Index

1. Introduction to Python

1.1 What is python ?

Python is a general-purpose, dynamic, high-level and interpreted programming language.
It is designed to be simple and easy to learn, making it an ideal choice for beginners. One of the key strengths of Python is its versatility.
Python supports the object-oriented programming approach, allowing developers to create applications with organized and reusable code.

1.2 Features of Python

Readability :- Python's syntax is designed to be clear and reusable, making it easy for both beginners and experienced programmers to understand and write code.

Simplicity :- Python emphasizes simplicity and avoids complex syntax, making it easy for learn and use compared to other programming language.

Dynamic typing: Python is dynamically typed, meaning you don't need to explicitly declare variable types.

Large standard Library: Python provides a vast standard library with ready-to-use modules and functions for various tasks, saving developers time and effort in implementing common functionalities.

Object-Oriented Programming (OOP) :- Python supports the object-oriented programming paradigm, allowing for the creation and manipulation of objects, classes and inheritance.

Cross-Platform Compatibility :- Python is available on multiple platforms, including Windows, macOS, and Linux, making it highly portable and versatile.

Extensive Third-Party Libraries :- Python has a vast ecosystem of third-party libraries and frameworks that expand its capabilities in different domains, such as web development, data analysis, and machine learning.

Interpreted Nature :- Python is an interpreted language, meaning it does not require compilation. This results in a faster development cycle as code can be executed directly without the need for a seprate compilation step.

Integration Capabilities :- Python can easily integrate with other languages like C, C++, and Java, allowing developers to leverage existing codebases and libraries.

1.3 Applications Of Python

Python is widely used in various domains and offers numerous applications due to its flexibility

ease of use. Here are some key areas where python finds application:

Web development :- Python is extensively used in web development Frameworks such as Django and Flask. These Frameworks provides efficient tools and libraries to build dynamic websites and web applications.

Data Analysis and Visualization :- Python's rich ecosystem of libraries, including NumPy, Pandas, and Matplotlib, make it a popular choice for data analysis and visualization. It enables professionals to process, manipulate, and visualize data effectively.

Machine learning and Artificial Intelligence :- Python has become the go-to language for machine learning and AI projects. Libraries like Tensorflow, keras, and scikit-learn provide powerful tools for implementing complex algorithms and training models.

Automation and Scripting: Python's easy-to-read syntax and rapid development cycle make it an ideal choice for automation and scripting tasks. It is commonly used for tasks such as file manipulation, data parsing, and system administration.

1.4 Python Installation

To download and install Python, follow these steps:

For Windows :

1. Visit the official Python website at www.python.org/downloads/
2. Download the python installer that matches your system requirements.
3. On the Python Releases for Window page, select the link for the latest Python 3.x.x release.
4. Scroll down and choose either the "Windows x86-64 Installer" for 64-bit or the "Windows x86 executable installer" for 32-bit.
5. Run the downloaded installer and follow the instructions to install python on your Windows system.

For Linux (specifically Ubuntu)

1. Open the Ubuntu software Center folder on your Linux system.
2. From the All software drop-down list box, select Developer Tools.
3. Locate the entry for Python 3.x.x and double-click on it.
4. Click on the install button to initiate the installation process.
5. Once the installation is complete, close the Ubuntu Software Center folder.

1.5 First Python Program :

Writing your first python program is an exciting step towards learning the language. Here's a simple example to get you started.

5.

```
# Printing Hello World using python

print ("Hello World!")
```

Let's break down the code:

- The print() function is used to display the specified message or value on the console.
- In this case, we pass the string "Hello, World!" as an argument to the print() function. The string is enclosed in double quotes.
- The # symbol indicates a comment in Python. Comments are ignored by the interpreter and are used to provide explanations or notes to the code.

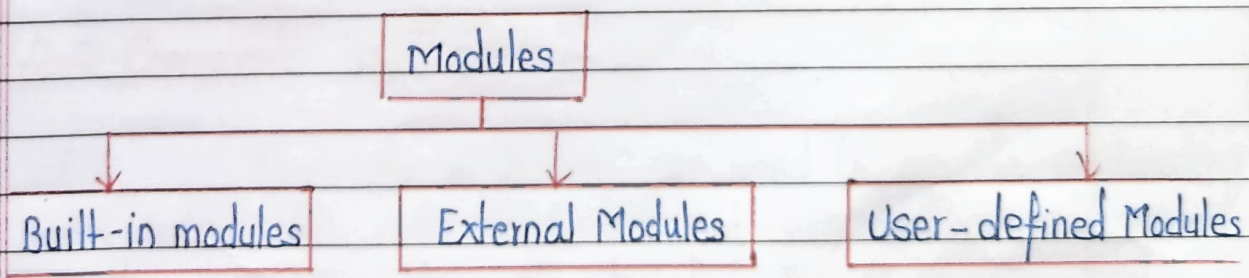
2. Modules, Comment & Pip

* 2.1 Modules in python:

Modules provide a way to organize your code logically. Instead of having all your code in a single file, you can split it into multiple modules based on their purpose. When you want to use the functionality from a module, you can import it into your current program or another module.

This allows you to access and use functions, classes and variables defined within that module, you can avoid writing the same code repeatedly and instead reuse the code defined in the module.

Three main types of modules:



1) **Built-in modules:**

These are modules that come pre-installed with python. They are part of the standard library and

provide a wide range of functionalities. Built in modules are readily available for use without the need for additional installations.

External Modules:

These are modules that are created by third-party developers and are not part of the standard library.

They extended Python's capabilities by providing additional functionalities for specific purposes. External modules can be downloaded and installed using package managers like pip (python package index).

Popular external modules include numpy for numerical computations.

User Defined Modules:

These modules are created by the python programmers themselves. They allow users to organize their code into seprate files and reuse functionality across multiple programs.

User-defined modules can contain functions, classes, variables, and other code that can be imported and used in other Python scripts or modules.

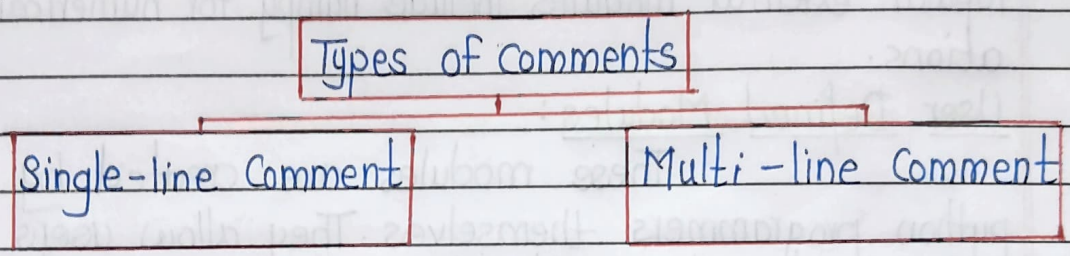
* 2.2 Comments in Python

Comments in Python are used to provide explanatory notes within the code that are not executed or interpreted by the computer. They are helpful for improving code readability and for leaving reminders or explainations for other developers who might work with the code in the future

8.

In Python, comments are denoted by the hash symbol (#) followed by the comment text. When the Python interpreter encounters a comment, it ignores it and moves on to the next lines of code. Comments can be placed at the end of line or an a line by themselves.
It's important to that comments are meant for human readers and are not executed by the Python interpreter. Therefore, they have no impact on the program's functionality or performance.

Types of Comments :



1. Single-line comments

Single-line comments are used to add explainatory notes or comments on a single line of code. They start with a hash symbol (#) and continue untill the end of the line. Anything written after the hash symbol is considered a comment and is ignored by the python interpreter.
Here's an example:

```
# This is a single-line comment
x = 5 # Assigning a value to the variable x
```

2. Multi-line Comments:

Multi-line comments, also known as block comments, allow you to add comments that span multiple lines. Python does not have a built-in syntax specifically for multi-line comments, but you can acheive this by using triple quotes (either single or double) to create a string that is not assigned to any variable. Since it is not used elsewhere in the code, it acts as a comment.
Here's an example:

```
""" This is a multi-line comment.
    It can span across multiple lines
    and is enclosed within triple quotes.
"""
x = 5  # Assigning a value to the variable x.
```

* 2.3 What is a pip?

In simple terms, pip is a package manager for python. It stands for "pip Installs Packages" or "pip install Python".
When working with python, you may need to use external libraries or modules that provide additional functionalities beyond what the standard library offers. These libraries are often developed by the Python community and are available for anyone to use.
Pip makes it easy to install a package, manage and uninstall these external libraries. It helps you

Find and download the libraries you need from the Python Package Index (PyPI), which is a repository of Python packages maintained by the community.
With pip, you can install a package by running a simple command in your terminal or command prompt. It will automatically fetch the package from PyPI and install it on your system, along with any dependencies it requires.

3. Variables, Data Types Keywords & Operators

3.1 Variables in Python

In Python, variables are used to store values that can be used later in a program. You can think of variables as containers that hold data. What makes python unique is that you don't need to explicitly declare the type of a variable. You simply assign a value to a variable using the "=" operator.
For example, you can create a variable called "name" and assign it a value like this:

```
name = "Yadnyesh"
```

Here, "name" is the variable name, and "Yadnyesh" is the value assigned to it. Python will automatically determine the type of variable based on the value assigned to it.

Variables in Python can hold different types of data, such as numbers, strings, lists or even more complex objects. You can change the value of a variable at any time by

assigning a new value to it. For instance:

```
age = 25
age = 26  # Updating the value of age variable.
```

Python also allows you to perform operations on variables. For example, you can add, substract, multiply, or divide variables containing numbers. You can even combine variables of different types using operators. For instance:

```
x = 5
y = 3
z = x+y  # The value of 'z' will be 8.

greeting = "Hello"
name = "John"
message = greeting + " " + name # The value of 'message' will be "Hello John".
```

Variables provide a way to store and manipulate data in python, making it easier to work with information throughput your program. By giving meaningful names to variables, you can make your code more readable and understandable.

3.1.1 Identifier in Python

In Python, an identifier is a name used to identify a variable, function, class, module, or any other user-defined object. An identifier can be made up of letters (both uppercase and lowercase), digits and underscores (_). However, it must start with a letter or an underscore.

Here are some important rules to keep in mind when working with identifiers in python.

1. Valid characters: An identifier can contain letters (a-z, A-Z), digit (0-9) and underscores (_). It cannot contain spaces, special characters like @, #, or $.
2. Case Sensitivity: Python is case-sensitive, meaning upper-case and lowercase letters are considered different. So, "myVar" and "myvar" are treated as two different identifiers.
3. Reserved Words: Python has reserved words, also known as keywords, that have predefined meanings in the language. These words cannot be used as identifiers. Examples of reserved words include "if", "While" and "def".
4. Length: Identifiers can be of any length. However, it is recommended to use meaningful and descriptive name that are not excessively long.
5. Readability: It is good practice to choose descriptive names for identifiers that convey their purpose or meaning. This helps make the code more readable and understandable.

Here are some examples of valid identifiers in python

- my_variable
- count
- total_sum
- PI
- Myclass

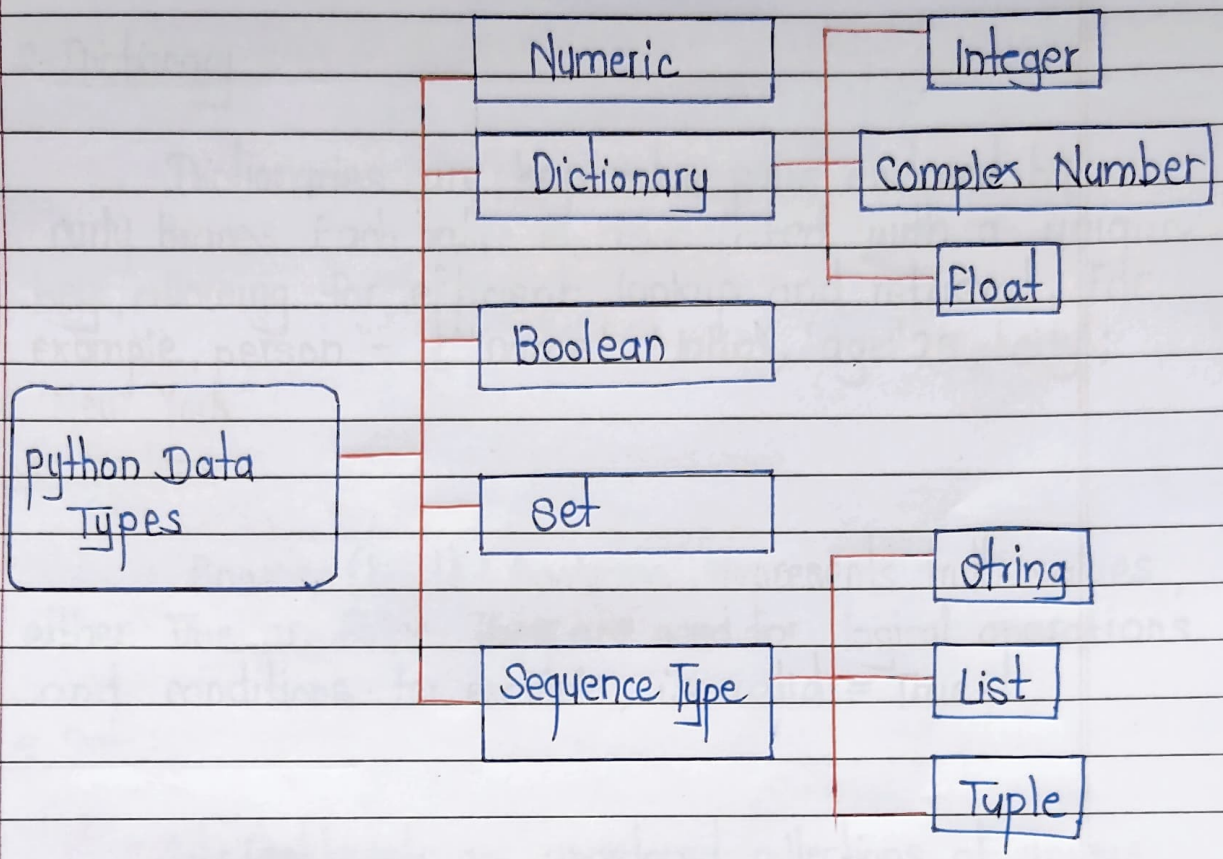
And here are some examples of invalid identifiers:

- 123abc (starts with a digit)
- my-variable (contains a hypen)
- if (a reserved word)
- my var (contains a space)

Understanding and following these rules for identifiers in Python is important to ensure clarity, reddability and proper functioning of your code.

3.2 Data Types in Python

Data types in Python refer to the different kinds of values that can be assigned to variables. They determine the nature of the data and the operations that can be performed on them. Python provides several built-in-data types, including: numeric, dictionary, Boolean, set, sequence types and numeric has Integer, Complex Number, Float. and Sequence Type has String, List and Tuple.



1. Numeric:

Python supports different numerical data Types, including integers (whole numbers), Floating-point numbers (decimal numbers), and complex numbers (numbers with real and imaginary parts)

a. Integers (int):- Integers represent whole numbers without any fractional part. For example, age = 25.

b. Floating-point Numbers (Float): Floating-point numbers represent numbers with decimal point or fractions. For example, pi = 3.14.

c. Complex Numbers (Complex): Complex numbers have a real and imaginary part. For ex: z = 2+3j where j is suffix with it.

2. Dictionary :

Dictionaries are key-value pairs enclosed by curly braces. Each value is associated with a unique key, allowing for efficient lookup and retrieval, For example, person = {'name': 'John', 'age': 25, 'city'; 'New York'

3. Boolean :

Boolean (bool) : Booleans represents truth values, either True or False. They are used for logical operations and conditions. For example, is_valid = True.

4. Set :

Sets (set) : sets are unordered collections of unique elements enclosed in curly braces. They are useful for mathematical operations such as union, intersection, and difference For example, fruits = {'apple', 'banana', 'orange'}

5. Sequence Type :

Sequences represent a collection of elements and include data types like strings, lists, and tuples. Strings are used to store textual data, while lists and tuples are used to store ordered collections of items.

a. Strings (str) : Strings represent sequences of character enclosed within single or double quotes. For example: name = 'John'.

b. List (list) : Lists are ordered sequences of element enclosed in square brackets. Each element

be of any data type. For example numbers = [1,2,3,4]

c. Tuples (tuple) : Tuple are similar to lists but are immutable, meaning their elements, cannot be changed once defined. They are enclosed in '()' parenthesis.

3.3 Keywords in Python

Keywords in python are special words that are having specific meanings and purposes within the python language. They are reserved and cannot be used as variable names or identifiers.

keywords play a crucial role in defining the structure and behaviour of python programmers.

Keywords are like building block that allow us to create conditional statements, loops, functions, classes, handle errors and perform other important operations.

* List of all the keywords in python :

```
False   await   else   import   pass
None    break   except   in   raise
True    class   finally   is   return
and     continue   for   lambda   try
as      def     from   nonlocal   while
assert  del     global   not   with
async   elif    if     or      yeild
```

3.4 Operators in Python :

Operators in Python are symbols or special

characters that are used to perform specific operations on variables and values. Python provides various types of operators to manipulate and work with different data types. Here are some important categories of operators in python:

- Arithmetic Operators
- Comparison Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identify Operators

* Arithmetic Operators:

Arithmetic Operators in python are used to perform mathematical calculations on numeric values. The basic arithmetic operators include:

- Addition (+): Adds two operands together. For example, if we have a=10 and b=10, then a+b equals 20.
- Subtraction (-): Subtracts the second operand from the first operand. If the first operand is smaller than the second operand, the result will be negative. For example: if we have a = 20 and b = 5, then a-b equals 15.
- Division (/): Divides the first operand by the other operand and returns the quotient. For example, if we have a=20 and b=10, then a/b equals 2.0.
- Multiplication (*): Multiplies one operand by the other

For example, if we have a = 20 and b = 4, then a*b equals 80

- Modulus (%) : Returns the remainder after dividing the first operand by the second operand. For example, if we have a = 20 and b = 10 then a%b equals 0.
- Exponentiation (**) or power : Raises the first operand to the power of second operand. For example, if we have a = 2 and b = 3, then a**b equals 8.
- Floor division (//) : provides the floor value of the quotient obtained by diving the two operands, it returns the largest integer that is less than or equal to the result. For example, if we have a = 20 and b = 3 then a//b equals 6.

* Comparison Operators:

Comparison operators in python are used to compare two values and return a Boolean value (True or False).

- Equal to (==) : checks if two operands are equal.
- Not equal to (!=) : check if two operands are not equal.
- Greater than (>) : checks if the left operand is greater than right operand.
- Less than (<) : check if the left operand is less than right operand.
- Greater than or equal to (>=) : checks if the left operand is greater than or equals to the right operand.
- Less than or equal to (<=) : checks if the left operand is less than or equals to the right operand.

* Assignment Operators :

Assignment Operators are used to assign values to variables. They include :

- Equal to (=) : Assign the value on the right to the variable on the left.
- Compound assignment operators (+=, -=, *=, /=) : perform the specified arithmetic operations and assign the result to variable.

* Logical operators :

Logical operators in python are used to perform logical operations on Boolean values.

- Logical AND (and) : Returns true if both operands are true, otherwise False.
- Logical OR (or) : Returns true if ~~both are~~ one of them is true, otherwise False.
- Logical NOT (not) : Returns the opposite boolean value of the operand.

* Bitwise Operators :

Bitwise operators perform operations on individual bits of binary numbers.

- Bitwise AND (&) : Performs a bitwise AND operations on the binary representations of the operands.

- Bitwise OR (|) : Performs a bitwise OR operation on the binary representations of the operands.
- Bitwise XOR (^) : Performs a bitwise exclusive OR operation on the binary representation of the operands.
- Bitwise complement (~) : Inverts the bits of the operands.
- Left shift (<<) : shifts the bit of the left operand to the left by the number of positions specified by the right operand.
- Right shift (>>) : shift the bit of the left operand to the right by the number of positions specified by the right operand.

* Membership Operators :

Membership operators are used to test whether a value is a member of a sequence.

- In : Returns true if both operands refer to the same object (sequence)
- Not In : Returns true if both operands do not refers to the same object (sequence)

* Identity operators :

Identity operators are used to compare the identity of two objects.

- Is : Returns True if both operands refer to the same object.
- Is not : Returns True if both operands do not refers to the same object.

4. List, Dictionary, set Tuples and Type Conversion

4.1 List:

In Python, a list is a versatile data structure that allows you to store and organize multiple items in a single variable. Lists are defined by enclosing a sequence of elements within square brackets [] and seprating them with commas.

- Lists in python are ordered collections of elements where each element is identified by its position or index.
- Lists are mutable, which means you can modify, add or remove elements after the list is created.

List declaration:

In python, you can declare a list by enclosing a sequence of elements within square brackets ([]). Here are a few examples:

```
numbers = [1,2,3,4,5]
Fruits  = ["apple", "banana", "orange", "grape"]
```

In this example, we have two lists. The numbers list contains integers, specifically the numbers 1, 2, 3, 4 &

5.

4.2 Dictionary:

A dictionary in python is a collection of key-value pairs. It allows you to store and retrieve values based on unique keys. Dictionaries are mutable, meaning you can modify them, and they provide fast access to values. They are useful for tasks like data mapping and storing structured information. key must be unique and python provides built-in methods to work with it.

Dictionary Declaration:

```
# Author: Codewithcurious.com
student = {"name": "John Doe", "age": 20, "major":
          "computer Science"}
```

4.3 set:

In python, a set is a collection of unique elements. It is used when you want to store a group of items without any duplicates.
Sets are flexible and allow you to add or remove elements. They don't have a specific order, so you can't access elements by their position.
Sets also support mathematical operations like combining sets or finding common elements.

4.4 Tuple

A tuple in python is an immutable ordered collection of elements enclosed in paranthesis (). It is used to store related data that should not be modified and supports indexing and unpacking for easy access to its elements. Tuples are memory-efficient and commonly used when data integrity and immutability are desired.

Tuple declaration

```
#Author : Codewithcurious.com
Student = ("John Doe", 20, "computer Science")
```

In this example, we have a tuple named student that contains three elements : the student's name ("John Doe"), age (20), and field of study ("computer science").

4.5 Type Convetsion

In python, type casting, also known as type conversion, refers to the process of changing the data type of a variable or value to a different type.

Python provides several built-in functions for type casting, allowing you to convert values between different data types. Here are some common type casting functions:

1. int() : Converts a value to an integer data type.
2. Float() : Converts a value to an float data type

3. str() : Converts a value to an string data type.
4. list() : Converts a value to an list data type
5. tuple() : Convets a value to a tuple data type.
6. bool() : converts a value to a boolean data type.

Type casting is very useful when you need to perform operations or comparisons with values of different data type.

Example:

```
#Author : CodewithCurious.com
# Integer to String
  num = 10
  num_str = str(num)
  print ("Number as a String :", num_str)
# String to Integer
  num_str = "20"
  num = int (num_str)
  print ("Number as a integer :", num)
# Float to Integer
  num_float = 3.14
  num_int = int (num_float)
  print ("Float as an integer :", num_int)
```

Output :

```
#Author : CodeWithCurious.com
Number as a string : 10
Number as an integer : 20
Float as an integer : 3
```

5. Flow Control

Flow control in programming refers to the ability to control the order in which statements and instructions are executed. It allows you to make decisions and repeat actions based on certain conditions. In python, Flow control is achieved using conditional statements (if, elif, else) and loops (for, while).

Conditional statements allows you to execute different blocks of code based on certain conditions, while loops enable you to repeat a block of code untill a specific condition is met.

Python Indentation

In python, indentation plays a crucial role in defining the structure and scope of code blocks. It is used to group statements together and indicate which statement belong to a particular block of code. Python uses indentation instead of traditional braces or brackets of define code blocks.

The standard convention in Python is to use four spaces for indentation. It is recommended to be consistent with indentation throughout your code to ensure readability and maintainability.

Here's an example that demonstrate the use of indentation in python.

```
# Author : CodewithCurious.Com
if condition
    # Code block 1
    Statement_1
    Statement_2
    Statement_3
else:
    # codeblock2
    statement_4
    statement_5
```

In above example, the if statement is followed by an indented block of code, which is considered as the if-block.

Decision-Making statements

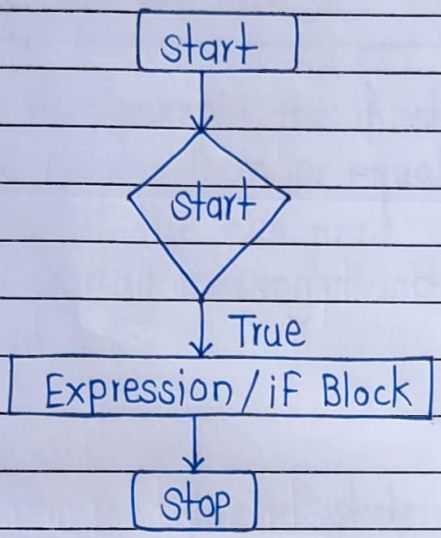
Decision-making is an important concept in programming, allowing us to execute specific blocks of code based on certain conditions. In python, we use different statements for decision making:

- IF statement : It tests a condition and executes a block of code if the condition is true.
- If-else statement : It test a condition and executes a block of code if the condition is true, and another block if the condition is false.
- Nested if statement : It allows us to use an if-else statement inside another if statement, creating multiple levels of conditions and decisions.

5.1 If-statement

The if statement in python is used to perform a specific action or execute a block of code if a condition is true. It allows us to make decisions in our programs based on the evaluation of the condition.

Flowchart :



syntax :

```
if condition
    #code to be executed if the condition is true
```

The condition is an expression that is evaluated to either True or False. If the condition is true, the code block indented below the if statement is executed. If the condition is false, the code block is skipped, and the program moves on the next statement after the if block.

Example:

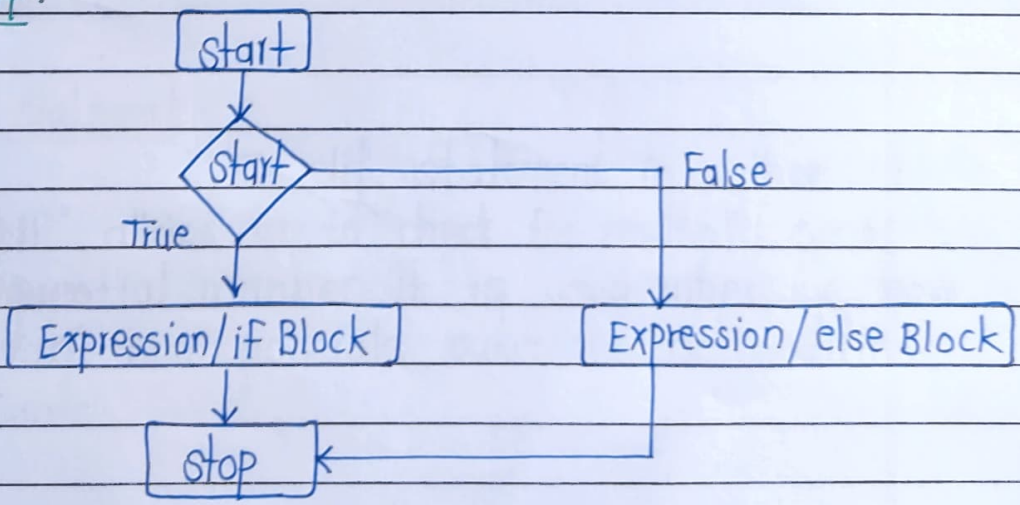
```
age = 25

if age >= 18 :
    print ("You are an adult.")
    print (" You can vote ")
```

In this example, the if statement checks if the variable "age" is greater than or equal to 18. If it is true, the program executes the two print statements within the if block, which display messages indicating that the person is an adult and can vote.

5.2 If-else statement

The if-else statement in python provides a way to perform different actions based on the evaluation of a condition. It allows us to execute one block of code if the condition is true, and another block of code if the condition is false.

Flowchart:



Syntax:

```
if condition
# code to be executed if the condition is true
else:
# code to be executed if the condition is false
```

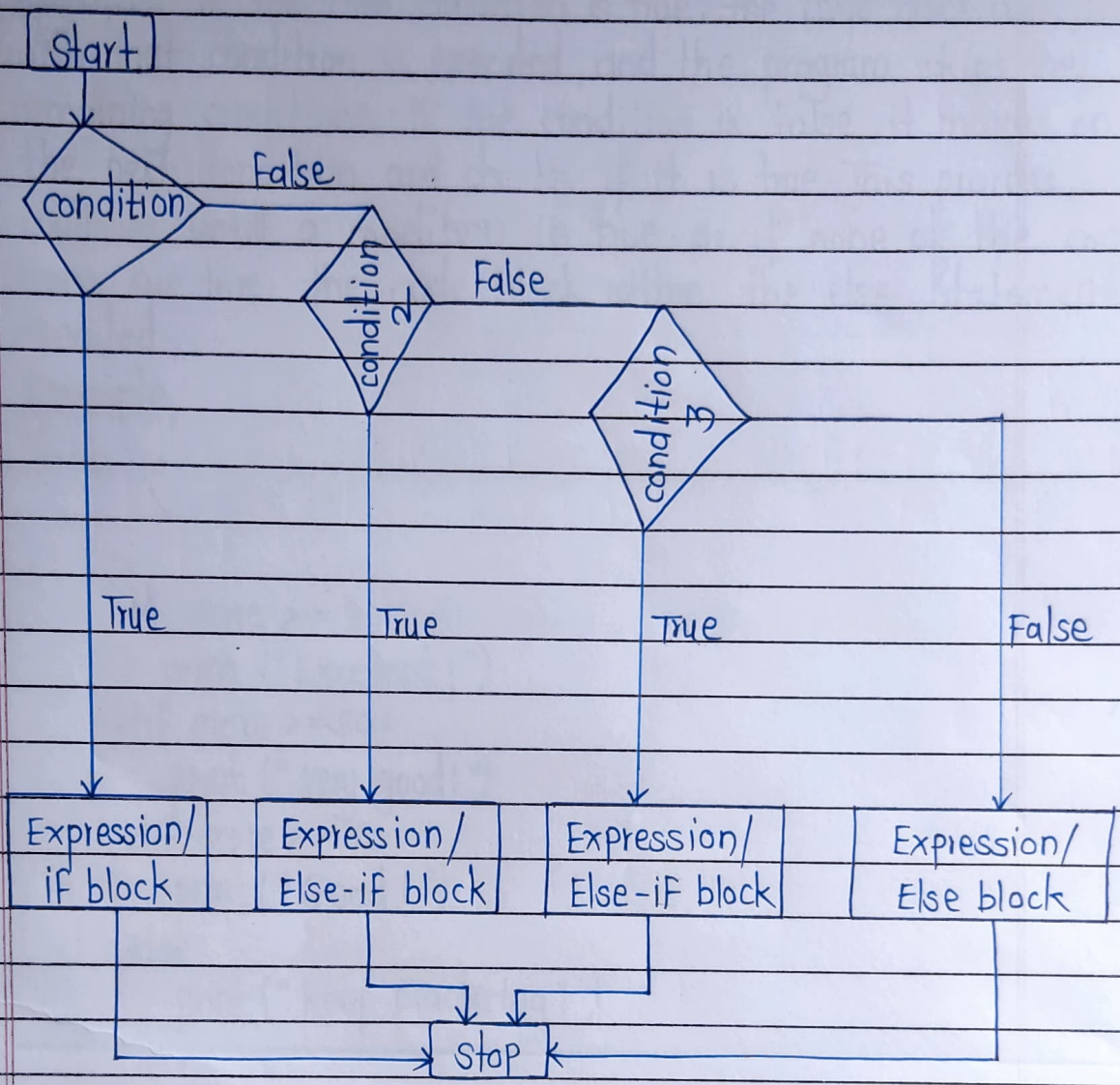
The condition is an expression that is evaluated to either True or False. If the condition is true, the code block indented below the if statement is executed. If the cond^{n} is false, the code block indented below the else statement is executed.

```
age = 15

if age >= 18:
    print ("You are an adult.")
    print ("You can vote.")
else:
    print ("You are not an adult.")
```

5.3 elif statement:

The elif statement in python, short for "else if", allows us to check for multiple conditions in a sequential manner. It is used when we have more than two possible outcomes or conditions to evaluate.

Flowchart



Syntax:

```
if condition1:
    # code to be executed if condition1 is true
elif condition2:
    # code to be executed if condition1 is false, condition2 is true.
elif condition3:
# code to executed if cond^n1, cond^n2 is false, condition3 is true.
else:
    # code to be executed if all conditions are false.
```

The conditions are evaluated in the order they are specified. If the first condition is true, the code block associated with that condition is executed, and the program skips the remaining conditions. If the condition is false, it moves on to the next condition and checks if it is true. This process continues untill a condition is true. or if none of the conditions are true. the code block within the else statement is executed.

Example:

```
Score = 75

if score >= 90:
    print ("Excellent!")
elif score >= 80:
    print ("Very good!")
elif score >= 70:
    print ("Good!")
else:
    print ("keep practicing!")
```

5.4 Nested if statement

The nested if statement in python allows us to have an if-else statement inside another if statement. It enable us to create multiple levels of conditions and decision in our code.

In a nested if statement, the inner if statement is indented under the outer if statement.

The inner if statement is evaluated only if the condition of the outer if statement is true.

Syntax:

```
if condition1:
  #code to be executed if condition1 is true
     if condition2:
  #code to be executed if condition1, condition2 is true
     else:
#code to be executed if cond^n1 is true but cond^n2 is false.
 else:
#code to be executed if condition1 is false
```

Example:

```
age=25
income=50000
if age>=18:
  print("You are eligible:")
 if income>=30000:
  print("You qualify for a loan.")
 else:
  print("You do not qualify for a loan")
else:
 print("You are not eligible:")
```

In this example, the outer if statement checks if the age is greater than or equal to 18. if it is true, the program executes the code block within the outer if statement. Inside this code block, there is an inner if statement that checks if the income is greater than or equal to 30000.

6. Loops

python provides several types of loops to cater to different looping needs. These loops offer different syntax and condition-checking approaches while serving the same purpose of executing statements repeatedly

The available loops in python are:

1. While loop : It represents a statement or group of statement as long as a specified condition is true. The loop body is executed only if the condition is evaluated as true.
2. For loop: This loop is used to iterate over a sequence of elements, such as a list or string. It simplifies the management of the loop variable and executes a code block for each item in the sequence.
3. Nested loops: Python allows us to have loops inside other loops. This concept of nesting loops enables us to iterate through multiple levels of iteration, executing a loop within another loop.

- Pass statement :

In Python, the pass statement is a placeholder statement that does nothing. It is used when a statement is syntactically required but you don't want to perform any action or write any code at that point. It acts as a null operation and helps in maintaining the structure of program.

The pass statement is commonly used as a placeholder for code that will be implemented later or as a placeholder for empty functions or classes. It allows you to write valid code without the need for immediate implementation.

Demonstrate the Use of the Pass Statement:

```
def my_function():
  pass  # placeholder for future code implementation
if x < 10:
  pass # placeholder for conditional code
class Myclass:
  pass # placeholder for class implementation
```

In the above examples, the pass statement is used to indicate that there will be code written in the future for the function, conditional block, or class.
It allows you to run the program without any syntax errors untill you are ready to implement the actual functionality.
The pass statement serves as a useful tool for maintaining code structure, enabling incremental development, and deferring the implementation of certain parts of the program until a later stage.

6.1 While loop :

The while loop in Python repeatedly executes

a block of code as long as a specified condition remains true. It tests the condition before each iteration and continues to execute the code block untill the condition evaluates to false.

Example:

```
Count = 0
while count < 5:
    print (count)
    count += 1
```

Here, the condition is the expression or condition that is evaluated before each iteration. IF the condition is true, the code block is executed.

IF the condition is false, the loop is excited, and the program continues with the next statement after the loop.

Syntax:

```
while condition:
    #code block
```

6.2 For loop:

The for loop in python is used to iterate over a sequence of elements, such as a list, tuple, string, or range.

It allows you to perform a set of actions for each item in the sequence. The loop variable takes the value of each item in the sequence one by one, untill the sequence is

exhausted.

Syntax:

```
for item in sequence
    # code block
```

Here, the item represents the loop variable that takes the value of each item in the sequence. The code block inside the loop is executed for each item in the sequence.

Example1 : Iterate over a list

```
Fruits = ["apple", "banana", "cherry"]
for fruit in fruits :
    print (fruit)
```

Output:

```
apple
banana
cherry
```

In the above example, the loop iterates over each item in the fruits list, and the loop variable fruit takes the value of each item successively. The print (fruit) statement is executed for each item, resulting in the names of fruits being printed.

Example2: Iterate using a range

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

Output:

```
1
2
3
4
```

In the above example, the range (1,5) function generates a sequence of numbers from 1 to 4 (exclusive). The loop variable i takes the value of each number in the sequence, and the print(i) statement prints each number.

6.3 For loop using range() Function:

The range() function in Python is often used in conjunction with the for loop to create a sequence of numbers that can be iterated over. The range() function generates a sequence of numbers based on the specified start, and step values.

The syntax of the range() function is as follows:

syntax:

```
range (start, stop, step)
```

Here, start specifies the starting value of the sequence (inclusive), stop specifies the ending value of the sequence (exclusive), and step specifies the increment

between each value in the sequence

Example1: Iterate over a range of numbers

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

Output:

```
0
1
2
3
4
```

In the above example, the range(5) function generates a sequence of numbers from 0 to 4. The for loop iterates over each number in the sequence, and the loop variable i takes the value of each number successively. The print(i) statement is executed for each number, resulting in the numbers being executed.

Example2: Iterate with a custom range and step

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

output:

```
1
3
5
7
9
```

In this example, the range (1, 10, 2) function generates a sequence of numbers from 1 to 9 with a step size of 2. The loop variable i takes the value of each number in the sequence, and the print(i) statements prints each number.

6.4 Nested For loop:

A nested for loop in python is a loop that is placed inside another for loop, it allows you to iterate over multiple sequences or perform repetitive actions within a nested structure. The inner for loop is executed for each iteration of the outer for loop, resulting in a combination of iterations.

Syntax:

```
for variable_outer in sequence_outer:
    #outer loop code block
    for variable_inner in sequence_inner:
        # Inner loop code block
```

Here, Variable_outer represents the loop variable for the outer loop, and sequence_outer is the sequence over which the outer loop iterates.

Example:

```
for i in range (1, 6):
    for j in range (1, 6):
        print (i*j, end= " ")
    print()
```

Output:

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 9 12 16 20
5 10 15 20 25
```

In the above example, the outer for loop iterates over the values 1 to 5. Inside the outer loop, there is an inner loop that also iterates from 1 to 5. For each value of the outer loop, the inner loop multiplies the current value of the outer loop (i) with the values of the inner loop (j), and the result is printed.

7. String

7.1 Python String

In Python, strings are a fundamental data type used to represent collections of characters. They can be defined by enclosing characters or sequences of characters in single quotes, double quotes or triple quotes.
Internally, the computer stores and manipulates characters as combinations of '0' and '1' using ASCII or Unicode encoding. Strings in python are also referred to as collections of Unicode characters.

Python provides flexibility in choosing the type of quotes to create strings, including single quotes (' '), double quotes (" ") or triple quotes (''' ''').

Using strings, we can handle text-based information, manipulate and process textual data, and perform string-specific operations in Python programming.

Syntax

```
str = "Hi there!"
```

7.2 Creating string in python:

In python, you can create strings by enclosing characters within single quotes (' '), double quotes (" ") or triple quotes (''' ''').

Example:

```
str1 = 'Hello'   #single quotes
str2 = 'World'   #Double quotes
str3 = """Python is powerful language."""  #Triple quotes

print(str1)
print(str2)
print(str3)
```

Output:

```
Hello
World
Python is powerful language.
```

In the above example, we create three different strings using single quotes, double quotes and triple quotes. All three ways are valid for creating strings in python.

7.3 strings indexing and splitting:

In python, strings are indexed starting From 0, just like in many other programming languages. For instance, the string 'HELLO' can be indexed as shown below:

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

```
str[0] = "H"
str[1] = "E"
str[2] = "L"
str[3] = "L"
str[4] = "O"
```

In this example, each letter of the string "HELLO" is assigned an id index starting from 0. The letter 'H' is at index 0, 'E' is at index 1, "L" is at index 2 and so on. Understanding string indexing is important beacause it allows you to access and manipulate specific characters within a string.

7.4 Deleting the string

In python, you can delete a string by assigning it the value None or using the del keyword. Here are the two methods to delete a string.

- Assigning None to the string

```
my_string = "Hello, World"
my_string = None
```

In this method, the variable my_string is assigned to value None, effectively deleting string.

- Using the del keyword.

```
my_string = "Hello, world"
del my_string
```

Here, the del keyword is used to delete the string object referred to by the variable my_string. After executing this statement, the variable my_string will no longer exist, and the memory occupied by the string will be freed.

7.5 String Formatting

In python, strings supports various operators that allow you to perform operations on strings. Here are some commonly used string operations.

1. Concatenation (+): The concatenation operator is used to combine two or more strings into a single string.

Example:

```
str1 = "Hello"
str2 = "World"
result = str1 + str2
print(result) #output : HelloWorld
```

2. Repetition (*): The repetition operator is used to repeat a string multiple times.

Example:

```
str1 = "Hello"
result = str1 *3
print(result) #output : HelloHelloHello
```

3. Indexing ([]): The indexing operator allows you to access individual characters of a string by their position (index).

Example:

```
str1 = "Hello"
print (str1[0])  # output : H
print (str1[3])  # output : l
```

4. Slicing ([:]) : The slicing operator is used to extract a substring from a string by specifying a range of indices.

Example:

```
str1 = "Hello World"
print (str1[6:11])  #output : World
```

5. Membership (in, not in) : The membership operators are used to check if a substring exists in a string.

Example:

```
str1 = "Hello World"
print ("World" in str1)      #output: True
print ("python" not in str1) #output : True
print ("print" in str1)      # output : False
print ("Hello" in str1)      # output : True
```