**PYTHON PROGRAMMING**

**UNIT-2**

**DATA TYPES, EXPRESSIONS, STATEMENTS**

### **Basic Syntax in Python**

**Basic syntax** refers to the set of rules that define the structure of Python programs. It includes how to write the code correctly so that Python can interpret and run it.

#### **Key Concepts:**

* **Indentation:** Python uses **indentation** (spaces or tabs) to define the structure of code blocks instead of curly braces {} used in languages like C, C++, or Java. Indentation is crucial, and all code inside a function or loop must be indented the same amount. If you mix spaces and tabs or misalign indentation, Python will raise an **IndentationError**.

def greet():

print("Hello, World!") # Correct indentation



Incorrect indentation will result in an error:

def greet():

print("Hello, World!") # IndentationError: expected an indented block



* **Statements:** A **statement** is a single line of code that performs a specific task, such as an assignment or print operation. In Python, statements end at the end of the line, and semicolons are optional.

x = 5 # This is a statement



* **Comments:** Comments are non-executable lines used for explanation and clarification. In Python, **single-line comments** begin with a #, while **multi-line comments** use triple quotes (''' or """).
  + Single-line comment:

# This is a comment



* + Multi-line comment:

'''

This is a

multi-line comment

'''



* **Case Sensitivity:** Python is case-sensitive, which means Variable, variable, and VARIABLE are all considered different identifiers.

variable = 10

Variable = 20

print(variable) # Prints 10

print(Variable) # Prints 20

#### **Example of Basic Syntax:**

# This is a simple program that prints a greeting

name = "Alice" # Variable assignment

print("Hello, " + name) # Output: Hello, Alice



In this program:

* name = "Alice" is a variable assignment.
* print("Hello, " + name) prints the greeting.
* Indentation, comments, and line breaks follow Python's syntax rules.

### **2. Python Identifiers**

**Identifiers** are names used to identify variables, functions, classes, modules, and other objects in Python. These names are used to refer to data stored in memory. It's essential to follow the **naming conventions** and **rules** when creating identifiers.

#### **Rules for Python Identifiers:**

* **Start with a letter or an underscore (\_):**  
  An identifier must begin with a letter (a-z, A-Z) or an underscore. It cannot start with a digit.

my\_variable = 10 # Valid

\_myVar = 20 # Valid

2variable = 5 # Invalid (starts with a number)



* **Can contain letters, digits, and underscores:**  
  After the first character, the identifier can include letters, digits (0-9), and underscores.

var\_1 = 100 # Valid

score\_2 = 50 # Valid

myName2 = "John" # Valid



* **No special characters:**  
  Identifiers cannot contain special characters like @, #, &, etc.

score@2 = 5 # Invalid

my-name = "Python" # Invalid



* **Case Sensitive:**  
  Identifiers are case-sensitive, so myvariable and MyVariable are two different identifiers.

name = "Alice"

Name = "Bob"

print(name) # Prints Alice

print(Name) # Prints Bob



* **Cannot be a Python reserved keyword:**  
  Keywords are predefined words in Python that have specific meanings, such as if, else, while, etc. These cannot be used as identifiers.

def = 10 # Invalid, "def" is a keyword

#### **Examples of Valid Identifiers:**

* my\_var
* \_age
* user\_name
* tempValue

#### **Examples of Invalid Identifiers:**

* 2variable (starts with a number)
* my-name (contains a hyphen)
* @user (contains special character @)
* False (Python keyword)

#### **How Python Identifiers Are Used:**

Identifiers are used to store data and refer to it later. For example:

# Assigning values to identifiers

age = 25

name = "Alice"

height = 5.5

# Using the identifiers

print(name) # Prints Alice

print(age) # Prints 25



### **3. Variable Assignment in Python**

**Variable assignment** in Python involves creating variables and assigning values to them. A variable is a name used to store data that can be referenced and manipulated later.

#### **Key Concepts:**

* **Dynamic Typing:**  
  Python is a **dynamically typed** language, meaning you don't have to declare the data type of a variable when you assign a value. The interpreter determines the type at runtime.

x = 5 # Integer

x = "Hello" # String (x's type is now a string)



* **Assigning Values:** To assign a value to a variable, you use the = operator.

age = 25 # Assigning the integer 25 to the variable age

name = "Alice" # Assigning a string value to name



* **Multiple Assignments:**  
  You can assign values to multiple variables in one line by separating them with commas.

a, b, c = 5, 10, 15



This assigns:

* + a = 5
  + b = 10
  + c = 15
* **Swapping Variables:** Python allows you to swap values between variables in a single line using tuple unpacking.

x, y = 10, 20

x, y = y, x # Swapping values



* **Variable Types:**  
  Python automatically detects the type of data assigned to a variable. Here are a few common data types:
  + Integer (int)
  + Floating point number (float)
  + String (str)
  + List (list)
  + Dictionary (dict)
  + Boolean (bool)

integer\_variable = 10 # Integer

float\_variable = 3.14 # Float

string\_variable = "Hello" # String

#### **Examples of Variable Assignment:**

# Assigning values to variables

x = 10 # Integer

y = 20.5 # Float

name = "Bob" # String

# Performing an operation

z = x + y # Adding an integer and a float

print(z) # Prints 30.5

# Reassigning a variable

x = "New value"

print(x) # Prints New value

#### **Important Notes About Variables in Python:**

* **No Need for Declaration:** In Python, you don’t need to declare the type of the variable beforehand. You just assign a value, and Python determines its type.

x = 10 # Python determines x is an integer

x = "hello" # Python reassigns x as a string



* **Immutable vs Mutable Variables:**  
  Some data types in Python (like strings, integers, and tuples) are **immutable**, meaning their values cannot be changed after assignment. Others (like lists and dictionaries) are **mutable** and can be modified.

x = 10 # Immutable

x = x + 5 # Creates a new value, x is reassigned to 15

my\_list = [1, 2, 3] # Mutable

my\_list.append(4) # List is modified directly



### **Summary**

1. **Basic Syntax**:
   * Python is indentation-sensitive.
   * Use # for comments, and Python doesn't require semicolons to end statements.
   * Case sensitivity and simple rules for structuring code are crucial.
2. **Python Identifiers**:
   * Must start with a letter or an underscore.
   * Can contain letters, digits, and underscores.
   * Cannot start with a digit or use special characters.
   * Must not be Python reserved keywords.
3. **Variable Assignment**:
   * Python uses the = operator to assign values to variables.
   * Variables are dynamically typed, meaning Python automatically assigns types based on the value.
   * Python allows reassigning values and swapping variables easily.

**Write a Python program using variables and identifiers**:

* Create variables for name, age, and city, and print them:

### **Python Program**

****# Creating variables to store the information

name = "Alice" # Assigning the string value "Alice" to the variable 'name'

age = 25 # Assigning the integer value 25 to the variable 'age'

city = "New York" # Assigning the string value "New York" to the variable 'city'

# Printing the values stored in the variables

print("Name:", name) # This prints "Name: Alice"

print("Age:", age) # This prints "Age: 25"

print("City:", city) # This prints "City: New York"

### **Explanation:**

1. **Variable Assignment:**
   * name = "Alice": Here, the variable name is assigned the value "Alice", a string.
   * age = 25: The variable age is assigned the integer value 25.
   * city = "New York": The variable city is assigned the value "New York", a string.
2. **Printing Variables:**
   * print("Name:", name) prints the string "Name:" followed by the value of the name variable ("Alice").
   * print("Age:", age) prints the string "Age:" followed by the value of the age variable (25).
   * print("City:", city) prints the string "City:" followed by the value of the city variable ("New York").

### **Output:**

When the program is executed, the output will be:

Name: Alice

Age: 25

City: New York



### Python Program with Arithmetic Operation

# Assigning values to variables

num1 = 5 # Assigning the integer 5 to variable 'num1'

num2 = 10 # Assigning the integer 10 to variable 'num2'

# Performing addition operation

sum = num1 + num2 # Adding num1 and num2 and storing the result in 'sum'

# Printing the result of the addition

print("The sum is:", sum) # This will print: The sum is: 15

### Explanation:

1. **Variable Assignment:**
   * num1 = 5: The variable num1 is assigned the value 5.
   * num2 = 10: The variable num2 is assigned the value 10.
2. **Arithmetic Operation:**
   * sum = num1 + num2: This line performs the addition operation (5 + 10) and stores the result (15) in the variable sum.
3. **Printing the Result:**
   * print("The sum is:", sum): This line prints the string "The sum is:" followed by the value stored in the sum variable (which is 15).

### Output:

When the program runs, the output will be:

The sum is: 15



### **Multiple Choice Questions (MCQs):**

1. **Which of the following is a valid variable name in Python?**
   * A) 2variable
   * B) variable\_2
   * C) @variable
   * D) variable#2
   * **Answer: B) variable\_2**
2. **In Python, which of the following can be used to assign a value to a variable?**
   * A) ==
   * B) =
   * C) :=
   * D) ===
   * **Answer: B) =**
3. **What is the rule for Python identifiers?**
   * A) Identifiers must begin with a digit
   * B) Identifiers can contain spaces
   * C) Identifiers must begin with a letter or an underscore
   * D) Identifiers can contain special characters like @ or #
   * **Answer: C) Identifiers must begin with a letter or an underscore**
4. **Which of the following is the correct way to assign a value to a variable in Python?**
   * A) x <- 10
   * B) 10 = x
   * C) x = 10
   * D) x == 10
   * **Answer: C) x = 10**
5. **Which of the following is not a valid Python identifier?**
   * A) my\_variable
   * B) \_variable
   * C) variable2
   * D) 2variable
   * **Answer: D) 2variable**

### **Key Discussion Points:**

1. **What is the basic syntax of Python?**
   * Python's syntax relies heavily on indentation to define code blocks. Statements are written without semicolons or parentheses, making the code cleaner.
2. **What are Python identifiers, and what are the rules for naming them?**
   * Identifiers are names for variables, functions, and classes. They must begin with a letter or an underscore and can contain letters, numbers, and underscores.
3. **How do you assign values to variables in Python?**
   * You assign a value using the = operator, e.g., x = 10.
4. **What is the significance of indentation in Python syntax?**
   * Indentation determines the grouping of statements. It is used instead of braces {} in other languages to define blocks of code.
5. **How do you create and use comments in Python code?**
   * Single-line comments use the # symbol. Multi-line comments are enclosed in triple quotes ('''comment''').
6. **What are the different data types available in Python?**
   * Python includes common data types like integers, floats, strings, lists, tuples, sets, and dictionaries.

**Input/Output Statements**

### **Introduction to Input/Output in Python**

In Python, **Input/Output (I/O)** refers to how we interact with the user and display information on the screen or get input from the user. There are two main functions to handle input and output in Python:

1. **Input Function**: This function allows the program to get data from the user.
2. **Output Function**: This function is used to display information on the screen.

Let's break down the different components mentioned.

### **1. Input Function**

In Python, the input() function is used to take input from the user.

#### Syntax:

input(prompt)



* The prompt is an optional argument. It’s a message that can be displayed to the user before asking for the input.

#### Example:

name = input("Enter your name: ")

print("Hello, " + name + "!")



**Explanation**:

* input("Enter your name: ") asks the user to enter their name.
* The entered value is stored in the variable name.
* The program then prints the greeting message using that input.

**Sample Output:**

Enter your name: John

Hello, John!



### **2. Output Function**

The print() function is used to display output in Python.

#### Syntax:

print(value1, value2, ..., sep=' ', end='\n')



* The sep parameter allows you to specify a separator between the printed values (default is space).
* The end parameter specifies what to print at the end (default is a new line).

#### Example:

print("Hello, World!")



**Explanation**:

* The print() function simply outputs the string "Hello, World!" to the screen.

**Sample Output:**

Hello, World!



### **3. Formatted Output**

You can format the output in Python using placeholders or f-strings.

#### Using .format() Method:

name = "Alice"

age = 30

print("My name is {} and I am {} years old.".format(name, age))



* The {} placeholders are replaced by the values provided in the format() method.

**Sample Output:**

My name is Alice and I am 30 years old.

#### Using f-strings (Python 3.6+):

name = "Bob"

age = 25

print(f"My name is {name} and I am {age} years old.")



* F-strings are a more modern and readable way to format strings in Python. The variable names are embedded directly into the string using curly braces {}.

**Sample Output:**

My name is Bob and I am 25 years old.



### **4. Multi-line Output**

You can print multi-line output using:

* \n to add a new line.
* Triple quotes (''' or """) for multi-line strings.

#### Using \n:

print("Line 1\nLine 2\nLine 3")



**Sample Output:**

Line 1

Line 2

Line 3

#### Using Triple Quotes:

print("""Line 1

Line 2

Line 3""")



**Sample Output:**

Line 1

Line 2

Line 3



### **5. Combining Input and Output**

You can combine both input and output in one program to make it more interactive.

#### Example:

name = input("Enter your name: ")

age = int(input("Enter your age: "))

print(f"Hello, {name}! You are {age} years old.")



**Explanation**:

* The program first asks for the name and age using input().
* It then uses print() to display a message with the provided input.

**Sample Output:**

Enter your name: John

Enter your age: 25

Hello, John! You are 25 years old.



### **Hands-on Activities**

Here are some practical coding activities:

1. **Program to take the name as input and print a greeting message:**

name = input("Enter your name: ")

print(f"Hello, {name}!")



**Output**:

Enter your name: Alice

Hello, Alice!



1. **Program to take the age as input, convert it to an integer, and display the result:**

age = int(input("Enter your age: "))

print(f"You are {age} years old.")



**Output**:

Enter your age: 30

You are 30 years old.



1. **Program to ask for the name and age, and print a formatted message using print():**

name = input("Enter your name: ")

age = int(input("Enter your age: "))

print("Hello, {}! You are {} years old.".format(name, age))



**Output**:

Enter your name: Bob

Enter your age: 22

Hello, Bob! You are 22 years old.



1. **Program to format the output using .format() or f-strings:**

name = "Eve"

age = 28

print(f"My name is {name} and I am {age} years old.")



**Output**:

My name is Eve and I am 28 years old.



1. **Program to display output on multiple lines using \n or triple quotes:**

address = """123 Main St

Springfield, IL 62701"""

print(address)



**Output**:

123 Main St

Springfield, IL 62701



### **Practice Problems**

1. **Write a program to input the user's birth year, calculate their age, and display it:**

birth\_year = int(input("Enter your birth year: "))

current\_year = 2025

age = current\_year - birth\_year

print(f"You are {age} years old.")



1. **Ask the user to input two decimal numbers and print their product:**

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

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

print(f"The product of {num1} and {num2} is {num1 \* num2}.")



1. **Print your full name, separated by underscores:**

first\_name = "John"

last\_name = "Doe"

print(f"{first\_name}\_{last\_name}")



1. **Write a program that prints three numbers (10, 20, 30) on the same line separated by commas:**

print(10, 20, 30, sep=", ")



1. **Write a program that takes the user's name and age, and displays: "My name is [name], and I am [age] years old.":**

name = input("Enter your name: ")

age = int(input("Enter your age: "))

print(f"My name is {name}, and I am {age} years old.")



1. **Format the output of a program that calculates the square of a number:**

num = int(input("Enter a number: "))

print(f"The square of {num} is {num \*\* 2}.")



1. **Write a program to display a multi-line address (e.g., street, city, state):**

address = """123 Elm St

Sunnydale, CA 12345"""

print(address)



1. **Display a motivational quote across three lines:**

quote = """Believe in yourself.

Keep pushing forward.

Success will follow."""

print(quote)



### **Multiple Choice Questions (MCQ)**

1. **Which function is used to take user input in Python?**
   * a. input()
   * b. print()
   * c. get()
   * d. scan()
   * **Answer**: a. input()
2. **Which function is used to display output in Python?**
   * a. input()
   * b. display()
   * c. print()
   * d. show()
   * **Answer**: c. print()
3. **How can you format a string in Python to include a variable?**
   * a. By using + operator
   * b. Using f-strings or .format()
   * c. By using print()
   * d. By concatenating with input()
   * **Answer**: b. Using f-strings or .format()
4. **How would you print multiple lines without using multiple print statements?**
   * a. Use the sep parameter in print()
   * b. Use \n or triple quotes
   * c. Use a loop
   * d. Use a list
   * **Answer**: b. Use \n or triple quotes
5. **Which of the following will take input and then print a message?**
   * a. print(input("Enter your name: "))
   * b. input("Enter your name: ")
   * c. input() + print()
   * d. print(input())
   * **Answer**: d. print(input())

**Understanding Variables in Python**

### **Understanding Variables in Python**

A **variable** in Python is like a container that holds data. Variables store information and allow you to reuse that data throughout your program.

### **1. Definition of Variables in Python**

A **variable** in Python is a symbolic name that refers to a value stored in the memory. Python uses variables to store data such as numbers, text, or even more complex objects.

### **2. Syntax of Variable Assignment**

To assign a value to a variable in Python, use the **assignment operator (=)**.

#### **Syntax:**

****variable\_name = value

* variable\_name is the name you choose for the variable.
* value is the data you want to store in the variable.

#### **Example:**

****name = "John"

age = 25

In this case:

* The variable name is assigned the string "John".
* The variable age is assigned the integer 25.

### **3. Rules for Naming Variables**

In Python, there are certain rules for naming variables:

1. **Variable names must start with a letter (a-z, A-Z) or an underscore (\_)**.
2. **Variable names can include letters, numbers (0-9), and underscores**.
3. **Variable names cannot start with a number**.
4. **Variable names are case-sensitive**, meaning age, Age, and AGE are all different variables.
5. **Variable names cannot be the same as Python reserved keywords** (like if, while, for, class, etc.).

#### **Valid variable names:**

****name, age, \_value, firstName

#### **Invalid variable names:**

****2name, my-name, class



### **4. Types of Variables**

Python has several **built-in data types** that variables can hold:

1. **Integer (int)**: Whole numbers without decimals.

age = 25

1.   
   **Floating-point (float)**: Numbers with decimals.

height = 5.8

1.   
   **String (str)**: Text or sequences of characters.

name = "Alice"

1.   
   **Boolean (bool)**: Represents True or False.

is\_student = True



### **5. Variable Assignment and Re-assignment**

Variables in Python can be reassigned. If you assign a new value to an existing variable, it overwrites the old value.

#### **Example of Re-assignment:**

****x = 10

print(x) # Output: 10

x = 20

print(x) # Output: 20

Here, the variable x initially holds 10, but after reassignment, it holds 20.

### **6. Multiple Assignment in One Line**

Python allows you to assign values to multiple variables in a single line.

#### **Syntax:**

****var1, var2, var3 = value1, value2, value3

#### **Example:**

****x, y, z = 10, 20, 30

print(x, y, z)

**Output**:

10 20 30



### **7. Variable Scope (Global vs Local Variables)**

* **Global Variables**: These are variables declared outside of a function and can be accessed anywhere in the program.
* **Local Variables**: These are variables declared inside a function and can only be accessed within that function.

#### **Example of Global Variable:**

****x = 10 # Global variable

def display():

print(x) # Accessing global variable inside a function

display()

**Output**:

10

#### **Example of Local Variable:**

****def display():

x = 20 # Local variable

print(x)

display()

**Output**:

20

Trying to access x outside the function would result in an error because it's a local variable.

### **Live Code Examples for Explanation**

Let's write some live code to demonstrate these concepts:

1. **Program to declare variables of different types (string, integer, float, boolean), assign values to them, and print their values.**

****name = "John"

age = 30

height = 5.9

is\_student = True

print(name)

print(age)

print(height)

print(is\_student)

**Output**:

John

30

5.9

True

1. **Program to assign values to multiple variables in one line and print the values of these variables.**

****x, y, z = 10, 20, 30

print(x, y, z)

**Output**:

10 20 30

1. **Program that assigns a value to a variable, prints it, then reassigns a new value to the same variable, and prints the new value.**

****value = 5

print(value)

value = 10

print(value)

**Output**:

5

10

1. **Program that demonstrates how the type of a variable can change during execution.**

****var = 5 # Integer

print(var, type(var))

var = "Hello" # String

print(var, type(var))

var = True # Boolean

print(var, type(var))

**Output**:

5 <class 'int'>

Hello <class 'str'>

True <class 'bool'>



### **Practice Programs for Students**

1. **Define variables to store your name, age, GPA, and whether you are a student, then print them.**

****name = "Alice"

age = 22

GPA = 3.8

is\_student = True

print(name, age, GPA, is\_student)

1. **Create variables to store the details of your favorite book: title, author, number of pages, and if it is currently available in the library.**

****book\_title = "To Kill a Mockingbird"

author = "Harper Lee"

pages = 281

available\_in\_library = True

print(book\_title, author, pages, available\_in\_library)

1. **Assign values to three variables and print their sum.**

****x = 10

y = 20

z = 30

sum\_value = x + y + z

print(sum\_value)

1. **Create two variables to store the lengths of two sides of a rectangle. Calculate and display the perimeter and area.**

****length = 5

width = 3

perimeter = 2 \* (length + width)

area = length \* width

print("Perimeter:", perimeter)

print("Area:", area)

1. **Create variables with valid names to store the details of your car (make, model, year). Print them.**

****car\_make = "Toyota"

car\_model = "Corolla"

car\_year = 2022

print(car\_make, car\_model, car\_year)

1. **Experiment with invalid variable names (e.g., 2name, my-name) and observe the errors.**
   * Try running invalid variable names in Python and observe the error messages.
2. **Declare variables of type integer, float, string, and boolean, then display their values and types using the type() function.**

****integer\_var = 10

float\_var = 3.14

string\_var = "Hello"

boolean\_var = True

print(type(integer\_var), type(float\_var), type(string\_var), type(boolean\_var))

1. **Write a program to track the current balance of a wallet. Update it after spending and adding money, displaying the balance at each step.**

****balance = 100 # Starting balance

# Spend some money

balance -= 30

print("Balance after spending $30:", balance)

# Add some money

balance += 50

print("Balance after adding $50:", balance)



### **Practice Questions (MCQs)**

1. **What happens if you try to use a variable that has not been defined in Python?**
   * a. It throws a SyntaxError
   * b. It throws a NameError
   * c. It automatically defines the variable
   * d. It returns None
   * **Answer**: b. It throws a NameError
2. **Which of the following is a valid Python variable?**
   * a. 2name
   * b. my-name
   * c. \_myname
   * d. for
   * **Answer**: c. \_myname
3. **What happens when you re-assign a variable in Python?**
   * a. The previous value is overwritten
   * b. A new variable is created
   * c. It throws an error
   * d. It keeps both old and new values
   * **Answer**: a. The previous value is overwritten
4. **What is the scope of a variable assigned inside a function?**
   * a. Global
   * b. Local to the function
   * c. Global to all functions
   * d. Only accessible inside classes
   * **Answer**: b. Local to the function
5. **Which statement correctly assigns multiple variables in one line?**
   * a. x = y = z = 10
   * b. x = 10, y = 20, z = 30
   * c. x = (y = 10)
   * d. 10 = x = y = z
   * **Answer**: a. x = y = z = 10

**Regular Expressions**

### Introduction to Regular Expressions

**Regular Expressions (RegEx)** are a powerful tool for pattern matching and text processing. They allow you to search, match, and manipulate strings based on specific patterns.

#### Python Regular Expressions

Python provides the re module to work with Regular Expressions. To use it, you need to import it:

import re

#### Example of RegEx in Python

Let's say you want to search for a pattern in a string that starts with "The" and ends with "Spain":

import re

txt = "TheraininSpain"

x = re.search("^The.\*Spain$", txt)



* The ^ asserts that the match starts at the beginning.
* .\* matches any characters in between.
* $ asserts that the match ends at the string's end.

### Regular Expression Functions

The re module provides several functions for working with regular expressions.

| **Function** | **Description** |
| --- | --- |
| findall() | Returns a list containing all matches. |
| search() | Returns a match object if a match is found. |
| split() | Splits the string where a match occurs. |
| sub() | Replaces the matches with a string. |

### Special Characters and Symbols

#### Metacharacters

Metacharacters have special meanings in regular expressions:

| **Character** | **Description** | **Example** |
| --- | --- | --- |
| [] | A set of characters | [a-m] |
| \ | Signals a special sequence | \d |
| . | Matches any character (except newline) | he..o |
| ^ | Matches the start of a string | ^hello |
| $ | Matches the end of a string | world$ |
| \* | Zero or more occurrences | `a |
| + | One or more occurrences | `a |
| {} | Matches a specific number of occurrences | `a |
| ` | ` | Matches either |

#### Special Sequences

These sequences start with a backslash and have special meanings:

| **Character** | **Description** | **Example** |
| --- | --- | --- |
| \A | Matches the beginning of the string | \AThe |
| \b | Matches the boundary of a word | r"\bain" |
| \B | Matches non-boundary of a word | r"\Bain" |
| \d | Matches any digit | "\d" |
| \D | Matches non-digit characters | "\D" |
| \s | Matches a whitespace character | "\s" |
| \S | Matches non-whitespace characters | "\S" |
| \w | Matches a word character (letters, digits, underscore) | "\w" |
| \W | Matches non-word characters | "\W" |
| \Z | Matches the end of the string | "Spain\Z" |

### Matching Patterns

#### Matching Multiple Patterns with Alternation (|)

You can use | to match either of several patterns:

| **RE Pattern** | **Strings Matched** |
| --- | --- |
| `at | home` |
| `r2d2 | c3po` |
| `bat | bet |

#### Matching Any Single Character (.)

The . character matches any character except a newline:

| **RE Pattern** | **Strings Matched** |
| --- | --- |
| f.o | Matches "fao", "f9o", "f#o" |
| .. | Matches any pair of characters |
| .end | Matches any character before the word "end" |

#### Anchors: Beginning (^) and End ($)

* ^ asserts the start of the string.
* $ asserts the end of the string.

| **RE Pattern** | **Strings Matched** |
| --- | --- |
| ^From | Matches any string starting with "From" |
| a$ | Matches any string ending with "a" |
| ^S : hi$ | Matches a string that is exactly "S: hi" |

#### Word Boundaries (\b and \B)

* \b matches the boundary of a word.
* \B matches positions where there isn't a word boundary.

| **RE Pattern** | **Strings Matched** |
| --- | --- |
| \bthe | Matches words starting with "the" |
| \bthe\b | Matches the exact word "the" |
| \Bthe | Matches strings containing "the" but not starting with it |

### Creating Character Classes ([ ])

Character classes define a set of characters to match:

| **RE Pattern** | **Strings Matched** |
| --- | --- |
| b[aeiu]t | Matches "bat", "bet", "bit", "but" |
| [cr][23][dp][o2] | Matches combinations like "c2do", "r3p2", "r2d2", etc. |

#### Ranges and Negation (-, ^)

* A hyphen between characters defines a range.
* A caret (^) at the beginning negates the match.

| **RE Pattern** | **Strings Matched** |
| --- | --- |
| [a-z] | Matches any lowercase letter |
| [^aeiou] | Matches any non-vowel |
| [0-9] | Matches any digit |

### Repetition Using Quantifiers (\*, +, ?, { })

These operators define how many times a pattern should be matched:

* \* matches 0 or more occurrences.
* + matches 1 or more occurrences.
* ? matches 0 or 1 occurrence.
* {n} matches exactly n occurrences.
* {m,n} matches between m and n occurrences.

### Core Functions in the re Module

#### 1. match()

The match() function checks if the pattern matches at the beginning of the string.

Example:

import re

m = re.match('foo', 'foo')

if m != None:

print(m.group()) # Output: foo

#### 2. search()

The search() function scans the string for a match.

Example:

import re

txt = "The rain in Spain"

x = re.search("\s", txt)

print("First white-space character is at position:", x.start()) # Output: 3

#### 3. findall()

The findall() function returns all matches in a list.

Example:

import re

txt = "The rain in Spain"

x = re.findall("ai", txt)

print(x) # Output: ['ai', 'ai']

#### 4. split()

The split() function splits the string wherever a match occurs.

Example:

import re

txt = "The rain in Spain"

x = re.split("\s", txt)

print(x) # Output: ['The', 'rain', 'in', 'Spain']

#### 5. sub()

The sub() function replaces matches with a specified string.

Example:

import re

txt = "The rain in Spain"

x = re.sub("\s", "9", txt)

print(x) # Output: The9rain9in9Spain



**Understanding Data Types in Python**

### **Understanding Data Types in Python**

In Python, **data types** refer to the type of value that a variable can store. Understanding data types is essential for writing efficient programs as different types of data are handled differently by the system.

#### **1. What is a Data Type?**

A **data type** is an attribute of data that tells the computer how to interpret the data. In Python, everything is an object, and variables hold references to these objects. There are several built-in data types that Python supports, which can be categorized as numeric, sequence, mapping, set, and Boolean types.

### **Types of Data Types**

1. **Numeric Data Types**
   * **int**: Integer numbers (positive or negative whole numbers).
   * **float**: Decimal numbers or floating-point numbers.
   * **complex**: Complex numbers, with a real and imaginary part.
2. **Sequence Data Types**
   * **list**: Ordered collection of elements, which are mutable (can be changed).
   * **tuple**: Ordered collection of elements, which are immutable (cannot be changed).
   * **string**: A sequence of characters.
3. **Mapping Data Types**
   * **dictionary**: A collection of key-value pairs. The keys must be immutable, and values can be of any type.
4. **Set Data Types**
   * **set**: An unordered collection of unique elements.
5. **Boolean Data Type**
   * **bool**: Represents Boolean values, either True or False.

### **Numeric Data Types**

In Python, there are three main numeric data types:

1. **int**: Represents whole numbers, both positive and negative.
   * Example: age = 25
2. **float**: Represents decimal numbers.
   * Example: height = 5.9
3. **complex**: Represents complex numbers with real and imaginary parts.
   * Example: complex\_number = 3 + 4j

#### **Example Code:**

# Using int

age = 25

# Using float

height = 5.9

# Using complex

complex\_number = 3 + 4j

print("Age:", age)

print("Height:", height)

print("Complex number:", complex\_number)



### **Sequence Data Types**

#### **List**

A list is an ordered, mutable collection of elements. You can add, remove, or modify elements.

* **Example:**

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

fruits[0] = "orange" # Changing an element

fruits.append("grape") # Adding a new element

print(fruits)

#### **Tuple**

A tuple is similar to a list, but it is immutable (cannot be changed after creation).

* **Example:**

person = ("John", 25, "Engineer")

# Tuples cannot be modified:

# person[1] = 30 # This will raise an error.

print(person)

#### **String**

Strings are sequences of characters and are immutable.

* **Example:**

greeting = "Hello, World!"

print(greeting)

#### **Example Code for List and Tuple:**

# List Example

my\_list = [1, 2, 3, 4]

my\_list.append(5)

print("List:", my\_list)

# Tuple Example

my\_tuple = (1, 2, 3)

print("Tuple:", my\_tuple)



### **Dictionary Data Type**

A **dictionary** is a collection of key-value pairs. It is unordered, and you can access values by their corresponding keys.

* **Example:**

person = {"name": "Alice", "age": 30, "job": "Engineer"}

person["age"] = 31 # Updating a value

person["location"] = "New York" # Adding a new key-value pair

print(person)



### **Boolean Data Type**

The **Boolean** data type can hold one of two values: True or False. It is commonly used for conditionals and comparisons.

* **Example:**

is\_adult = True

is\_student = False

print("Is Adult:", is\_adult)

#### **Example Code for Boolean:**

age = 60

is\_senior\_citizen = age >= 60 # True if the person is 60 or older

print("Eligible for senior citizen discount:", is\_senior\_citizen)



### **Set Data Type**

A **set** is an unordered collection of unique elements. Sets do not allow duplicate elements.

* **Example:**

numbers = {1, 2, 3, 4, 5}

numbers.add(6) # Adding an element

numbers.remove(2) # Removing an element

print(numbers)

#### **Example Code for Set Operations:**

even\_numbers = {2, 4, 6, 8}

odd\_numbers = {1, 3, 5, 7}

# Union

union\_set = even\_numbers | odd\_numbers

print("Union:", union\_set)

# Intersection

intersection\_set = even\_numbers & odd\_numbers

print("Intersection:", intersection\_set)

# Difference

difference\_set = even\_numbers - odd\_numbers

print("Difference:", difference\_set)



### **Interactive Classroom Activities**

1. **Code Examples for Practice**:
   * **Program to calculate the area and circumference of a circle**:

radius = float(input("Enter the radius of the circle: "))

area = 3.14159 \* radius \*\* 2

circumference = 2 \* 3.14159 \* radius

print(f"Area: {area}, Circumference: {circumference}")



* + **Program to create and modify a list**:

colors = ["red", "blue", "green"]

colors.append("yellow") # Adding an element

colors.remove("blue") # Removing an element

print(colors)



* + **Program to demonstrate dictionary usage**:

contact = {"name": "John", "phone": "123-456-7890"}

contact["email"] = "john@example.com" # Adding a new key-value pair

print(contact)



1. **Discussion**:
   * **Why are sets unordered?**
     + **Explanation**: Sets are unordered collections, meaning there is no guaranteed order of elements. This makes accessing elements based on their index not possible in sets. Adding the same element twice does not change the set as elements in sets are unique.

### **Assignment Programs for Students**

1. **Finding the Number of Diagonals in a Polygon**:

sides = int(input("Enter the number of sides of the polygon: "))

diagonals = sides \* (sides - 3) // 2

print(f"Number of diagonals: {diagonals}")



1. **Converting a Number from One Base to Another**:

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

base = int(input("Enter the base (2 for binary, 16 for hexadecimal): "))

converted = format(number, f"{base}")

print(f"Converted number: {converted}")



### **Conceptual Questions**

1. **Which of the following is a complex number in Python?**
   * a) 3 + 4j
   * b) 3.14
   * c) 5
   * d) TRUE

**Answer**: a) 3 + 4j

1. **Which datatype is mutable?**
   * a) Tuple
   * b) String
   * c) List
   * d) Integer

**Answer**: c) List

1. **What is the correct way to create a dictionary in Python?**
   * a) my\_dict = {}
   * b) my\_dict = []
   * c) my\_dict = ()
   * d) my\_dict = set()

**Answer**: a) my\_dict = {}

1. **What is the result of the expression 5 == 5 in Python?**
   * a) True
   * b) False
   * c) Error
   * d) None

**Answer**: a) True

1. **Which of the following is a valid set in Python?**
   * a) {1, 2, 3}
   * b) [1, 2, 3]
   * c) (1, 2, 3)
   * d) 1 2 3

**Answer**: a) {1, 2, 3}

**Type Conversion and Memory Management in Python**

### **Type Conversion and Memory Management in Python**

In Python, understanding **type conversion** and **memory management** is crucial for efficient programming. These concepts help in ensuring data is appropriately handled during operations and optimizing the allocation of memory.

### **1. Implicit Type Conversion**

**Implicit Type Conversion** (also known as **type coercion**) is when Python automatically converts one data type to another during an operation without the programmer needing to do it manually. This happens automatically in most arithmetic and logical operations.

#### **Real-World Example:**

If you're calculating the sum of an integer and a float, Python automatically converts the integer to a float for the operation.

#### **Example Code:**

# Implicit Type Conversion

a = 10 # int

b = 3.14 # float

# The integer 'a' is implicitly converted to float to match the type of 'b'

result = a + b # result is a float

print(result) # Output: 13.14

#### **Other Examples:**

1. **Boolean Conversion**: In Python, True is implicitly converted to 1, and False is converted to 0 in arithmetic operations.

a = True # Equivalent to 1

b = 5

print(a + b) # Output: 6 (True is converted to 1)



1. **String Concatenation**: Implicit conversion doesn’t happen when strings are concatenated with non-string types.

a = "Hello"

b = 5

# This will throw an error because you can't concatenate a string with an integer implicitly

print(a + b) # TypeError



**Solution**: You can explicitly convert the integer to a string:

print(a + str(b)) # Output: Hello5



### **2. Explicit Type Conversion**

**Explicit Type Conversion** (also known as **type casting**) is when you manually convert one data type to another using functions like int(), float(), and str().

#### **Example Code:**

# Explicit Type Conversion

x = 10 # int

print(float(x)) # Convert to float, Output: 10.0

print(str(x)) # Convert to string, Output: '10'

#### **Real-World Example:**

1. **Convert a string to an integer**:

string\_value = "123"

integer\_value = int(string\_value) # Converts string to integer

print(integer\_value) # Output: 123



If the string is not a valid number:

string\_value = "abc"

# This will throw an error

# integer\_value = int(string\_value) # ValueError: invalid literal for int() with base 10: 'abc'



1. **Convert a float to an integer**:

float\_value = 3.14

integer\_value = int(float\_value) # Explicit conversion drops the decimal

print(integer\_value) # Output: 3



### **3. Random Number Generation**

Python has a built-in **random module** that allows you to generate random numbers for various purposes, such as simulations or games.

#### **Examples**:

1. **Generate a Random Integer Between 1 and 100**:

import random

random\_number = random.randint(1, 100)

print(random\_number) # Random integer between 1 and 100



1. **Simulate the Rolling of Two Dice**:

dice1 = random.randint(1, 6)

dice2 = random.randint(1, 6)

print(f"Dice 1: {dice1}, Dice 2: {dice2}")

print(f"Sum of Dice: {dice1 + dice2}")



1. **Generate 5 Unique Lottery Numbers**:

lottery\_numbers = random.sample(range(1, 51), 5)

print("Lottery Numbers:", lottery\_numbers)



### **4. Memory Management in Python**

Memory management in Python is handled automatically by the Python **memory manager**. This includes memory allocation, deallocation, and garbage collection. Understanding how Python manages memory is key to writing efficient programs.

#### **Memory Allocation**:

* **Immutable Types**: For example, integers, floats, strings, and tuples are **immutable**. When you modify an immutable object, a new object is created.
* **Mutable Types**: Lists, dictionaries, and sets are **mutable**. They allow changes to their content without creating new objects.

#### **Memory and the id() function**:

The id() function returns the **memory address** of an object, helping us understand how Python manages memory.

#### **Example**:

a = [1, 2, 3] # List (mutable)

b = (1, 2, 3) # Tuple (immutable)

print(id(a)) # Memory address of list

print(id(b)) # Memory address of tuple

a[0] = 4 # Modifying a mutable object

print(id(a)) # The id of 'a' remains the same since we only modified it

b = (4, 5, 6) # Reassigning an immutable object

print(id(b)) # The id of 'b' changes because we created a new tuple



### **5. Garbage Collection**

**Garbage collection** is the process by which Python automatically manages memory by reclaiming unused memory. When an object no longer has any references pointing to it, it is considered **garbage** and can be removed by the garbage collector.

* Python uses **reference counting** and a **cyclic garbage collector** to track and collect unused objects.

#### **When Does Garbage Collection Happen?**

* **When there are zero references**: If no variable is referring to an object, Python can reclaim the memory used by that object.

#### **Real-Time Example**:

import gc

# Create objects

x = [1, 2, 3]

y = x

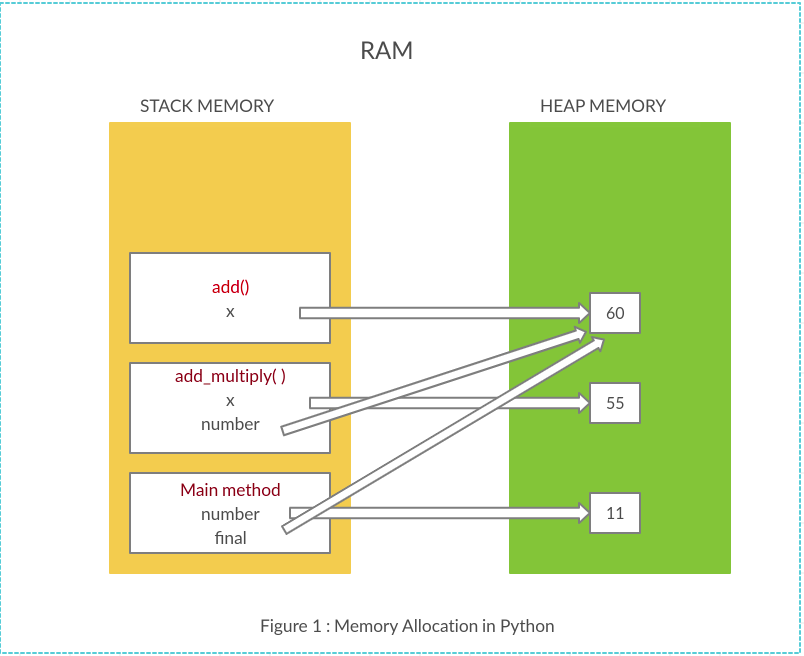
x = None # No references to the list object

# The object [1, 2, 3] is now eligible for garbage collection

gc.collect() # Force garbage collection

#### **Memory Management in Python (Pictorial Explanation)**:

* **Mutable vs Immutable**: The memory allocation and management between mutable (like lists) and immutable types (like tuples and strings) are different. When a mutable object is modified, its memory address remains the same. In contrast, when an immutable object is modified, a new memory address is created.



### **Classroom Activity**

1. **Implicit Type Conversion Demo**:
   * Show how Python converts integers and floats automatically in an arithmetic operation.
2. **Explicit Type Conversion**:
   * Write a program that converts a string input to an integer and handle invalid input.
3. **Random Number Generator**:
   * Generate random numbers and simulate a dice roll. Have students experiment with the random.randint() and random.sample() functions.
4. **Memory Management Activity**:
   * Show how mutable and immutable objects affect memory management.
   * Use id() to track memory locations of variables before and after modifications.

### **Discussion Questions**

1. **What happens during implicit type conversion in Python?**
   * **Answer**: Python automatically converts data types to a compatible type during operations, such as converting an integer to a float during arithmetic.
2. **How would you explicitly convert a string '123' to an integer?**
   * **Answer**: Use int('123') to convert the string to an integer.
3. **Which method is used to generate a random number in Python?**
   * **Answer**: The random.randint() method is used to generate random integers, and random.sample() is used for generating random samples.
4. **What is garbage collection in Python?**
   * **Answer**: It is the process by which Python reclaims memory by deleting objects that are no longer referenced by any variables.
5. **Which Python module is used for garbage collection?**
   * **Answer**: The gc module is used for interacting with the garbage collector in Python.

**Operators in Python**

#### Operators in programming are special symbols or keywords used to perform operations on variables and values. They act on data and produce results. Essentially, operators are the building blocks of expressions in a program.

#### There are several types of operators in programming, each used for different purposes. In Python, operators are classified into various categories:

#### Here are the different types of operators in programming:

#### Arithmetic Operators

#### Comparison Operators

#### Logical Operators

#### Assignment Operators

#### Bitwise Operators

#### Membership Operators

#### Identity Operators

#### Operator Precedence and Associativity

#### **1. Arithmetic Operators**

**Definition**: Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, division, etc.

**Operators**:

* + : Addition
* - : Subtraction
* \* : Multiplication
* / : Division
* // : Floor Division (divides and rounds down to nearest integer)
* % : Modulo (returns remainder of division)
* \*\* : Exponentiation (raises the left operand to the power of the right operand)

**Example 1: Basic Arithmetic Operations**

****# Input two numbers

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

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

# Perform basic arithmetic operations

addition = num1 + num2

subtraction = num1 - num2

multiplication = num1 \* num2

division = num1 / num2

print(f"Addition: {addition}")

print(f"Subtraction: {subtraction}")

print(f"Multiplication: {multiplication}")

print(f"Division: {division}")



**Example 2: Modulo Operation**

****# Modulo to get remainder when 25 is divided by 4

remainder = 25 % 4

print(f"Remainder when 25 is divided by 4: {remainder}")



**Practice**:

1. **Gross Salary Calculation**: (Example of using arithmetic operators)

basic\_pay = float(input("Enter basic pay: "))

HRA = basic\_pay \* 0.2

DA = basic\_pay \* 0.5

deductions = basic\_pay \* 0.1

gross\_salary = basic\_pay + HRA + DA - deductions

print(f"Gross Salary: {gross\_salary}")

1. **Percentage Calculation**:

total\_marks = float(input("Enter total marks: "))

marks\_obtained = float(input("Enter marks obtained: "))

percentage = (marks\_obtained / total\_marks) \* 100

print(f"Percentage: {percentage}%")



#### **2. Comparison Operators**

**Definition**: Comparison operators are used to compare two values. These operators return a boolean value: True or False.

**Operators**:

* == : Equal to
* != : Not equal to
* > : Greater than
* < : Less than
* >= : Greater than or equal to
* <= : Less than or equal to

**Example 1: Check Fever Condition**

****temperature = float(input("Enter patient's temperature: "))

if temperature >= 100.4:

print("The patient has a fever.")

else:

print("The patient does not have a fever.")



**Example 2: Prime Number Check**

****num = int(input("Enter a number: "))

is\_prime = True

if num > 1:

for i in range(2, num):

if num % i == 0:

is\_prime = False

break

if is\_prime:

print(f"{num} is a prime number.")

else:

print(f"{num} is not a prime number.")



#### **3. Logical Operators**

**Definition**: Logical operators are used to combine conditional statements.

**Operators**:

* and : Returns True if both conditions are true
* or : Returns True if at least one condition is true
* not : Reverses the boolean value (True becomes False, and vice versa)

**Example 1: User Validation**

****username = input("Enter username: ")

password = input("Enter password: ")

if username == "admin" and password == "admin123":

print("Login successful.")

else:

print("Login failed.")



**Example 2: Eligibility Check for Insurance**

****age = int(input("Enter age: "))

medical\_test\_pass = input("Did the medical test pass (yes/no)? ").lower()

if age >= 18 and age <= 60 and medical\_test\_pass == "yes":

print("Eligible for insurance.")

else:

print("Not eligible for insurance.")



#### **4. Assignment Operators**

**Definition**: Assignment operators are used to assign values to variables.

**Operators**:

* = : Assigns a value to a variable
* += : Adds and assigns
* -= : Subtracts and assigns
* \*= : Multiplies and assigns
* /= : Divides and assigns
* %= : Modulo and assigns

**Example: Retail Store Scenario**

****stock = 50

# Add new stock

stock += 20

print(f"New stock count: {stock}")

# Subtract sold items

stock -= 10

print(f"Remaining stock: {stock}")

# Apply discount on all items

price = 100

price \*= 0.9 # 10% discount

print(f"Discounted price: {price}")

# Calculate per-item cost

total\_cost = 500

items = 5

per\_item\_cost = total\_cost / items

print(f"Cost per item: {per\_item\_cost}")



#### **5. Bitwise Operators**

**Definition**: Bitwise operators are used to manipulate the individual bits of integers.

**Operators**:

* & : Bitwise AND
* | : Bitwise OR
* ^ : Bitwise XOR
* ~ : Bitwise NOT (inverts bits)
* << : Left shift
* >> : Right shift

**Example 1: Displaying Binary Representation**

****num1 = 5 # Binary: 101

num2 = 3 # Binary: 011

# AND operation

result = num1 & num2

print(f"5 & 3 = {result} (Binary: {bin(result)})")



**Example 2: Swapping Two Numbers Using Bitwise XOR**

****a = 5

b = 10

# Swapping using XOR

a = a ^ b

b = a ^ b

a = a ^ b

print(f"Swapped values: a = {a}, b = {b}")



#### **6. Membership Operators**

**Definition**: Membership operators are used to check if a value exists in a sequence (list, string, etc.).

**Operators**:

* in : Returns True if value is found in sequence
* not in : Returns True if value is not found in sequence

**Example 1: Check if Item Exists in List**

****fruits = ["apple", "banana", "orange"]

item = input("Enter a fruit to check: ")

if item in fruits:

print(f"{item} is available.")

else:

print(f"{item} is not available.")



#### **7. Identity Operators**

**Definition**: Identity operators are used to compare the memory locations of two objects.

**Operators**:

* is : Returns True if two variables refer to the same object
* is not : Returns True if two variables refer to different objects

**Example 1: Comparing Two Variables**

****a = 5

b = 5

if a is b:

print("a and b are the same object in memory.")

else:

print("a and b are different objects.")



### **Operator Precedence and Associativity**

**Precedence** refers to the order in which operations are performed in an expression. For example, multiplication and division have higher precedence than addition and subtraction.

**Associativity** refers to how operators of the same precedence are evaluated. Most operators in Python follow left-to-right associativity.

**Example:**

****T = 10 + 2 \* 3 - 4 / 2

1. 2 \* 3 is evaluated first (multiplication has higher precedence).
2. 4 / 2 is evaluated next (division has the same precedence as multiplication).
3. Then, addition and subtraction are performed from left to right.

Final output:

T = 10 + 6 - 2 = 14



**Associativity Example:**

****T = 2 \*\* 3 \*\* 2 # Right-to-left associativity for the exponentiation operator

This evaluates as:

T = 2 \*\* (3 \*\* 2) # First 3\*\*2 is calculated

T = 2 \*\* 9 = 512



### **Basic Syntax**

1. **What is the basic syntax of Python?**
   * Hint: Python uses indentation instead of braces and semicolons.
2. **What are Python identifiers, and what are the rules for naming them?**
   * Hint: Identifiers are names given to variables, functions, classes, or modules. They should start with a letter or an underscore.
3. **How do you assign values to variables in Python?**
   * Hint: Use the = operator to assign a value to a variable.
4. **What is the significance of indentation in Python syntax?**
   * Hint: Python uses indentation to define blocks of code, such as loops and functions.
5. **How do you create and use comments in Python code?**
   * Hint: Use the # symbol for single-line comments.
6. **What are the different data types available in Python?**
   * Hint: Python supports types like int, float, string, list, tuple, set, and dictionary.

### **Operators Quiz**

1. **Which of the following is a valid variable name in Python?**
   * A) 2variable
   * B) variable\_2
   * C) @variable
   * D) variable#2
   * **Answer:** B) variable\_2
2. **In Python, which of the following can be used to assign a value to a variable?**
   * A) ==
   * B) =
   * C) :=
   * D) ===
   * **Answer:** B) =
3. **What is the rule for Python identifiers?**
   * A) Identifiers must begin with a digit
   * B) Identifiers can contain spaces
   * C) Identifiers must begin with a letter or an underscore
   * D) Identifiers can contain special characters like @ or #
   * **Answer:** C) Identifiers must begin with a letter or an underscore
4. **Which of the following is the correct way to assign a value to a variable in Python?**
   * A) x <- 10
   * B) 10 = x
   * C) x = 10
   * D) x == 10
   * **Answer:** C) x = 10
5. **Which of the following is not a valid Python identifier?**
   * A) my\_variable
   * B) \_variable
   * C) variable2
   * D) 2variable
   * **Answer:** D) 2variable

### **User Input & Output**

1. **Which function is used to take user input in Python?**
   * Hint: Look for a function that allows user interaction.
2. **Which function is used to display output in Python?**
   * Hint: It’s the function that prints to the screen.
3. **How can you format a string in Python to include a variable?**
   * Hint: You can use f-strings or the format() method.
4. **How would you print multiple lines without using multiple print statements?**
   * Hint: Use escape characters or string concatenation.
5. **Which of the following will take input and then print a message?**
   * A) print(input("Enter your name: "))
   * B) input("Enter your name: ")
   * C) input() + print()
   * D) print(input())
   * **Answer:** D) print(input())

### **Data Types and Variables**

1. **What happens if you try to use a variable that has not been defined in Python?**
   * Hint: Look for an error message related to undefined variables.
2. **Which of the following is a valid Python variable?**
   * A) 1name
   * B) name\_1
   * C) @name
   * D) #name
   * **Answer:** B) name\_1
3. **What happens when you re-assign a variable in Python?**
   * Hint: The variable is updated to reference the new value.
4. **What is the scope of a variable assigned inside a function?**
   * Hint: Think about where the variable is accessible within the code.
5. **Which statement correctly assigns multiple variables in one line?**
   * A) x = y = z = 10
   * B) x y z = 10
   * C) x, y, z = 10
   * D) 10 = x = y = z
   * **Answer:** A) x = y = z = 10

### **String Manipulation**

1. **Write a Python program to replace all vowels in a given string with asterisks (\*).**
   * Hint: Think about iterating through the string and checking each character.
2. **Create a program to extract all hashtags (e.g., #example) from a sentence.**
   * Hint: Use string methods and regular expressions to extract hashtags.

### **Regular Expressions**

1. **What is the purpose of regular expressions in Python?**
   * Hint: They are used for pattern matching in strings.
2. **Which module in Python is used to work with regular expressions?**
   * Hint: The module is called re.
3. **What does the re.match() function do?**
   * Hint: It checks if the pattern matches at the beginning of the string.
4. **Which metacharacter is used to represent the start of a string?**
   * Hint: It’s the caret symbol ^.
5. **What does \d represent in regular expressions?**
   * Hint: It represents a digit.

### **Data Structures**

1. **Which of the following is a complex number in Python?**
   * A) 3 + 4j
   * B) 3.14
   * C) 5
   * D) TRUE
   * **Answer:** A) 3 + 4j
2. **Which datatype is mutable?**
   * Hint: A mutable type can be changed after creation.
3. **What is the correct way to create a dictionary in Python?**
   * A) {"key": value}
   * B) {"key" => value}
   * C) key: value
   * D) key = value
   * **Answer:** A) {"key": value}
4. **What is the result of the expression 5 == 5 in Python?**
   * Hint: It checks for equality.
5. **Which of the following is a valid set in Python?**
   * A) {1, 2, 3}
   * B) [1, 2, 3]
   * C) (1, 2, 3)
   * D) 1 2 3
   * **Answer:** A) {1, 2, 3}

### **Control Flow**

1. **Write a program that checks if a number is within a specified range (e.g., 10 to 20).**
   * Hint: Use conditional statements to check the range.
2. **Create a program to compare the prices of two products and determine which is cheaper.**
   * Hint: Use comparison operators to compare the prices.
3. **Write a program to check if a student is eligible for a scholarship based on their score (e.g., score > 85).**
   * Hint: Use an if statement to check eligibility.

### **Assignment Operators**

1. **Which of the following is used for shorthand assignment in Python?**
   * A) +=
   * B) ==
   * C) =
   * D) -=
   * **Answer:** A) +=
2. **What is the result of 5 & 3 in Python?**
   * Hint: This involves bitwise AND.
3. **Which operator is used for bitwise OR in Python?**
   * Hint: It’s the | symbol.
4. **What will be the result of "Python" in "Python is great"?**
   * Hint: Look for the substring in the string.
5. **What is the result of 5 is 5 in Python?**
   * Hint: This checks for object identity, not equality.

**END OF UNIT-2**