PYTHON

History of python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido Van Rossum in 1991 and developed by python Software foundation.
It was mainly developed for emphasis on code readability and its syntax allows programmers to express concepts in fewer lines of code.

Features in python

There are many features in python, some of which are discussed below:-

1. Easy to code:
Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C++, Javascript, Java etc. It is very easy to code in python language.

2. Free and open source:
Since it is open source, this means that source code is also available to public. So you can download it as, use it

as well as share it.

3. Object - oriented language:-
One of the key features of python is object-oriented programming. python support object oriented language and concept of classes, object encapsulation, etc.

4. GUI programming support:-
Graphical user interface can be made using a module in python. PyQt5 is the most popular option for creating graphical apps in python.

5. High-level language
python is a high level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

6. Python is Portable language:
Python language is also a portable language.
For example, if we have a python code on other platform such as Linux, unix, mac then we do not need to change it we can run this code on any platform.

7. Python is Integrated language:-
Python is also an Integrated language because we can easily integrated python with other languages like C, C++ etc.

✗ Application of python

- Embeded Scripting language:-
Python is used as an embeded & Scripting language for various testing / building / deployment / monitoring framework, Scientific apps, and quick Script.

- 3D Software:-
3D Software like: Maya uses python for automating small user task, or for doing more complex integration such as talking to database and asset management system.

- Web development:-
Python is an easily extensible language that provide good integration with database and other web standards.

- GUI- based desktop application:-
Simple Syntax, modular architecture, rich text processing tool and the ability to work on multiple operating system makes python a preferred choice for developing

desktop based application.

Games:-
python has various module, libraries and platform that support development of games. Games like civilization-IV disney Toontown online, Vega Strike, etc are coded using python.

- Operating System:-
python forms as integral part of linux distribution.

* Python Distribution

* Cpython:- The standard and most widely used python implementation.

* Anaconda:- A distribution focused on data science and machine learning comes with many pre-installed libraries.

* PyPy:- A fast and efficient implementation of python.

* Iron python:- Runs on the .NET framework, allowing integration with .NET application.

* Python Version

- Python 3.12.2, documentation released on 6 Feb 2024
- Python 3.12.2, documentation released on 8 dec 2023.
- python 3.12.0, documentation released on 2 oct 2023.

* python has two major versions that are widely used: Python 2 and Python 3. Python 2 reached its end of life in 2020. and python 3 is the current and recommended version.

* Python IDEs :-

* Pycharm :- A popular IDE developed by Jet Brains. Known for its robust features.

* Visual Studio Code :- A versatile code editor with python support through extension.

* Jupyter Notebook :- An interactive environment for data science and research.

* IDLE :- Python is built in IDE, simple and lightweight.

* python Interpreter :-
The python Interpreter is the program that executes python code. When you write and run a python script, the interpreter reads and executes the code line by line producing the output.

Execution of Python programs

Python program are executed by running the python interpreter. you can run a python script from the command line using the python command.

- python program can be executed in various ways:-
- using an Integrated Development Environment (IDE) like pycharm, VScode or Spyder.
- using Command line Interface (CLI) by running the python interpreter (eg. python myscript.py).
- using an online python interpreter or REPL (Read-Eval-print Loop) like Google Colab or Replit.

- Debugging Python Code
- Debugging is the process of finding and fixing errors in code.
- Python has several built in tools and techniques to help with debugging:
- using the Pdb module for interactive debugging.

* Best practices for python programming

- Follow the python Enhancement proposals (PEPs) for coding style and conventions.
- use meaningful variable names and follow naming conventions.
- Write readable, concise, and modular code.
- use comment and documentation to explain the code.
- Test and validate your code regularly.

* Character set

- python supports a wide range of character, including:
- ASCII character (e.g. letters, digits, punctuation)
- unicode character (e.g. emojis, non-ASCII letters).
- Special character (e.g. newline, tab, backslash)

* Tokens

Tokens are the smallest individual element of a program. In python, tokens include keywords, identifiers, literals operators and punctuation symbol.

- Keywords (e.g. if, for, def)
- Identifiers (e.g. variable names, function names).

* Best practice for variable Name

- use meaningful and descriptive names for variables.
- use underscore to seprate words in a variable name (my_variable_Name).
- Avoid using abbreviations or single-letter variable names unless they are commonly used (x for a coordinate.)

* Assignment

- Assignment is the process of assigning a value to a variable.
- In python, assignment is done using the assignment operator (=).
- The syntax for assignment is:
  Variable_name = value

Examples

x = 5 assign the value of 5 to the variable x.

Types of Assignment

- python support several types of assignment;
  - Simple assignment: x = 5
  - multiple assignment: x, y, z = 1, 2, 3
  - Augmented assignment: x += 5
    (equivalent to x = x + 5)

* Visual studio code:- (famously known as vs code. is a free. open source. text editor by microsoft, vs code. is available. for windows, linux, and macos.

- Although the editor is relatively lightweight it includes some powerful features that mas vs code. one of the most popular development environment tools in recent time.

- vs code. support a wide array of programming language. from java, c++ and python to CSS, go and Dockerfile.

* pycharm

- pycharm is a hybrid platform developed by jetbrains as an IDE for python. It is commonly used for python application development.

- Some of the unicorn organization such as twitter, facebook, Amazon and Pinterest use pycharm in their python IDE.
- He can run pycharm on windows, linux or Mac. Additonally it contains modules and packages that help programming or develop software using python.

DATE
PAGE NO.

Python Interpreter

- A python Interpreter is a computer program that converts each high level program statement into machine code.

Compiler	Interpreter
• It scans the entire program first and translate it into machine code.	It scan the program line by line and translate it into machine code.
• Compiler show all error and warning at same time.	Interprete, show one error at a time.
• Error occur after scanning the whole program.	Error occurs after scanning each line.
• Debugging is slow	Debugging is faster.
• Execution time is less	Execution time is more.
• Compiler is used for by language such as C, C++ etc.	An interpreter is used as Java, python etc.

* Indentation:-

Indentation in python, a significant respect of the language. Syntax used to define code blocks.

- python uses indentation instead of brackets to indicates blocks of code.

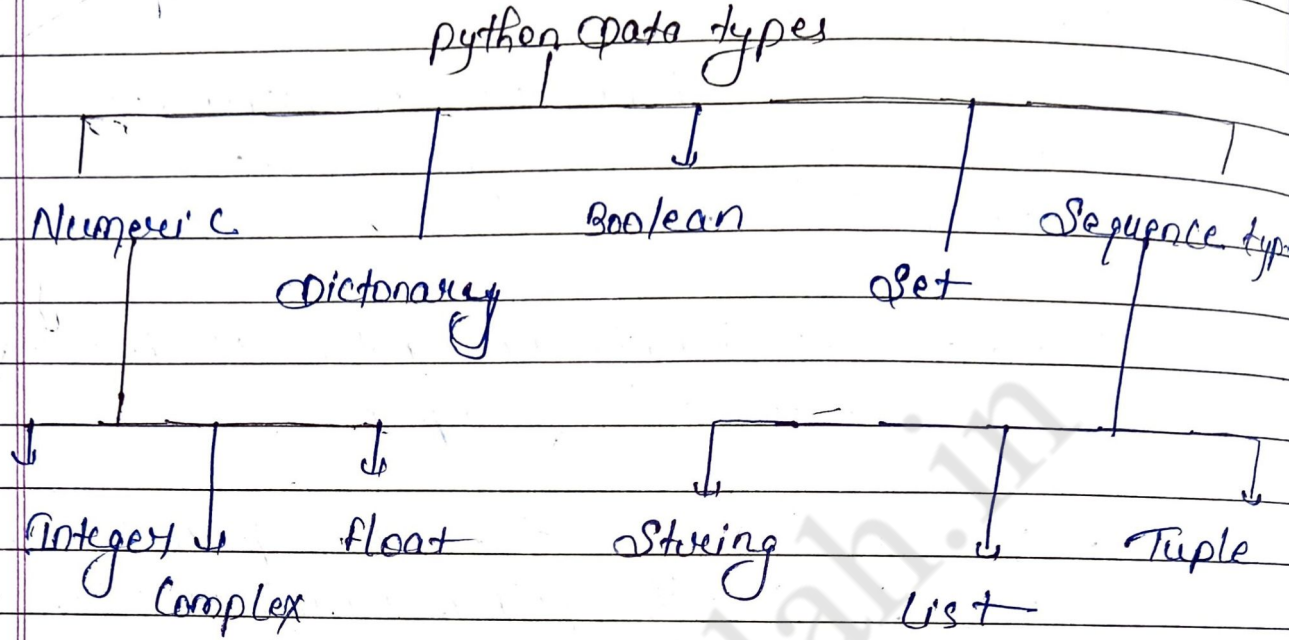
- If we change the line of code were not enough indented, python would not be able to execute the code within the correct order, and the program would not work as intended.

Ex- If we are writing a code if a particular is even or odd.

```
Number = 20
If number != 0;
    if number % 2 == 0;
        print ["Number is even")
    else:
        print ("Number is odd")
else:
    print (" Number is neither even or odd")
```

* What is data types?

→ Data type represent the different kinds of values that we stored on the variable.



~~X~~ What is Set?

→ A Set is a datatype in python to store several items in a single variable. It is one of the four built in data types. having qualities and usage different from the other three.

It is a collection that is written with ~~curly~~ curly brackets and is both unindexed and unordered.

- A Set is mutable ie we can remove or add elements to it, Set in python.
- Items of a Set in python are immutable do not duplicate value and unordered.

and in underscore (_)
* An identifier cannot begin with a digit if an identifier start with a digit it will give a syntax error.

* In python Keywords are reserved word that are built in to python, so, a Keyword cannot be used as an identifier.

* Special Symbols like !, @, #, $ etc are not allowed in identifiers.

* python identifiers cannot only contain digit.

* Identifiers names are case sensitive.

* What is function? full explanation.

function is nothing but group a statement which only executed when we call the function.

Note:-

i) code reusability
ii) function return values a result.
iii) we can pass value to ~~this~~ the function called parameters.

* Input() function:-

- In python programs we use the input() function to take any kind of data input from the user through the keyboard.

- The input() function waits for the user's input and as soon as the users enters the data and presses the enter key, the input() function reads that data and assign it to the variable.

Syntax - var_name = input().

- input() function always return string value.

- print() function :-
- It is used to display the output on the console.

Some output method

```
var = eval(input("enter data:"))
print(var)
print(type(var)).
```

give all type of datatype ex- suppose we enter 12 then it return int, if we enter 12.5 then it return float.

② print ("Hello," "Ankush", "Sep =>)

output => (Hello — Ankush — Ankit)

③ print ("Hello," end = " ")
print ("world")

output
Hello world.

Datatypes
Datatypes represent the different kinds of values that we stored on the Variables.
Datatypes are -
Integer — +ve, -ve, 0, (25, -25, 0)
String — "Sagar".
float — 3.99, 2.89
Boolean — True, False
None — none

Unit-2

Input/output

* How to take input from user in python

Sometimes a developer might want to take user input at some point in the program. To do this python provides an input() function.

Syntax:-

```
Input('prompt')
```

where prompt is an optional string that displayed on the string at the time of taking input.

Example:-

```
# Taking input from user

name = input("Enter your name")
# output
print("Hello", +name)
print(type(name))
```

output

```
Enter your name : Sagar Kumar
Hello, Sagar Kumar
<class 'str'>
```

* How to take multiple input in python:
We can take multiple input of the same data type at a time in python, using map() method in python.

Example

```
a,b,c = map(" Enter the Numbers:").split()
print("The numbers are:", end=" ")
print(a,b,c)
```

Output

```
Enter the Numbers : 2 3 4
The numbers are : 2 3 4
```

* How take inputs for the sequence data types

In case of list and set the input can be taken from the user in two ways.

1. Taking list/set elements one by one by using the append()/add() methods.
2. using map() and list()/set() methods.

→ Taking list/set element one by one.

Take the element of list/set one by one and use the append() methods in the case of list, and add() method in the case of a set, to add the element to the list/set.

```
List = list()
Set = set()
I = int(input("Enter the size of list:"))
S = int(input("Enter the size of Set:"))
print("Enter the list elements:")
for i in range(0,I)
list.append(int(input()))
print("Enter the Set element:")
for i in range(0,S)
    Set.add(int(input()))
print(list)
print(Set)
```

Output

```
Enter the size of the list: 4
Enter the size of the Set: 3
Enter the list elements:
9
0
1
3
Enter the Set elements:
2
9
1
[9, 0, 1, 3]
[9, 2, 1]
```

Formatting output

- Formatting output in python can be done in many ways. Let's discuss them below:

- using formatted String literals.

We can use formatted Strings literals, by starting a string with f or F before opening quotation marks or triple quotation marks. In this string, we can write python expressi between { and } that can refer to a variable or any literal value.

Example

```
# Declaring a variable.
name = "Sagar Kumar"
# output
print(f'Hello {name}! How are you?')
```

Output

'Hello Sagar Kumar! How are you?'

- using format()

We can also use format() function to our output to make it look presentable. The curly braces { } work as placeholders. We can specify the order in which variable occur in the output.

Operators

- The operator is a symbol that performs a specific operation between two operands, according to one definition.
- operators serve as the foundation upon which logic is constructed in a program in a particular programming language.
- In every programming language, some operators perform several task. Same as other language python also has some operators, and these are given below:-
  - Arithmetic operators
  - Comparison operators
  - Assignment operators
  - Logical operators
  - Bitwise operators
  - Membership operators
  - Identity operators

* Arithmetic operators in python

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

* operator	Description	Syntax
+	Addition: adds two operands	x+y
-	Subtraction: subtract two operands	x-y
*	multiplication: multiply two operands	x*y
/	Division: divides the first operands by the second	x/y

Operator	Description	Syntax
//	Division (floor): divides the first operands by the second.	x // y
%	modulus: returns the remainder when the first operands is divided by the second.	x % y
**	power: Returns first raised to power second.	x ** y

Example

```
a = 9
b = 4
# Addition of number
add = a+b
# Subtraction of number
Sub = a-b
# multiplication of number
mul = a*b
# modulo of both number
mod = a%b

print(add)
print(sub)
print(mul)
print(mod)
```

output

```
13
5
36
1
```

* Comparision operators in python

In python comparision or Relational operators compares the values. It either returns True or false according to the condition.

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right.	x > y
<	less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal.	x == y
!=	Not equal to - True if both operands are not equal	x != y
>=	Greater than or equal to true if the left operand is greater than or equal to the right	x >= y
<=	less than or equal to true if the left operand is less than or equal to the right.	x <= y

* Assignment operators in python

python Assignment operators are used to assign values to the variables.

DATE
PAGE NO.

Operator	Description	Syntax
=	Assign the value of the right side of the expression to the left side operands.	x = y+2
+=	Add AND: Add right-side operand with left-side operand and then assign to left operand.	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand.	a-=b a=a-b
*=	multiply AND: multiply right operand with left operand and then assign to left operand.	a*b a=a*b
/=	divide AND: divide left operand and then assign to left operand.	a/=b a=a/b.

Example

```
a=10
# Assign value
b=a
print(b)
# Add and assign value
b+=a
print(b)
# Subtract and assign value
b-=a
```

```
print(b)
# multiply and assign
b* = a
print(b)
```

Output

10
20
10
100

* logical operators in python

python logical operators perform logical AND, logical OR and logical NOT operations.

It is used to combine the conditional statements.

operator	Description	Syntax
and	logical AND: True if both the operands are true.	x and y
or	logical OR: True if either of the operands is true.	x or y
not	logical NOT: True if the operands is false	not x

Example

```
a = True
b = false
# print a and b is false
print (a and b)
# print a or b is True
print (a or b)
# print not a is false
print (not a)
```

output

false
True
false

Bitwise operators in python

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

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x >>
<<	Bitwise left shift	x <<

Example

```
# Example of Bitwise operators.

a=10
b=4
# print bitwise AND operation
print(a & b)
# print bitwise OR operation
print(a | b)
```

output

```
0
14
```

* Identity operators in python

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.

Ch-3

Control flow : Conditional Blocks

In python, control flow is managed using conditional blocks that allow you to execute different parts of the code based on certain condition. The main constructs for conditional control flow are 'if', 'elif' & 'else' statements.

1. 'If' Statement :-
   The if statement evaluates a condition. If the condition is 'True' the block of code inside the 'if' statement is executed.

```
age = 18
if age >= 18:
    print("you are an adult")
```

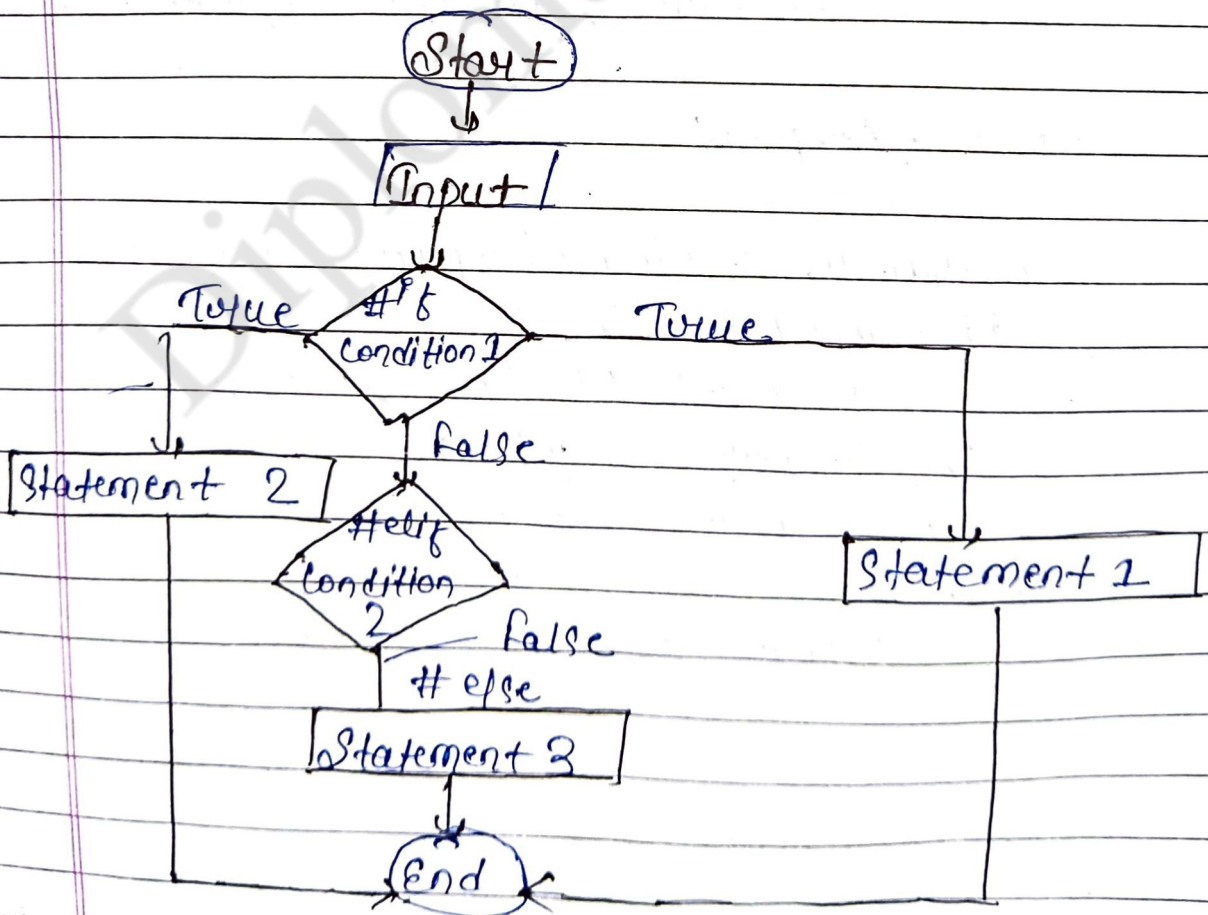
2. elif Statement :
   The 'elif' (short for 'else if') statement allows you to check multiple expression for 'True' and execute a block of code as soon as one of the conditions is 'True'. It is optional and comes after an 'if' statement.

3. 'else' Statement:

The else statement catches anything that isn't caught by the preceeding conditions. It is optional and comes after the 'if' and 'elif' statements.

```
age = 10
if age > 18:
    print("you are an adult")
elif age >= 13:
    print("you are a teenager")
else:
    print("you are a child").
```

Flow chart four conditional statements



* Nested 'if' Statements

you can nest 'if', 'elif' and 'else' statement within each other to create more complex conditions.

```
num = 10
if num > 0
    print("The number is positive")
    if num % 2 == 0:
        print("The number is even.")
    else:
        print("The number is odd.")
else:
    print("The number is not positive")
```

more examples:

```
temperature = float(input("enter the Temp"))
if temperature > 30:
    print("It hot outside.")
elif temperature > 20:
    print("The weather is nice")
else:
    print("Its cold outside")
```

PAGE NO.

* Key points

- Indentation :- Python uses indentation to define the scope of the 'if', 'elif' & 'else' blocks, proper identation is crucial.

- order of conditions :- conditions are evaluated in 'True' ~~the~~ ~~true~~ order once a condition is 'True' the corresponding block of code is executed and the remaining conditions are not checked.

- chaining conditions :- use logical operators to chain multiple conditions within a single 'if' statement.

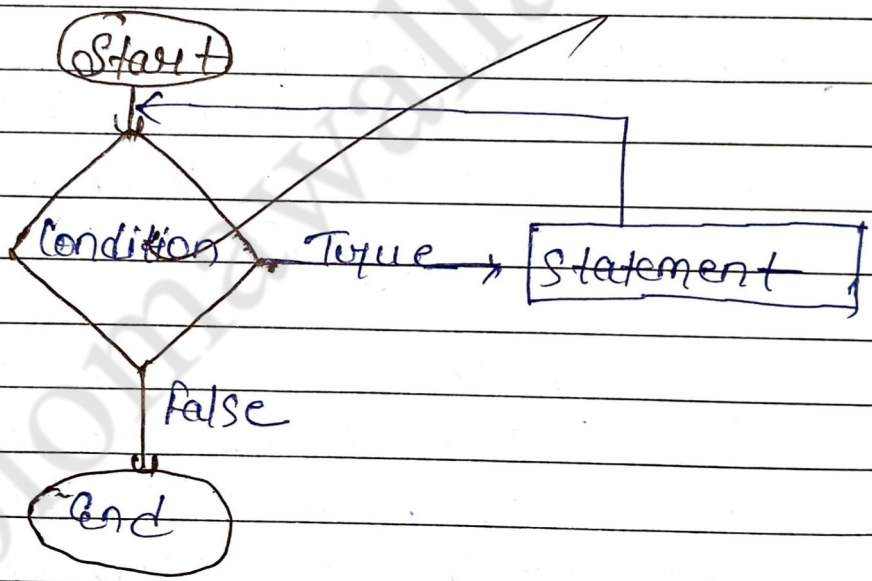
using conditional blocks effectively allows you to control the flow of your program based on dynamic conditions and user input, making your code more flexible and interactive.

Ch-4

Control flow: loops

In python, loops are used to execute a block of code repeatedly as long as a specified condition is met. There are two types of loop in python: 'for' loops & 'while' loops.

We can easily understand the flow of loops by the flow chart below:



Flow Chart (Control flow: loops)

1. While loop:

The While loop in python repeatedly execute a block of code as long as a specified condition evaluates to 'True'

- The general format of a while loop is straight forward and involves

Initializing a condition, checking the condition, executing a block of code if the condition is 'True' and then updating the condition.

General format of a 'while' loop:

```
while condition:
    # code block to be executed
    # (usually involves updating the condition
      to eventually break the loop)
```

Component of a while loop:

1. Initialization: Before the while loop starts you can often initialize one or more variables that will be used in the loop's condition.

2. Condition: The loop continues to execute as long as this condition remains 'True'. If the condition is 'False', the loop terminates.

3. Code Block:- The block of code that is executed on each iteration of the loop as long as the condition is True.

4. update:- within the code block, you typically update variables to eventually make the condition 'False'. If the condition is not properly updated, you might end up

with an infinite loop.

Example:-

```
count = 1                    # Initialization.
while count <= 5             # condition
    print(count)             # code block
    count += 1               # update.
```

2. For loop:
The for loop in python is used to iterate over a sequence. It is commonly used for iterating through items in a collection.

General format of a for loop:

```
for variable in sequence:
    # code block to be executed.
```

Component of a for loop.

1. Initialization: The loop variable ('variable') is initialized to the first item in the sequence.

2. Condition:- The loop continues as long as there are items remaining in the sequence.

3. Code block:- The block of code that is executed for each item in the sequence.

4. update: After executing the code block, the loop variable is automatically updated to the next item in the sequence.

Example:

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(fruit)
```

Range():

The range() function is used to generate a sequence of number. It is commonly used in for loops to iterate over a sequence of numbers. The 'range()' function can take one, two, or three arguments.

Basic usage of 'range()'

1. one argument: 'range(stop)'
when range() is called with a single argument it generates numbers from 0 up to but not including the specified 'stop' value.

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

2. Two Arguments : range (start, stop)

When range() is called with two arguments it generates numbers from the start value up to but not including the 'stop value'

```
for i in range (2,6):
    print (i)
```

3. Three Argument: range (start, stop, step)

When range () is called with three argument it generates numbers from the start value up to but not including the stop value incrementing by the 'step value'.

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

* Default value:
- The default value of 'start' is 0.
- The default value of 'step' is 1.

* Using range() to Iterate over Indexes:

The range() function is often used to iterate over the indexes of a list or other sequences:

```
fruits = ["apple", "banana", "cherry"]

for i in range(len(fruits)):
    print(f"Index: {i}, Fruit: {fruits[i]}")
```

* Converting range() to a list:

You can convert the sequence generated by 'range()' to a list using the list() function:

```
numbers = list(range(5))
print(numbers)
```

* Nesting loops and Conditional Statements:

Nesting loops and contional statement allows for more complex and dynamic control flows within your programs.

* Nested loops:

Nested for loops.
You can nest for loops to iterate over multi-dimensional data structure or perform repeated actions within repeated actions.

Example: multiplication table

```
for i in range(1, 11):        # outer loop
    for j in range(1, 11):    # inner loop
        print(f"{i} * {j} = {i*j}")
    print()                   # for better readability.
```

* Nested while loops

You can also nest while loops to perform repeated actions within repeated actions.

Ex- Counting Down

```
i = 3
while i > 0:
    j = 3
    while j > 0:
        print(f"i: {i}, j: {j}")
        j -= 1
    i -= 1
    print()
```

Nesting conditional statement within loops

You can also nest conditional statements within loops to create more dynamic behaviour based on specific conditions.

```
for i in range(1,6):
    if i%2 == 0:
        print(f"{i} is even")
    else:
        print(f"{i} is odd")
```

* Controlling loop execution:

Controlling loop execution involves using statements that alter the normal flow of the loop. Python provides several ways to control loop execution using 'break', 'continue' and 'pass' statements, as well as combining loops with conditional statements. Lets explore these in detail.

Break statement:

The break statement is used to exit the loop immediately, regardless of the loops condition. It stops the execution of the loop and transfer control statement following the loop.

Example:

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

Chapter - 5

Data collection

Python offers several built-in data collections to store and manage data efficiently. These collection includes list, tuple, dictionaries, and sets. Each type has different properties and use cases.

Concept of Mutability :-

Mutability refers to whether or not an object can be changed after it has been created.

- Mutable objects :- These objects can be changed after creation. Examples include lists, dictionaries and sets.

- Immutable objects :- These objects cannot be changed after creation.
  Examples include tuples, String and numbers.

* Sets in python :

Sets are a type of mutable data collection in python. They are unordered collection of unique elements, meaning they do not allow duplicate values. Sets are useful for storing elements where order does not matter and for performing membership tests.

Chapter - 5

Data collection

python offers several built - in data collections to store and manage data efficiently.
These collection includes list, tuple, dictonaries, and sets. Each type has different properties and use cases.

Concept of Mutability :-

Mutability refers to whether or not an object can be changed after it has been created.

- Mutable objects :- These objects can be changed after creation. Examples include lists, dictonaries and sets.

- Immutable Objects :- These objects cannot be changed after creation.
  Examples include tuples, string and numbers.

* Sets in python :

Sets are a type of mutable data collection in python. They are unordered collection of unique elements, meaning they do not allow duplicate values. Sets are useful for storing elements where order does not matter and for performing membership tests.

Features of Sets:

- Unordered: The item have no defined order.
- unique: No duplicate elements are allowed.
- mutable: you can add and remove element.
- Dynamic size: They can grow and shrink as needed.
- Efficient membership Testing: checking for the existence of an element is fast.

* Declaration and Initialization:
you can declare and initalize sets using curly braces '{}' or the set() function.

using curly braces

```
my_set = {1,2,3,4}
print(my_set)  # output {1,2,3,4}
# using the set() function
another_set = set([1,2,3,4])
print(another_set)  # output {1,2,3,4}
```

Creating an empty set (using empty curly braces creates an empty dictionary)

```
empty_set = set()
print(empty_set)  # output: set()
```

* Set operations:
Set support a variety of operations, including union, intersection, difference and symmetric difference.

1. Adding Element:- use the 'add()' method.

```
my_set = {1,2,3}
my_set.add(4)
print(my_set)  # output: {1,2,3,4}
```

2. Removing Elements: use the 'remove()' or 'discard()' method.

```
my_set = {1,2,3,4}
my_set.remove(4)
print(my_set)  # output {1,2,3}
# my_set.remove(5)

my_set.discard(3)
print(my_set)  # output: {1,2}
my_set.discard(5) # does not raise an error.
```

3. union:- Combines elements from both sets, removing duplicates.

```
Set 1 = {1, 2, 3}
Set 2 = {3, 4, 5}
union_set = Set 1 | Set 2
print(union_set)  # output {1, 2, 3, 4, 5}
```

4. Intersection: Contains only elements that are in both sides.

```
intersection_set = Set 1 & Set 2
print(intersection_set)  # : {3}
```

* Set Comprehension

Set comprehension is a concise way to create sets. It follows the same principles as list comprehension but uses curly braces.

* Summary

- Sets are unordered, mutable, collections of unique elements.
- Declaration and Initialization can be done using curly braces or the 'set()' function.
- Set operation include adding and removing elements union, intersection difference, and

Symmetric difference.

- Set comprehension provides a concise way to create sets based on existing ~~iterates~~ iterables.

Tuples in python:

Tuples are an ordered, immutable collection type in python that can hold a sequence of element of a tuple cannot be changed once the tuple is created. Tuples are often used to represent fixed collection of items and their immutability makes them hashable and suitable for use as key in dictionaries.

Features of Tuples:

- Ordered: The elements have a defined order, which will not change.
- Immutable: once a tuple is created, its elements cannot be changed, added or removed.
- Heterogeneous: Tuples can contain elements of difference type.
- Hashable: Since tuples are immutable, they can be used as keys in dictonaries if all the elements are hashable.

* Declaration and Initialization
Tuples are declared using parenthesis () or without any brackets (just commas separating the values)

using parenthesis.

```
my_tuple = (1,2,3)
print(my_tuple)      # output (1,2,3)
```

* Basic operations:
Tuples support various operations similar to list, but since they are immutable, operations, that modify the tuple directly are not supported.

* Concatenation
you can concatenate two tuples using the '+' operator.

```
tuple1 = (1,2,3)
tuple2 = (4,5,6)
concatenated_tuple = tuple1 + tuple2
print(concatenated_tuple).
```

* Repetition:
you can repeat the element of a tuple using the '*' operator

```
tuple1 = (1,2,3)
repeated_tuple = tuple * 3
print(repeated_tuple)
```

* Indexing and Slicing:
Tuples support indexing and slicing operation similar to lists.

- Indexing:
Access elements by their position using square brackets '[]'.

```
my_tuple = (1,2,3,4,5)
print(my_tuple[0])
print(my_tuple[-1])
```

- Slicing:
Access a range of elements using the slicing syntax 'start:stop:step'.

```
my_tuple = (1,2,3,4,5)
print(my_tuple[1:4])
print(my_tuple[:3])
print(my_tuple[::2])
```

* Built in Functions
Several built in functions can be used with tuples.

len(): Returns the number of elements in the tuple.

```
my_tuple = (1, 2, 3, 4, 5)
print (my_tuple)
```

min() and max() : Return the smallest and the largest element in the tuple, respectively.

Sum() : Returns the sum of all elements in the tuple.

Sorted() : Returns a new list containing all elements of the tuple in sorted order.

* Summary

- Tuples are ordered, immutable collections
- Declaration can be done using parentheses or commas.
- Basic operation includes concatenation and repetation.
- Indexing and Slicing allow access to individual elements or range of elements.
- Nested Tuples allows tuples to contain other tuples, enabling complex data structure.

Ch.6

List

List are one of the most versatile and widely used data structure in python. They are mutable, ordered collections of items that can be of any data type list are used to store multiple items in a single variable and are highly flexible in terms of what they can hold and how they can be manipulated.

* Features of lists:

- **ordered**: The elements in a list have a defined order.
- **Mutable**: List can be changed after creation (elements can be added, removed, or changed)
- **Heterogeneous**: List can contain elements of different data types.
- **Dynamic size**: List can grow and shrink in size as needed.

* Declaration and Initialization:

List can be declared and initialised using square bracket [] or the list() function.

using Square bracket—

```
my_list = [1,2,3,4]
print(my_list)
```

using the list() function

```
another_list = list([1,2,3,4])
print(another_list)
```

Creating an empty list—

```
empty_list = []
print(empty_list)
```

Basic operation :

* Adding element

- append() : Adds a single element to the end of the list.
- extend() : Adds multiple elements to the end of the list.
- insert() : Adds an element at a specified position.

* Removing element—

- remove() : Remove the first occurance of a specified value.
- pop() : Remove and returns the element at a specified position (or the last element if no position is specified)
- clear() : Removes all elements from the list.

* modifying elements:
you can change the value of elements in a list by assigning anew value to a ~~Spec~~ specific index.

* Indexing and Slicing
Indexing: Access elements by their position using square brackets '[]'.

Slicing:- Access a range of elements using the slicing syntax 'start: stop: step'.

* List Iterations:
You can iterate over the elements of a list using loops.

```
my_list = [1,2,3,4,5]
for item in my_list:
    print(item)
```

Built-in function:- "Same as tuple".

* Nested List
Lists can contain other lists as elements, creating nested structure.

```
nested_list [[1,2], [3,4], [5,6]]
# Accessing elements in a nested list
print(nested_lists[2][0]) #output: 5
```

List Comprehensions:

List comprehensions provide a concise way to create lists based on existing lists.

Create a list of squares of numbers from 1 to 5.

```
squares = [ x **2 for x in range (1,6)]
print (squares) # output: [1,4,9,16,25]
```

Application of lists.

lists are used in a variety of application including

- Data Storage :- Store collection of item.
- Iteration :- perform operation on each term in collection

Dynamic Arrays: Resize array as needed during program execution.

- Matrix Representation :- Represent matrix using nested list.
- Data filtering :- Filter and process data using list comprehension.
- Queue and Stack Implementations :- Implement queues and stacks using list operations.

DATE
PAGE NO.

Summary

- Lists are ordered, mutable collections of elements.
- Declaration and Initialization can be done using square brackets or the list() function.
- Basic operation include adding, removing and modifying elements.
- Indexing and slicing allows to individual elements or range of ~~elem~~ elements.
- Built in function like ~~len~~ len(), min(), max(), sum() and sorted(), are useful for working with list.
- Nested lists allows lists to contain other lists, enabling complex data structure.
- List comprehension provides a consise way to create list based on existing iterables.
- Application of lists are vast and include data storage, iteration, matrix representation and more.

Chapter - 7

Dictionaries

Dictionaries are an essential and powerful built in data structure in python, used to store data in key-values pairs. They are mutable, unordered collection that map unique keys to values, allowing for efficient data retrieval and manipulation.

* Features of Dictionaries :-

- unordered : The item have no define order.
- mutable ; Dictionaries can be changed after creation you can change (add, remove, or modify items)
- Heterogeneous ; Keys and values can be of any data type.
- Key - value pairs ; Each item in a dictonary is a pair of a key and a value.
- Key - must be unique :- Each key in a dictionary must be unique.
- Key must be Immutable :- Keys must be of an immutable data type. eg- Tuples, Strings.

Declaration and Initialization:
Dictonaries can be declared and initialized using curley braces '{ }' or the 'dict()' function.

using curley braces

```
dict = {"name": "Sagar", "age": 25, "city": 'Tokoyo"}
print(dict)
```

* Basic operations:
Dictonaries support various operations including adding, removing and accessing items.

Accessing values:
you can access values using their keys:

```
dict = {'name': 'Alice', 'age': 25, 'city': 'Sagar'}
print: ( Dict ['name']
print ( Dict.get ('age'))
```

* Adding and modifying Items:
you can add new key-value pairs or modify existing ones by assigning a value to a key:

Dict = {'name': 'Sagar', 'age': 25}

Adding new element
Dict['City'] = 'New York'

Removing Items:
You can remove items using the 'del' statement or the pop statement.

Iterating through Dictonaries:
you can iterate through Dictonaries using loops.

Iterating through keys:

```
for key in dict.keys():
    print(key, dict[key])
```

Iterating through values:

```
for value in dict.values():
    print(value)
```

Iterating through key-value pairs:

```
for key, value in dict.items():
    print(key, value)
```

Built in function:
Several built function can be used with Dictionaries.

len()
Returns the numbers of key-value pairs in the dictionary.

```
dict = {'name': 'Sagar', 'age': 25, 'city': 'Tokyo'}
print(len(dict))
```

* .keys(), values(), and items()
Return views of the dictionaries keys, values, and key-value pairs respectively.

```
dict = {'name': 'Sagar', 'age': 25, 'city': 'Tokyo'}
print(dict.keys())
print(dict.values())
print(dict.items())
```

clear()
Removes all items from the dictionary.

```
dict = {'name': 'Sagar', 'age': 25, 'city': 'Tokyo'}
dict.clear()
print(dict)
```

* Dictionary comprehensions:
Dictionary comprehension provide a concise way to create dictionaries from existing iterables.

```
# Create a dictonary with their no. and squares.
squares = {x: x**2 for x in range(1, 6)}
print(squares)
```

Chapter – 8

Arrays and Strings

Arrays :-

Array are a collection of items stored at contiguous memory location. In python arrays are available through the 'array' module though lists are more commonly used. However for numerical operations the 'numpy' library provides a more powerful array object.

Features of Arrays :—

- Fixed size :— The size of an array is defined at the time of creation.
- Homogeneous elements :— Arrays typically store elements of the same type.
- Efficient memory usage :— Arrays use less memory and are more efficient for numerical operations compared to lists.

Creating and Initialization Arrays :—

using the 'array' module

```
import array
# Creating an Integer array
int_array = array.array('i', [1,2,3,4])
```

print(int_array)

using the 'numpy' library

```
import numpy as np
# Creating a numpy array
np_array = np.array([1,2,3,4])
print(np_array)
```

Indexing and Transversal:-

```
# Accessing elements.
print(int_array[0])
print(int_array[2])

# Transversing an array
for elements in int_array:
    print(element)
```

* Manipulation:-

```
# Adding element using array
int_array.append(5)
print(int_array)

# Remove element using array
int_array.remove(2)
print(int_array)

# Adding elements (using numpy)
np_array = np.append(np_array, [5,6])
print(np_array)
```

Strings

Strings in python are sequence of character. They are immutable, meaning once a string is created, it cannot be changed.

Creating And Assignment Strings.

```
# Creating a String
my_string = "Hello, world!"
print (my_string)

# Assigning a string
another_string = 'python is fun'
print (another_string)
```

Indexing and Slicing

```
# Accessing character by index
print (my_string[0])

# Slicing String.
print (my_string[0:5])
print (my_string[7:])
print (my_string[::-1])
```