NAME :- ABHISHEK BHIM YADAV
EMAIL :- abhishekbyadav2019@gmail.com
ASSSINGMENT LINK :- Python Assingment 1
DRIVE LINK :- Python Assingment 1
GITHUB LINK :- Python_Assignment

1. What is Python, and why is it popular ?

Ans:- **Python** is a high-level, interpreted programming language known for its **simplicity**, **readability**, and **versatility**. It was created by **Guido van Rossum** and first released in **1991**.

✅ **Why is Python Popular?**

1. **Easy to Learn and Use**
    - Python has a simple, English-like syntax which makes it beginner-friendly.
2. **Versatile Applications**

- Used in web development, data science, machine learning, automation, game development, and more.

3. **Large Standard Library**
    - Comes with built-in modules and functions for almost every task.
4. **Strong Community Support**
    - A vast, active community means plenty of tutorials, libraries, and frameworks are available (like Django, Flask, NumPy, Pandas, etc.).
5. **Cross-Platform Compatibility**
    - Works on Windows, macOS, Linux, etc., without changing the code.
6. **Great for Prototyping**
    - Quick development cycles make it ideal for testing and prototyping.
7. **Backed by Big Tech**
    - Used by companies like Google, Netflix, Facebook, and NASA.

2. What is an interpreter in Python?

Ans:-An **interpreter** in Python is a program that **executes Python code line by line**, converting it into **machine-readable instructions** at runtime.

How it works:

When you run a Python script:

1. The **interpreter reads** the code **one line at a time**.
2. It **converts** each line into **bytecode** (an intermediate form).
3. The **Python Virtual Machine (PVM)** then executes the bytecode.

🧠 **Key Points:**

- Python is an **interpreted language**, not compiled like C/C++.
- You don't need to compile Python code before running it.
- This makes Python **easy to debug** and **faster for development**.

3.What are pre-defined keywords in Python?

Pre-defined keywords in Python are **reserved words** that have **special meaning** to the Python interpreter. These words are **part of the language syntax**, and you **cannot use them as variable names, function names, or identifiers**.

🔑 **Examples of Python Keywords:**

Here are some commonly used keywords:

python

```
False      True       None       and        or
if         else       elif       while      for
def        return     import     from       as
class      try        except     finally    raise
```

```
with      assert    break     continue  pass
global    nonlocal  lambda    is        in
```

📌 **Key Points:**

- Python currently has **35+ keywords** (may vary by version).
- They are **case-sensitive** (e.g., `True` is valid, but `true` is not).
- You can list all keywords using this code:

python

```
import keyword
print(keyword.kwlist)
```

❗ **Example (Invalid Usage):**

python
CopyEdit

```
if = 5  # ❌ Error: 'if' is a reserved keyword
```

✅ **Summary:**

Pre-defined keywords in Python are **built-in reserved words** that define the **rules and structure** of Python code. They **cannot be redefined** or used for anything else

4. Can keywords be used as variable names?

❌ **No, keywords cannot be used as variable names in Python.**

🔍 Why?

Keywords are **reserved words** that Python uses to define its **syntax and structure**. Using them as variable names would **confuse the interpreter**, leading to **syntax errors**.

❗ Example (Incorrect):

python

```
def = 10  # ❌ Error: "def" is a keyword used to define functions
```

✅ Example (Correct):

python

CopyEdit

```
number = 10  # ✔️ "number" is a valid variable name
```

🧠 How to check Python keywords?

You can check all keywords with:

python

```
import keyword
```

```
print(keyword.kwlist)
```

✅ **Summary:**

No, Python **keywords cannot be used as variable names** because they are **reserved for special purposes** in the language. Always use unique, non-keyword names for variables.

5.What is mutability in Python?

Mutability in Python refers to whether an object's value can be changed after it is created.

🔄 **Two Types of Objects:**

1. ✅ **Mutable Objects – Can be changed after creation.**
2. ❌ **Immutable Objects – Cannot be changed after creation.**

🔧 **Mutable Examples:**

- `list`
- `dict`
- `set`

python

```
my_list = [1, 2, 3]

my_list[0] = 10  # ✅ Allowed
```

```
print(my_list)  # Output: [10, 2, 3]
```

🔒 **Immutable Examples:**

- `int`
- `float`
- `str`
- `tuple`

python

```
my_str = "Hello"

my_str[0] = "h"  # ❌ Error: 'str' object does not support item assignment
```

✅ **Summary:**

Mutability means the ability to change an object's value after creation.

- **Mutable: Can change (like `list`)**
- **Immutable: Cannot change (like `str`, `int`)**

Understanding mutability helps avoid bugs and write efficient Python code.

6.Why are lists mutable, but tuples are immutable?

The difference lies in how lists and tuples are designed in Python:

🔧 Lists are mutable because:

- They are meant to store collections of items that can change over time.
- You can add, remove, or modify elements in a list.

python

```
my_list = [1, 2, 3]
my_list[0] = 10     # ✅ Modification allowed
my_list.append(4)   # ✅ Can add elements
```

🔒 Tuples are immutable because:

- They are used to store fixed collections of items.
- Their immutability makes them faster, hashable, and safer for use as keys in dictionaries or constants.

python

```
my_tuple = (1, 2, 3)
my_tuple[0] = 10     # ❌ Error: Tuples do not support item assignment
```

⚙️ Internal Reason:

- Lists are built with methods to alter data (`append()`, `pop()`, `remove()`).
- Tuples do not include any methods for changing contents—intentionally, for data integrity and performance.

✅ Summary:

- Lists are mutable to allow dynamic data handling.
- Tuples are immutable for data safety, performance, and to enable use in hash-based structures like dictionaries and sets.
-

7. What is the difference between “==” and “is” operators in Python ?

Though both are used for **comparison**, they check **different things**:

✅ **== (Equality Operator):**

- Checks **if the values are equal**.
- Compares the **data/content** of two objects.

python

```
a = [1, 2, 3]
b = [1, 2, 3]
```

```
print(a == b)  # ✅ True – because the values inside are the same
```

🔍 **`is` (Identity Operator):**

- Checks **if two variables point to the same object** in memory.
- Compares **object identity**, not content.

python

```
a = [1, 2, 3]
b = [1, 2, 3]
print(a is b)  # ❌ False – because they are two different objects in memory
```

🧠 **Example with `is` and `==`:**

python

```
x = 1000
y = 1000
print(x == y)  # ✅ True – same value
print(x is y)  # ❌ False – different objects (for large integers)
```

```
x = y = 10

print(x is y)  # ✅ True – same object (small integers are cached)
```

✅ **Summary:**

Operator	Compares	Returns True If
`==`	**Values**	Values are **equal**
`is`	**Memory Address**	Variables point to **same object**

Use `==` for **value comparison**, and `is` for **identity checks** (especially for `None`, like `if x is None`).

8. What are logical operators in Python?

Logical operators in Python are used to **combine conditional statements** (boolean expressions). They return either **`True`** or **`False`** based on the logic.

🔑 **Python has three logical operators:**

Operator	Description	Example	Result
`and`	True if **both** conditions are True	`True and False`	`False`
`or`	True if **at least one** is True	`True or False`	`True`
`not`	**Inverts** the result	`not True`	`False`

✅ Examples:

python

```
a = 10
b = 5

print(a > 5 and b < 10)   # True – both are True
print(a < 5 or b < 10)    # True – one is True
print(not(a > 5))         # False – inverts True
```

🧠 Usage in Conditions:

python

```
if age > 18 and age < 60:
    print("Eligible")
```

✅ Summary:

Logical operators in Python – `and`, `or`, and `not` – are used to build **complex conditions** by combining **boolean expressions**.

9.What is type casting in Python?

Type casting in Python refers to the conversion of one data type to another manually, using built-in functions.

🔁 Why Use Type Casting?

- To perform operations between different data types.
- To format data (e.g., string to int for math operations).

🔧 Common Type Casting Functions:

Function	Converts to	Example

`int()`	Integer	`int("5") → 5`
`float()`	Floating point	`float("3.14") → 3.14`
`str()`	String	`str(100) → "100"`
`bool()`	Boolean	`bool(0) → False`
`list()`	List	`list("abc") → ['a', 'b', 'c']`

✅ Example:

python

```
x = "10"
y = int(x)  # Type casting from str to int
print(y + 5)  # Output: 15
```

❗ Note:

- Type casting can fail if the value is not compatible:

python

```
int("abc")  # ❌ Error: invalid literal for int()
```

✅ Summary:

Type casting is the process of converting one data type into another using functions like `int()`, `float()`, `str()`, etc. It's useful when you need to perform operations between different types of data.

10.What is the difference between implicit and explicit type casting ?

Type Casting means converting one data type to another.

---

1. Implicit Type Casting (Type Coercion)

- Done **automatically by Python** without user intervention.
- Happens when Python **converts data types to avoid errors** during operations.
- Usually converts a smaller data type to a larger or more general type.

Example:

python

CopyEdit

```
x = 5       # int
```

```
y = 3.2     # float
z = x + y   # int is implicitly converted to float
print(z)    # Output: 8.2 (float)
```

---

2. Explicit Type Casting (Type Conversion)

- Done **manually by the programmer** using functions like `int()`, `float()`, `str()`.
- You **explicitly specify** the target type.

Example:

python

```
x = "10"
y = int(x)  # explicitly convert string to int
print(y + 5)  # Output: 15
```

✅ Summary:

Feature	Implicit Type Casting	Explicit Type Casting
Performed by	Python interpreter automatically	Programmer manually

Requires function call	No	Yes (e.g., `int()`, `float()`)
Use case	Safe automatic conversions	When you want to force type change
Example	`5 + 3.2` → `8.2`	`int("10")` → `10`

11.What is the purpose of conditional statements in Python?

Purpose of Conditional Statements:

Conditional statements allow a program to **make decisions** and **execute different blocks of code based on certain conditions**.

Why are they important?

- To control the **flow of execution** depending on different inputs or situations.
- To **perform actions only when certain conditions are met**.
- To enable **dynamic behavior** in programs.

Common Conditional Statements in Python:

- `if`

- `if...else`
- `if...elif...else`

Example:

python

```
age = 18

if age >= 18:

    print("You are eligible to vote.")

else:

    print("You are not eligible to vote.")
```

✅ **Summary:**

Conditional statements in Python are used to **check conditions** and **control which code runs**, enabling programs to **respond differently** based on input or situations.

12.How does the elif statement work?

What is `elif`?

- `elif` stands for **“else if”**.
- It allows you to check **multiple conditions** sequentially.

- It comes **after an `if` statement** and before an optional `else`.

How it works:

1. Python evaluates the **`if` condition** first.
2. If the `if` condition is **False**, Python moves to the **`elif` condition**.
3. If the `elif` condition is **True**, that block executes, and the rest are skipped.
4. If none of the conditions are true, the **`else` block** runs (if provided).

Example:

python

```
num = 10

if num > 10:
    print("Greater than 10")
elif num == 10:
    print("Equal to 10")
else:
    print("Less than 10")
```

Output:

```
Equal to 10
```

✅ **Summary:**

- Use `elif` to **check multiple conditions in order**.
- Only **one block** among `if`, `elif`, and `else` executes.
- Helps write **clean and readable decision-making code**.

13.What is the difference between for and while loops?

Feature	`for` Loop	`while` Loop
Purpose	**Used to iterate over a sequence (like list, string, range)**	**Used to repeat a block while a condition is True**
Syntax	`for variable in sequence:`	`while condition:`
Number of Iterations	**Known or fixed (based on sequence length)**	**Not necessarily known, depends on condition**
Use Case	**When you know the number of iterations or want to iterate over items**	**When you want to repeat until a condition changes**

Example	**```python**	**```python**
	for i in range(5):	**i = 0**
	print(i)	**while i < 5:**
		print(i)
		i += 1
	```	**```**

Summary:

- **Use a `for` loop to iterate over items or a fixed range.**
- **Use a `while` loop to run as long as a condition remains True.**

14. Describe a scenario where a while loop is more suitable than a for loop.

Scenario:

When you don’t know beforehand how many times you need to repeat an action, and the loop should continue until a certain condition is met, a `while` loop is more suitable.

Example:

User input validation:

You want to keep asking the user to enter a positive number. You don't know how many attempts it will take, so you repeat until the user enters a valid number.

python

```
number = -1

while number <= 0:

    number = int(input("Enter a positive number: "))

print("Thank you! You entered:", number)
```

- Here, the loop runs until the condition (number > 0) is true.
- The number of iterations is not fixed — depends on user input.

Why not a `for` loop?

- A `for` loop works best when you know the number of iterations or have a sequence to loop over.
- Since the number of inputs is unknown, `while` is the right choice.

✅ Summary:

Use a `while` loop when the number of iterations depends on a dynamic condition, such as waiting for valid user input or an event to occur.

PRACTICAL QUESTIONS AND ANSWERS

1.Write a Python program to print "Hello, World!

Ans :- Print(“hello world”)

2.Write a Python program that displays your name and age?

```
name = "Abhishek Yadav"  # Replace with your name

age = 25                 # Replace with your age


print("Name:", name)

print("Age:", age)
```

3.Write code to print all the pre-defined keywords in Python using the keyword library

Ans:- import keyword

```
# Print the list of all Python keywords

print(keyword.kwlist)
```

4.Write a program that checks if a given word is a Python keyword.

```
import keyword


word = input("Enter a word: ")


if keyword.iskeyword(word):

    print(f'"{word}" is a Python keyword.')

else:

    print(f'"{word}" is NOT a Python keyword.')
```

5. Create a list and tuple in Python, and demonstrate how attempting to change an element works differently for each.

```
# Create a list (mutable)

my_list = [1, 2, 3]

print("Original list:", my_list)


# Change an element in the list

my_list[0] = 10

print("Modified list:", my_list)
```

```
# Create a tuple (immutable)
my_tuple = (1, 2, 3)
print("Original tuple:", my_tuple)

# Attempt to change an element in the tuple
try:
    my_tuple[0] = 10  # This will raise an error
except TypeError as e:
    print("Error while modifying tuple:", e)
```

Output:

```
python
Copy
Edit
Original list: [1, 2, 3]
Modified list: [10, 2, 3]
Original tuple: (1, 2, 3)
Error while modifying tuple: 'tuple' object does not support item assignment
```

6. Write a function to demonstrate the behavior of mutable and immutable arguments.

```
def modify_args(num, lst):
```

```
    # num is immutable (int)
    num += 10
    print("Inside function - num:", num)

    # lst is mutable (list)
    lst.append(100)
    print("Inside function - lst:", lst)

# Immutable argument example
x = 5
# Mutable argument example
my_list = [1, 2, 3]

print("Before function call - x:", x)
print("Before function call - my_list:", my_list)

modify_args(x, my_list)

print("After function call - x:", x)          # x remains unchanged
print("After function call - my_list:", my_list)  # my_list is modified
```

Explanation:

Immutable object (num = int): Changes inside the function do not affect the original variable outside.

Mutable object (lst = list): Changes inside the function modify the original list outside the function.

Output:

pgsql

Copy

Edit

Before function call - x: 5

Before function call - my_list: [1, 2, 3]

Inside function - num: 15

Inside function - lst: [1, 2, 3, 100]

After function call - x: 5

After function call - my_list: [1, 2, 3, 100]

7. Write a program that performs basic arithmetic operations on two user-input numbers.

```
# Take input from the user and convert to float

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))


# Perform arithmetic operations

addition = num1 + num2

subtraction = num1 - num2

multiplication = num1 * num2
```

```
# Check division by zero
if num2 != 0:
    division = num1 / num2
else:
    division = "Undefined (division by zero)"

# Display results
print(f"{num1} + {num2} = {addition}")
print(f"{num1} - {num2} = {subtraction}")
print(f"{num1} * {num2} = {multiplication}")
print(f"{num1} / {num2} = {division}")
```

8. Write a program to demonstrate the use of logical operators.

```
# Take two boolean inputs from the user
a = input("Enter True or False for a: ").strip().capitalize()
b = input("Enter True or False for b: ").strip().capitalize()

# Convert string input to boolean
a_bool = True if a == "True" else False
b_bool = True if b == "True" else False

print(f"a = {a_bool}, b = {b_bool}")
```

```
# Logical AND
print("a and b =", a_bool and b_bool)

# Logical OR
print("a or b =", a_bool or b_bool)

# Logical NOT
print("not a =", not a_bool)
print("not b =", not b_bool)
```

9. Write a Python program to convert user input from string to integer, float, and boolean types.

```
user_input = input("Enter something: ")

# Convert to integer
try:
    int_value = int(user_input)
except ValueError:
    int_value = "Conversion to int failed"

# Convert to float
try:
    float_value = float(user_input)
```

```
except ValueError:
    float_value = "Conversion to float failed"

# Convert to boolean
# For boolean, non-empty strings are True, empty string is False
bool_value = bool(user_input)

print(f"String input: {user_input}")
print(f"Integer conversion: {int_value}")
print(f"Float conversion: {float_value}")
print(f"Boolean conversion: {bool_value}")
```

10. Write code to demonstrate type casting with list elements.

```
# List of strings representing numbers
str_list = ["10", "20", "30", "40"]

# Convert each string element to integer using list comprehension
int_list = [int(item) for item in str_list]

print("Original list (strings):", str_list)
print("Converted list (integers):", int_list)

# Now convert integers back to strings
```

```
str_list_again = [str(num) for num in int_list]


print("Converted back to strings:", str_list_again)
```

11. Write a program that checks if a number is positive, negative, or zero.

```
num = float(input("Enter a number: "))


if num > 0:
    print("The number is positive.")
elif num < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

12. Write a for loop to print numbers from 1 to 10.

Here's a simple `for` loop in Python to print numbers from 1 to 10:

python

```
for i in range(1, 11):
    print(i)
```

13. Write a Python program to find the sum of all even numbers between 1 and 50.

```
total = 0

for num in range(2, 51, 2):  # Start at 2, step by 2 (even numbers)

    total += num


print("Sum of even numbers between 1 and 50 is:", total)
```

14. Write a program to reverse a string using a while loop.

Here's a Python program to reverse a string using a `while` loop:

python

```
input_str = input("Enter a string: ")

reversed_str = ""

index = len(input_str) - 1


while index >= 0:

    reversed_str += input_str[index]

    index -= 1


print("Reversed string:", reversed_str)
```

15. Write a Python program to calculate the factorial of a number provided by the user using a while loop.

```
num = int(input("Enter a non-negative integer: "))


if num < 0:
    print("Factorial is not defined for negative numbers.")
else:
    factorial = 1
    i = 1
    while i <= num:
        factorial *= i
        i += 1
    print(f"Factorial of {num} is {factorial}")
```