Programming in Python

Module-1

- Introduction to Python , features and applications.
- Datatypes , Keywords , Identifiers , Literals , Constants, Variables.
- Python indentation , Concept of indentation.
- Operators and Expressions , Naming conventions, Managing input & output.
- Conditional Statements , Looping statements , Nesting of loops.
- break , continue , pass & return statements.

Introduction to Python

- High-level, interpreted & general-purpose programming language.
- Created by Guido van Rossum in 1991 and further developed by the Python Software Foundation.
- It was designed with an emphasis on code readability.
- Helps the programmers to write the code in fewer lines.
- A programming language that lets you work quickly and integrate systems more efficiently.

Features of Python

- Easy to use and Read
- Dynamically Typed
- High-level
- Compiled and Interpreted Garbage Collected
- Purely Object-Oriented
- Cross-platform Compatibility
- Rich Standard Library

Applications of Python

- Data Science
- Desktop Applications
- Console-based Applications
- Mobile Applications ,Web Applications & Software Development
- Machine Learning
- Computer Vision or Image Processing Applications & Speech Recognition
- Testing
- Gaming

Variables

Variables are used to store data that can be referenced and manipulated during program execution. A variable is essentially a name that is assigned to a value. Unlike many other programming languages, Python variables do not require explicit declaration of type. The type of the variable is inferred based on the value assigned.

Rules for naming variables

- Variable names can only contain letters, digits and underscores (_).
- A variable name cannot start with a digit.
- Variable names are case-sensitive (myVar and myvar are different).
- Avoid using Python keywords (e.g., if, else, for) as variable names.

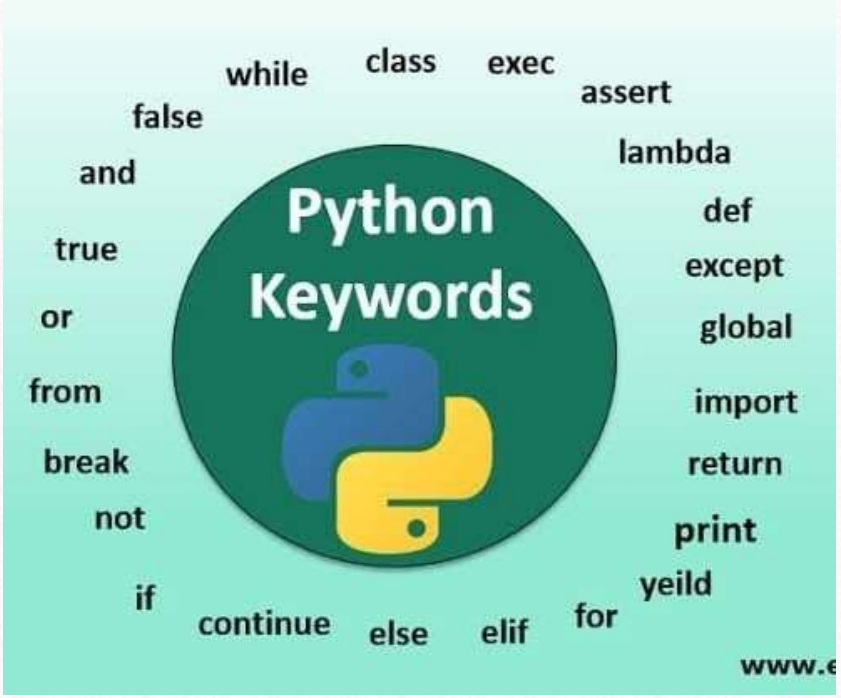
Module-1

Data types

Data types in Python define the type of data a variable can hold. Common types include **integers**, **floats**, **strings, sequence, mapping, set types, binary types , tuple, List and Boolean** . Each type has unique properties and behaviors, influencing how data is manipulated and stored in your programs.

Keywords & Identifiers

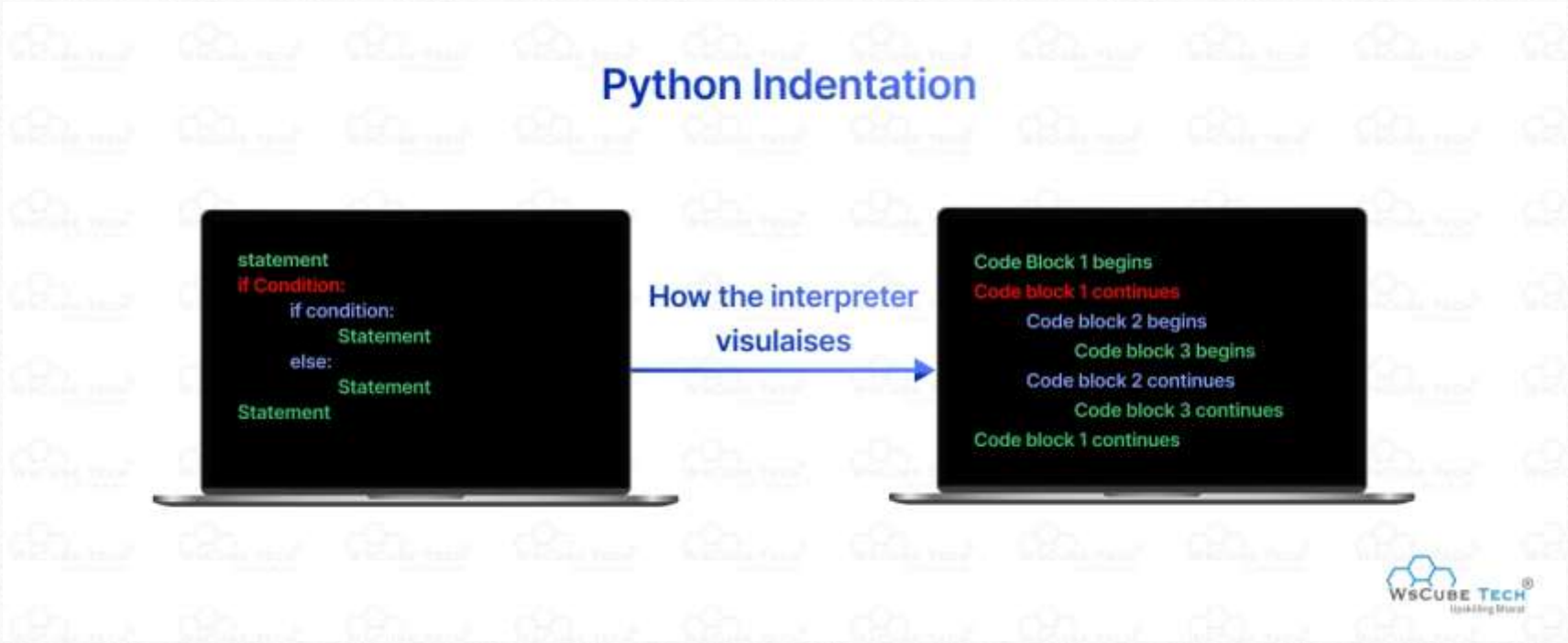
- In Python, **keywords** are reserved words that have specific meanings, such as **if**, **else**, and **def**. **Identifiers**, on the other hand, are names you choose for variables and functions.
- Understanding the distinction is vital for writing syntactically correct code.



Literals & Constants

- Literals are raw data values directly assigned to variables.
- They can be numeric (int, float, complex), string, Boolean (True, False), or special (None).
- Constants in Python are variables whose values remain unchanged throughout the program
- Example: PI = 3.14159

Python Indentation



Concept of Indentation

- Python Indentation is the whitespace (space or tab) before a block of code. It is the cornerstone of any Python syntax and is not just a convention but a requirement. In other words, indentation signifies that every statement with the same space to the right is a part of the same code block.
- Indentation is significant in Python as it ensures code readability. Unlike C, C++, and other languages, where curly braces represent a block of code, Python uses indentation level (number of leading whitespaces) to show a black with the same group of statements.

Module-1

Operators

Definition:Operators are symbols or keywords used to perform operations on values and variables in Python.

Types of Operators:

1. Arithmetic Operators: Perform mathematical operations.

+, -, *, /, %, **, //

2. Relational (Comparison) Operators: Compare two values

==, !=, >, <, >=, <=

3. Logical Operators: Combine conditional statements.

and, or, not

4. Assignment Operators: Assign values to variables.

=, +=, -=, *=, /=, %=

5. Bitwise Operators: Perform operations on binary numbers.

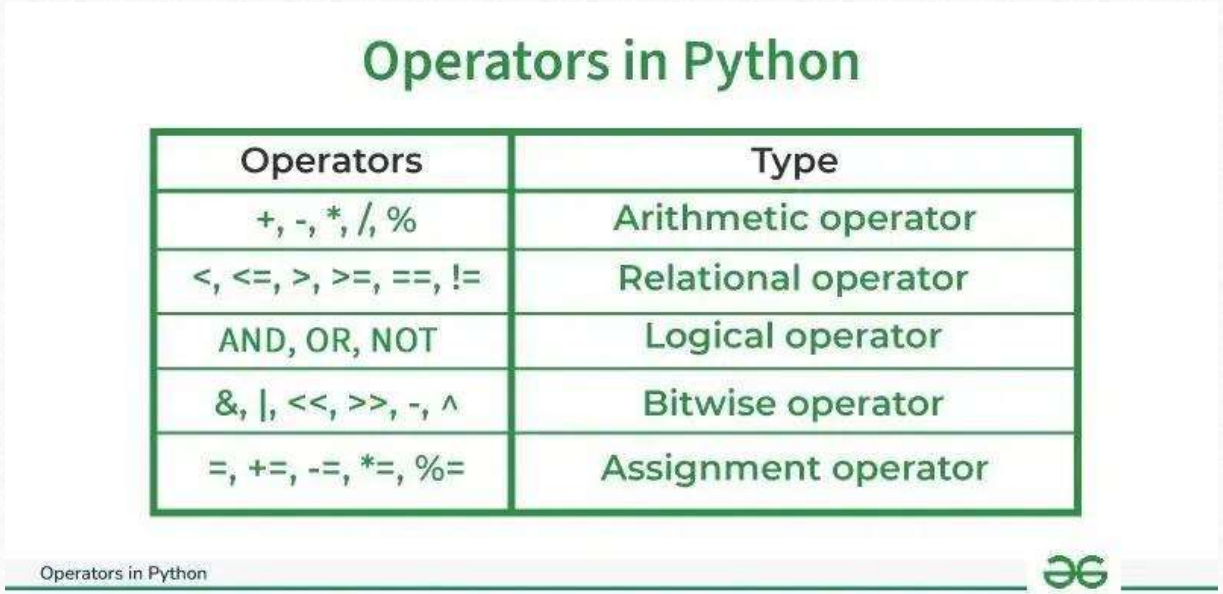
&, |, ^, ~, <<, >>

6. Membership Operators: Test if a value is in a sequence.

in, not in

7. Identity Operators: Compare objects by memory location.

is, is not



Naming Conventions

Best Practices:

1. Use meaningful names (age, total_price).
2. Use snake_case for variables and functions (my_variable).
3. Use PascalCase for classes (MyClass).
4. Constants in UPPERCASE (MAX_LIMIT).

Code Example: # Variable student_name = "Ray"

Function def calculate_area(radius):

return 3.14 * radius ** 2

Class class MyClass:

pass

Code Element	Naming Convention	Example
Package/Module Names	lowercase_with_underscores	`my_package`, `utils`
Class Names	CapitalizedWords	`MyClass`, `CustomerOrder`
Exception Names	CapitalizedWords	`CustomError`, `InvalidInputError`
Global Variables	lowercase_with_underscores	`global_variable`
Function Names	lowercase_with_underscores	`calculate_sum`, `process_data`
Method Names	lowercase_with_underscores	`get_name`, `set_value`
Function/Method Arguments	lowercase_with_underscores	`first_name`, `input_data`
Instance Variables	lowercase_with_underscores	`self.instance_variable`
Local Variables	lowercase_with_underscores	`local_variable`
Constants	UPPERCASE_WITH_UNDERSCORES	`PI`, `MAX_VALUE`
Type Variable Names	CapitalizedWords	`T`, `AnyStr`

Input & Output

Input in Python

- With the print() function, you can display output in various formats, while the input() function enables interaction with users by gathering input during program execution.

Print Output in Python

- At its core, printing output in Python is straightforward, thanks to the print() function. This function allows us to display text, variables, and expressions on the console. Let's begin with the basic usage of the print() function:
- In this example, "Hello, World!" is a string literal enclosed within double quotes. When executed, this statement will output the text to the console.

Ex-print("Hello, World!")

Output

Hello, World!

Conditional Statements

Conditional Statements are statements in Python that provide a choice for the control flow based on a condition.

1. If Conditional Statement in Python

If the simple code of block is to be performed if the condition holds then the if statement is used. Here the condition mentioned holds then the code of the block runs otherwise not.

Syntax

```
if condition:
    # statement
```

Example

```
if 10 > 5:
    print("10 greater than 5")
```

If Else Statement

2. If else Conditional Statements in Python

In a conditional if Statement the additional block of code is merged as an else statement which is performed when if condition is false.

Syntax

```
if (condition):
# Executes this block if
# condition is true
else:
# Executes this block if
# condition is false
```

Exa

```
x = 3
if x == 4:
    print("Yes")
else:
    print("No")
```

Nested If Else

- Nested if..else means an if-else statement inside another if statement.
- We can use one if or else if statement inside another if or else if statements.

```
# if..else chain statement
letter = "A"
if letter == "B":
    print("letter is B")
else:
    if letter == "C":
        print("letter is C")
    else:
        if letter == "A":
            print("letter is A")
        else:
            print("letter isn't A, B and C")
```

If-elif-else

The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final "else" statement will be executed.

```
letter = "A"

if letter == "B":
    print("letter is B")

elif letter == "C":
    print("letter is C")

elif letter == "A":
    print("letter is A")

else:
    print("letter isn't A, B or C")
```

Ternary Expression

The Python ternary Expression determines if a condition is true or false and then returns the appropriate value in accordance with the result.

Syntax: [on_true] if [expression] else [on_false]

expression: conditional_expression | lambda_expr

```
a, b = 10, 20
print("Both a and b are equal" if a == b else "a is greater than b"
      if a > b else "b is greater than a")
```

Module-1

Loops

- In programming, the loops are the constructs that repeatedly execute a piece of code based on the conditions.
- There are two types of loops in Python and these are for and while loops.
- Both of them work by following the below steps:
- 1. Check the condition
  2. If True, execute the body of the block under it. And update the

     iterator/the value on which the condition is checked.
  3. If False, come out of the loop

While Loop

- While loops execute a set of lines of code iteratively till a condition is satisfied. Once the condition results in False, it stops execution, and the part of the program after the loop starts executing.
- The syntax of the while loop is :

```
While condition:

    statement(s)
```

An example of printing numbers from 1 to 5 is shown below.

Example of Python while loop:

```
i=1

while (i<=5):

    print(i)

    i=i+1
```

For Loop

- For loop in Python works on a sequence of values. For each value in the sequence, it executes the loop till it reaches the end of the sequence.
- The syntax for the for loop is:

```
for iterator in sequence:
    statement(s)
```

Example of Python for loop:

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

Nested Loop

- A nested loop is a loop inside the body of the outer loop.
- The inner or outer loop can be any type, such as a while loop or for loop. For example, the outer for loop can contain a while loop and vice versa.
- The outer loop can contain more than one inner loop. There is no limitation on the chaining of loops.
- In the nested loop, the number of iterations will be equal to the number of iterations in the outer loop multiplied by the iterations in the inner loop.

```
Nested for loop
for i in range(1, 11):
    for j in range(1, 11):
        print(i * j, end=' ‘)
    print()
While inside for
names = ['Kelly', 'Jessa', 'Emma’]
for name in names:
    count = 0
    while count < 5:
        print(name, end=' ‘)
        count = count + 1
    print()
```

Break

The `break` statement will completely break out of the *current loop*,meaning it won't run any more of the statements contained
inside of it.

```
names = ["Rose", "Max", "Nina", "Phillip"]
for name in names:
    print(f"Hello, {name}")
    if name == "Nina":
        break
```

Continue

- continue works a little differently.
- Instead, it goes back to the start of the loop, skipping over any other statements contained within the loop.

```
for name in names:
    if name != "Nina":
        continue
    print(f"Hello, {name}")
```

Pass

- Do nothing. Ignore the condition in which it occurred and proceed to run the program as usual.
- We use pass statements to write empty loops. Pass is also used for empty control statements, functions, and classes.

```
# An empty loop

for letter in ‘Rose':

pass

print('Last Letter :', letter)
```

Return

- A **return statement** is used to end the execution of the function call and it “returns” the value of the expression following the return keyword to the caller.
- The statements after the return statements are not executed.

```
def add(a, b):


    # returning sum of a and b

    return a + b

def is_true(a):

    return bool(a)


# calling function

res = add(2, 3)

print(res)
```

Assignment 1

1: Print the first 10 natural numbers using for loop.
2: Python program to print all the even numbers within the given range.
3: Python program to calculate the sum of all numbers from 1 to a given number.
4: Python program to calculate the sum of all the odd numbers within the given range.
5: Python program to print a multiplication table of a given number
6: Python program to display numbers from a list using a for loop.
7: Python program to count the total number of digits in a number.
8: Python program to check if the given string is a palindrome.
9: Python program that accepts a word from the user and reverses it.
10: Python program to check if a given number is an Armstrong number
11: Python program to count the number of even and odd numbers from a series of numbers.
12: Python program to display all numbers within a range except the prime numbers.
13: Python program to get the Fibonacci series between 0 to 50.
14: Python program to find the factorial of a given number.
15: Python program that accepts a string and calculates the number of digits and letters.
16: Write a Python program that iterates the integers from 1 to 25.
17: Python program to check the validity of password input by users.
18: Python program to convert the month name to a number of days.