1)Introduction to Python and its Features (simple, high-level, interpreted language).

-> Python is a popular, powerful, and easy-to-learn programming language. It was created by Guido van Rossum and released in 1991.

-> It is used for **web development, data analysis, AI, machine learning, scripting, automation**, and many other areas.

-> Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

-> Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development.

-> Python supports modules and packages, which encourages program modularity and code reuse.

\*Features of Python

-> Python has a **clean and easy-to-read syntax**.

-> Python is a **high-level language**, meaning you don’t need to manage low-level details like memory management.

-> Python code is **executed line by line** by the Python interpreter.

-> This makes it **easy to test and debug**.

-> You don’t need to declare the type of a variable.

-> Python supports **object-oriented programming (OOP)** and **functional programming**.

-> Free and Open Source

2)History and evolution of Python.  
-> Created by Guido van Rossum in 1989 (Netherlands).

-> Named after *Monty Python* comedy group.

-> Goal: Easy, readable, powerful language.

-> 1991: First version (0.9.0) released. Included functions, modules, exceptions, classes.

-> 1994: Python 1.0 released. Added functional tools (map, filter, reduce).

-> 2000: Python 2.0 released. New features: List comprehensions, Unicode support, garbage collection. Last version: 2.7 (2010).Official support ended in 2020.

-> 2008: Python 3.0 released (not backward-compatible). Improved Unicode, cleaner syntax.Key features over time:

* f-strings
* async/await
* pattern matching (3.10+)

-> Still evolving (3.12, 3.13…).

3) Advantages of using Python over other programming languages

-> Easy to read and write.

-> Works on any platform.

-> Has lots of libraries for many tasks.

-> Big community for help.

-> Great for web, data science, AI, automation, etc.

-> Free and open source.

-> Fast to develop and test ideas.

4) Understanding Python’s PEP 8 guidelines.

5) Indentation, comments, and naming conventions in Python.

Indentation: -

Comments: -

-> A comment is text in your code that Python ignores when running the program. It’s for humans to read—notes, explanations, or reminders.

-> Single line comments Use the # symbol.  
-> multi-line comments Use multiple docstrings (“”” “”“) (’’’ ’’’) for documentation.

Naming conventions: -

General Rules

-> Allowed Characters: Identifiers (names for variables, functions, classes, etc.) can contain letters (a-z, A-Z), digits (0-9), and underscores (\_).

-> Start: They must start with a letter or an underscore, not a digit.

-> Case Sensitivity: Python is case-sensitive, so myVar and myvar are different names.

-> No Spaces: Spaces are not allowed in names. Use underscores to separate words (e.g., my\_variable).

-> No Special Characters: Special characters other than the underscore are not allowed.

-> Reserved Words: Avoid using Python keywords (e.g., if, else, for, while, print) as names.

Specific Naming Conventions

-> Variables: Use lowercase with words separated by underscores (e.g., user\_name, total\_count).

-> Constants: Use uppercase with underscores separating words (e.g., MAX\_VALUE, PI).

-> Functions: Use lowercase with underscores (e.g., calculate\_area, get\_user\_data).

-> Classes: Use CamelCase, where each word starts with a capital letter (e.g., MyClass, UserData).

-> Modules and Packages: Use lowercase names, avoid underscores and hyphens in package names.

-> Methods: Use lowercase with underscores. Non-public methods should start with a single underscore.

5) Writing readable and maintainable code

6) Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.

String: -

-> Represent whole numbers, positive or negative, without decimal points

Example:

a = 10 b = -5

print (a, b)

Floats: -

-> Represent numbers with decimal points or in exponential form

Example:

pi = 3.14

temperature = -2.5

print(pi, temperature)

String: -

-> Represent sequences of characters, enclosed in single or double quotes

Example:

name = “Alice”

message = ‘Hello, world!’

print(name, message)

List: - []

-> Ordered, mutable collections of items, allowing duplicates. Items are enclosed in square brackets []

Example:

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

numbers = [1, 2, 3, 4]

mixed = [1, "hello", 3.5]

print(fruits)

print(numbers)

print(mixed)

Tuples: - ()

-> Ordered, immutable collections of items, allowing duplicates. Items are enclosed in parentheses ()

Example:

coordinates = (10, 20)

colors = ("red", "green", "blue")

print(coordinates)

print(colors)

Dictionaries: - {}

-> Unordered collections of key-value pairs. Keys must be unique and immutable, while values can be of any type. Enclosed in curly braces {}

Example:

person = {"name": "Alice", "age": 25, "city": "New York"}

print(person)

Sets: - {}

-> Unordered collections of unique items. Duplicate elements are automatically removed. Enclosed in curly braces {}

Example:

numbers = {1, 2, 3, 3, 4}

print(numbers)

7) Python variables and memory allocation.

Variables: -

-> Variables are containers for storing data values.

Example:

a = 10

b = “hello”

print(a)

print(b)

-------------------------------------

a = 10

b = “hello”

print(type(a))

print(type(b))

------------------------------------

A = “kamini”

a = ‘kamini’

print(a)

print(A)

------------------------------------------

#mutiple values to multiple variables

x, y, z = “first”,”second”, ”third”  
print(x)

print(y)

print(z)

--------------------------------------------

# one value to multiple variables

x = y = z = “Hello”

print(x)

print(y)

print(z)

--------------------------------------------------

x = "Python"  
y = "is"  
z = "awesome"  
print (x, y, z)

------------------------------------------------

x = "Python "  
y = "is "  
z = "awesome"  
print (x + y + z)

Memory Allocation: -

-> Memory allocation in Python means giving space in memory to store data. Python does this automatically. When you create a variable, Python allocates memory for it. When you don't need it anymore, Python's garbage collector frees the memory.

8) Python operators: arithmetic, comparison, logical, bitwise.  
Arithmetic: -

->Used for basic mathematical operations.

-> + (Addition), - (Subtraction), \* (Multiplication), / (Division), % (Modulus (remainder)),

// (Floor division), \*\* (Exponentiation)

Comparison: -

-> Used to compare values; result is **True** or **False**.

-> == (Equal to) , != ( Not equal to), < (Grater then), > (Less then), <= (Less or equal),

>= (Grater or equal)

Logical: -

-> Used to combine conditional statements.

-> and -True if both are True

-> or - True if least one is True

-> not – Inverts the result

Bitwise: -

-> Operate on **bits** (binary representations).

& (AND), | (OR), ^ (XOR), ~ (NOT), << (Left shift),>> (Right shift)

9) Introduction to conditionalstatements: if, else, elif.

If: -

-> The if statement runs a block of code **if** the condition is true.

-> check condition

Syntax:

if condition:

Example:

age = 18

if age >= 18:

print ("You are an adult.")

If else: -

-> The else block runs if all previous if/elif conditions are false.

-> do something if all previous conditions were false

Syntax:

if condition:

# code if condition is true

else:

# code if condition is false

Example:

age = 16

if age >= 18:

print("You are an adult.")

else:

print("You are a minor.")

Elif: -

-> elif means else if. It lets you check multiple conditions.

-> check another condition if the first was false

Syntax:

if condition1:

# code if condition1 is true

elif condition2:

# code if condition2 is true

else:

# code if none are true

Example:

marks = 85

if marks >= 90:

print("Grade: A")

elif marks >= 80:

print("Grade: B")

else:

print("Grade: C")

10) Nested if-else conditions

-> A *nested if-else* is an if-else statement *inside* another if-else. It's used when you want to make further decisions based on a previous condition.

Syntax:

if condition1:

# block of code if condition1 is true

if condition2:

# block if both condition1 and condition2 are true

else:

# block if condition1 is true but condition2 is false

else:

# block if condition1 is false

Example:

num = 5

if num >= 0:

print("Number is non-negative.")

if num == 0:

print("Number is zero.")

else:

print("Number is positive.")

else:

print("Number is negative.")

11) Introduction to for and while loops

For: -

-> A **for** loop is used to iterate over a sequence (like a list, string, or range).

Syntax:

for variable in sequence:

# code block

Example: Using range ()

for i in range(5):

print(i)

Example: Iterating over a list

fruits = ['apple', 'banana', 'cherry']

for fruit in fruits:

print(fruit)

While: -

-> A **while** loop runs **as long as** a condition is true.

Syntax:

while condition:

# code block

Example:

count = 0

while count < 5:

print(count)

count += 1

12) How loops work in Python.

For loop: -

-> Used to iterate over a sequence (like a list, tuple, string, or range).

Example:

for i in range(5):

print(i)

While loop: -

-> Runs as long as a condition is True.

Example**:**

count = 0

while count < 5:

print(count)

count += 1

Break and Continue: -

-> break = exit the loop immediately.

-> continue = skip the rest of the current iteration.

Example: (break)

for i in range(10):

if i == 5:

break

print(i)

Example: (Continue)

for i in range(5):

if i == 2:

continue

print(i)

Loop with else:

-> Python loops can have an else clause. It runs only if the loop **did not break**.

Example:

for i in range(5):

print(i)

else:

print("Loop finished without break")

13) Using loops with collections (lists, tuples, etc.).

-> Using loops with collections like **lists**, **tuples**, and **sets** in Python is a common way to iterate over elements and perform actions. Here are some examples:

-> For loop with List:-

l = [“First”,”second”,”Third”,”Fourth”]:

for i in l:

print(i)

-> For loop with tuple:-

t = (‘red’,’green’,’white’,’orange’)

for i in t:

print(i)

-> For loop with set:-

s = {1,2,3,4,5}

for i in s:

print(i)

14) Defining and calling functions in Python.

-> In Python, **functions** are blocks of code that perform a specific task and can be reused.

-> Use def to define a function.

-> Use return to send a result back to the caller (optional).

-> Functions help keep your code organized and reusable

Syntax:

def function\_name(parameters)

return result

Example:

def greet ():

print ("Hello, welcome to Python!")

greet ()

15) Function arguments (positional, keyword, default).