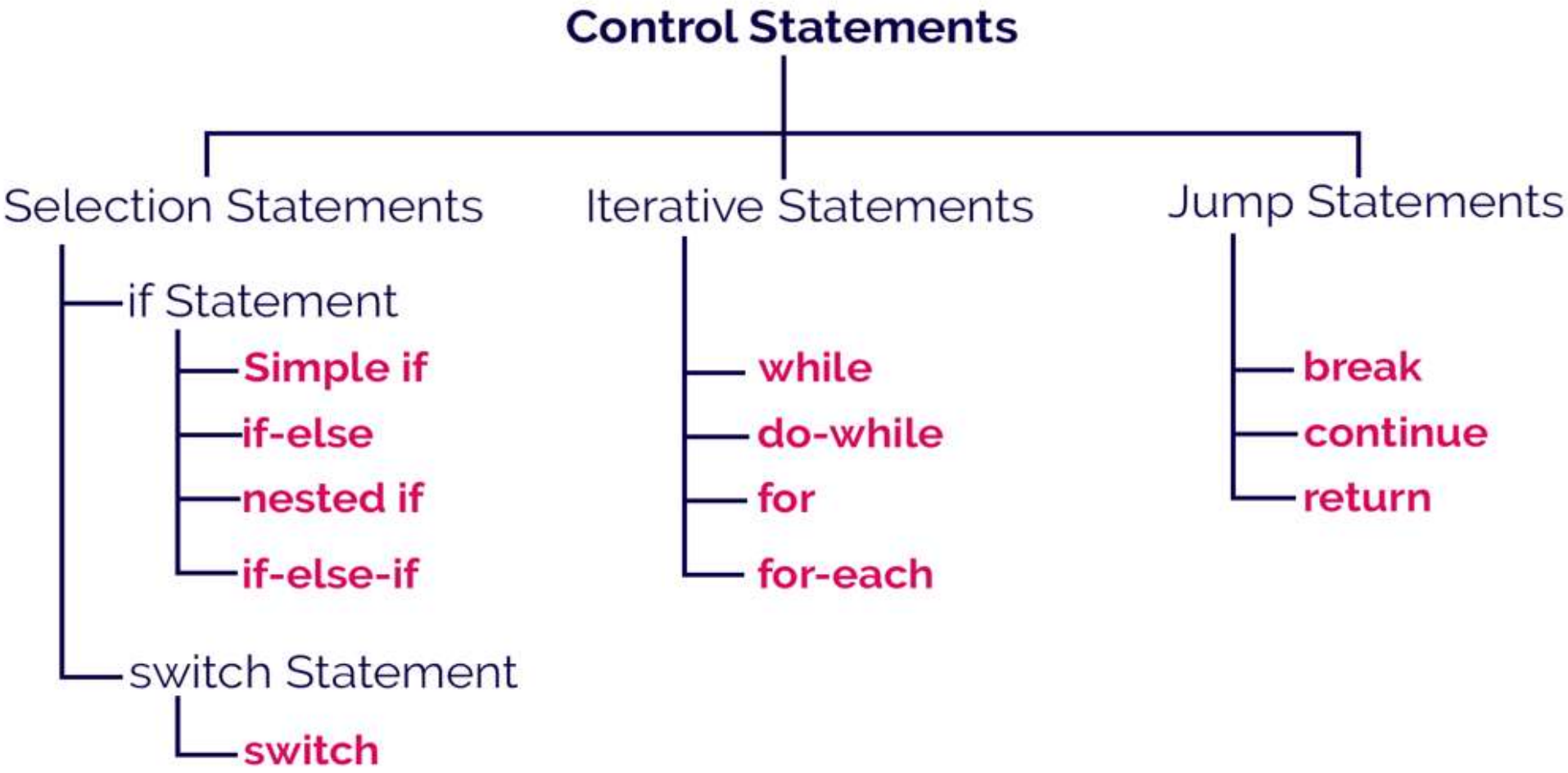
6. Assignment operators

Python

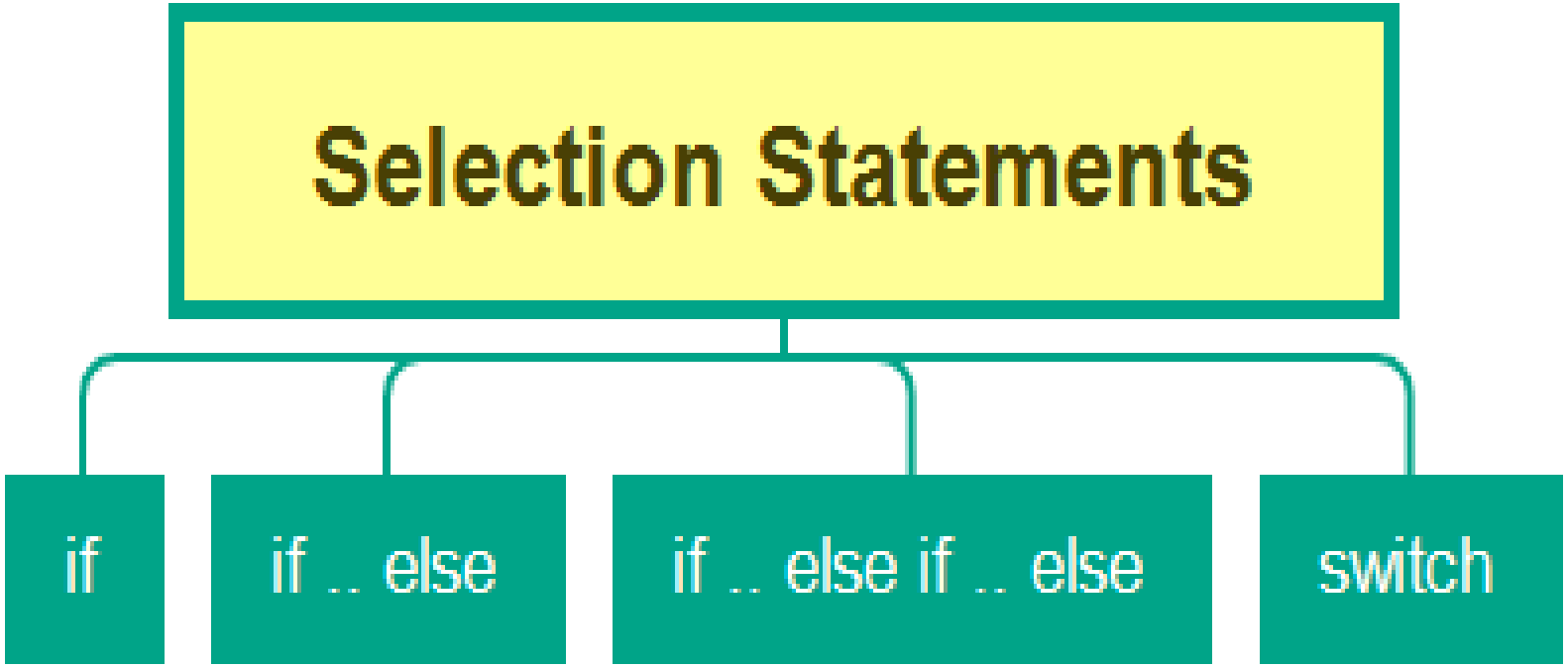
```
x = 5
x += 3 * 2 #Multiplication happens before +=
print(x) #output: 11
```

7. Selective and Iterative Statements

- In Python, conditional statements are used to control the flow of a program based on certain conditions.



Selection Statements (or) Branching Statements



Python If statements

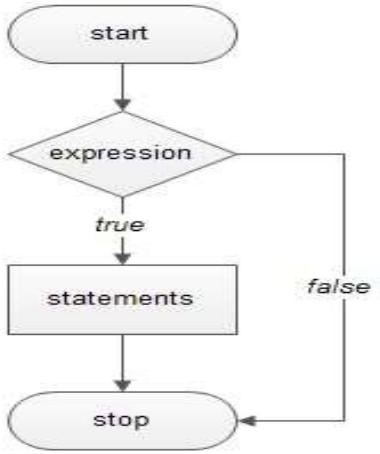
This construct of python program consist of one if condition with one block of statements.

When condition becomes true then executes the block given below it.

Syntax:

```
if ( condition):
        ……..
```

Flowchart



Find the Given Number is Even

Example-1:

```
n=int(input("Enter n Value: "))
 if (n%2==0):
        print(n," is a Even Number")
```

```
1  n = int(input("Enter the n Value"))
2  if(n%2==0):
3      print(n,"is a Even Number")
4
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Padmaja R> & "C:/Users/Padmaja R/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Padmaja R/OneDrive/Desktop/Even.py"
Enter the n Value4
4 is a Even Number
PS C:\Users\Padmaja R>
```

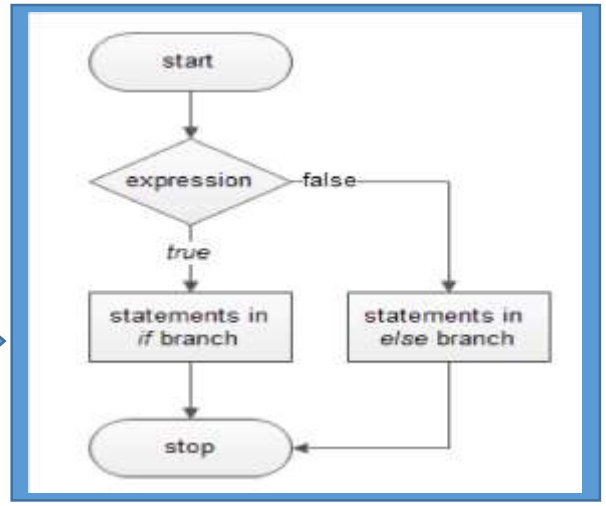
Python if - else statements

This construct of python program consist of one if condition with two blocks. When condition becomes true then executes the block given below it. If condition evaluates result as false, it will executes the block given below else.

Syntax:

```
if ( condition):
        ……….
else:
        ……….
```

Flowchart




```
1   n = int(input("Enter the n Value"))
2   if(n%2==0):
3       print(n,"is a Even Number")
4   else:
5       print(n,"is a Odd Number")
6
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\Padmaja R> & "C:/Users/Padmaja R/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Padmaja R/OneDrive/Desktop/Even.py"
Enter the n Value7
7 is a Odd Number
PS C:\Users\Padmaja R>
```

To Check Given Number is Even or Odd

Example-1:

```
n=int(input("Enter n Value: "))
 if ( n%2==0):
        print(n, "is Even")
else:
        print(n, "is Odd")
```

To Check Given age is eligible for voting or not

Example-2:

```
Age=int(input("Enter Age: "))
 if ( age>=18):
        print("You are eligible for vote")
else:
        print("You are not eligible for vote")
```

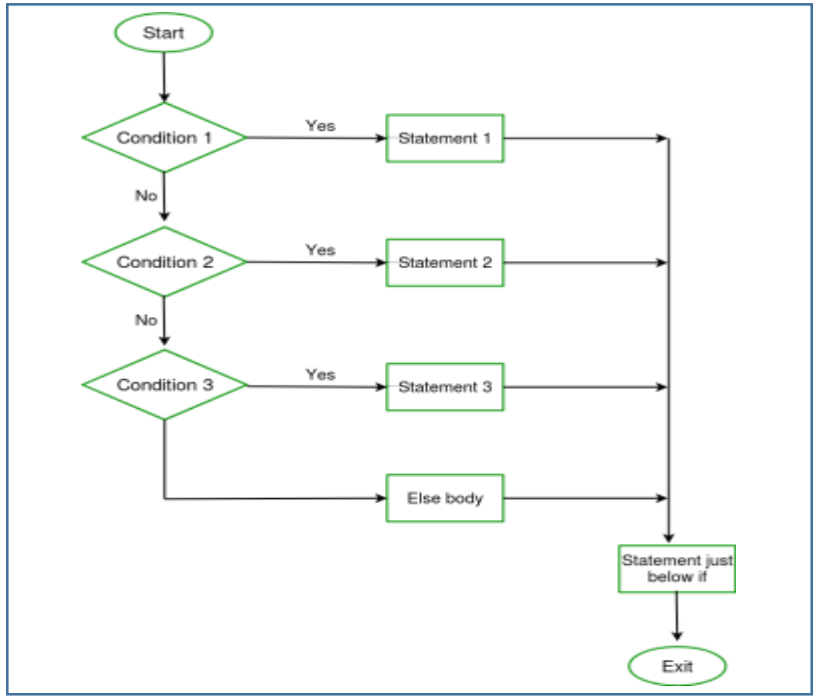
Python Ladder if else statements (if-elif-else)

This construct of python program consist of more than one if condition. When first condition evaluates result as true then executes the block given below it. If condition evaluates result as false, it transfer the control at else part to test another condition. So, it is multi-decision making construct.

Syntax

```
if ( condition-1):
        ……..
        ……..
 elif (condition-2):
        ……..
        ……..
elif (condition-3):
        ……..
        ……..
 else:
        ……..
        ……..
```

Flow Chart



Example

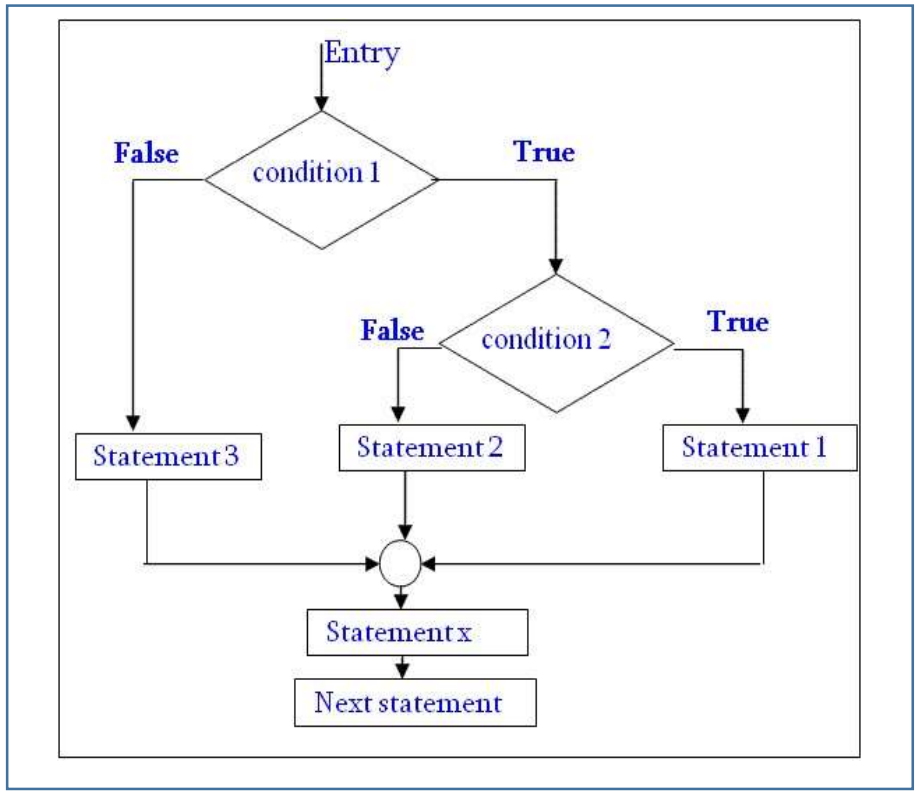
```
C: > Users > Padmaja R > OneDrive > Desktop > Weekday.py > ...
1   n = int(input("Enter the weekday number"))
2   if(n==1):
3       print("Today is a Monday")
4   elif(n==2):
5       print("Today is a Tuesday" )
6   elif(n==3):
7       print("Today is a Wednesday" )
8   elif(n==4):
9       print("Today is a Thursday" )
10  elif(n==5):
11      print("Today is a Friday" )
12  elif(n==6):
13      print("Today is a Saturday" )
14  elif(n==7):
15      print("Today is a sunday")
16  else:
17      print("Not a week day")
18
19
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\Padmaja R> & "C:/Users/Padmaja R/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Padmaja R/OneDrive/Desktop/Even.py"
Enter the weekday number4
Today is a Thursday
PS C:\Users\Padmaja R>
```

Python Nested if statements

It is the construct where one if condition take part inside of other if condition. This construct consist of more than one if condition. Block executes when condition becomes false and next condition evaluates when first condition became true. So, it is also multi-decision making construct.

```
if ( condition -1):
        if (condition -2):
                ..............
                ..............
        else:
                ..............
                ..............
else:
        ..........................
        ..........................
```

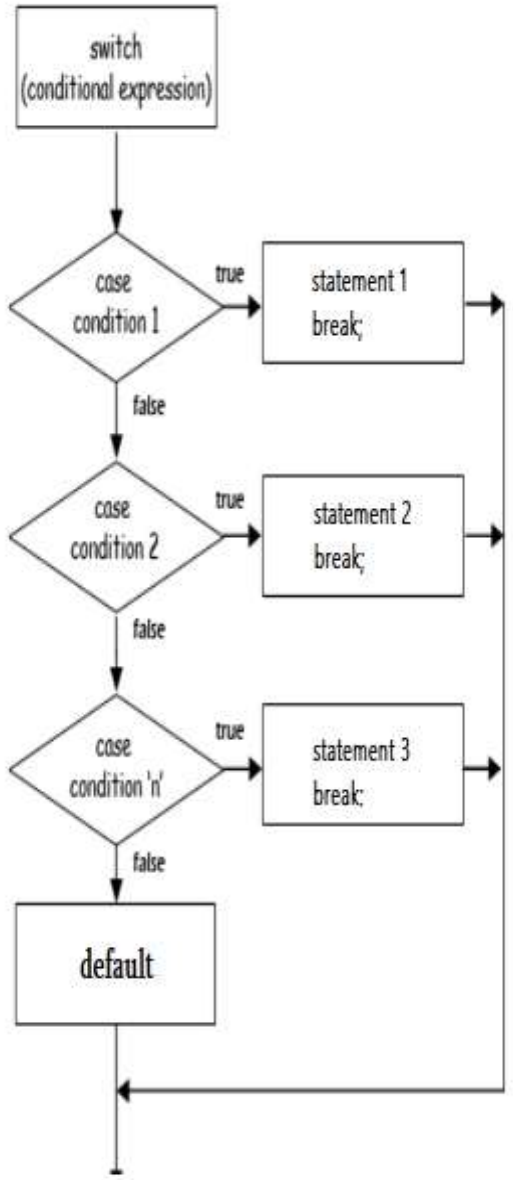


Example: to check the given number is negative, zero or positive

```
num=int(input("Enter Number:"))
If ( num<=0):
        if ( num<0):
                Print("You entered Negative number")
        else:
                Print("You entered Zero ")
else:
        Print("You entered Positive number")
```

7. Selective and Iterative Statements

- Python Switch Case is a selection control statement.
- The switch expression is evaluated once. ***The value of the expression is compared with the values of each case; if there is a match, the associated block of code is executed.***
- Then it makes a decision based on whether the condition is true or not.
- If the condition is true, it evaluates the indented expression; however, if the condition is false, the indented expression under else will be evaluated.
- When we need to run several conditions, you can place as many elif conditions as necessary between the ***if*** condition and the ***else*** condition.
- **Switch case statements** are a substitute for long if statements that compare a variable to several integral values
- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.



- Since Python 3.10, we can now use a new syntax to implement this type of functionality with a match case. The match case statement allows users to implement code snippets exactly to switch cases.

Syntax:

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

```python
n = int(input("Enter the weekday number"))
match n:
    case 1:
        print("Today is Monday")
    case 2:
        print("Today is Tuesady")
    case 3:
        print("Today is Wednesday")
    case 4:
        print("Today is Thursday")
    case 5:
        print("Today is Friday")
    case 6:
        print("Today is Saturday")
    case 7:
        print("Today is Sunday")
    case _1:
        print("Not a Week day")

```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Padmaja R> & "C:/Users/Padmaja R/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Padmaja R/OneDrive/Desktop/matchCase.py"
Enter the weekday number6
Today is Saturday
PS C:\Users\Padmaja R>
```

7. Selective and Iterative Statements

- The flow of the programs written in any programming language is sequential by default.
- The first statement of the program is executed first, followed by the second, and so on.
- There may be a situation when the programmer needs to execute a block of code several times.
- For this purpose, The programming languages provide various kinds of loops that are able to repeat some particular code numerous numbers of times.
- Looping simplifies complicated problems into smooth ones.
- It allows programmers to modify the flow of the program so that rather than writing the same code, again and again, programmers are able to repeat the code a finite number of times.
- In Python, there are three different types of loops:

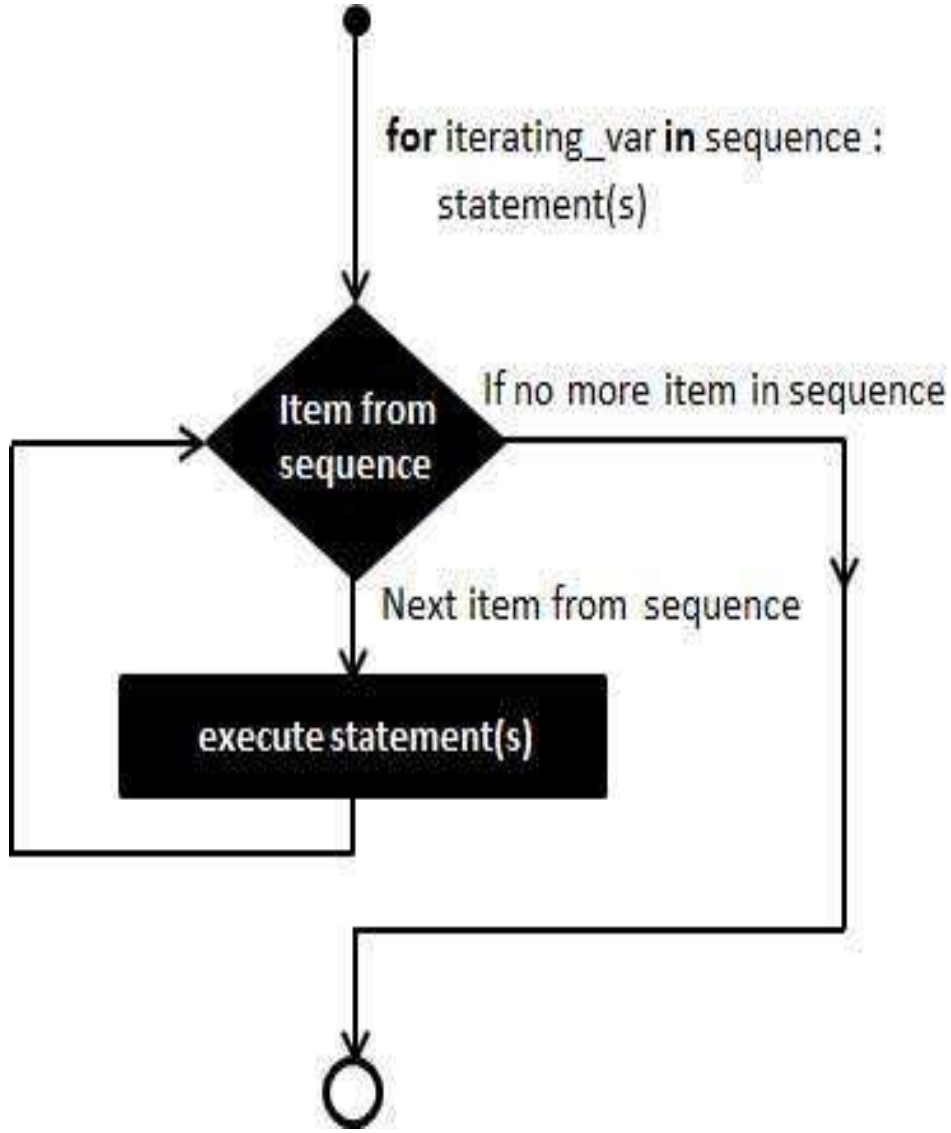
  12.1) For Loop

  12.2) While Loop and

  12.3) Nested Loop.

For Loop

- The for loop is used in the case where a programmer needs to execute a part of the code until the given condition is satisfied.
- The for loop is also called a pre-tested loop.
- It is best to use for loop if the number of iterations is known in advance.



- **Python For Loops** are used for iterating over a sequence like **lists, tuples, strings, and ranges.**
- The range() function is commonly used with for loops to generate a sequence of numbers.
- It can take one, two, or three arguments:

  range(stop): Generates numbers from 0 to stop-1.

  range(start, stop): Generates numbers from start to stop-1.

  range(start, stop, step): Generates numbers from start to stop-1, incrementing by step.

```
# Iterate from 0 to 4
for i in range(5):
    print(i) # prints from 0 to 4

# Iterate from 2 to 9
for i in range(2, 10):
    print(i) # prints 2,3,4,5,6,7,8,9

# Iterate from 0 to 9, with a step of 2
for i in range(0, 10, 2):
    print(i) # prints 0,2,4,6,8
```

- ✓ for loop to iterate over a string and print each character on a new line.
- ✓ The loop assigns each character to the variable i and continues until all characters in the string have been processed.

```
s = "Geeks"
for i in s:
    print(i)
```

Output

```
G
e
e
k
s
```

Iterating from Sequence

```
for val in sequence:
    statements
```

- Here, val accesses each item of sequence on each iteration.
- The loop continues until we reach the last item in the sequence.

Example

```
#!/usr/bin/python

for letter in 'Python':     # First Example
   print 'Current Letter :', letter

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
   print 'Current fruit :', fruit

print "Good bye!"
```

When the above code is executed, it produces the following result –

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Iterating by Sequence Index

- An alternative way of iterating through each item is by index offset into the sequence itself. Following is a simple example –

```
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
   print 'Current fruit :', fruits[index]

print "Good bye!"
```

When the above code is executed, it produces the following result –

```
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

- Here, len() is a built-in function, which provides the total number of elements in the tuple as well as the range() is another built-in function to give us the actual sequence to iterate over.

Using else Statement with For Loop

- Python supports to have an else statement associated with a loop statement
- If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.

```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

Run Code »

Output

```
0
1
5
No items left.
```

Here, the `for` loop prints all the items of the `digits` list. When the loop finishes, it executes the `else` block and prints `No items left.`

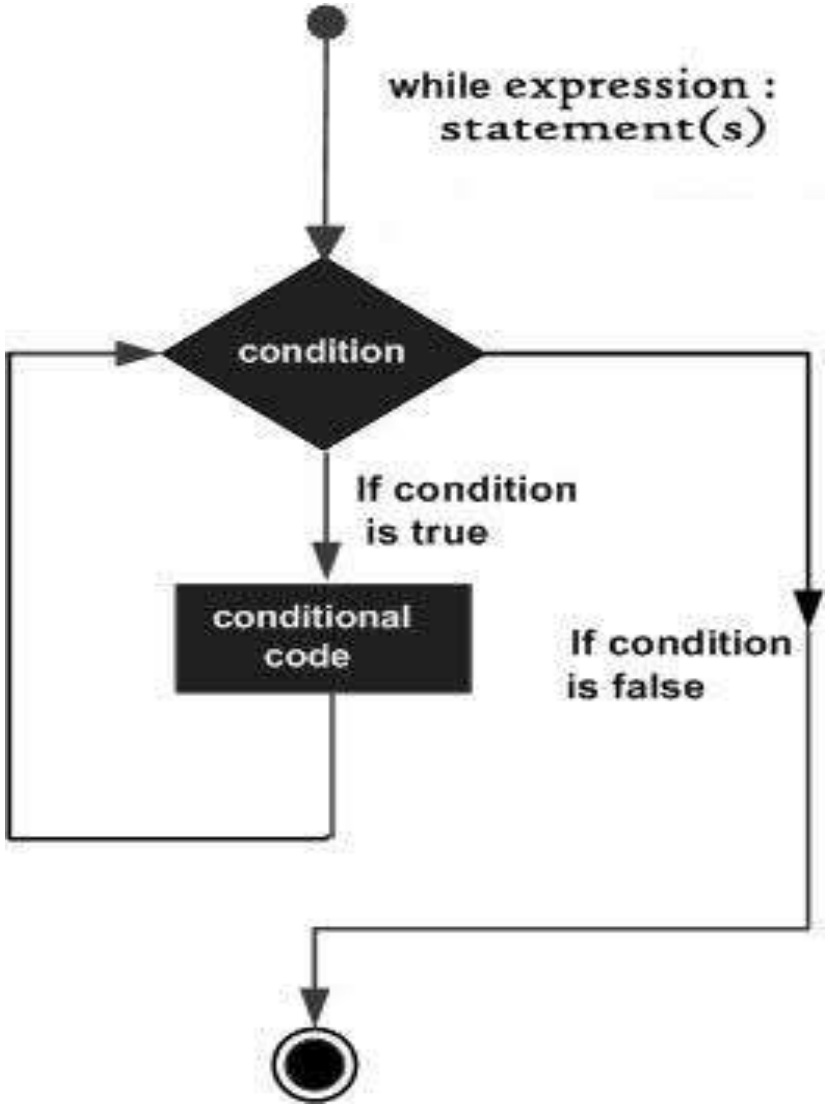
> **Note**: The else block will not execute if the for loop is stopped by a break statement.

While Loop

- A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
- The syntax of a **while** loop in Python programming language is

```
while expression:
   statement(s)
```

- Here, **statement(s)** may be a single statement or a block of statements.
- The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.
- In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code.
- Python uses indentation as its method of grouping statements.



While Loop

Example

```
#!/usr/bin/python

count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

When the above code is executed, it produces the following result –

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

Using else Statement with While Loop

- Python supports to have an **else** statement associated with a loop statement.
- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise else statement gets executed.

```
#!/usr/bin/python

count = 0
while count < 5:
   print count, " is  less than 5"
   count = count + 1
else:
   print count, " is not less than 5"
```

When the above code is executed, it produces the following result –

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

Nested Loop

- Python programming language allows to use one loop inside another loop.

Syntax

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

The syntax for a **nested while loop** statement in Python programming language is as follows –

```
while expression:
   while expression:
      statement(s)
   statement(s)
```

Nested Loop

Nested For Example

```
1   for i in range(2,11):
2       c=0
3       for j in range(1,i+1):
4           if(i%j==0):
5               c=c+1
6       if(c==2):
7           print(i,"is Prime")
```

OUTPUT

```
2 is Prime
3 is Prime
5 is Prime
7 is Prime
```

Nested While Example

```
1   i=2
2   while(i<=10):
3       c=0
4       j=1
5       while(j<=i):
6           if(i%j==0):
7               c=c+1
8           j=j+1
9       if(c==2):
10          print(i,"is a prime number")
11      i=i+1
```

OUTPUT

```
2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
```

Jumping Statements

Break statement	Continue statement
Terminates the loop	*Skips the current iteration*

`for i in range(1,10):`
`  if(i==5):`
`  break`
`  else:`
`  print(i)`
	`for i in range(1,10):`
	`  if(i==5):`
	`  continue`
	`  else:`
	`  print(i)`

OUTPUT 1 2 3 4	**OUTPUT** 1 2 3 4 6 7 8 9

STRING HANDLING

- Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes.
- The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.
- Each character is encoded in the ASCII or Unicode character.
- So we can say that Python strings are also called the collection of Unicode characters.

Creating a String in Python

STRINGS IN PYTHON CAN BE CREATED USING SINGLE QUOTES OR DOUBLE QUOTES OR EVEN TRIPLE QUOTES.

```
# Creating a String with single Quotes
String1 = 'Welcome to the Python World'
print("String with the use of Single Quotes: “ ,String1)

# Creating a String # with double Quotes
String1 = “Python Strings"
print("String with the use of Double Quotes: “,String1)

# Creating a String # with triple Quotes
String1 = '''I'm a Geek and I live in a world of "Geeks"'''
print("String with the use of Triple Quotes: “,String1)
```

Accessing characters in Python String

❖ **In Python, individual characters of a String can be accessed by using the method of Indexing.**

CODE	OUTPUT
# **Python Program to Access characters of String** String1 = “Python World" print("Initial String: “,String1) # **Printing First character** print("First character of String is: “,String1[0]) # **Printing Last character** print("Last character of String is: “,String1[-1])	Initial String: Python World First character of String is: P Last character of String is: d

Python Program to get a string made of the first 2 and last 2 characters of a given string.

```
str = "PythonWorld"
print(str[:2]+str[-2:]) \\ prints Pyld
```

Python program to get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.

```
str = "restart"
char = str[0]
str = str.replace(char,'$')
str = char +str[1:]
print(str) \\ prints resta$t
```

Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string

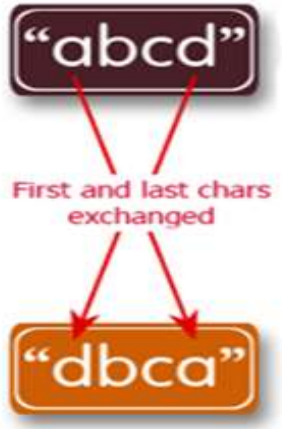
```
str1 = 'abc'
str2 = 'xyz'
new_str1 = str2[:2]+str1[2:]
new_str2 = str1[:2]+str2[2:]
print(new_str1+' '+new_str2)
```

Python program to remove the nth index character from a nonempty string.

```
str = input("Enter the String : ")
n = int(input("Enter the position of the character to be removed : "))
print(str[:n]+str[n+1:])
```

Write a Python program to change a given string to a newly string where the first and last chars have been exchanged.

```
str ='abcd'
print(str[-1:]+str[1:-1]+str[:1])
```



Write a Python function to get a string made of 4 copies of the last two characters of a specified string (length must be at least 2).

```
str='Python'
sub_str = str[-2:]
res = sub_str*4
print(res)
```