**Assignment 1**

**1.Explain the key features of Python that make it a popular choice for programming**

**Readable Syntax:** Python’s syntax is straightforward, like reading English, so it’s easy to understand and write.

**Versatile:** It’s used in almost everything—from web apps and data science to automation and AI.

**Huge Libraries:** Python has tons of built-in and third-party libraries, saving time on complex tasks.

**Cross-Platform:** Code written in Python works on any operating system without changes.

**Dynamic Typing:** No need to declare types; Python does it automatically, speeding up development.

**Interpreted:** Runs code line-by-line, making testing and debugging easier.

**Supports Multiple Styles:** Works well for different coding styles—object-oriented, functional, or procedural.

**Strong Community:** A huge support community means lots of resources, tools, and answers to questions.

**2.Describe the role of predefined keywords in Python and provide examples of how they are used in a program.**

**Define Structure**: Keywords shape the code’s structure, like setting up loops or conditions.

**Control Flow**: They manage how code runs, letting you execute parts based on conditions.

**Handle Errors**: Keywords like try and except help handle errors gracefully.

**Define Functions and Variables**: Keywords like def let you create functions, while class defines classes.

**Examples**

* **Control Flow :**if, else**:** Check conditions and choose between actions.

age=int(input())

if age< 18:

print("You are a minor")

else:

print("You are a major")

* **Looping :** for, in, range: Loop over a sequence of numbers.

continue: Skip the current iteration.

for i in range(1, 6):

if i == 3:

continue

print(i)

* **Functions:** def:Define a function.

return: Send back a result.

def greet(name):

return "Hello, " + name

print(greet("Ram"))

* **Exception Handling**:try, except: Handle errors gracefully**.**
* **Classes:** class:Define a blueprint for objects.

self: Refer to the instance of the class.

**3.Compare and contrast mutable and immutable objects in Python with examples.**

**Modification:**

* Mutable objects can be changed in place (e.g., adding an element to a list).
* Immutable objects require creating a new instance if we need any change.

**Memory Efficiency:**

* Mutable objects are more memory-efficient because changes happen in the same memory location.
* Immutable objects may be less efficient in terms of memory if modified frequently, as each change results in a new object.

**Usage in Data Structures:**

* Mutable like lists are useful for collections that need to change.
* Immutable like tuples are good when you need a fixed collection of items.

**Example**

**Mutability:**

**list\_cont = [1, 2, 3, 4.3, 3+5j, True, "Ajay"]**

**list\_cont[6] = "Bijay**

in this the item can be added to the list.

**Imutability:**

list\_cont[1] = "pwskills"

b[0] = "I"

this will throw an error because strings are imutable.

**4.Discuss the different types of operators in Python and provide examples of how they are used.**

* **Arithmetic Operators**

Used for mathematical calculations

* **+ (Addition)**: Adds two numbers.

Example:x = 5 + 3

* **- (Subtraction)**: Subtracts the second number from the first.

Example:x=5-2

* **\* (Multiplication)**: Multiplies two numbers.

z = 4 \* 3

* **/ (Division)**: Divides the first number by the second.

a = 16 / 4

* **% (Modulus)**: Returns the remainder after division.

b = 10 % 3

* **\*\* (Exponentiation)**: Raises the first number to the power of the second.

c = 2 \*\* 3

* **// (Floor Division)**: Divides and rounds down to the nearest integer.

d = 10 // 3 # 3

* **Comparison Operators:** The comparison operators are

== (equal to), != (not equal to), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to)

a = 10

b = 5

print(a == b) # False, checks if a is equal to b

print(a != b) # True, checks if a is not equal to b

print(a > b) # True, checks if a is greater than b

print(a < b) # False, checks if a is less than b

print(a >= b) # True, checks if a is greater than or equal to b

print(a <= b) # False, checks if a is less than or equal to b

* **Logical Operators:** The Logical operators are

and, or, not

Example:

x = True

y = False

print(x and y) # False, True and False is False

print(x or y) # True, True or False is True

print(not x) # False, not True is False

print(not y) # True, not False is True

* **Assignment Operators:** The assignment operators are

=,+=

Example:1

x = 5

Example:2

x += 3

* **Bitwise Operator:** The bitwise operators are

& (AND), | (OR), ^ (XOR), ~ (NOT), << (left shift), >> (right shift).

Example:

a = 5 # binary: 0101

b = 3 # binary: 0011

print(a & b) # 1 (AND: 0101 & 0011 = 0001)

print(a | b) # 7 (OR: 0101 | 0011 = 0111)

print(a ^ b) # 6 (XOR: 0101 ^ 0011 = 0110)

print(~a) # -6 (NOT: ~0101 = -0110 in two's complement)

print(a << 1) # 10 (left shift: 0101 << 1 = 1010)

print(a >> 1) # 2 (right shift: 0101 >> 1 = 0010)

* **Membership Operators:** Check if a value is in a sequence .

The membership operators are

In , not in

'a' in 'apple' # True

'b' not in 'apple' # True

**5.** **Explain the concept of type casting in Python with examples.**

Type casting is the process of converting one data type to another. This is useful when we want to perform operations that require specific types, such as adding numbers or working with strings. In Python, type casting is often done using built-in functions that convert values between common types.

**Implicit Type Casting (Automatic)**  
Python automatically converts one data type to another when it’s safe. This usually happens with compatible types like integers and floats.

x = 5 # int

y = 2.5 # float

result = x + y # Python converts `x` to float automatically

print(result) # 7.5 (float)

**Explicit Type Casting (Manual)**  
Here, you explicitly tell Python to convert a variable to a specific type using functions like int(), float(), str(), etc.

a = "10" # a is a string

b = int(a) # b is now an integer

print(b + 5) # 15

**6. How do conditional statements work in Python? Illustrate with examples**

Conditional statements allow you to execute specific code blocks based on certain conditions, making decisions within your program. The main conditional statements in Python are if, elif, and else.

**if statement**: Used to check a condition. If the condition is True, the indented block of code below it will run

Example: age = 18

if age >= 18:

print("You are an adult.") # Output: "You are an adult."

**else statement**: Runs if none of the previous if or elif conditions are True. It doesn’t require a condition.

**elif (else if) statement**: Used for additional conditions. If the previous if condition is False, Python checks the elif conditions next.

Example: age = 16

if age >= 18:

print("You are an adult.")

elif age >= 13:

print("You are a teenager.") # Output: "You are a teenager."

else:

print("You are a child.")

**7.Describe the different types of loops in Python and their use cases with examples.**

Loops allow us to execute a block of code repeatedly, either for a specific number of times or until a certain condition is met. Python has two main types of loops: for loops and while loops. Each has its own use cases and is suited to different situations.

**for Loop**

A for loop iterates over a sequence (like a list, tuple, string, or range) and executes a block of code for each item in the sequence. This type of loop is typically used when we know the number of iterations in advance or when we want to loop through each item in a collection.

Example:

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

for fruit in fruits:

print(fruit)

Output: apple, banana, cherry

**while Loop**

A while loop keeps running as long as a given condition is True. It’s commonly used when the number of iterations isn’t known in advance and depends on some condition that changes within the loop.

Examples

count = 5

while count > 0:

print(count)

count -= 1

Output: 5, 4, 3, 2, 1