## Q. What do you know about Python?

### Python

**Python** is one of the most popular and widely used programming languages today. It is a versatile language used in various fields such as web development, data analysis and artificial intelligence. Python's simple syntax and clear structure make it an excellent choice for beginners to learn programming concepts easily.

## Q. What is a development environment in programming?

### Development Environment

A **development environment** is a collection of all the necessary tools used to write, run and debug computer programs. It serves as a basic platform for writing and executing computer programs. Setting up the development environment is essential before starting to write programs.

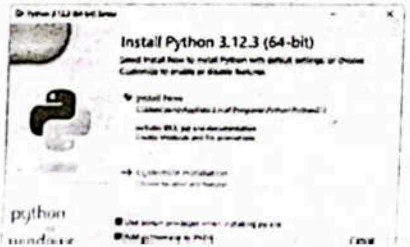
An **Integrated Development Environment (IDE)** is a type of development environment that combines several tools into a single application. These tools typically include code editor, debugger and compiler or interpreter. **IDLE** is a simple IDE that is automatically installed with Python. It is suitable for beginners because it is easy to use. **PyCharm** is a powerful and widely used IDE designed for Python programming. It offers advanced features that support professional software development.

## Q. Write the procedure to download and install the Python development environment.

The following procedure is used to download and install Python development environment:

1. Open the web browser.
2. Type **https://www.python.org/downloads** and press **Enter**. The following web page will appear:


3. Click **Download Python** button. A file will be downloaded.
4. Double click the file. The following window will appear:





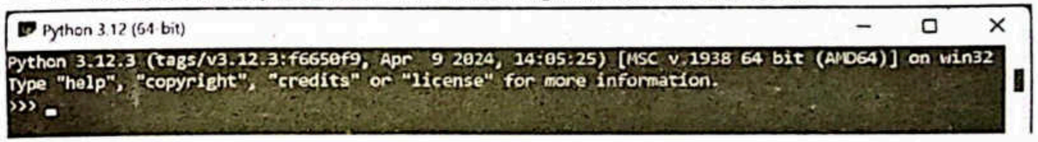

5. Select **Use admin privileges when installing py.exe** and **Add python.exe to PATH** checkboxes. This makes it easier to run Python from the command line.
6. Click **Install Now**. Windows will display a message "Do you want to allow this app to make changes to your device?"
7. Click **Yes**. The installation will start and the following window will appear when the installation process is completed:


8. Click **Close**.

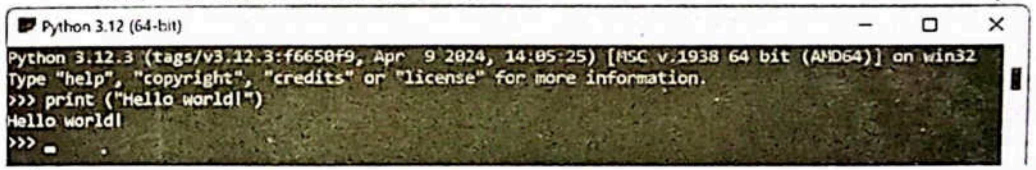
## Q. Write the procedure to run a command in Python.

The **print** command is used in Python to display a message on the command prompt. The followong procedure is used to run a command in Python:

1. Double click **Python** icon. The following window will appear:



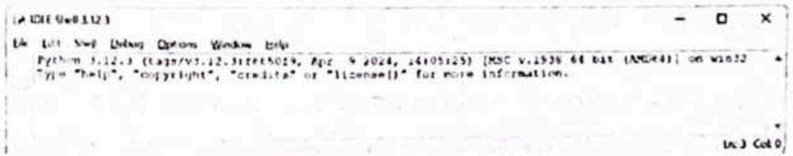

2. Type the command **print ("Hello world!")** and press **Enter**. The output "Hello world" will appear on the screen:



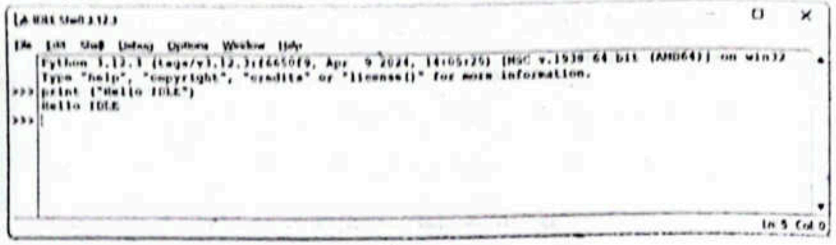

## Q. Write the procedure to start IDLE and write Python commands.

The followong procedure is used start IDLE and run a command:

1. Type **IDLE** in search bar of Windows . The **IDLE (Python 3.12 64-bit)** will appear in search results.
2. Click **IDLE (Python 3.12 64-bit)** icon IDLE (Python 3.12 64-bit). The **IDLE** will start and the following window will appear:



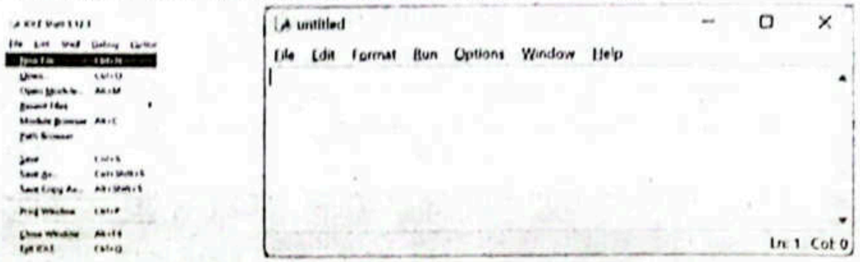
3. Type the command **print ("Hello IDLE")** and press **Enter**. The output "Hello IDLE" will appear on the screen:



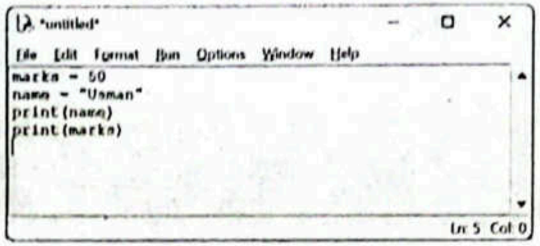

## Q. Write the procedure to create, save and execute a program in Python.

The following procedure is used to create, save and execute a program in Python:

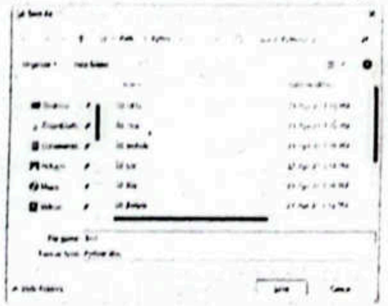
1. Start **IDLE**.
2. Click **File > New File** or press **Ctrl+N**. A blank file will appear.




3. Type the desired commands in the file as follows:



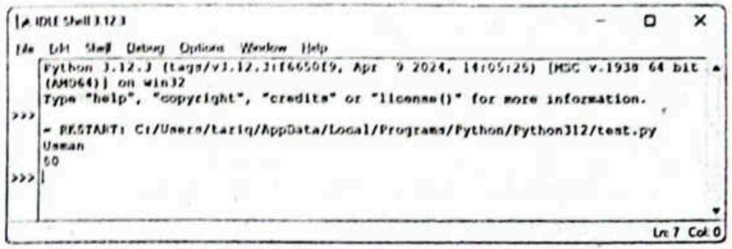

4. Click **File > Save** or press **Ctrl+S**. The **Save As** dialog box will appear.



5. Type the desired file name in **File name** box and click **Save**. The program will be saved.



6. Click **Run > Run module** or press **F5**. The program will run and output will appear.




## Q. What do you know about keywords of Python? Give some examples.

### Keywords

**Keyword** is a word in a programming language that has a predefined meaning and purpose. The meaning and purpose of a keyword is defined by the developer of the language. It cannot be changed or redefined by the user. Keyword can be used for the same purpose for which it is defined. Keywords are also called **reserved words.**

### Examples

Some examples of keywords in Python are as follows:

| | | | | |
|---|---|---|---|---|
| and | or | if | elif | else |
| def | For | in | or | return |

## Q. What are comments in Python? Describe their purpose and syntax.

### Comments

**Comments** are the lines of text in the program that are not executed by Python interpreter. The comments are used to provide explanations or notes for the code. They allow other programmers to understand the code more easily. The comments can be added anywhere in the program.

### Types of Comments

Comments can be added to the programs in single line or multiple lines.

### 1. Sing-line Comments

The single-line comments start with #. Anything after # on the same line is considered as comments. An example of single-line comments is as follows:

```
# Program 2.5: Finding the area of a rectangle
```

### 2. Multi-line Comments

In Python, multi-line comments are often written using triple quotes (''' or """). Anything written between triple quotes is considered a comment. Multi-line commends are typically used for longer explanations. An example of adding multi-line comments is as follows:

```
'''
Program 2.7
Description: Finding the area of a circle
Written by: Abdullah
'''
```

## Q. What is a variable? Write the rules to specify variable names in Python.

### Variable

A **variable** is a named memory location. It is used to store program's input data and its computational results during execution. The value of a variable may change during the execution of program. However, the name of variable cannot be changed. If a new value is stored in a variable, it replaces the previous value. Each variable has a unique name called **Identifier**. A variable also has a data type that indicates the type of data that can be stored in the variable.

### Rules for Variable Name

Different rules for naming variables in Python are as follows:

- The first character of variable name must be an alphabet or underscore. For example, the variables **9minute**, **#home** and **2kg** are invalid variable names.
- The characters allowed in variable name include uppercase letters (A to Z), lowercase letters (a to z), digits (0 to 9) and underscore (_).
- Python is case-sensitive language. The uppercase and lowercase letters are considered different in Python. For example, **MARKS** is different from **marks**.
- The special symbols such as @, $, % are not allowed in variable names. For example, **$cost** and **tot.al** are invalid variable names.
- The spaces are not allowed in variable names. For example, **my var** and **your car** are invalid variable names.

### Examples

Some examples of valid variables are as follows:

Marks    sub1    _test    first_name

## Q. What is data type? List the basic data types in Python.

### Data Types

The **data type** indicates the type of data and a set of operations that can be performed on data. The variables are used without specifying the data types. Python automatically detects the data type based on the type of data stored in variables.

### Basic Data Types in Python

Python provides following basic data types:

**Integer** It is used to store whole numbers that do not have a decimal point or fraction. Some examples of integers are 10, 520, and -20.

**Floating-point** It is used to store real numbers. A real number is a numeric value with a decimal point or fractional part such as 2.5, 4.0, and -5.91.

**String** It is used to store text data. The text must be enclosed in either single quotes ('Python') or double quotes ("Hello"). Strings are commonly used to represent names, messages or any sequence of characters.

**Boolean** It stores only two possible values which are **True** or **False**.



## Q. How variables are created in Python? Explain with examples.

### Creating Variables

Python does not require variable declaration because it automatically detects the type based on the assigned value. The variables can be created by writing a variable name and assigning a value. The symbol = is used to assign a value to a variable. The variable name is written on left side and the value is written on the right side of = symbol. The symbol = is called the **assignment operator**.

### Syntax

The syntax of creating a variable is as follows:

var = value

**var** It indicates the name of the variable.

**=** It is the assignment operator used to assign a value to the variable.

**value** It is the value to be assigned to a variable.

### Example

Some examples of creating variables are as follows:

```
n = 100
avg = 50.73
grade = 'A'
city = "Islamabad"
```

**PROGRAM 2.1** **Write a program in Python that creates three variables to store name, marks and average of a student and displays them.**

1. Start **IDLE** and open a new file.
2. Type the following code:

```
Prog3_1.py - C:/Users/tariq/AppData/Local/Programs/Python/Python312/Prog3_1.py (3.12.3)
File Edit Format Run Options Window Help
Name = "Ali"
Marks = 561
Average = 93.5
print(Name)
print(Marks)
print(Average)
Ln: 7 Col: 0
```

3. Save the file as "Prog3_1.py".
4. Click **Run > Run module** or press **F5**. The output will appear as follows:

```
IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
    Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit
    (AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    = RESTART: C:/Users/tariq/AppData/Local/Programs/Python/Python312/Prog3_1.py
    Ali
    561
    93.5
>>>
Ln: 8 Col: 0
```

### Explanation

The above program creates three variables **Name, Marks** and **Average** and store the initial values in them. It uses three **print** statements to display the values of these variables.

## Q. What is input and output in computer programming?

### Input

In computer programming, the process of providing data to the program is known as **input**. The input is mostly given by the keyboard. The input given to the computer through keyboard is known as **standard input**.

### Output

In computer programming, the process of displaying the result of the program is known as **output**. The output is mostly displayed on the monitor. The output displayed on the monitor is known as **standard output**.

## Q. What is the use of print() function in Python? Describe with examples.

### The print() Function

The **print()** function in Python is used to display standard output on the screen. It can display text or the value of the variable on the screen. The **print()** function adds a newline character at the end of the output by default.

The values to be printed on the screen are given as arguments in parenthesis. Multiple values given to the function are separated with comma. The plus (+) symbol is used to concatenate two strings given in this function. A string is a set of characters enclosed in single quotes (') or double quotes ("). It is used to represent textual data in programming.

### Syntax

The syntax of using **print()** function is as follows:

```
print(val)
```

**val** It indicates the text or value to be displayed.

**Example 1** The following example has two print statements. The first statement will display "Pakistan Zindabad". The string is enclosed in single or double quotes. The second statement will display 720. The numeric constant is not enclosed in quotation marks.

```
print("Pakistan Zindabad")
print(720)
```

**Output:** Pakistan Zindbad
720

**Example 2** The following example will display a message along with the value of variable **a**. The variables are not enclosed in quotation marks. The string and the variable are separated with comma in **print()** function.

```
a = 100
print("The value of a is ", a)
```

**Output:** The value of a is 100

**Example 3** The following example will display two strings concatenated with the plus (+) symbol:

```
str1 = "Hello"
str2 = "World"
print(str1 + str2)
```

**Output:** HelloWorld



## Q. What is the use of end character in print() function?

### The end Character

By default, the **print()** function adds a newline character at the end of the output. It causes the next output to appear on the next line. The **end** character can be used in **print()** function to change this behavior. For example, the **end** character can be set to a space so that the output of the next **print** statement appears on the same line instead of moving to the next line.

**Example** The following example has two print statements. It uses **end** character in the first statement to end the output with space so the output of the second statement starts after the output of the first statement.

```
print("Python", end=" ")
print("Programming")
```

**Output:** Python Programming

## Q. What is the use of input() function in Python? Describe with example.

### The input() Function

The **input()** function in Python is used to accept input from the user during program execution. It inputs the value from the user via keyboard and returns the entered value as a string. The function is normally used with a variable that stores the value entered by the user.

### Syntax

The syntax of using **input()** function is as follows:

```
var = input(prompt)
```

**var** It indicates the variable in which the given input will be stored.

**prompt** It indicates the message given to the user to enter input. It is optional.

### Example

The following example will display "Enter a number: " on the screen. It will get the input from the user and store it in the variable **num**.

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

## Q. What is the data type of values returned by input() function in Python?

### Data Type of Input Value

The value returned by **input()** function is always a string even if the user enters a number. For example, the addition operator applied on two input values will work as a concatenation operator as follows:

```
a = input("Enter a number: ")
b = input("Enter a number: ")
print("a + b = ", a + b)
```

**Output:**
Enter a number: 3
Enter a number: 5
a + b = 35

The **input()** function in above example inputs two numbers but returns them as strings. The addition operator + in the last line performs a concatenation operation and displays 35 instead of 8.

## Q. How are the integer and float values handled in Python. Explain with examples.

### Handling Integer and Float Inputs

Python considers all input values as string by default. For example, the input value 50 is considered as "50" and the input value 3.1 is considered as "3.1". These string values need to be converted into integer and float values in order to use them in the program properly. Python provides **int()** function to handle integer inputs and **float()** function to handle the float inputs.

### 1. The int() Function

The **int()** function is used to handle integer input. It converts the input value to an integer data type. The syntax of using **int()** function is as follows:

```
var = int(input(prompt))
```

| | |
|---|---|
| **var** | It indicates the variable in which the given input will be stored. |
| **int** | It indicates the name of the function that converts the input to integer. |
| **prompt** | It indicates the message given to the user to enter input. It is optional. |

### Example

The following example inputs two numbers using **input()** function. It uses int() function to convert the input values to integers. The addition operator + in the last line performs an addition operation and displays the result 8.

```
a = int(input("Enter a number: "))
b = int(input("Enter a number: "))
print("a + b = ", a + b)
```

**Output:**
Enter a number: 3
Enter a number: 5
a + b = 8

### 2. The float() Function

The **float()** function is used to handle float input. It converts the input value to a float data type. The syntax of using **float()** function is as follows:

```
var = float(input(prompt))
```

| | |
|---|---|
| **var** | It indicates the variable in which the given input will be stored. |
| **float** | It indicates the name of the function that converts the input to floating-point data type. |
| **prompt** | It indicates the message given to the user to enter input. It is optional. |

### Example

The following example inputs two numbers using **input()** function. It uses **float()** function to convert the input values to floating-point.

```
math = float(input("Enter marks in Math: "))
eng = float(input("Enter marks in Englis: "))
print("Average = ", (math + eng) / 2)
```

**Output:**
Enter marks in Math: 97
Enter marks in Englis: 92
Average = 94.5



## Q. What are operators? List different types of operators in Python.

### Operators

**Operators** are symbols used to perform various operations on data in a program. Python provides a variety of operators. Different operators are used to perform different operations on data. For example, the operator + is used to add two numbers.

Different types of operators in Python are as follows:

- Arithmetic Operators
- Assignment Operators
- Arithmetic assignment Operators
- Comparison Operators
- Logical Operators

## Q. What is an expression? Give some examples of expressions in Python.

### Expression

An **expression** is a combination of constants, variables and operators. It returns a single value when it is executed. The variables and constants in an expression are called **operands**. An **operator** is a symbol that tells the computer to perform certain type of operation on operands.

### Examples

Different examples of expressions in Python are as follows:

- A + B
- m / n
- x + 100
- 12 + 2

## Q. What is an assignment operator? Write its syntax and give examples.

### Assignment Operator

The **assignment operator** is used to assign a value to a variable. The symbol = is used as assignment operator in Python. A statement that assigns a value to a variable is known as **assignment statement**.

The name of variable is written on the left side of the assignment operator and the value is written on the right side of assignment operator. The value can be a constant, variable, expression or function.

### Syntax

The syntax of using assignment operator is as follows:

```
var = expr
```

| | |
|---|---|
| **var** | It is the name of variable to which the value is assigned. |
| **=** | It is the assignment operator used to assign a value to the variable. |
| **expr** | It is the expression whose value is assigned to the variable. It can be a constant, variable or an arithmetical expression. |

### Examples

Some examples of using assignment operator are as follows:

```
a = 100
c = a + b
x = c - d + 10
```

**PROGRAM 2.2** **Write a program that swaps the values of two integer variables.**

```
# Program to swap the values of two integer variables
a = 2
b = 3
print("Values before swapping: ")
print("a = ", a, ", b = ", b)
temp = a
a = b
b = temp
print("Values after swapping: ");
print("a = ", a, ", b = ", b)
```

**Output:**
Values before swapping:
a = 2, b = 3
Values after swapping:
a = 3, b = 2

**PROGRAM 2.3** **Write a program that swaps the values of two integer variables without using temporary variable.**

```
a = 5       # First variable
b = 10      # Second variable
print("Values before swapping: ")
print("a = ", a, ", b = ", b)
a = a + b
b = a - b
a = a - b
print("Values after swapping. ");
print("a = ", a, ", b = ", b)
```

**Output:**
Values before swapping:
a = 5, b = 10
Values after swapping:
a = 10, b = 5

**Q. What is arithmetic operator? List different arithmetic operators in Python. Also describe arithmetic expression.**

### Arithmetic Operator

**Arithmetic operator** is a symbol that performs mathematical operation on data. Python provides many arithmetic operators which are as follows:

| Operation | Symbol | Description |
|---|---|---|
| Addition | + | It adds two numbers. |
| Subtraction | – | It subtracts one number from another number. |
| Multiplication | * | It multiplies two numbers. |
| Division | / | It divides one number by another number and give result as a floating-point number. |
| Floor division | // | It divides one number by another and gives the result as a whole number. |
| Modulus | % | It divides one number by another number and gives the remainder. |
| Exponent | ** | It raises a number to the power another number. |

**Table:** Arithmetic operators in Python



Some important points about modulus operator are as follows:

- Modulus operator is also called **remainder operator.**
- The modulus operator works only with integer values.
- If modulus operator is used with the division of 0, the result will always be 0. For example, the expression 0 % 5 will give 0 as a result.
- In expression like 3 % 5, 3 is not divisible by 5. Its result is 3.

### Arithmetic Expression

A type of expression that consists of constants, variables and arithmetic operators is called **arithmetic expression.** These expressions are used to perform arithmetic operations such as addition, subtraction and division etc.

### Examples

Suppose we have two variables A and B where A = 5 and B = 2. The following table shows the arithmetic expressions and their result:

| Arithmetic Expression | Result |
|---|---|
| A + B | 7 |
| A – B | 3 |
| A * B | 10 |
| A / B | 2.5 |
| A // B | 2 |
| A % B | 1 |
| A ** B | 25 |

**Table:** Examples of arithmetic expression

**PROGRAM 2.4** **Write a program that performs the arithmetic operations on two variables.**

```
a = 10
b = 5
print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b)
print("a / b =", a / b)
print("a % b =", a % b)
print("a ** b =", a ** b)
```

**Output:**
a + b = 15
a – b = 5
a * b = 50
a / b = 2.0
a % b = 0
a**b= 100000

**Q. What is compound assignment operator. Write its syntax and examples.**

### Compound Assignment Operator

Python provides **compound assignment operators** that combine assignment operator with arithmetic operators. These operators are used to perform arithmetic operations more easily.

### Syntax

The general syntax of compound assignment operator is as follows:

```
variable op = expression;
```

**variable** The variable to assign a value.
**op** It can be any arithmetic operator.
**expression** It can be a constant, variable or arithmetic expression

### Examples

Some examples of compound assignment operator are as follows:

| Operator | Example | Equivalent to |
|---|---|---|
| += | A += 10 | A = A + 10 |
| -= | A -= 10 | A = A - 10 |
| *= | A *= 10 | A = A * 10 |
| /= | A /= 10 | A = A / 10 |
| %= | A %= 10 | A = A % 10 |

**PROGRAM 2.5 Write a program that performs all compound assignment operations on an integer.**

```
a = 10
print("Value of a: ", a)
a += 5
print("Value of a after a+=5: ", a)
a -= 5;
print("Value of a after a-=5: ", a)
a *= 2
print("Value of a after a*=2: ", a)
a /= 2
print("Value of a after a/=2: ", a)
a %= 2
print("Value of a after a%=2: ", a)
```

**Output:**

```
Value of a : 10
Value of a after a+=5 : 15
Value of a after a-=5 : 10
Value of a after a*=2 : 20
Value of a after a/=2 : 10.0
Value of a after a%=2 : 0.0
```

### Q. Describe precedence of arithmetic operators in Python with example.

### Operator Precedence

The order in which different types of operators in an expression are evaluated is known as **operator precedence**. It is also known as **hierarchy of operators**. Each operator has its own precedence level. If an expression contains different types of operators, the operators with higher precedence are evaluated before the operators with lower precedence.

The following table shows the order of precedence for arithmetic operators in Python. The operators with the highest precedence are written at the top of the table. The operators with the lowest precedence are written at the bottom of the table. Any expression written in parenthesis is always evaluated first. In case of equal precedence, the operator on left side is evaluated before the operator on right side.

| Precedence | Operator | Operator Name |
|---|---|---|
| 1 | () | Parentheses |
| 2 | ** | Exponentiation |
| 3 | *, /, //, % | Multiplication, division and remainder operator |
| 4 | +, - | Addition, subtraction |

### Example

The expression **10 * (24 / (5 – 2)) + 13** is evaluated in the following order:



10 * (24 / (5 – 2)) + 13
↓
10 * (24 / 3) + 13
↓
10 * 8 + 13
↓
80 + 13
↓
93

1. First of all, the expression 5-2 will be evaluated. It gives a value 3.
2. Secondly, 24 will be divided by the result of last line i.e. 24 / 3 giving value 8.
3. Thirdly, 10 will be multiplied by 8 i.e. giving a result 80.
4. Finally, 80 will be added in 13 and the last result will be 93.

### Q. Compute the following expressions:

a) 10 + 3 * 2 ** 2 - 5 / 5

```
= 10 + 3 * 4 - 5 / 5        # 2 ** 2 = 4
= 10 + 12 - 5 / 5           # 3 * 4 = 12
= 10 + 12 – 1               # 5 / 5 = 1
= 22 – 1                    # 10 + 12 = 22
= 21
```

b) (10 + 3) * (2 ** (2 - 1)) / 5

```
= 13 * (2 ** 1) / 5         # 10 + 3 = 13, 2 - 1 = 1
= 13 * 2 / 5                # 2 ** 1 = 2
= 26 / 5                    # 13 * 2 = 26
= 5.2
```

**PROGRAM 2.6 Write a program that inputs two numbers in the variables x and y. It adds the numbers and displays the result.**

```
x = int(input("Enter first number: "))
y = int(input("Enter second number: "))
z = x + y;
print("The result = ", z)
```

**Output:**

```
Enter first number: 10
Enter second number: 20
The result = 30
```

**PROGRAM 2.7 Write a program that reads four integers and prints their sum, product and average.**

```
a = int(input("Enter first integer: "))
b = int(input("Enter second integer: "))
c = int(input("Enter third integer: "))
d = int(input("Enter fourth integer: "))
s = a + b + c + d
p = a * b * c * d
avg = s / 4.0
print("Sum = ", s)
print("Product = ", p)
print("Average = ", avg)
```

**Output:**

```
Enter first integer: 2
Enter second integer: 3
Enter third integer: 5
Enter fourth integer: 7
Sum = 17
Product = 210
Average = 4.25
```

**PROGRAM 2.8 Write a program that reads the base and height of a triangle and prints the area of the triangle. Hint: The formula Area = ½ * Base * Height.**

```
base = float(input("Enter base: "))
height = float(input("Enter height: "))
area = 0.5 * base * height
print("Area = ", area)
```

**Output:**
Enter base: 10.5
Enter height: 5.4
Area = 28.35

**PROGRAM 2.9** **Write a program that reads length and breadth of rectangle and prints its area.**

```
length = int(input("Enter length of rectangle: "))
b = int(input("Enter breadth of rectangle: "))
area = length * b
print("Area of rectangle = ", area)
```

**Output:**
Enter length of rectangle: 5
Enter breadth of rectangle: 4
Area of rectangle = 20

**PROGRAM 2.10** **Write a program that gets temperature from the user in Celsius and converts it into Fahrenheit using the formula F = 9/5 * C + 32.**

```
c = float(input("Enter temperature in Celsius: "))
f = 9.0 / 5.0 * c + 32
print("Temperature in Fahrenheit Is ", round(f, 2))
```

**Output:**
Enter temperature in Celsius: 37
Temperature in Fahrenheit is 98.6

**Note:** The **round** function in the above program is used to round the value to 2 decimal points.

**PROGRAM 2.11** **Write a program that reads temperature in Fahrenheit and prints its equivalent temperature in Celsius using the formula c = 5/9 * (f – 32).**

```
f = float(input("Enter temperature in Fahrenheit:"))
c = 5.0 / 9.0 * (f - 32)
print("Temperature in Celsius is ", round(c, 2))
```

**Output:**
Enter temperature in Fahrenheit: 85
Temperature in Celsius is 29.44

**PROGRAM 2.12** **Write a program that converts a person's height from inches to centimeters using the formula 2.54 * height.**

```
height = int(input("Enter height in inches: "))
cent = height * 2.54
print("Height in centimeters is: ", round(cent, 2))
```

**Output:**
Enter height in inches: 20
Your height in centimeters is: 50.8

**PROGRAM 2.13** **Write a program that inputs radius from the user and calculates area and circumference of circle using formulas Area = $\pi R^2$ and circumference = $2\pi R$.**

```
radius = float(input("Enter radius: "))
area = radius * radius * 3.141
cir = 2.0 * 3.141 * radius
print("Area: ", round(area, 2))
print("Circumference: ", round(cir, 2))
```

**Output:**
Enter radius: 5
Area: 78.53
Circumference: 31.41

**PROGRAM 2.14** **Write a program that inputs a three-digit number from the user and displays its digits in reverse order. For example, if the user enters 123, it displays 321.**

```
n = int(input("Enter 3-digit number: "))
a = n // 100
n = n % 100
b = n // 10
n = n % 10
print("Digits in reverse order is", n, b, a)
```

**Output:**
Enter 3-digit number: 512
Number in reverse order is 2 1 5



**PROGRAM 2.15** **Write a program that inputs the distance traveled and the speed of vehicle. It calculates the time required to reach the destination and displays it.**

```
d = int(input("Enter distance in miles: "))
s = int(input("Enter vehicle speed (mph): "))
time = d / s
print("Time to reach destination: ", time, " hours")
```

**Output:**
Enter distance in miles: 100
Enter vehicle speed (mph): 50
Time to reach destination: 2.0 hours

**Q. What are comparison operators? Discuss various comparison operators in Python?**

### Comparison Operators

The **comparison operators** compare two values and produce result as **true** or **false**. These operators are used to specify conditions in programs. The comparison operators are also called **relational operators**.

Different comparison operators in Python are as follows:

- `>` It is called **greater than operator**. It returns **true** if the value on left side of > is greater than the value on right side. Otherwise, it returns **false**.
- `<` It is called **less than operator**. It returns **true** if the value on left side of < is less than the value on right side. Otherwise, it returns **false**.
- `==` It is called **equal to operator**. It returns **true** if the values on both sides of == are equal. Otherwise, it returns **false**.
- `>=` It is called **greater than or equal to operator**. It returns **true** if value on left side of >= is greater than or equal to the value on right side. Otherwise, it returns **false**.
- `<=` It is called **less than or equal to operator**. It returns **true** if the value on left side of <= is less than or equal to the value on right side. Otherwise, it returns **false**.
- `!=` It is called **not equal to operator**. It returns **true** if the value on the left side of != is not equal to the value on right. Otherwise, it returns **false**.

### Examples

Suppose the value of **A = 10** and the value of **B = 5**. The result of different comparison operators will be as follows:

| Relation Expression | Result |
|---|---|
| A > B | True |
| A < B | False |
| A <= B | False |
| A >= B | True |
| A == B | False |
| A != B | True |

**Q. What are logical operators? Discuss various logical operators in Python.**

### Logical Operators

The **logical operators** are used to combine multiple conditions or reverse a condition. There are three logical operators in Python called **and** operator, **or** operator and **not** operator.

### 1. The and Operator

The **and** operator is used to form a compound condition in which two conditions are evaluated. The conditions are given using comparison operators. The compound condition returns **true** if both conditions are **true**. It returns **false** if any one condition is **false**.

The following table shows all possible results of **or** operator:

| Condition 1 | Operator | Condition 2 | Result |
|---|---|---|---|
| False | and | False | False |
| False | and | True | False |
| True | and | False | False |
| True | and | True | True |

**Example**

Suppose we have two variables A = 100 and B = 50. The compound condition **(A>10) and (B>10)** is **true**. It contains two conditions and both are **true**. So the whole compound condition is also **true**. The compound condition **(A>50) and (B>50)** is **false**. It contains two conditions. One condition **(A>50)** is **true** and second condition **(B>50)** is **false**. So the whole compound condition is **false**.

### 2. The or Operator

The **or** operator is used to form a compound condition in which two conditions are evaluated. The conditions are given using comparison operators. The compound condition returns **true** if any condition is **true**. It returns **false** if both conditions are **false**.

The following table shows all possible results of **or** operator:

| Condition 1 | Operator | Condition 2 | Result |
|---|---|---|---|
| False | or | False | False |
| False | or | True | True |
| True | or | False | True |
| True | or | True | True |

**Example**

Suppose we have two variables A = 100 and B = 50. The compound condition **(A>50) or (B>50)** is **true**. It contains two conditions and one condition **(A>50)** is **true**. So the whole compound condition is also **true**. The condition **(A>500) or (B>500)** is **false** because both conditions are **false**.

### 3. The not Operator

The **not** operator is used with single condition. It is used to reverse the result of a condition. It returns **true** if the condition is **false** and returns **false** if the condition is **true**. The following table shows all possible results of **not** operator:

| Operator | Condition | Result |
|---|---|---|
| not | True | False |
| not | False | True |

**Example**

Suppose we have two variables A = 100 and B = 50. The condition **not(A==B)** is **true**. The result of **(A==B)** is **false** but **not** operator converts it into true. The condition **not(A>B)** is **false**. The condition **(A>B)** is **true** but **not** operator converts it into **false**.



### Q. What is a condition? Give some examples of conditions.

**Condition**

A **condition** is an expression that is either **true** or **false**. A condition may consist of arithmetic expressions, relational expressions and logical expressions.

**Examples**

Some examples of conditions are as follows:

| Condition | Result |
|---|---|
| 5 > 3 | True |
| !(10 > 5) | False |
| 25 == 25 | True |
| (2 <= 10) and (3 >= 5) | False |
| (9 < 50) or (7 > 15) | True |

### Q. What are control structures? Describe different types of control structures in Python.

**Control Structure**

**Control structure** is a statement used to control the flow of execution in a program. Python executes statements sequentially from top to bottom by default. It means the statements are executed in the same order in which they are written in the program. Control structures can be used to alter the default flow of execution. Two types of control structures in Python are decision-making and looping.

### 1. Decision Making

**Decision-making structures** are used to execute a statement or a set of statements based on a condition. The statement is executed if the condition is **true**. The statement is skipped if the condition is **false**. It is also called **selection structure** or **conditional structure**. The common decision-making structures in Python are as follows:

- if
- if-else
- if-elif-else

**Example**

Suppose a program inputs the marks of a student and displays **Pass** if the student gets 40 or more than 40 marks. It displays **Fail** when the marks are below 40. The program checks the marks before displaying the message. This process is known as **decision-making**.

### 2. Looping

**Looping structures** are used to execute a statement or set of statements repeatedly. The common looping structures in Python are as follows:

- for loop
- while loop

**Example**

Suppose the user needs to display a message "I love Pakistan" for 100 times. It is time consuming to perform this task using sequence control structure. However, a single loop statement can be used to display the message for 100 times.

**Q. Explain if statement in Python with the help of example.**

**if Statement**

If statement is the simplest form of conditional statement. It is used to execute or skip a statement or set of statements based on a condition. If the condition is **true**, the statement or set of statements after if statement is executed. If the condition is **false**, the statement or set of statements after if statement is not executed.

**Syntax**

The syntax of **if** statement is as follows:

```
if condition:
    statement
```

| | |
|---|---|
| **if** | It is the keyword that represents the **if** statement. |
| **condition** | It indicates the condition that will be checked. |
| **statement** | It indicates the statement that will be executed if the condition is **true**. It must be written with an indentation by pressing **tab** key before the statement. All statements written with indentation become part of if statement and are executed when the condition is **true**. |

**Example**

```
n = 12
if (n % 2 == 0):
    print("The value of n is ", n)
    print("n is an even value.")
```

**Output:**
The value of n is 12
n is an even value.

**Explanation**

The above example checks the value of the variable **n**. It divides **n** by 2 using % operator to get the remainder. The value of **n** is 12 so the condition **n%2==0** is **true**. The two statements after **if** condition are given with **tab** indentation so the program executes both statements.

**Q. Explain if-else statement in Python with the help of example.**

**if-else Statement**

**If-else** statement is another type of **if** statement. It is used for making two-way decisions. It executes one block of statement(s) when the condition is **true** and the other when it is **false**. In any situation, one block is executed and the other is skipped.

**Syntax**

The syntax of **if-else** statement is as follows:

```
if condition:
    statement
else:
    statement
```

| | |
|---|---|
| **if** | It is the keyword that represents the **if** statement. |
| **condition** | It indicates the condition that will be checked. |
| **statement** | It indicates the statement to be executed. The statement after the condition is executed if the condition is **true**. The statement after **else:** is executed if the condition is **false**. The statement must be written with an indentation by pressing **tab** key. |
| **else** | It is the keyword that represents another part of **if** statement. |



**Example**

```
n = int(input("Enter a number: "))
if(n%2 == 0):
    print(n, " is even.")
else:
    print(n, " is odd.");
```

**Output:**
Enter a number: 10
10 is even.

**Explanation**

The above example inputs a number from the user. It checks if it is even or odd using the condition **n%2==0**. The result of the condition will be **true** if the value of **n** is even and the statement **print(n, " is even.")** will execute. The result of the condition will be **false** if the value of **n** is odd and the statement **print(n, " is odd.")** will execute.

**Q. What is indentation? Give an example.**

**Indentation**

**Indentation** refers to the spaces or tabs used at the beginning of lines to show which lines of code belong together. It indicates which statements are part of a block of code such as the conditional statement, body of a loop or function.

**Example**

```
marks = eval(input("Enter marks. "))
If marks > 40:
    print("Pass")
    print("Well done!")
else:
    print("Fail")
    print("Good luck next time")
```

The indentation in the third and fourth line indicates that both statements are related to the **if** condition. They will be executed if the condition is **true**. Similarly, the Indentation in the last two lines indicates that these statements are related to the **else** statement. They will be executed if the condition is **false**.

**PROGRAM 2.16 Write a program that reads marks and prints the message "You have passed." If the marks are greater than or equal to 33.**

```
marks = int(input("Enter your marks: "))
If(marks >= 33):
    print("You have passed.")
```

**Output:**
Enter your marks: 50
You have passed.

**PROGRAM 2.17 Write a program that inputs two numbers and finds whether both are equal.**

```
a = int(input("Enter a number: "))
b = int(input("Enter a number: "))
if(a == b):
    print("Both numbers are equal.")
```

**Output:**
Enter a number: 15
Enter a number: 15
Both numbers are equal.

**PROGRAM 2.18 Write a program that inputs three numbers and prints the largest one.**

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
```

```
c = int(input("Enter third number: "))
max = a
if(b > max):
    max = b
if(c > max):
    max = c
print("Largest number is ", max)
```

**Output:**
Enter first number: 15
Enter second number: 31
Enter third number: 9
Largest number is 31

**PROGRAM 2.19 Write a program that inputs two numbers and displays the greater number.**

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
if(a > b):
    print("The greater number is ", a)
else:
    print("The greater number is ", b)
```

**Output:**
Enter first number: 5
Enter second number: 10
The greater number is 10

**PROGRAM 2.20 Write a program that reads a number and print its square if the number is greater than 10 otherwise prints it cube.**

```
n = int(input("Enter a number: "))
if(n > 10):
    print("Square of ", n, " is ", n*n)
else:
    print("Cube of ", n, " is ", n*n*n)
```

**Output:**
Enter a number: 5
Cube of 5 is 125

**PROGRAM 2.21 Write a program that inputs a character and displays whether it is a vowel or not.**

```
ch = input("Enter any character: ")
if(ch=='A' or ch=='a' or ch=='E' or ch=='e' or ch=='I' or ch=='i' or ch=='O' or ch=='o' or ch=='U' or
ch=='u'):
    print("You entered a vowel: ", ch)
else:
    print("You did not enter a vowel: ", ch)
```

**Output:**
Enter any character: A
Your entered a vowel: A

## Q. Describe shorthand if-else with example.

### Shorthand if-else Statement

Python also allows a shorthand **if-else** statement that can be written in a single line. This is useful for writing simple conditional expressions easily.

### Syntax

The syntax of shorthand if-else is as follows:

```
value_if_true if condition else value_if_false
```

**condition** It is the expression to be evaluated.
**value_if_true** It is the value that is returned if the condition is True.
**value_if_false** It is the value that is returned if the condition is False.

### Example

```
n = int(input("Enter a number: "))
print("Even" if n % 2 == 0 else "Odd")
```

The example inputs a number in **n** and evaluates the condition **if n % 2 == 0.** It displays "Even" if the condition is **true** and "Odd" if the condition is **false**.



## Q. Explain if-elif-else statement in Python with the help of example.

### if-elif-else Statement

**If-elif-else** statement can be used to choose one block of statements from many blocks of statements. It is used when there are many options and only one block of statements should be executed on the basis of a condition.

### Syntax

The syntax of **if-elif-else** is as follows:

```
if condition:
   statement
elif condition:
   statement
else:
   statement
```

**if** It is the keyword that represents the **if** statement.
**condition** It indicates the condition that will be checked.
**statement** It indicates the statement to be executed. The statement after the condition is executed if the relevant condition is **true**. The statement after **else:** is executed if all preceding conditions is **false**. The statement must be written with an indentation by pressing **tab** key.
**elif** It is the keyword that represents another part of **if** statement.
**else** It is the keyword that represents the last part of **if** statement.

### Working of if-elif-else

The test conditions in **if-elif-else** statement are executed in a sequence until a **true** condition is found. If a condition is **true**, the statement following the condition is executed. The remaining blocks are skipped. If a condition is **false**, the statement following the condition is **skipped**. The statement after **else** is executed if all preceding conditions are **false**.

**PROGRAM 2.22 Write a program that inputs weather condition and displays a message to the user based on the weather condition.**

```
weather = input("Weather is sunny or rainy? ")
if weather == "sunny":
  print("Wear sunglasses")
elif weather == "rainy":
  print("Take an umbrella")
else:
  print("Enjoy your day!")
```

**Output:**
Weather is sunny or rainy? sunny
Wear sunglasses

**PROGRAM 2.23 Write a program to input a number and prints the message whether it is zero, odd or even.**

```
n = int(input("Enter a number "))
if(n%2 == 0):
  print("The number is even.")
elif(n%2 == 1):
  print("The number is odd.")
else:
  print("The number is zero.")
```

**Output:**
Enter a number: 21
The number is odd.

**PROGRAM 2.24** **Write a program that inputs marks and prints grade based on the obtained marks according to given scheme:**

| Marks | >= 80 <= 100 | >= 70 <= 79 | >= 60 <= 69 | >= 50 <= 59 | >= 0 <= 49 |
|---|---|---|---|---|---|
| Grade | A | B | C | D | F |

```
marks = int(input("Enter your marks: "))
if(marks >= 80):
    print("Your grade is A")
elif(marks >= 70):
    print("Your grade is B")
elif(marks >= 60):
    print("Your grade is C")
elif(marks >= 50):
    print("Your grade is D")
else:
    print("Your grade is F")
```

**Output:**
Enter your marks: 74
Your grade is B

**PROGRAM 2.25** **Write a program to calculate Body Mass Index (BMI). Ask the user for their weight and height, then compute and display their BMI and classification.**

```
w = float(input("Enter weight in kg: "))
h = float(input("Enter height in feet: "))
m = h * 0.3048     # convert height to meters
bmi = round(w / (m**2), 2)
print("Your BMI is ", bmi)
if bmi < 18.5:
   print("Classification: Underweight")
elif 18.5 <= bmi < 24.9:
   print("Classification: Normal weight")
elif 25 <= bmi < 29.9:
   print("Classification: Overweight")
else:
   print("Classification: Obese")
```

**Output:**
Enter weight in kg: 82
Enter height in feet: 5.7
Your BMI is 27.17
Classification: Overweight

## Q. What is loop? What is the purpose of loops in programs?

### Loop

The control structure that executes a statement or set of statements repeatedly is called **loop**. Loops are also known as iteration or repetition structure. Python provides two main loop structures known as **for** loop and **while** loop.

### Purpose of Loops

Loops are used in programs for two purposes:

- The loops are used to execute a statement or number of statements for a specified number of times. For example, the programmer can display a message on the screen for 10 times.
- The loops are also used to get a sequence of values. For example, a loop can be used to display a sequence of numbers from 1 to 10.

## Q. Discuss "while" loop in Python. Explain it with an example.

### The while Loop

The **while** loop executes one or more statements as long as the given condition is **true**. It checks the condition before each iteration and stops when the condition is **false**. It is useful where the number of iterations is not known in advance.



### Syntax

The syntax of this loop is as follows:

```
while condition:
   statement
```

**while** It is the keyword that indicates the **while** loop.

**condition** It indicates the condition that can be **true** or **false**. The loop is executed as long as the condition is **true**.

**statement** It indicates the statement or set of statements that will be executed by **while** loop.

### Example

The following example displays counting from 1 to 5 using **while** loop.

```
n = 1
while n <= 3:
   print(n)
   n = n + 1
```

**Output:**
1
2
3

### Explanation

The above example stores 1 in the variable **n**. The loop works as follows:

- In first iteration, the value of **n** is 1 so the condition **n <= 3** is **true**. The statement **print(n)** displays the value of **n** and statement **n = n + 1** changes its value to 2.
- In second iteration, **n** is 2 so the condition **n <= 3** is **true**. The statement **print(n)** displays the value of **n** and the statement **n = n + 1** changes its value to 3.
- In third iteration, **n** is 3 so the condition **n <= 3** is **true**. The statement **print(n)** displays the value of **n** and statement **n = n + 1** changes its value to 4.
- In fourth iteration, the value of **n** is 4 so the condition is **false** and the loop exits.

**PROGRAM 2.26** **Write a program that displays back-counting from 10 to 1 using while loop.**

```
c = 10
while c >= 1:
   print(c, end=" ")
   c = c - 1
```

**Output:**
10 9 8 7 6 5 4 3 2 1

**PROGRAM 2.27** **Write a program that displays first five numbers with their squares using while loop.**

```
n = 1
while n <= 5:
    print(n, n * n)
    n += 1
```

**Output:**
1 1
2 4
3 9
4 16
5 25

**PROGRAM 2.28** **Write a program that displays first five numbers and their sum using while loop.**

```
c = 1
sum = 0
while c <= 5:
    print(c)
    sum = sum + c
    c = c + 1
print("Sum is", sum)
```

**Output:**
1
2
3
4
5
Sum is: 15

**PROGRAM 2.29** Write a program that inputs a number from the user and displays a table of that number using while loop.

```
n = int(input("Enter a number: "))
c = 1
while c <= 10:
    print(n, "*", c, "=", n * c)
    c += 1
```

**Output:**
```
Enter a number: 3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
...
3 * 10 = 30
```

**PROGRAM 2.30** Write a Python program using a while loop that prints all the odd numbers between 1 and 20. Also, count and print the total number of odd numbers.

```
c = 1
odd = 0
while c <= 20:
    if c % 2 != 0:
        print(c)
        odd = odd + 1
    c += 1
print("Total odd numbers = ", odd)
```

**Output:**
```
1
3
5
7
9
11
13
15
17
19
Total odd numbers = 10
```

**PROGRAM 2.31** Write a program that inputs a number from the user and displays the factorial of that number using while loop.

```
n = int(input("Enter a number: "))
c = 1
f = 1
while c <= n:
    f = f * c
    c = c + 1
print("Factorial of", n, "is", f)
```

**Output:**
```
Enter a number: 3
Factorial of 3 is 6
```

**PROGRAM 2.32** Write a Python program that prints even and counts the odd numbers from 1 to 20 using a while loop.

```
n = 1
odd = 0

while n <= 20:
    if n % 2 == 0:
        print("Even number:", n)
    else:
        odd += 1
    n += 1
print("Total odd numbers from 1 to 20:", odd)
```

**Output:**
```
Even number: 2
Even number: 4
Even number: 6
Even number: 8
Even number: 10
Even number: 12
Even number: 14
Even number: 16
Even number: 18
Even number: 20
Total odd numbers from 1 to 20: 10
```



### Q. Discuss "for" loop in Python. Explain it with an example.

#### The for Loop

The **for** loop executes one or more statements for a specified number of times. This loop is also called **counter** loop. It is the flexible and most frequently used loop.

#### Syntax

The syntax of this loop is as follows:

```
for var in (value1, value2, value3...):
    statement
```

- **for** It is the keyword that indicates the **for** loop.
- **var** It indicates the name of a variable used with **for** loop.
- **in** It indicates the keyword before the list of values.
- **values** The number of values indicate the number of times **for** loop will execute. For example, the loop will execute three times if there are three values in the parenthesis.
- **statement** It indicates the statement or set of statements that will be executed by the **for** loop.

#### Example

The following example displays counting from 1 to 5 using **for** loop.

```
for n in (1, 2, 3, 4, 5):
    print(n)
```

**Output:**
```
1
2
3
4
5
```

#### Explanation

The above example uses the variable **n** with **for** loop. There are five values given in parenthesis so the loop executes five times. It executes the statement **print(n)** against each value in the parenthesis. It displays 1 in the first iteration, 2 in the second iteration and so on.

### Q. Describe the use of range() function? Explain with examples.

#### The range() Function

The **range()** function returns a sequence of numbers in a given range. The most common use of this function is to iterate a sequence of numbers using loops.

The syntax of using **range()** function is as follows:

```
range(start, stop, step)
```

- **start** It indicates the starting value in the sequence. It is optional. The value starts from 0 if it is not given.
- **stop** It indicates the ending value of the sequence. This value is not included in the sequence.
- **step** It indicates the value by which the sequence is incremented or decremented. It is optional. Its default value is 1.

**Example 1** The following **range()** function will return the values from 0 to 4. It only uses the **stop** value so starting value will be 0 by default. The difference between each number in the sequence will be 1 by default.

range(5)

**Example 2** The following **range()** function will return the values from 1 to 4. It uses **start** value of 1 and **stop** value of 5. The difference between each number in the sequence will be 1 by default.

range(1, 5)

**Example 3** The following **range()** function will return the values 1, 3, 5, 7, 9. It uses **start** value of 1, **stop** value of 10 and **step** value of 2.

range(1, 10, 2)

### Q. Describe the use of range() function with "for" loop in Python.

#### The range() Function with for Loop

The **range()** function can be used with **for** loop to give a range of values. This function returns a sequence of numbers. The **for** loop will execute according to the number of values returned by **range()** function.

#### Syntax

The syntax of for loop with **range()** function is as follows:,

```
for var in range(n):
    statement;
```

**for** It is the keyword that indicates the **for** loop.

**var** It indicates the name of a variable used with **for** loop. It is used to store each value in the loop sequence.

**In** It indicates the keyword before the list of values.

**range** It indicates the name of the function.

**n** It indicates the number up to which the function will generate values.

**statement** It indicates the statement or set of statement that will be executed by the **for** loop.

**Example 1** The following example displays counting from 0 to 4:

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

**Example 2** The following example displays even numbers from 2 to 10:

```
for n in range(2, 11, 2):
    print(n)
```

**Example 3** The following example displays odd numbers from 1 to 9:

```
for n in range(1, 10, 2):
    print(n)
```

**PROGRAM 2.33** **Write a program that displays "Pakistan" for five times using for loop.**

```
for n in range(5):
    print(n, ". Pakistan")
```

**Output:**
0 . Pakistan
1 . Pakistan
2 . Pakistan
3 . Pakistan
4 . Pakistan



**PROGRAM 2.34** **Write a for loop using range() to print the even numbers from 2 to 10.**

```
for n in range(2, 11, 2):
    print(n, end=" ")
```

**Output:**
2 4 6 8 10

**PROGRAM 2.35** **Write a program that displays back-counting from 10 to 1 loop.**

```
for i in range(10, 0, -1):
    print(i, end=" ")
```

**Output:**
10 9 8 7 6 5 4 3 2 1

**PROGRAM 2.36** **Write a program that prints the first 10 multiples of 3 using a for loop and the range() function.**

```
for i in range(3, 31, 3):
    print(i, end=" ")
```

**Output:**
3 6 9 12 15 18 21 24 27 30

**PROGRAM 2.37** **Write a program that prints the sum of first ten positive numbers.**

```
sum = 0
for i in range(10):
    sum = sum+(i+1)
print("Sum of first ten positive number is ", sum)
```

**Output:**
Sum of first ten positive numbers is 55

**PROGRAM 2.38** **Write a program that inputs a number from the user and displays the factorial of that number using for loop.**

```
f = 1
n = int(input("Enter a number: "))
for c in range(1, n+1):
    f = f * c
print("Factorial of ", n, " is ", f)
```

**Output:**
Enter a number: 5
Factorial of 5 is 120

**PROGRAM 2.39** **Write a program that inputs a number from the user and displays a table of that number using for loop from 1 to 10.**

```
t = int(input("Enter number for table: "))
for c in range(1, 11):
    print(t, "*", c, "=", t*c)
```

**Output:**
Enter number for table: 2
2 * 1 = 2
2 * 2 = 4
...
2 * 9 = 18
2 * 10 = 20

### Q. Describe module and function in Python? Discuss the advantages of functions.

#### Module

A **module** in Python is a file that contains Python definitions and statements. A module can define functions, classes and variables. It can also include executable code. The related code can be arranged into a module that makes it easier to understand and use. A module file is saved with **.py** extension and can be imported into other programs using **import** statement

### Function

A **function** is a block of code that performs a specific task. The statements written in a function are executed when it is called by its name. A function can be called many times to perform the same task again and again. A program can have many functions. Each function in the program must have a unique name.

### Advantages of using Functions

Some advantages of using functions in programs are as follows:

**1. Easy to Understand**

A program can be divided into small blocks using functions. It is easier to write small functions instead of writing one long program. It makes the code clear and easy to understand.

**2. Reusability**

The code in function is written once but can be reused many times. It allows the user to execute the same instructions multiple times without writing it again.

**3. Easy to Test**

Each function is written as an independent block. It is easier to test and debug it. Any error in the function can be identified and corrected easily.

**4. Easier to Modify**

Functions are easier to modify than long programs. Each function contains independent code. A change in the function does not affect other parts of program.

**5. Less Programming Time**

Different functions in a program are written independently. Many programmers can work on different functions at the same time. It takes less time to complete the program.

**Q. Describe the process of defining a function in Python with example.**

### Defining a Function

Each user-defined function must be defined before it is used in the program.

**Syntax**

The syntax of defining a user-defined function is as follows:

```
def fname (param):
  statement
```

| | |
|---|---|
| **def** | It indicates the keyword that is used to define a function in Python. |
| **fname** | It indicates the name of the function. Each function should have a unique name. A function is called by its name. |
| **param** | These are the function parameters. They are given in parentheses. If there are many parameters, these are separated by commas. The empty parentheses are used if there is no parameter. |
| **statement** | It indicates the statement or set of statements that are executed when the function is called. |



**Example**

The following example defines a function **show()** that displays a message:

```
def show():
  print("Hello world")
```

The above example defines a function **show()** that has one statement in it.

**Q. What is function call? Give an example.**

### Function Call

**Function call** is the statement that invokes a function. A function is executed when it is called by its name. The function name is followed by necessary arguments in parentheses. If there are many parameters, these are separated by commas. The empty parentheses are used if no parameter is required.

The following steps take place when a function is called:

1. The control moves to the function that is called.
2. All statements in the function body are executed.
3. The control returns back to the calling function.

**Example**

The following example displays a message "Hello world" on screen using function.

```
def show():
  print("Hello world")
show()
show()
show()
```

**Output:**

```
Hello world
Hello world
Hello world
```

**Explanation**

The above program defines a function **show()** that displays a message. The program calls the function **show()** three times. Each time the control moves to the function and executes the single statement in it.

**Q. What are parameters? How are parameters passed to the functions?**

### Parameters

A **parameter** is the data that is passed into a function when the function is called. The function can use its parameters in calculations or other operations.

**Example**

```
def show(msg):
  print(msg)

show("Python")
show("Programming")
show("Language")
```

**Output:**

```
Python
Programming
Language
```

**Explanation**

The above program defines a function **show()** that displays the value of the parameter **msg**. The program calls the function **show()** three times with three different values. The first function call passes "Python" as parameter. The second function call passes "Programming" as parameter. The third function call passes "Language" as parameter. The function displays the value of parameter each time it is called.

**PROGRAM 2.40** **Write a program to take a number as input from the user and display a table using a function.**

```
def table(t):
  for c in range(1, 11):
    print(t, "*", c, "=", t*c)
n = int(input("Enter number for table: "))
table(n)
```

**Output:**
Enter number for table: 5
5 * 1 = 5
5 * 2 = 10
...
5 * 9 = 45

## Q. How is a value returned from the function? Explain with example.

### Returning Value from Function

A function can return a single value. The keyword **return** is used to return the value back to the calling function. When **return** statement is executed in the function, the control moves back to the calling function along with the returned value. It is a good practice to write the returned value in parenthesis "( )" after **return** keyword.

### Syntax

The syntax for returning a value is as follows:

```
return expression
```

**expression** It can be a variable, constant or an arithmetic expression whose value is returned to the calling function.

### Example

The following example finds the product of two numbers using user-defined function.

```
def product(x, y):
  z = a * b
  return (z)
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = product(a, b)
print(a, "*", b, "=", c)
```

**Output:**
Enter first number: 5
Enter second number: 10
5 * 10 = 50

### Explanation

The above program defines a function **product()** that accepts two parameters **x** and **y**. It calculates the product of two values and stores it in variable **z**. The value of **z** is returned using **return** statement. The program inputs two numbers in **a** and **b**. It calls **product()** function and passes the values of **a** and **b** as parameters. The returned value is stored in **c** and the program displays the value of **c**.

**PROGRAM 2.41** **Write a program that inputs a number and passes it to a function that displays its factorial.**

```
def factorial(n):
  fact = 1
  i = 1
  while(i <= n):
    fact = fact * i
    i = i + 1
  print("Factorial of ", n, " is ", fact)
num = int(input("Enter a number: "))
factorial(num)
```

**Output:**
Enter a number: 5
Factorial of 5 is 120



**PROGRAM 2.42** **Write a program having two functions names area and perimeter to find the area and perimeter of a square.**

```
def area(s):
    a = s * s;
    print("Area = ", a)
def perimeter(s):
    p = s * 4
    print("Perimeter = ", p)
side = float(input("Enter the length of side: "))
area(side)
perimeter(side)
```

**Output:**
Enter the length of side: 2.5
Area = 6.25
Perimeter = 10.0

## Q. What are default parameters? Explain with example.

### Default Parameters

The default parameters refer to the default values given to the parameters. The default values are used if the function is called without passing the values for the parameters. The syntax of default parameters is as follows:

```
def fname (param=dvalue)
  statement
```

**def** It indicates the keyword that is used to define a function in Python.
**fname** It indicates the name of the function.
**param** It indicates the function parameter.
**dvalue** It indicates the default value of the parameter.

### Example

```
def show(str="Hello"):
    print(str)
show("Python")
show("Programming")
show()
```

**Output:**
Python
Programming
Hello

The above program defines a function show() with a default value "Hello". The function is called three times. In the first function call, the value "Python" is given as parameter that is shown on the screen. In the second function call, the value "Programming" is given as parameter that is shown on the screen. In the third function call, no value is given as parameter so the default value "Hello" is shown on the screen.

## Q. What are libraries in Python? How are these libraries imported and used?

### Libraries

A **library** is a collection of related modules that contain pre-written code. The code in a library can be used in different programs. Libraries simplify programming by providing ready-to-use functions for common tasks such as mathematical calculations and random number generation etc.

### Importing and Using a Library

A library must be imported into a program to use. The syntax of importing a library in Python is as follows:

```
import lib
```

**import** It is the keyword that is used to import a library in Python.
**lib** It indicates the name of the library to be imported.

**Examples**

Some examples of importing and using libraries are as follows:

**Example 1** The following example imports **random** library to generate random numbers.

```
import random
n = random.randint(1, 10)
print("The random number is:", n)
```

**Example 2** The following example imports **datetime** library to show current date and time.

```
import datetime
t = datetime.datetime.now()
print("Current date and time:", t)
```

**Example 3** The following example imports **statistics** library to calculate the mean of a list of numbers.

```
import statistics
d = [21, 43, 60, 87, 19, 49, 55]
m = statistics.mean(data)
print("The mean value is:", m)
```

### Q. What is package in Python? Give an example of using it in Python program.

**Package**

A **package** is a way to organize and structure code by grouping related modules into directories. A package refers to a folder that contains one or more Python modules and a special file name **__init__.py**. This organization helps to manage and reuse code effectively. For example, a package named **ecommerce** can be created for e-commerce platform. The package may contain the modules such as **products.py**, **customers.py** and **orders.py** etc.

**Example**

Suppose a function **List_Products()** is available in **products.py** module of the **ecommerce** package as follows:

```
def List_products ():
    return ["Laptop", "Mobile", "Tablet"]
```

The above function can be used in a Python program by importing **product** module from **ecommerce** package as follows:

```
from ecommerce import products
available_products = products.list_products()
print(available_products)
```

### Q. What are built-in data structures in Python?

**Built-in Data Structures**

**Data structure** is a container to store the collection of data items in a specific layout. Python provides several built-in data structures that help in organizing and managing data efficiently. The most commonly used data structures are lists, tuples and dictionaries. Each data structure offers unique features to handle various types of data. Many common operations can be performed on these data structures such as insertion, deletion and searching.



### Q. What is a list in Python? Describe its purpose.

**List**

A **list** is a data structure in Python that can store multiple values. The list is **mutable** that means its elements can be created, accessed, modified or removed easily. Each item stored in a list is called an **element**. Lists can store elements of different data types, including integers, floats, strings etc. Each element in the list can be accessed with reference to its position in the list. This position is called **index**. Each element in the list has a unique index. The index of first element is 0 and the index of last element is length -1. The value of the index is written in brackets along with the name of the list.

The following example shows a list **Num** with three elements:

| Index of each element → | 0 | 1 | 2 |
|---|---|---|---|
| | | | |

**Figure:** A list **Num** with three elements

**Purpose of Lists**

Lists are used to store and organize a large collection of data items easily. The basic variables can only store one value at a time. For example, a variable **marks** can store the marks of one student. The user needs to declare 50 variables to store the marks of 50 students that is very time-consuming process. Lists become very useful in these situations. A list of 50 elements can be used to store the marks of 50 students.

### Q. Describe the process of creating a list with an example.

**Creating a List**

A list is created by giving the name of list and the initial values for the list elements. The values are separated with commas and enclosed within square brackets [ ].

The syntax of creating a list in Python is as follows:

```
list = [value 1, value 2, ... value N]
```

**list** It indicates the name of the list.
**value** It indicates the values of the list. Each value is separated by comma.

**Example**

An example of list is as follows:

```
Marks = [91, 52, 75, 66, 87]
```

The above example creates a list **Marks** with five elements. The index of first element is 0 and the index of last element is 4.

| | | | | | |
|---|---|---|---|---|---|
| Index of each element → | 0 | 1 | 2 | 3 | 4 |
| Value of each element → | 91 | 52 | 75 | 66 | 87 |

**Figure:** A list **Marks** with five elements

### Q. How an individual element of a list is accessed?

Each individual element of a list is accessed by specifying the following:

- Name of the list
- Index of the element

### Syntax

The syntax of accessing an individual element of a list is as follows:

list_name[index];

**list_name** It indicates the name of the list.
**index** It indicates the index of the element to be accessed.

### Example

```
fav = ["Pakistan", "Mango", "Potato"]
print("My favorite country: ", fav[0])
print("My favorite fruit: ", fav[1])
print("My favorite vegetable: ", fav[2])
```

The above example defines a list **fav** with three elements. It initially stores the names of favorite things. It displays the list elements individually with a message using **print** statement.

| Index of each element → | 0 | 1 | 2 |
|---|---|---|---|
| Value of each element → | Pakistan | Mango | Potato |

**Figure:** A list **fav** with three elements

**PROGRAM 2.43** **Write a program in Python that creates a list city. It stores capital city of the country and provinces in the list and displays the list.**

1. Start **IDLE** and open a new file.
2. Type the following code:

```
p3_9.py - C:/Users/tariq/AppData/Local/Programs/Python/Python312/p3_9.py (3.12.3)
File Edit Format Run Options Window Help
city = ["Islamabad", "Lahore", "Karachi", "Peshawar", "Quetta"]
print("Pakistan's capital city: ", city[0])
print("Punjab's capital city: ", city[1])
print("Sindh's capital city: ", city[2])
print("KPK's capital city: ", city[3])
print("Balochistan's capital city: ", city[4])
Ln: 7 Col: 0
```

3. Save the file as "p3_9.py".
4. Click **Run > Run module** or press **F5**. The output will appear as follows:

**PROGRAM 2.44** **Write a program that stores different types of values in a list and displays these values.**

```
my_list = [42, "Python", 3.14]
print("First item:", my_list[0])
print("Second item:", my_list[1])
print("Third item:", my_list[2])
```

**Output:**
First item: 42
Second item: Python
Third item: 3.14

## Q. How the individual elements of a list can be accessed using loop? Give an example.

Each individual element of a list can be accessed using loops in Python. The counter variable of the loop can be used as the index to access a different element in each iteration of loop. This is an easier and quicker way of accessing all elements of a list.



### Example

```
fruit = ["apple", "banana", "orange", "grape", "mango"]
for i in range(5):
    print(fruit[i])
```

The above example defines a list **fruit** with five elements with initial values. It uses a **for** loop to access all elements of the list individually and display its value. The value of the counter variable **i** is 0 in first iteration. The statement **print(fruit[i])** displays the value of the first element that is "apple". Similarly, the value of counter variable is 1 in second iteration so the statement **print(fruit[i])** displays the value of second element that is "banana" and so on.

| Index of each element → | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Value of each element → | apple | banana | orange | grape | mango |

**Figure:** A list **fruit** with five elements

## Q. Discuss different methods for the lists with examples.

Python provides different methods to perform different operations on the lists. Some important methods for the lists are as follows:

### 1. The append() Method

The **append()** method is used to add an item to the end of the list. The syntax of this method is as follows:

list_name.append(val);

**list_name** It indicates the name of the list.
**val** It indicates the value that will be appended to the list.

### Example

```
Num = [10, 20, 30, 40, 50]
Num.append(100)
print(Num)
```

**Output:** [10, 20, 30, 40, 50, 100]

### 2. The remove() Method

The **remove()** method removes the first occurrence of an item from the list. The syntax of this method is as follows:

list_name.remove(val);

**list_name** It indicates the name of the list.
**val** It indicates the value that will be removed from the list.

### Example

```
Num = [10, 20, 30, 40, 50]
Num.remove(40)
print(Num)
```

**Output:** [10, 20, 30, 50]

### 3. The sort() Method

The **sort()** method is used to sort the elements of a list in ascending order. The syntax of this method is as follows:

list_name.sort();

**list_name** It indicates the name of the list.

**Example**

```
team = ["Faheem", "Zaid", "Daood", "Wasim", "Ali"]
team.sort()
print(team)
```

**Output:** ['Ali', 'Daood', 'Faheem', 'Wasim', 'Zaid']

### 4. The reverse() Method

The **reverse()** method is used to reverse the order of elements in a list. The syntax of this method is as follows:

list_name.reverse();

**list_name** It indicates the name of the list.

**Example**

```
team = ["Faheem", "Zaid", "Daood", "Wasim", "Ali"]
team.reverse()
print(team)
```

**Output:** ['Ali', 'Wasim', 'Daood', 'Zaid', 'Faheem']

### 5. The insert() Method

The **insert()** method is used to add an element at a specific index in a list. If there is already an element at that index, it will be shifted to the right to make space for the new element. The syntax of this method is as follows:

list_name.insert(index, element);

**list_name** It indicates the name of the list.
**index** It indicates the index where the element will be inserted in the list.
**element** It indicates the value to be inserted in the list.

**Example**

```
country = ["Turkey", "Iran", "Oman"]
country.insert(0, "Pakistan")
print(country)
```

**Output:** ['Pakistan', 'Turkey', 'Iran', 'Oman']

### 6. The index() Method

The **index()** method returns the index of the given value in the list. If the value occurs more than once in the list, it will return the index of the first occurrence.

The syntax of this method is as follows:

list_name.index(val);

**list_name** It indicates the name of the list.
**val** It indicates the value whose index will be returned.

**Example**

```
Num = [10, 12, 14, 16, 18, 10]
print("10 occurs at index", Num.index(10))
```

**Output:** 10 occurs at index 0

## Q. Describe the len() function in Python with example.

### The len() Function

The **len()** function returns the total number of elements of the list or tuple. The syntax of this function is as follows:



len(list_name);

**len** It indicates the name of the function.
**list_name** It indicates the name of the list.

**Example**

```
num = [10, 20, 30, 40, 50]
print("Total elements in the list are", len(num))
```

**Output:** Total elements in the list are 5

**PROGRAM 2.45** **Write a program that inputs five integers from the user and stores them in a list. It then displays all values in the list.**

```
num = [ ]
for i in range(5):
    n = int(input("Enter an integer: "))
    num.append(n)
print("The values in list are as follows: ")
print(num)
```

**Output:**
Enter an integer: 10
Enter an integer: 20
Enter an integer: 30
Enter an integer: 40
Enter an integer: 50
The values in list are as follows:

**PROGRAM 2.46** **Write a program that inputs five numbers in a list and displays them in actual and reverse order.**

```
num = [ ]
for i in range(5):
    n = int(input("Enter an integer: "))
    num.append(n)
print("The list in actual order: ")
print(num)
print("The list in reverse order: ")
num.reverse()
print(num)
```

**Output:**
Enter an integer: 35
Enter an integer: 47
Enter an integer: 83
Enter an integer: 66
Enter an integer: 72
The list in actual order:
[35 47 83 66 72]
The list in reverse order:

## Q. Describe the slicing and concatenation operation on list.

The lists or tuples in Python support various operations such as slicing and concatenation as follows:

### 1. Slicing Operation

The slicing operation is used to extract a portion of the list. This operation does not affect the original list but stores the portion in the new list.

The syntax for list slicing in Python is as follows:

new_list = original_list[start: end: step]

**new_list** It is the name of the list created to store the sliced elements.
**original_list** It is the original list that is being sliced.
**start** It is the index of the element at which the slicing starts.
**end** It is the index of the element before which the slicing ends.
**step** It is the increment value for selecting elements. It is optional.

**Example**

```
num = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

```
num2 = num[1:5]
print(num2)
```

**Output:** [20, 30, 40, 50]

The above example creates a list **num** with ten numbers. It slices the list from index 1 to 4. The statement **num[1:5]** indicates that slicing will start from index 1 and end before index 5. The sliced potion is stored in the list **num2** that is displayed on the screen.

### 2. Concatenation Operation

The concatenation operation is used to combine two or more lists into a single list. This operation can be performed by using **concatenation operator +**. The syntax for concatenating the lists is as follows:

```
new_list = list1 + list2
```

**new_list** It is the name of list in which two or more lists will be concatenated.
**list1** It is the first list to be concatenated.
**list2** It is the second list to be concatenated.

**Example**

```
odd = [1, 3, 5, 7, 9]
even = [2, 4, 6, 8, 10]
com = odd + even
print(com)
```

**Output:** [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]

The above example creates two lists **odd** and **even.** It concatenates both lists into a third list **com.** The new list is then displayed on the screen.

### Q. Describe indexing and slicing with negative index with example.

Negative index can be used in Python to access the elements from the end of a list. For example, -1 refers to last element, -2 refers to second-last element and so on.

**Example**

```
fruits = ["Apple", "Banana", "Cherry", "Date", "Elderberry"]
# Indexing
print("First fruit:", fruits[0])      # Positive index
print("Last fruit:", fruits[-1])      # Negative index
# Slicing with positive indices
print("Fruits from index 1 to 3:", fruits[1:4])
# Slicing with negative indices
print("Fruits from index -4 to -2:", fruits[-4:-1])
Output
First fruit: Apple
Last fruit: Elderberry
Fruits from index 1 to 3: ['Banana', 'Cherry', 'Date']
Fruits from index -4 to -2: ['Banana', 'Cherry', 'Date']
```

### Explanation

This example demonstrates:

- **Indexing** using both positive and negative values.
- **Slicing** using ranges of positive indices (1:4) and negative indices (-4:-1).
  - fruits[1:4] refers to elements from index 1 to 3 (Banana, Cherry, Date).
  - fruits[-4:-1] refers to elements from index -4 to -2 (Banana, Cherry, Date).



**PROGRAM 2.47** **Imagine you are maintaining a list of your favorite books: ["To Kill a Mockingbird", "1984", "The Great Gatsby", "Pride and Prejudice"]. Perform the following tasks using Python:**

1. **Add a new book "Moby Dick" to the list.**
2. **Replace "1984" with "Brave New World".**
3. **Remove "The Great Gatsby" from the list.**
4. **Merge this list with another list of books: ["War and Peace", "Hamlet"].**
5. **Print the final list of books.**

```
books = ["To Kill a Mockingbird", "1984", "The Great Gatsby", "Pride and Prejudice"]
books.append("Moby Dick")                    # 1. Add a new book "Moby Dick"
ind = books.index("1984")                    # 2. Replace "1984" with "Brave New World"
books[ind] = "Brave New World"
books.remove("The Great Gatsby")             # 3. Remove "The Great Gatsby"
new_books = ["War and Peace", "Hamlet"]      # 4. Merge with another list of books
books = books + new_books
print("Final list of favorite books:")       # 5. Print the final list of books
print(books)
```

**Output:**

```
Final list of favorite books:
['To Kill a Mockingbird', 'Brave New World', 'Pride and Prejudice', 'Moby Dick', 'War and Peace', 'Hamlet']
```

### Q. What is a tuple in Python? How is it defined?

### Tuple

A **tuple** is a data structure in Python that can store multiple values. The tuple is **immutable** that means its elements cannot be modified after creation. Each item stored in a tuple is called an **element.** Tuples can store elements of different data types including integers, floats, strings etc. Each element in the tuple can be accessed with reference to its position in the tuple. This position is called **Index.** Each element in the tuple has a unique index. The index of first element is 0 and the index of last element is length - 1. The value of index is written in brackets with the name of tuple.

### Defining a Tuple

A tuple is defined by giving its name and the initial values for the tuple elements. The values are separated with commas and enclosed within parenthesis ().

The syntax of defining a tuple in Python is as follows:

```
tup = (value 1, value 2, ... value N)
```

**tup** It indicates the name of the tuple.
**value** It indicates the values of the tuple separated by comma.

**Example**

An example of tuple is as follows:

```
fav = ("Pakistan", "Mango", "Potato")
print("My favorite country: ", fav[0])
print("My favorite fruit: ", fav[1])
print("My favorite vegetable: ", fav[2])
```

The above example defines a tuple **fav** with three elements. It initially stores the names of favorite things. It displays the elements individually with a message using **print** statement. Each individual element of a tuple is accessed by specifying its name and index of the element.

**Q. Describe modular programming in Python.**

### Modular Programming in Python

**Modular programming** is a technique used to divide a program into smaller, manageable and reusable parts called **modules**. This makes the code easier to read, test and maintain. Modules can be reused in different programs that reduces code duplication. It also supports teamwork as different developers can work on separate modules independently.

### The main Function

The **main** function in Python defines where the program starts. It is usually placed in a block that checks if the script is being run directly or imported as a module.

An example of main function is as follows:

```
def main ( ) :
   print("This is the main function.")
if __name__ == "__main__" :
   main()
```

### Explanation

The example defines **main()** function. The condition **if __name__ == "__main__":** ensures that **main()** function is executed only when the script is run directly. This block is not executed if the script is imported into another module. This is useful in modular projects where files are reused across different programs.

**PROGRAM 2.48** **Create a Python module calculator.py with two functions. The add (a, b) adds and returns the sum of two numbers. The subtract (a, b) function returns the difference between two numbers. Then, write a script named main.py that imports calculator module and uses these functions to perform the following:**

1. **Print the result of adding 15 and 8.**
2. **Print the result of subtracting 10 from 25.**

**Make sure to run main.py script and verify that the output is correct.**

```
# Save the following code in a separate file calculator.py
def add(a, b):
   return a + b
def subtract(a, b):
   return a - b
# Save the following code in a separate file main.py
import calculator
print(calculator.add(18, 8))
print(calculator.subtract(25, 10))
```

**Note:** The above example assumes that both files are saved in the same folder. When **main.py** is executed, it imports **calculator.py**" and uses its functions.

**Q. What is object-oriented programming in Python.**

### Object-Oriented Programming

**Object-Oriented Programming (OOP)** is a programming technique in which programs are written on the basis of classes and objects. It is a way of writing and organizing code to make it easier to manage and understand.



### Class and Objects

An **object** is a collection of data (attributes) and functions (methods). It may represent a person, thing or place in the real world. In OOP, data and all possible functions on data are grouped together.

A **class** is a template for creating objects. It defines the characteristics and functions that the objects will have. For example, a class **Person** can be used to define the characteristics such as Name, Address and Phone etc. It also defines the functions of a person such as move, talk, read etc. The class can be used to create different objects such as Ali, Usman, Abdullah etc. All objects of **Person** class will have the same characteristics and functions.

**Q. Describe the procedure of defining a class and creating object in Python.**

### Defining Class

The syntax of defining a class in Python is as follows:

```
class class_name:
    atr = val
```

| | |
|---|---|
| **class** | It indicates the keyword that is used to define a class. |
| **class_name** | It indicates the name of the class. |
| **atr** | It indicates the attribute of the class. |
| **val** | It indicates the initial value assigned to the attribute. |

### Creating Object

The syntax of creating an object in Python is as follows:

```
obj class_name()
```

| | |
|---|---|
| **obj** | It indicates the name of the object to be created. |
| **class_name** | It indicates the name of the class from which the object is created. |

### Example

```
class Test:
    n = 10

mytest = Test()
print(mytest.n)
```

### Explanation

The above example defines a class **Test** that has an attribute **n** with a value of 10. It creates an object of the class named **mytest**. The statement **print(mytest.n)** displays the value of the attribute **n** which is 10.

**Q. What is the use of _init_ method in classes. Give an example.**

### The _init_ Method

The **__init__** method is a special built-in function in Python classes. It is automatically called when an object of the class is created. This method is typically used to initialize the object's attributes and to perform other necessary operations.

### Example

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p = Person("Ali", 21)
print(p.name)
print(p.age)
```

### Explanation

The above example defines a class **Person** as follows:

- It defines **__init__** method in which the first parameter **self** refers to the object being created. The other two parameters are named **name** and **age**.
- The statement **self.name = name** assigns the value of **name** parameter to **name** attribute of the object being created.
- The statement **self.age = age** assigns the value of **age** parameter to **age** attribute of the object being created.
- The statement **p = Person("Ali", 21)** creates an object **p** of class Person with two values. Both values are passed to **__init__** method that assigns these values to **name** and **age** respectively.
- The statement **print(p.name)** displays the value of **name** attribute of **p** object.
- The statement **print(p.age)** displays the value of **age** attribute of **p** object.

**PROGRAM 2.49** **Write a program that creates a class Car. It has three properties named color, size and wheels. It defines __init__ method to initialize its properties with given values as parameters. It defines a method describe() that displays the details of the car.**

```
class Car:
  def __init__(self, color, size, wheels):
    self.color = color
    self.size = size
    self.wheels = wheels

  def describe(self):
    print(f"This car is {self.color}, {self.size} and has {self.wheels} wheels.")

mycar = Car("red", "small", 4)
yourcar = Car("blue", "large", 6)
mycar.describe()
yourcar.describe()
```

### Explanation

The above example defines a class **Car** as follows:

- It defines **__init__** method in which **self** refers to the object being created. The other three parameters are named **color, size** and **wheels**.
- The statement **self.color = color** assigns the value of **color** parameter to **color** attribute of the object.
- The statement **self.size = size** assigns the value of **size** parameter to **size** attribute of the object.
- The statement **self.wheels = wheels** assigns the value of **wheels** parameter to **wheels** attribute of the object.



- The statement **mycar = Car("red", "small", 4)** creates an object **mycar** of class **Car**. The parameter values are passed to **__init__** method that assigns these values to **color, size** and **wheels** attributes respectively.
- The statement **yourcar = Car("blue", "large", 6)** creates an object **yourcar** of class **Car**. The parameter values are passed to **__init__** method that assigns these values to **color, size** and **wheels** attribute respectively.
- The class defines another method **describe()** that displays the details of the object using a **print** statement with **f** format string. The **f** letter before double quotes display the values of object properties using { }. For example, **{self.color}** in the string will be replaced with the value of the color.

**Q. What is exception handling in Python? Discuss Try-Except block with example.**

### Exception Handling

**Exception handling** is a method to manage run-time errors that occur during program execution. It allows a program to continue running or terminate if an error occurs instead of crashing. It can also be used to shows user-friendly error messages.

Some common exceptions in Python include the following:

- **ZeroDivisionError:** It occurs when dividing a number by zero.
- **ValueError:** It occurs when a function receives an argument of the right type but inappropriate value.
- **TypeError:** It occurs when an operation is applied to an object of incorrect type.
- **FileNotFoundError:** It occurs when a file or directory is requested but does not exist

### Try-Except Blocks

The **try** block in Python is used to test a block of code for errors and the **except** block is used to handle errors. The syntax of try-except block is as follows:

```
try:
  # Code that may cause an error
except ExceptionType:
  # Code to handle the error
```

### Example

```
try:
  num = int(input("Enter a number: "))
  result = 10 / num
  print("Result:", result)
except ZeroDivisionError:
  print("You cannot divide by zero!")
```

### Explanation

In the above example:

- The **try** block contains three statements. The first statement inputs a number and stores it in **num**. The second statement divides 10 by the input number and the third statement displays the result.
- The except block handles an exception **ZeroDivisionError** that occurs if the user inputs 0. It displays an error message.

**Q. What is file handling? Describe the method of opening, reading and closing file in Python.**

### File Handling

**File handling** refers to the process of reading from and writing to files in a program. A **file** is a collection of related **records**. A record contains data about an entity such as student, employee or product. A file can be used to provide input to a program. It can also be used to store the output of a program permanently.

### Opening a File

A file should be opened to access its contents. A file can be opened using **open()** function in Python. The syntax of **open()** function is as follows:

```
with open(file_name, mode) as file:
```

**with** It is a keyword that automatically closes opened file after being used.

**file_name** It indicates the file name to be opened. The full path of the file is required if the file is not in the current folder.

**mode** It indicates the mode in which the file will be opened. Some common modes are as follows:

- **r** It opens file in read-only mode for reading.
- **w** It opens the file for write mode for writing.
- **a** It opens the file in append mode to add data at the end of file.

**file** It indicates the name of the object that will refer to the opened file.

### Reading a File

The **read()** function is used to read the contents of a file. The syntax of **read()** function is as follows:

```
var = file.read()
```

**var** It indicates the variable in which the file contents will be stored.

**file** It indicates the name of the object that will refer to the opened file.

### Closing a File

The **close()** function is used to close an opened file. It is a good practice to close the files. The file opened using **with** statement is automatically closed. The syntax of **close()** function is as follows:

```
file.close()
```

**file** It indicates the name of the object that will refer to the opened file.

### Example

```
with open("D:\\Test.txt", "r") as f:
    d = f.read()
    print(d)
```

### Explanation

In the above examples:

- The file **Test.txt** is opened using **with** statement. It closes the file after it has been used.
- The variable **f** refers to the opened file.
- The parameter **r** indicates that the file is opened in read mode.
- The statement **d = f.read()** reads file contents and stores them in variable **d**.
- The last statement prints the value of **d**.



**Q. Describe the method of writing to the file in Python.**

### Writing to Files

Python allows to write new data to the file. A file must be opened in **write** or **append** mode to write data to it. The **write()** function is used to write data to the file. This method will overwrite the existing data if the file is opened in **write** mode. If the file is opened in **append** mode, the new data is added to the end of existing data in the file. The syntax of this function is as follows:

```
file.write(data)
```

**file** It indicates the name of the object that will refer to the opened file.

**data** It indicates the data that will be written to the file.

### Example

```
with open("Test.txt", "w") as f:
  f.write("Python programming")
```

### Explanation

In the above examples:

- The file **Test.txt** is opened using **with** statement. It closes the file automatically after it has been used.
- The variable **f** refers to the opened file.
- The parameter **w** indicates that the file is opened in **write** mode.
- The last statement writes the given text to the file. If the file contains any previous data, it will be overwritten.

**Q. What is testing? Discuss different types of testing in Python.**

### Testing

**Testing** is the process of running the code with various inputs to check if it works as expected. The goal of testing is to find and fix any issues before the code is used in real-world applications. It ensures the reliability of the code.

### Types of Testing

Different types of testing in Python are as follows:

- **Unit Testing:** It is used to test the individual parts of the code such as functions or classes. The **unittest** module in Python is commonly used for unit testing.
- **Integration Testing:** It checks how different parts of the code work together.
- **Functional Testing:** It validates that the software works as expected from the user's point of view. It focuses on what the system does, not how it does it.
- **Regression testing:** It ensures that new changes or updates to the code do not break or negatively affect the existing functionality of software. It is commonly performed after bug fixes or adding new features. It ensures that the previously working parts of the application still function as expected.

**Q. What is debugging? Discuss different debugging options in Python.**

### Debugging

An error in a computer program is known as **bug**. The process of finding and removing errors in the program is called **debugging**. It involves identifying the cause of problems and making the necessary changes.

### Common Debugging Techniques

The common debugging techniques are as follows:

- **Print Statements:** The **print** statements can be used in the code to check the values of variables at different stages of the code.
- **Debugging Tools:** Python provides various debugging tools such as **Python Debugger (pdb)**. It can be used to step through the code, inspect variables and understand the flow of execution.
- **Error Messages:** The error message can be read to locate the source of the problem in the program. It helps to find and remove bugs.

---

## Exercise Solution

### Multiple-Choice Questions (MCQs)

1. **An action needed during Python installation to run from the command line easily:**
   a) Uncheck "Add Python to PATH" b) Choose a different IDE
   c) Check "Add Python to PATH" d) Install only the IDE
2. **A valid variable name in Python is:**
   a) variable1 b) 1variable
   c) variable-name d) variable name
3. **Output of following piece of code is:**
   ```
   age = 25
   print(" Age : ", age)
   ```
   a) Age: 25 b) 25 c) Age d) age
4. **The operator used for exponentiation in Python is:**
   a) * b) ** c) // d) /
5. **A loop used to iterate over a collection such as lists is:**
   a) while b) for c) do-while d) repeat
6. **The range() function:**
   a) Generates a list of numbers b) Creates a sequence of numbers
   c) Calculates the sum of numbers d) Prints a range of numbers
7. **A keyword used to define a function in Python:**
   a) define b) function c) def d) func
8. **The Output of the following code is:**
   ```
   temperature, humidity, wind_speed = 25, 60, 15
   print("Hot and humid" if temperature > 30 and humidity > 50 else
   "Warm and breezy" if temperature == 25 and wind_speed > 10 else
   "Cool and dry" if temperature < 20 and humidity < 30 else
   "Moderate")
   ```
   a) Hot b) Warm c) Cool d) Nothing
9. **The operation used to combine two lists in Python:**
   a) combine() b) concat() c) + d) merge()

**Answers**

| | | | | | |
|---|---|---|---|---|---|
| 1. c | 2. a | 3. a | 4. b | 5. b | 6. b |
| 7. c | 8. b | 9. c | | | |

### Short Questions

**Q.1. Explain the purpose of using comments in Python code**

The purpose of using comments in Python is to make the code easier to understand. They are notes that explain the purpose and logic of specific parts of the code. They are especially helpful when reviewing, updating or sharing code with others.



**Q.2. Describe the difference between integer and float data types in Python. Provide an example of each.**

An integer is a data type used to represent whole numbers without a decimal point such as 25. A float is used to represent numbers that include a decimal point such as 36.5. Integers are typically used for counting while floats are used for measurements and precise calculations. For example, age = 25 is an integer, and temperature = 36.5 is a float.

**Q.3. Define operator precedence and give an example of an expression where operator precedence affects the result.**

Operator precedence in Python refers to the order in which operators are evaluated in an expression. Operators with higher precedence are executed before those with lower precedence. For example, in the expression 10 + 5 * 2, multiplication has higher precedence than addition, so 5 * 2 is evaluated first. The result is 10 + 10 = 20.

**Q.4. How does the short-hand if-else statement differ from the regular if-else statement?**

The short-hand if-else statement in Python is a single-line expression used for simple conditions whereas the regular if-else statement spans multiple lines and is used for more complex logic. The syntax of short-hand if-else statement is: value_if_true if condition else value_if_false. For example: result = "Pass" if score >= 50 else "Fail".In contrast, a regular if-else uses block structure with indentation and can include multiple statements.

**Q.5. Explain the use of the range() function in a for loop?**

The range() function can be used with for loop to give a range of values. This function returns a sequence of numbers. The for loop will execute according to the number of values returned by range() function. The following example displays counting from 0 to 4:

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

**Q.6. Explain how default parameters work in Python functions.**

The default parameters in Python refer to the default values given to the parameters. The default values are used if the function is called without passing the values for the parameters. The default values are given in function definition using = sign such as def greet(name="Guest").

**Q.7. Explain why modular programming is useful in Python.**

Modular programming is useful in Python because it allows to break a large program into smaller and manageable parts called modules. This makes the code easier to read, test and maintain. Modules can be reused in different programs that reduces code duplication. It also supports teamwork as different developers can work on separate modules independently.

**Q.8. Explain the difference between a class and an object in Python.**

A class in Python is a template that defines the attributes and functions of its objects. It specifies the data and behavior of each object created from the class. An object is a real instance of a class that represents entities like a person or place. For example, Ali and Usman are objects of the class Person.

### Long Questions

**Q.1. Evaluate the following Python expressions**

**a. (18 / 3 + 4 ** 2) - (2 * (7 - 3)) / (9 * 7 - 4)**

= (6 + 16) - (2 * 4) / (63 - 4) # 18 / 3 = 6, 4 ** 2 = 16, 7 - 3 = 4, 9 * 7 = 63
= 22 - 8 / 59 # 2 * 4 = 8
= 22 - 0.13559322 # 8 / 59 ≈ 0.1356
= 21.86440678

**Answer: 21.8644**

b. (25 + 3 * 4 ** 2 - 6) / (2 ** 3 + 1) - 7
= (25 + 3 * 16 - 6) / (8 + 1) - 7    # 4 ** 2 = 16, 2 ** 3 = 8
= (25 + 48 - 6) / 9 - 7    # 3 * 16 = 48
= 67 / 9 - 7
= 7.4444 - 7
= 0.4444
**Answer: = 0.4444**

c. (12 + 6 * (5 - 2)) ** 2 / ((4 ** 2 - 7) + 10)
= (12 + 6 * 3) ** 2 / (16 - 7 + 10)    # 5 - 2 = 3, 4 ** 2 = 16
= (12 + 18) ** 2 / 19
= 30 ** 2 / 19
= 900 / 19
= 47.3684
**Answer: = 47.3684**

d. 45 / (2 ** 2 + 3 * 4) + 8 * (7 - 3)
= 45 / (4 + 12) + 8 * 4    # 2 ** 2 = 4, 3 * 4 = 12, 7 - 3 = 4
= 45 / 16 + 32
= 2.8125 + 32
= 34.8125
**Answer: 34.8125**

**Q.2. Translating the following Mathematical Expressions to Python Syntax**

a. 15 x (3 + $2^2$)
= 15 * (3 + 2**2)

b. 6-2x3
= 6 - 2 * 3

c. 7 + $2^2$
= 7 + 2 ** 2

**Q.3. Explain the concept of variables in Python.** (See the chapter for answer)

**Q.4. Write a Python program that takes a number as input and checks whether it is positive, negative, or zero using an if-elif-else statement.** (See the chapter for answer)

**Q.5. Write a Python program using a while loop that prints all the odd numbers between 1 and Also, count and print the total number of odd numbers.** (See the chapter for answer)

## SHORT QUESTIONS

**Q.1. Define computer program and programmer.**

A set of well-defined instructions that can be executed by a computer to solve a problem is called computer program. A computer program is written in a programming language. A person who knows how to write computer program correctly is called programmer.

**Q.2. What is the use of programming language? Give three examples of programming languages.**

The programming languages are used to write computer programs. A programming language defines the rules for writing the computer programs. A large number of programming languages are available for writing programs. Three examples of programming languages are Java, Python and C++.



**Q.3. What is computer programming?**

Computer programming is the process of writing instructions that tell a computer how to perform a specific task. These instructions are written in a programming language.The instructions are given to the computer to execute. The problem is solved when the computer executes these instructions.

**Q.4. What are the basic steps involved in writing a computer program?**

The basic steps in writing a computer program include writing code, compiling or interpreting, executing the code, and displaying the output.

**Q.5. What is syntax?**

A set of rules for writing computer programs in a programming language is known as syntax. All program statements must be written according to the syntax. The syntax works as the grammar of a programming language.

**Q.6. Do all programming languages use the same syntax?**

No, each language has its own syntax. The programs written in a specific programming language must follow its own syntax.

**Q.7. What is Python?**

**Python** is one of the most popular and widely used programming languages today. It is a versatile language used in various fields such as web development, data analysis and artificial intelligence. Python's simple syntax and clear structure make it an excellent choice for beginners to learn programming concepts easily.

**Q.8. What is a development environment in Python programming?**

A development environment in Python is a setup that includes tools to write, run, and debug Python code. It typically consists of a code editor, Python interpreter and useful libraries or packages. Setting up the development environment is essential before starting to write python programs.

**Q.9. Why is setting up the Python development environment important?**

Setting up the development environment ensures that the computer is ready for programming in Python. It includes installing Python, setting the PATH and choosing an IDE. A properly set environment makes it easier to write, test and debug programs.

**Q.10. Can Python be used online without installation?**

Yes, Python programs can be written and run using online services like Replit. It allows users to write and run Python code in the browser.

**Q.11. Define keyword and give three examples.**

Keyword has a predefined meaning and purpose in a programming language. The meaning and purpose of a keyword is defined by the developer of the language. It can be used for same purpose for which it is defined. Three examples of keywords are if, for and in.

**Q.12. Can the reserved words be used as variable names in Python?**

No, the reserved words cannot be used as variable names in Python because they have predefined meanings.

**Q.13. What are comments? Where can comments be added in the program?**

Comments are the lines of text in the program that are not executed by Python interpreter. They are used to provide explanations or notes for the code. They allow other programmers to understand the code more easily. The comments can be added anywhere in the program.

**Q.14. How are comments added on single line?**

Single-line comments start with #. Anything written after # on the same line is considered as comments.

**Q.15. How are comments added on multiple lines in Python?**

Multi-line comments are added in Python using triple quotes (''' or """). Anything written between triple quotes are considered comments.

**Q.16. What is the difference between single-line and multi-line comments in Python?**

Single-line comments start with the # symbol and are used to comment on one line only. Multi-line comments appear on multiple lines and are written using triple quotes (''' or """).

**Q.17. What is variable? Why is it used in programs?**

A variable is a named memory location. It is used to store program's input data and its computational results during execution. The value of variable may change during the execution of the program. However, the name of variable cannot be changed.

**Q.18. Write any four rules for declaring variables in Python.**

The first character of variable must be a letter or underscore. The blank spaces are not allowed in variable names. The reserved words cannot be used as variable names. The special symbols such as % and @ cannot be used as variable name.

**Q.19. Give any five examples of valid variable names.**

income _area total_price MAX_SPEED Marks1

**Q.20. Write down the reasons of the invalid variable names 3a and s$.**

The variable name 3a is invalid because it starts with a digit. The variable name s$ is invalid because it contains the special character $.

**Q.21. What happens if a Python variable name starts with a number?**

If a Python variable name starts with a number, it will result in a syntax error. Variable names must begin with a letter (a-z, A-Z) or an underscore (_).

**Q.22. How can you differentiate keyword from identifier?**

Keyword has a predefined meaning and purpose in a programming language such as if and for. However, an identifier is defined by the user such as variable name etc.

**Q.23. What does it mean that Python variables are case-sensitive?**

It means that Python treats variable names with different cases as different variables. For example, Var and var are considered two separate variables in Python.

**Q.24. Describe three basic data types in Python.**

The three basic data types in Python are integer, floating-point and string. The integer is used to store whole numbers. For example, x = 10 stores an integer value 10 in x. The floating-point is used to store real numbers. For example, y = 3.14 stores a floating-point value 3.14 in y. The string is used to store text or characters. For example, name = "Ali" stores the text "Ali" in name.

**Q.25. What is the function of the input() function in Python?**

The input() function in Python is used to accept input from the user during program execution. It inputs the value from the user via keyboard and returns the entered value as a string. The function is normally used with a variable that stores the value entered by the user.

**Q.26. Give an example of using the input() function in Python.**

An example is: n = input("Enter a number: ")

This line shows a message to the user to enter a number and stores the given number in the variable n. The input value can then be used in the program.

**Q.27. What is the purpose of the print() function in Python?**

The print() function in Python is used to display output on the screen. It can display text or the value of the variable on the screen. The print() function adds a newline character at the end of the output by default.

**Q.28. Give an example of using the print() function in Python.**

An example is: print("Hello world!")

This line displays a simple message "Hello world!" on the screen.

**Q.29. What is string concatenation in Python and how does it differ from numerical addition?**

String concatenation is the process of joining two or more strings together to form a single string. It differs from addition as it does not perform arithmetic operation but it combines strings.

**Q.30. How does Python treat input values by default?**

Python treats all input values as strings by default. For example, the input value 50 is considered as "50" and the input value 3.1 is considered as "3.1". These string values need to be converted into integer and float values in order to use them in the program properly.



**Q.31. What is the purpose of the float() function in Python?**

The float() function converts string input into a floating-point number. It allows the user to enter real values and use them in calculations.

Example: height = float(input("Enter your height: "))

**Q.32. When is it appropriate to use the float() function instead of the int() function in Python?**

The float() function should be used when the input is expected to be a decimal or fractional number. It can handle values like 3.14, 0.5, or -2.75 accurately. In contrast, the int() function should be used only when whole numbers are required.

**Q.33. What is the use of operators? List basic types of operators in Python.**

Operators are the symbols that are used to perform different operations on data. Some basic types of operators in Python are arithmetic, comparison, logical and assignment operator.

**Q.34. What is an expression?**

An expression is a combination of constants, variables and operators. It returns a single value when it is executed. The variables and constants in an expression are called operands. An operator is a symbol that performs some operation on operands.

**Q.35. Define arithmetic operator and list different arithmetic operators in Python.**

Arithmetic operator is a symbol that performs mathematical operations on data. Python language provides different arithmetic operators such as + , -, *, /, //, ** and %.

**Q.36. What does the modulus operator (%) do in Python? Give example.**

The modulus operator (%) returns remainder of a division. For example, 5 % 4 will return 1.

**Q.37. What is the purpose of the integer division operator (//) in Python? Give example.**

The integer division operator (//) returns only the integral part of a division. It discards any decimal part. For example, 5 // 4 will return 1.

**Q.38. How is exponentiation performed in Python? Give example.**

Exponentiation in Python is performed using ** operator. For example, 2 ** 3 will return 8 that is 2 raised to the power of 3.

**Q.39. List six comparison operators in Python.**

The six comparison operators in Python are ==, !=, >, <, >= and <=. They compare two values and return either True or False.

**Q.40. Differentiate between assignment operator (=) and equal to operator (==).**

The symbol = is an assignment operator that assigns a value to a variable. The symbol == is a relational operator that checks if two values are equal or not.

**Q.41. What are compound assignment operators in Python?**

Compound assignment operators combine assignment operator with the arithmetic operators. The examples of compound assignment operators include +=, -=, *= and /=.

**Q.42. What are logical operators in python? Give examples.**

The logical operators are used to combine multiple conditions or reverse a condition. Python provides three logical operators called and perator, or operator and not operator. Some examples are (a>50)and(b>50), (x>50)or(y<90) and not(a>b).

**Q.43. What is the use of "and" operator? Give example.**

The **and** operator is used to form a compound condition in which two conditions are evaluated. The conditions are given using comparison operators. The compound condition returns true if both conditions are true. It returns false if any one condition is false. An example of **and** operator is (A>10) and (B>10)

**Q.44. What is the use of "or" operator? Give example.**

The **or** operator is used to form a compound condition in which two conditions are evaluated. The conditions are given using comparison operators. The compound condition returns true if any condition is true. It returns false if both conditions are false. An example of **or** operator is (A>50) or (B>50).

**Q.45. What is the use of "not" operator?**

The **not** operator is used with single condition. It is used to reverse the result of a condition. It returns true if the condition is false and returns false if the condition is true. An example of **not** operator is not(A==B).

**Q.46. Differentiate between comparison and logical operators.**

The comparison operators compare two values and produce result as true or false. The logical operators combine multiple conditions or reverse a condition. The number of comparison operators is six and the number of logical operators is three.

**Q.47. What is meant by operator precedence?**

The order in which different types of operators in an expression are evaluated is known as operator precedence. Each operator has its own precedence level. If an expression contains different types of operators, the operators with higher precedence are evaluated before the operators with lower precedence.

**Q.48. What is the order of operator precedence in Python for arithmetic operations?**

Operator precedence determines the order in which operations are performed. The order of precedence of arithmetic operators are **, *, /, //, %, +, -.

**Q.49. What are control structures in Python?**

Control structures are statements used to control the flow of execution in a Python program. They are used in programs to make decisions or repeat actions based on certain conditions. The two main types of control structures are decision making and looping.

**Q.50. Why are control structures important in programming?**

Control structures allow the programmer to change the flow of execution. They help in making decisions or repeating blocks of code efficiently.

**Q.51. Describe a condition with an example.**

A condition is an expression that is either true or false. A condition may consist of arithmetic expressions, relational expressions and logical expressions. An example of a condition is a > b that will check if the value of a is greater than b.

**Q.52. Write three conditional statements and two repetition statements in Python.**

The three conditional statements in Python are if, if-else, and if-elif-else. The two repetition or looping statements are while and for.

**Q.53. Why do we need conditional statement?**

A conditional statement is needed to make decisions in programming based on a condition. It executes a statement if the condition is true and ignore when the condition is false. For example, a conditional statement may display "Pass" if the marks of a student are 33 or more.

**Q.54. Write the syntax of if-else statement.**

The syntax of an **if-else** statement in Python is as follows:

```
if condition:
    statement
else:
    statement
```

**Q.55. How if-elif-else is used to handle multiple conditions?**

The **if-elif-else** structure is used to handle multiple conditions by checking each condition in sequence until a true condition is found. If a condition is true, the statement following the condition is executed. The remaining blocks are skipped. The statement after **else** is executed if all preceding conditions are false.

**Q.56. How do if, elif and else statements work together in Python?**

The **if** statement is used to check a condition and execute a block of code if the condition is true. The **elif** statement allows additional conditions to be checked if the preceding **if** condition is false. The **else** statement is executed if all preceding **if** and **elif** conditions are false.

**Q.57. Differentiate between if-else and if-elif-if statements.**

**if-else** statement is used when there are two options and only one block of statements is to be executed on the basis of condition. **if-elif-if** statement is used when there are many options and only one block of statements is to be executed on the basis of condition.



**Q.58. Define loop structure. Write two uses or advantages of loop.**

A type of control structure that executes a statement or set of statements repeatedly is called loop structure. Loops are used to execute a statement or number of statements for a specified number of times. Loops are used to get a sequence of values.

**Q.59. Write two main types of loops in Python.**

The two main types of loops in Python are the **while** loop and the **for** loop. The **while** loop runs as long as a condition is true whereas the **for** loop iterates for specific number of times.

**Q.60. Why loops are important in programming languages? Give three reasons.**

Loops are important in programming languages. Firstly, they are used to repeat a statement or set of statements repeatedly without writing the statements again and again. Secondly, they save time and effort. Thirdly, the size of the program is reduced using loops.

**Q.61. What do you mean by an iteration in a loop?**

An iteration refers to a single execution of the loop body. Each time the loop completes its cycle, it is considered one iteration.

**Q.62. What is while loop?**

The **while** loop executes one or more statements as long as the given condition is true. It checks the condition before each iteration and stops when the condition is false. It is useful where the number of iterations is not known in advance.

**Q.63. Give an example of a while loop in Python.**

```
n = 1
while n <= 3:
  print(n)
  n = n + 1
```

**Q.64. What happens if you forget to update the loop variable in a while loop?**

If you forget to update the loop variable, the condition may never become false. As a result, the **while** loop will continue to run indefinitely that is called an infinite loop. This can cause the program to freeze or crash if not stopped manually.

**Q.65. Write the basic syntax of a for loop in Python.**

The basic syntax of for loop in Python is as follows:

```
for var in (value1, value2, value3...):
  statement
```

**Q.66. How does a programmer decide to use a while loop versus a for loop?**

The **for** loop is used when the number of iterations is known in advance such as display a message for 100 times. The **while** loop is used when the number of repetitions depends on a condition and is not known in advanced.

**Q.67. Define infinite loop.**

An infinite loop is a loop in which the ending condition never becomes false. It continues to repeat the same block of code endlessly. The loop runs forever until the user or system manually stops the program.

**Q.68. What is a function?**

A function is a named block of code that performs some action. The statements written in a function are executed when it is called by its name. Functions are the building blocks of python programs.

**Q.69. List any three benefits of using functions.**

Firstly, the functions are easier to code and modify. Secondly, they are easier to maintain and debug. Thirdly, they can be reused and reduces programming time.

**Q.70. What is the syntax for defining a function in Python?**

A function is defined using the def keyword followed by the function name and parameters.

```
def fname (param):
  statement
```

**Q.71. What is the difference between a function and a module?**

A function is a reusable block of code that performs a specific task. A module is a file that can contain multiple functions and other code. Functions are defined inside modules and can be reused when imported.

**Q.72. What is a function call?**

Function call is the statement that invokes a function. A function is executed when it is called by its name. The function name is followed by necessary parameters in parentheses. If there are many parameters, these are separated by commas. The empty parentheses are used if no argument is required.

**Q.73. How does a function return value?**

A function can return a single value. The keyword return is used to return the value back to the calling function. When return statement is executed in the function, the controls moves back to the calling function along with the returned value. It is a good practice to write the returned value in parenthesis "( )" after return keyword.

**Q.74. What is a library in Python?**

A **library** is a collection of related modules that contain pre-written code. The code in a library can be used in different programs. Libraries simplify programming by providing ready-to-use functions for common tasks such as mathematical calculations and random number generation etc.

**Q.75. What is the difference between a module and a library in Python?**

A module is a single Python file that contains functions, classes or variables. A library is a collection of related modules that contain pre-written code. The code in a library can be used in different programs. Libraries simplify programming by providing ready-to-use functions for common tasks such as mathematical calculations and random number generation etc

**Q.76. What does the randint() function do in Python? Write its syntax.**

The **randint()** function is used to generate a random number within given range. The syntax for the **randint()** function is as follows:

```
random.randint(n1, n2)
```

**Q.77. Write an example to generate a random number between 5 and 15.**

```
import random
rn = random.randint(5, 15)
print(rn)
```

**Q.78. What is a package in Python?**

A package is a way to organize and structure code by grouping related modules into directories. A package refers to a folder that contains one or more Python modules and a special file name **__init__.py**. This organization helps to manage and reuse code effectively.

**Q.79. How is a package different from a module in Python?**

A module is a single Python file containing code like functions or classes. A package is a directory that holds multiple related modules.

**Q.80. What are built-in data structures in Python?**

Built-in data structures are pre-defined formats in Python used to store and organize data. Common examples include lists, tuples and dictionaries. They help to manage and process data efficiently in programs.

**Q.81. What is a list in Python?**

A list is a data structure in Python that can store multiple values. The list is mutable that means its elements can be created, accessed, modified or removed easily. Each item stored in a list is called an element. Lists can store elements of different data types, including integers, floats, strings etc.

**Q.82. How does list differ from simple variable?**

A list can store multiple values but a simple variable can store only one value. Different types of processing can be applied on list easily. However, it is difficult to process multiple values using simple variables.



**Q.83. How is an element accessed in List?**

Each element in the list is accessed with reference to its position in the list. This position is called index. The index of first element is 0 and the index of last element is length – 1. The value of the index is written in brackets along with the name of list.

**Q.84. How a list is created in Python?**

A list can be created by assigning values to a variable using square brackets. For example:

```
my_list = [1, 2, 3, 4, 5]
```

An empty list can be created as follows:

```
empty_list = []
```

**Q.85. Write a Python code to create a list of even numbers from 2 to 10 and print it.**

```
even = [2, 4, 6, 8, 10]
print(even)
```

**Q.86. How an element can be inserted into a list? Give an example.**

An element can be inserted into a list using insert() method. For example:

```
test = [1, 2, 3]
test.insert(1, 10)
print(test)
```

**Q.87. What does the index() method do in a list?**

The index() method returns the index of the first occurrence of a specified value in the list. For example:

```
test = [1, 2, 3, 2]
print(test.index(2))
```

**Q.88. What is the purpose of the append() method in a Python list?**

The append() method is used to add an item to the end of the list. It modifies the original list by including the new element as the last item.

Example: students.append("Hina")

**Q.89. How would you add the number 100 to the end of a list and then print the updated list?**

```
test = [10, 20, 30]
test.append(100)
print(test)
```

**Q.90. How does the remove() method work in a list?**

The remove() method removes the first occurrence of an item from the list.

Example: student_names.remove("Sara")

**Q.91. How can you combine two lists in Python?**

The concatenation operation is used to combine two or more lists into a single list. This operation can be performed by using concatenation operator +. An example is as follows:

```
odd = [1, 3, 5, 7, 9]
even = [2, 4, 6, 8, 10]
com = odd + even
print(com)
```

**Q.92. What is list slicing in Python?**

The slicing operation is used to extract a portion of the list. This operation does not affect the original list but stores the portion in the new list. an example is as follows:

```
num = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
num2 = num[1:5]
print(num2)
```

**Q.93. What is a tuple in Python?**

A tuple is a data structure in Python that can store multiple values. The tuple is immutable that means its elements cannot be modified after creation. Each item stored in a tuple is called an element. Tuple can store elements of different data types, including integers, floats, strings etc.

**Q.94. How does a list differ from a tuple in Python?**

A list is a mutable data structure that means it can be modified after it is created. A tuple is an immutable data structure that means it cannot be modified after it is created. Lists are defined using square brackets [] but tuples are defined using parentheses (). Lists are commonly used when data needs to be modified while tuples are preferred to store fixed or constant data.

**Q.95. How do you create a tuple in Python?**

You create a tuple by placing items inside parentheses, separated by commas.
Example: my_tuple = (1, 2, 3, "Hello", 4.5)
Tuples can contain mixed data types like integers, strings, and floats.

**Q.96. Can a tuple be modified after it is created? Why or why not?**

No, a tuple cannot be modified after it is created because it is immutable. This means you cannot add, remove or change its items. This makes tuples useful for storing fixed data.

**Q.97. What function is used to find the length of a tuple?**

The len() function can be used to find the lenght of a tuple.
Example: len(my_tuple) returns the length of the tuple.

**Q.98. How does negative index work in Python?**

The negative index counts from the end of the sequence. For example, -1 refers to the last item, -2 to the second last, and so on.
Example: my_tuple[-1] returns the last element.

**Q.99. What is Object-Oriented Programming (OOP)?**

Object-Oriented Programming (OOP) is a technique in which programs are written using objects. An object is a combination of data and functions that may represent real-world entities like a person, place or thing. In OOP, data and all possible functions on data are grouped together. Object oriented programs are easier to understand and modify. Python, C++ and Java are popular object-oriented programming languages.

**Q.100. How are classes and objects related in OOP?**

A class defines the structure and behavior of objects. An object is created from a class and contains actual values. You can create multiple objects from the same class with different attributes.

**Q.101. How do you define a class in Python?**

A class is defined using the class keyword followed by the class name. The attributes and methods are defined inside the class.

**Q.102. What is the purpose of the __init__ method in a class?**

The __init__ method initializes the object with specific attributes when it is created. It assigns values to the object's properties using **self**. The example of this method is as follows:

```
def __init__(self, color, size, wheels):
    self.color = color
```

**Q.103. What is meant by self in a Python class?**

The **self** refers to the current object of the class. It is used to access variables and methods within the class. It must be the first parameter in the class methods.

**Q.104. How do you create an object from a class?**

An object can be created by writing the object name followed by the assignment operator and class name. For example, the following example creates an object of class Test.

```
mytest = Test()
```

**Q.105. What is exception handling in Python?**

Exception handling is a method to manage run-time errors that occur during program execution. It allows a program to continue running or terminate if an error occurs instead of crashing. It can also be used to shows user-friendly error messages.

**Q.106. What is file handling in Python?**

File handling refers to the process of reading from and writing to files in a Python program. Files can be used to provide input to a program or store its output permanently. Python provides built-in functions like open(), read(), and write() to perform file operations.



**Q.107. What function is used to open a file in Python?**

The open() function is used to open a file. It takes the file name and mode such as "r", "w", "a" as arguments. Example: open("example.txt", "r") opens a file in read mode.

**Q.108. What does the "r" mode do when opening a file?**

The "r" mode stands for read mode. It is used to open an existing file and read its contents.

**Q.109. What is the purpose of the "w" mode in file handling?**

The "w" mode is write mode that is used to create a new file or overwrite an existing file. All previous contents are erased and new data is written.

**Q.110. Why should we use the with statement for file handling?**

The **with** statement automatically closes the file after it has been used. It helps to manage resources efficiently and avoids common mistakes like forgetting to close the file. It is the best practice for working with files in Python.

**Q.111. How do you read the contents of a file in Python?**

The contents of a file can be read using read() method. An example is as follows:

```
with open("example.txt", "r") as file:
    content = file.read()
```

**Q.112. How do you append text to a file in Python?**

The text can be appended to a file by opening the file in append ("a") mode and using the write() method. An example is as follows:

```
with open("example.txt", "a") as file:
    file.write("New line added.\n")
```

**Q.113. How do you write data to a file in Python?**

The write() method is used to write data in a file. The file must be opened in in write ("w") or append ("a") mode. An example is as follows:

```
with open("example.txt", "w") as file:
    file.write("Hello, World!\n")
```

**Q.114. What is pdb in Python?**

pdb stands for Python Debugger. It is a built-in tool used to debug Python programs. It can be used to step through the code, inspect variables and understand the flow of execution.

**Q.115. How can error message help in debugging?**

An error message provides information about the error and its location in the code. The error message can be used to locate the source of the problem in the program. It helps to find and remove bugs.

**Q.116. Write Python expression for the following statements:**

a. **Age is from 18 to 25**
(age >= 18 and age <= 25)

b. **Temperature is less than 40.0 and greater than 25.0**
(temperature < 40.0 and temperature > 25.0)

c. **Year is divisible by 4**
(year%4 == 0)

d. **Speed is not greater than 80**
not (speed > 80)

e. **Y is greater than x and less than z**
(y > x and y < z)

f. **W is either equal to 6 or not greater than 3**
((w == 6) or not (w > 3))

g. **score is between 50 and 60 inclusive**
(score >= 50 and score <= 60)

h. **p and q are less than zero**
(p < 0 and q < 0)

**Q.117. Determine the output of the following code:**

**a.**
```
x = 7
y = 3
print(x // y, "and", x % y)
```
**Ans:** 2 and 1

**b.**
```
num = 10
num *= 5
print(num)
```
**Ans:** 50

**c.**
```
x = 4 * 5 // 2 + 9
print(x)
```
**Ans:** 19

**d.**
```
name = "Ali"
print("Hello, " + name + "!")
```
**Ans:** Hello, Ali!

**e.**
```
n = 10
n %= 2
n += 5
print(n)
```
**Ans:** 5

**f.**
```
x = 2
y = 3
z = (x + 5) // (y + 5)
print(x + y + z)
```
**Ans:** 5

**g.**
```
a = 10
b = 3
print(a, "//", b, "=", a // b)
```
**Ans:** 10 // 3 = 3

**h.**
```
a = 10
b = 3
print(a, "%", b, "=", a % b)
print(a, "**", b, "=", a ** b)
```
**Ans:** 10 % 3 = 1
10 ** 3 = 1000

**i.**
```
x = 0
y = 5
z = 4
x = y + z * 2
print(x)
```
**Ans:** 13

**j.**
```
ch1 = 'A'
ch2 = 'B'
ch1 = ch2
ch2 = ch1
print(ch1, ch2)
```
**Ans:** B B

**k.**
```
temperature = 15
if temperature > 30:
    print("It's a hot day")
else:
    print("It's not a hot day")
```
**Ans:** It's not a hot day

**l.**
```
a = 5
if a % 2 == 0:
    print("Even")
else:
    print("Odd")
```
**Ans:**Odd

**m.**
```
if 2 % 3 >= 1:
    print("Great")
else:
    print("Bad")
```
**Ans:** Great

**n.**
```
p = 21
q = 4
if p % q == 4:
    x = 0
else:
    x = 1
print("x =", x)
```
**Ans:** x = 1

**o.**
```
k = 0
while k <= 4:
    print("ok", end=" ")
    k = k + 1
```
**Ans:**
ok ok ok ok ok

**p.**
```
n = 1
while n <= 3:
    print("Pakistan")
    n = n + 1
```
**Ans:** Pakistan
Pakistan
Pakistan

**q.**
```
x = 3
while x < 9:
    print("x is", x)
    x += 2
```
**Ans:** x is 3
x is 5
x is 7

**r.**
```
a = 1
while a % 3 > 0:
    a += 4
    print(a, "x")
```
**Ans:**
5 x
9 x



**Q.118. Determine the output of the following code:**

**a.**
```
num = 10
num += 5
num -= 2
num *= 3
print(num)
```
**Ans:** 39

**b.**
```
x = 3
y = 17
print(x // y, y // x, y % x)
```
**Ans:** 0 5 2

**c.**
```
base = 2
exponent = 3
result = base ** exponent
print(result)
```
**Ans:** 8

**d.**
```
num1 = 14
num2 = 4
result = num1 % num2
print(result)
```
**Ans:** 2

**e.**
```
x = 13
y = 3
print(x // y, y // x, x % y)
```
**Ans:** 4 0 1

**f.**
```
n = 500
a = n % 100
b = n // 10
n = n % 10
print(n, b, a)
```
**Ans:** 0 50 0

**g.**
```
a = 1
b = 6
if a + b < 7:
    print(a)
else:
    print(b)
```
**Ans:** 6

**h.**
```
p = 10
q = 3
if p % q == 3:
    r = 0
else:
    r = 1
print(r)
```
**Ans:** 1

**i.**
```
x = 5
y = 5
z = 10
if x == y and y < z:
    print("Condition met")
else:
    print("Condition not met")
```
**Ans:** Condition met

**j.**
```
a = 5
b = 0
if a or b:
    print("if part")
else:
    print("else part")
```
**Ans:** if part

**k.**
```
x = 25
if x % 7 + 3 == 17:
    print("Correct")
else:
    print("Incorrect")
```
**Ans:** Incorrect

**l.**
```
grade = 100
if grade > 60 and grade < 100:
    print("Pass")
else:
    print("Fail")
```
**Ans:** Fail

**m.**
```
j = 2
for i in range(1, 4):
    print(i, j, end=" ")
```
**Ans:** 1 2 2 2 3 2

**n.**
```
for i in range(1, 6, 2):
    print(i, end="*")
```
**Ans:** 1*3*5*

**o.**
```
for i in range(2, 8, 2):
    print(i, end=", ")
```
**Ans:** 2, 4, 6,

**p.**
```
for i in range(5, 1, -1):
    print(i, end=" ")
```
**Ans:** 5 4 3 2

**q.**
```
for i in range(4):
    print("Hello", end=" ")
```
**Ans:** Hello Hello Hello Hello

**r.**
```
for i in range(0, 10, 2):
    print(i)
```
**Ans:**
0
2
4
6
8

**Q.119. Determine the output of the following code:**

**a.**
```
for i in range(3):
    print("Pakistan")
```
**Ans:**
Pakistan
Pakistan
Pakistan

**b.**
```
myList = [1, 2, 3];
myList[1] = 4;
print(myList)
```
**Ans:** [1, 4, 3]

**c.**
```
for i in range(2, 5):
    print(i)
```
**Ans:**
2
3
4

**d.**
```
sum = 0
for i in range(1, 6):
    sum += i
print(sum)
```
**Ans:** 15

**e.**
```
for num in range(4):
    print(num)
else:
    print("Done")
```
**Ans:**
0
1
2
3
Done

**f.**
```
for i in range(3):
    for j in range(2):
        print(i, j)
```
**Ans:**
0 0
0 1
1 0
1 1
2 0
2 1

**g.**
```
for i in range(1, 16):
    if i % 2 == 0:
        print(i, end=' ')
```
**Ans:** 2 4 6 8 10 12 14

**h.**
```
items = [10, 20, 30]
items[0] = 100
print(items)
```
**Ans:** [100, 20, 30]

**i.**
```
ch_list = ['a', 'b', 'c', 'd']
ch_list.append('e')
print(my_list)
```
**Ans:** ['a', 'b', 'c', 'd', 'e']

**j.**
```
my_list = [10, 20, 30, 40, 50]
print(my_list[-1])
```
**Ans:** 50

**k.**
```
my_list = [1, 2, 3, 4, 5]
print(len(my_list))
```
**Ans:** 5

**l.**
```
my_list = ['a', 'b', 'c']
my_list.insert(1, 'd')
print(my_list)
```
**Ans:** ['a', 'd', 'b', 'c']

**Q.120. Find errors in the following code:**

**a.**
```
m = = 7
if m >= 10,
    print("Good");
```
1. m == 7 is incorrect. The assignment operator should be written as = instead of ==.
2. The comma at the end of if statement must be replaced with colon.
3. The semicolon at the end of the last line must be removed.

**b.**
```
if (7 ≠ 10
   print("hello")
else
   prints(("welcome")
```
1. The symbol ≠ must be replaced with !=.
2. There must be a colon after else.
3. The prints must be replaced with print.

**c.**
```
For x in rang(50, 0, -1):
    print(x):
```
1. The keyword For should be written in lowercase as for.
2. The function rang must be replaced with range.
3. The colon at the end of the last line must be removed.



**d.**
```
k = 1
while(k <= 5):
{
   k = k + 1
   print('k' k)
}
```
1. The semicolon after the condition while(k <= 5); must be replaced with colon.
2. The braces { } must be removed as Python uses indentation to define code blocks.
3. There should be a comma or + between 'k' and K in the last line.

**e.**
```
k <= 15;
while k <== 10:
   print("k"; k)
   k += 1
```
1. The symbol <= in the first line must be replaced with = to initialize the variable k.
2. The condition k<==10 in the second line must be written as k<=10.
3. There should be a comma or + between 'k' and k in print().

## Multiple Choice Questions

1. **Which of the following fields is NOT commonly associated with Python?**
   a. Web development  b. Data analysis
   c. Hardware circuit design  d. Artificial Intelligence
2. **Python is used in various fields such as web development, data analysis and AI that shows its:**
   a. Rigidity  b. Exclusivity  c. Complexity  d. Versatility
3. **_______ is the process of creating a set of instructions for a computer to perform a task.**
   a. Programming  b. Debugging  c. Compiling  d. Executing
4. **Which of the following is NOT a part of creating a computer program?**
   a. Writing code  b. Compiling/Interpreting code
   c. Executing code  d. Printing code
5. **Which of the following is the first basic step involved in writing a program?**
   a. Execute  b Output
   c. Write code  d. Compile/Interpret
6. **_______ refers to the process of preparing a computer to write, run and debug Python programs.**
   a. Development environment  b Software compilation
   c. Code debugging  d. System administration
7. **Which of the following is NOT valid Python syntax?**
   a. print('text')  b print("text")
   c. print text  d. print("'text'")
8. **What is missing in the following statement: print("Edhi Foundation)?**
   a. Opening quote  b. Closing parenthesis
   c. Closing quote  d. Comma
9. **Which of the following is a valid print statement in Python?**
   a. print(Hello)  b. print("Hello")
   c. printf "Hello"  d. print(Hello)
10. **What is the output of: print("Hello")?**
    a. Hello  b. "Hello"  c. print(Hello)  d. Error
11. **Which of the following characters are used to enclose string values in Python?**
    a. Quotes  b. Parentheses only
    c. Square brackets  d. Angle brackets

12. The text in quotes inside print("Hello World") is called a:
    a. Statement   b. Function
    c. String   d. Comment
13. Which of the following is used to explain or document code in a Python program?
    a. Variables   b. Loops   c. Comments   d. Functions
14. Which of the following symbols is used to start a single-line comment in Python?
    a. //   b. <!--   c. #   d. *
15. Which of the following is a valid comment?
    a. -- This is a comment   b. # This is a comment
    c. // This is a comment   d. /* This is a comment */
16. How are multi-line comments created in Python?
    a. Using angle brackets <!-- -->   b. Using /* and */
    c. Using triple quotes (''' or """)   d. Using REM statements
17. Where the comments can be added in Python program?
    a. Top of the program   b. Bottom of the program
    c. Middle of the program   d. Anywhere
18. What happens to comments during program execution?
    a. They are printed   b. They are ignored
    c. They cause errors   d. They become variables
19. A storage container in computer memory that can hold changing values is called:
    a. Variable   b. Data type   c. Parameter   d. Operator
20. What can you do with a variable in Python?
    a. Store data   b. Retrieve data   c. Manipulate data   d. All
21. In Python, variable names can start with a:
    a. Digit   b. Letter or underscore
    c. Special symbol   d. Space
22. Which of the following is a valid variable name in Python?
    a. 1st_var   b. _var1   c. for   d. var-name
23. Which symbol can be used at the beginning of a Python variable name?
    a. @   b. _   c. #   d. $
24. Which of the following is NOT a valid variable name?
    a. myVar   b. MyVar1   c. 123var   d. var_123$
25. Which of the following is NOT a keyword in Python?
    a. for   b. while   c. if   d. Class
26. How are 'Num' and 'num' treated in a Python program?
    a. Same variable   b. Different variables
    c. Invalid variable names   d. None
27. Which of the following data types is used to store whole numbers in Python?
    a. Floating point   b. String   c. Integer   d. Boolean
28. What is the data type of the variable price = 19.99?
    a. Integer   b. Floating point
    c. String   d. Boolean
29. Which of the following is a floating-point variable?
    a. score = "8.5"   b. score = 8.5
    c. score = 'float'   d. score = '8.5'
30. Which of the following is an integer variable?
    a. temp = 22.5   b. is_raining = "False"
    c. student_count = 100   d. message = True
31. What is the data type of the variable that stores a person's name?
    a. Integer   b. Floating point
    c. String   d. Boolean
32. Which data type is used to represent True or False values?
    a. Character   b. Integer   c. Floating point   d. Boolean



33. Which statement correctly creates a string variable?
    a. item = True   b. item = 23
    c. item = 14.6   d. item = "Book"
34. Which of the following is NOT a valid data type in Python?
    a. Integer   b. String   c. Decimal   d. Boolean
35. Which function is used in Python to get input from the user?
    a. read()   b. input()   c. get()   d. scan()
36. What type of value does input() return by default?
    a. Integer   b. String
    c. Boolean   d. Floating point
37. If you write input("Enter your age: ") and enter 25, what type is the result?
    a. Integer   b. String
    c. Boolean   d. Floating point
38. Which function is used to display output in Python?
    a. echo()   b. print()   c. show()   d. display()
39. The process of joining two strings in Python is called:
    a. Addition   b. Multiplication
    c. Concatenation   d. Subtraction
40. The operator used for string concatenation in Python is:
    a. +   b. -   c. *   d. /
41. What is the purpose of the + operator in this statement: print("Hello, " + name + "!")?
    a. Repeats strings   b. Joins strings
    c. Multiplies strings   d. Compares string lengths
42. Which of the following correctly prints both text and a variable?
    a. print("Age is: " + age)   b. print("Age is:", age)
    c. print("Age is:" + , age)   d. echo("Age is:", age)
43. How should you take an integer input from the user in Python?
    a. age = str(input("Enter age: "))   b. age = input("Enter age: ")
    c. age = int(input("Enter age: "))   d. input(int(age))
44. Which of the following functions converts input to a floating-point value?
    a. toFloat()   b. str()   c. float()   d. convert()
45. A symbol that performs operations on variables and values is called:
    a. Function   b. Keyword
    c. Operator   d. Expression
46. A combination of variables, operators and values that produces a result is called:
    a. Function   b. Statement
    c. Condition   d. Expression
47. Which of the following is NOT an arithmetic operator in Python?
    a. -   b. +   c. **   d. ==
48. The operator used to calculate the remainder of a division in Python is:
    a. * (multiplication)   b. / (division)
    c. % (modulus)   d. - (subtraction)
49. The modulus operator in Python is:
    a. +   b. **   c. //   d. %
50. The result of the expression 10 % 3:
    a. 1.25   b. 1   c. 5   d. 4
51. The // operator in Python represents:
    a. Exponentiation   b. Modulus
    c. Floor division   d. Logical AND
52. Floor division is also called:
    a. True division   b. Integer division
    c. Modulus division   d. Floating-point division
53. The result of the expression 10 // 3 in Python is:
    a. 3.33   b. 10   c. 0   d. 3

54. The result of the expression 3 // 5 in Python is:
a. 1.25 b. 5 c. 0 d. 1
55. The data type of the result from the expression 10 / 3 in Python is:
a. Integer b. Float c. Boolean d. String
56. The result of 45 / 4 in Python is:
a. 10 b. 11 c. 11.25 d. 12
57. The result of 7 / 4 in Python is:
a. 2.75 b. 2 c. 1.75 d. 1
58. What is the result of 45 // 4?
a. 10 b. 11 c. 11.25 d. 12
59. Which operator is used for exponentiation in Python?
a. power() b. ** c. exp() d. ^
60. How do you calculate the square of a number in Python?
a. 5**2 b. 5 ^ 2 c. 5 % 2 d. 5>>2
61. The result of the expression 10 ** 3 in Python is:
a. 30 b. 13 c. 10 d. 1000
62. How many comparison operators does Python provide?
a. 4 b. 5 c. 6 d. 7
63. Which of the following is not a comparison operator in Python?
a. = = b. != c. >= d. +=
64. The result of a comparison operation in Python is always a:
a. String value b. Floating-point values
c. Boolean value d. Integers
65. Which operator checks if two values are NOT equal?
a. != b. =! c. == d. <>
66. The result of the expression 5 != 3 in Python is:
a. True b. False c. 5 d. Error
67. The result of the expression 5 < 3 in Python is:
a. True b. False c. 5 d. Error
68. Which operator is used to check for equality in Python?
a. = b. == c. := d. ===
69. Which symbol is used as the assignment operator in Python?
a. = b. == c. := d. =>
70. To add number to sum, you write:
a. sum += number b. sum = sum + number
c. number = sum + number d. Both a and b
71. If a = 10, what will be the value of a after executing the code a += 5?
a. 5 b. 10 c. 15 d. 20
72. If a = 10, what will be the value of a after executing the code a -= 5?
a. 15 b. 5 c. 10 d. 0
73. If a = 10, what will be the value of a after executing the code a *= 5?
a. 5 b. 50 c. 15 d. 10
74. If a = 10, what will be the value of a after executing the code a /= 5?
a. 2 b. 2.0 c. 5 d. 10
75. If a = 10 and b = 5, what will be the value of a after executing a **= b?
a. 50 b. 100 c. 100000 d. 2
76. In Python, multiple conditions or expressions can be joined by using:
a. Arithmetic operators b. Comparison operators
c. Logical operators d. Bitwise operators
77. Which of the following is NOT a logical operator in Python?
a. and b. or c. and d. xor
78. How many logical operators are there in Python?
a. 2 b. 3 c. 4 d. 5
79. Which of the following is a logical and operator in Python?
a. & b. && c. and d. d. ||



80. Which of the following is a logical OR operator in Python?
a. & b. || c. or d. |
81. Which of the following is a logical NOT operator in Python?
a. ! b. not c. ~ d. !=
82. Which logical operator in Python returns True only if all conditions are True?
a. or b. and c. not d. xor
83. Which logical operator in Python returns True if at least one condition is True?
a. or b. and c. not d. xor
84. Which logical operator is used to reverse the result of a condition?
a. or b. and c. not d. xor
85. Assume x = 3 and y = 4, Which of the following is true?
a. x < 4 and y < 4 b. x < 4 or y < 4
c. x > 4 and y > 4 d. x > 4 or y > 4
86. The order in which operations are performed in an expression is called:
a. Operator grouping b. Expression priority
c. Operator precedence d. Execution pattern
87. Which of the following is a valid expression in Python?
a. 3 + 6 = s b. s = 3 + 5 c. while = 10 d. result =
88. Which symbol is used in Python to control the order of operations in expressions?
a. {} b. [] c. () d. < >
89. Which operator has the highest precedence in Python?
a. * (multiplication) b. + (addition)
c. () (parentheses) d. - (subtraction)
90. Which of the following operations comes after parentheses in operator precedence?
a. Addition (+) b. Exponentiation (**)
c. Modulus (%) d. Subtraction (-)
91. Which has lower precedence in Python?
a. Multiplication b. Division c. Modulus d. Addition
92. The expression 2 * 3 ** 2 evaluates to:
a. 36 b. 18 c. 12 d. 81
93. What is the result of evaluating 2 + 2 ** 3 / 2?
a. 5 b. 3 c. 4.0 d. 6.0
94. In the expression 5 + 2 * 3, which operation is performed first?
a. 5+2 b. 2 * 3 c. 5 * 2 d. 5 * 3
95. What is the output of the following Python statement: print(14 + 10 // 4 - 2)?
a. 4 b. 12 c. 14 d. 15
96. Decision making and looping are types of:
a. Input methods b. Functions
c. Control structures d. Data types
97. Which of the following is NOT a control structure in Python?
a. if-else b. for loop c. while loop d. print
98. Which control structure is used to make decisions in a program?
a. for loop b. if statement
c. while loop d. function call
99. Which of the following statements is used to write a single-alternative decision structure in Python?
a. if-elif-else b. if c. if-else d. if-call
100. ________ statement tests a condition and executes one block of code if condition is true and another block if it's false.
a. if b. if-else c. test-jump d. if-call
101. Which is the best option for making decisions based on multiple conditions in Python?
a. if b. if-else c. if-elif-else d. for
102. In Python, which keyword is used to check additional conditions if previous conditions are false?
a. elif b. else if c. then d. for

**103. For num = 64, which of the following is true?**
a. if (num % 14 == 0) b. if (num % 15 == 0)
c. if (num % 16 == 0) d. if (num % 17 == 0)

**104. If p=21, which statement will be true?**
a. if(p%2==0) b. if(p%3==0)
c. if(p%4==0) d. if(p%5==0)

**105. What will be the output of following code?**

```
m = 33
If m >= 33:
    print("Pass")
else:
    print("Fail")
```

a. Fail b. Pass c. PassFail d. FailPass

**106. What will be the output of following code?**

```
m = 0
n = 5
if m % n > 1:
    print("Success")
else:
    print("Looser")
```

a. Success b. Looser c. "Success" d. "Looser"

**107. Which control structure in Python is used to repeat a block of code multiple times?**
a. Loop b. Conditional c. Sequential d. None

**108. Which of the following are types of loops in Python?**
a. for b. while c. do-while d. a and b

**109. A single execution of a loop's code block is called a(n):**
a. Sprinting b. Duration c. Iteration d. Testing

**110. Which type of loop executes its block of code as long as its condition remains true?**
a. for loop b. while loop
c. if-else statement d. function call

**111. In a while loop, when is the condition evaluated?**
a. Only at the start b. Before each iteration
c. After each iteration d. Only at the end

**112. Which of these is a correct while loop syntax in Python?**
a. while (condition) { code } b. while condition: code
c. while condition then code d. while: if condition code

**113. Which loop is best for iterating through a list of items?**
a. while b. for c. if-else d. do-while

**114. Which function in python is used to generate a sequence of numbers?**
a. range() b. len() c. randint() d. eval()

**115. The function range(5) in Python returns a sequence of:**
a. 1, 2, 3, 4, 5 b. 0, 1, 2, 3, 4, 5
c. 1, 2, 3, 4 d. 0, 1, 2, 3, 4

**116. Which of the following correctly generates a sequence of 0, 1, 2, 3?**
a. range(0, 3) b. range(0, 4) c. range(3) d. range(5)

**117. Which of the following refers to a particular format or method for organizing and storing data?**
a. Function b. Data structure
c. Module d. Statement

**118. Which of the following is a built-in data structure in Python?**
a. print b. List c. File d. Import

**119. Modules and built-in data structures are included in:**
a. Python interpreter b. User-defined files
c. Python standard library d. Operating system



**120. A block of code that can be reused to perform a specific task is called:**
a. Function b. Variable c. List d. Constant

**121. A set of related functions grouped together is called a:**
a. Module b. Library c. Loop d. Package

**122. The keyword used to define a function in Python is:**
a. def b. function c. define d. func

**123. What is the correct syntax for defining a Python function?**
a. function name(): b. def name():
c. def name() d. define name():

**124. What are the values passed into a function called?**
a. Returns b. Parameters c. Conditions d. Variables

**125. Parameters in Python function definitions always appear within:**
a. Square brackets ([]) b. Parentheses (())
c. Curly braces ({}) d. Quotation marks ("")

**126. A function is executed when it is:**
a. defined b. prototyped c. declared d. called

**127. What is the correct syntax for calling a function named greet?**
a. greet{} b. greet() c. call.greet d. greet[]

**128. How many values can be passed to the function?**
a. One b. two c. Three d. Many

**129. In Python, a function can typically return:**
a. One value b. Two values
c. Three values d. Many values

**130. The keyword used to specify the value returned by a function in Python is:**
a. return b. call c. output d. exit

**131. Which feature allows a function to use a predefined value when a parameter is missed during a function call?**
a. Required parameters b. Return values
c. Default parameters d. Keyword parameters

**132. A pre-built toolkit of reusable code in Python is called:**
a. Library b. List c. Keywords d. None

**133. Which keyword is used to include a library in Python?**
a. include b. load c. import d. use

**134. Python's "math" and "random" are examples of:**
a. Variables b. Data types
c. Libraries d. Operators

**135. The function to generate random numbers in Python is:**
a. range() b. randint() c. len() d. eval()

**136. Which library would you use to generate a random integer?**
a. random b. math c. randomize d. rand

**137. Which of the following is a standard Python library for working with dates and times?**
a. datetime b. calendar c. timecheck d. clock

**138. What does `datetime.datetime.now()` return?**
a. Only the date b. Only the time
c. Current date and time d. Random timestamp

**139. How do you access the current time using the `datetime` library?**
a. now.datetime() b. datetime.now()
c. datetime.datetime.now() d. date.time.now()

**140. Which of the following is NOT a valid way to import and use a library function in Python?**
a. import random b. import datetime
c. import statistics.mean d. import statistics

**141. A directory containing related modules is called:**
a. Function b. Package
c. Data types d. Application

142. Which of the following is NOT a built-in data structure in Python?
   a. Set   b. List
   c. Tuple   d. Dictionary
143. Which Python data structure is versatile and can store multiple items of different data types?
   a. Tuple   b. List
   c. Set   d. Dictionary
144. Which Python data structure is mutable and ordered?
   a. Tuple   b. List
   c. String   d. Dictionary
145. A list is created by placing items inside ______ separated by commas.
   a. Parentheses ( )   b. Curly braces { }
   c. Square brackets [ ]   d. Angle brackets < >
146. Which of the following can a Python list contain?
   a. Numbers   b. Strings   c. Other lists   d. All
147. Which of the following is the correct way to create list in Python?
   a. list = (1, 2, 3)   b. list = {1, 2, 3}
   c. list = [1, 2, 3]   d. list = <1, 2, 3>
148. Which of the following is the correct way to create list in Python?
   a. list1 = []   b. list1 = [12, 4, 4]
   c. list1 = [1, "3", "red"]   d. All
149. Which of the following is used to access a specific item of a list in Python?
   a. Value   b. Index   c. Length   d. Name
150. What is the index of the first item in a Python list?
   a. 1   b. -1   c. 0   d. first
151. How do you access the second item in the list fruits = ["Mango", "Apple", "Banana"]?
   a. fruits[0]   b. fruits[1]
   c. fruits[2]   d. fruits.second()
152. How do you access the last element of a list named myList?
   a. myList[0]   b. myList[-1]
   c. myList[len(myList)]   d. myList[-2]
153. What does fruits[-1] return for the list fruits = ["Mango", "Apple", "Banana"]?
   a. [illegible]   b. "Apple"   c. Banana"-   d. Error
154. Suppose list1 is [3, 4, 5, 20, 5, 25, 1, 3], what is list1.index(5)?
   a. [illegible]   b. 4   c. 1   d. 2
155. Which of the following is NOT a list method in Python?
   a. append()   b. [illegible]   c. sort()   d. split()
156. Which of the following is used to add a new item to the end of a list in Python?
   a. append()   b. add()   c. insert()   d. extend()
157. Which of the following can be used to add 5 to the end of list1?
   a. list1.add(5)   b. [illegible]1.append(5)
   c. list1.addLast(5)   d. list1.addEnd(5)
158. Which of the following methods deletes the first occurrence of an item from a list in Python?
   a. delete()   b. remove()   c. pop()   d. discard()
159. How do you remove the name "Sara" from the list student_names?
   a. student_names.delete("Sara")   b. student_names.pop("Sara")
   c. student_names.remove("Sara")   d. student_names.discard("Sara")
160. Which of the following methods sorts a list in ascending order in Python?
   a. sort()   b. order()   c. arrange()   d. sorted()
161. Which operation extracts a specific portion such as items from index 2 to 5) of a list in Python?
   a. Indexing   b. Appending   c. Slicing   d. Sorting
162. Which operator is used to combine two lists in Python?
   a. *   b. -   c. +   d. &



163. Which of the following statements is correct for list concatenation?
   a. list1.add(list2)   b. list1.append(list2)
   c. list1 + list2   d. list1.concat(list2)
164. How do you slice a list to get items from index 1 to 3?
   a. list[1:3]   b. list[1:4]
   c. list[0:3]   d. list.slice(1, 3)
165. What does numbers[1:4] return for numbers = [1, 2, 3, 4, 5]?
   a. [2, 3, 4]   b. [1, 2, 3]   c. [2, 3, 4, 5]   d. [1, 2, 3, 4]
166. Which Python data structure is immutable?
   a. List   b. Dictionary   c. Tuple   d. Set
167. How is a tuple created in Python?
   a. Using curly braces {}   b. Using square brackets []
   c. Using parentheses ()   d. Using angle brackets <>
168. Which of the following is a correct way to create a tuple in Python?
   a. my_tuple =[1, 2, 3]   b. my_tuple ={1, 2, 3}
   c. my_tuple =(1, 2, 3)   d. my_tuple =<1, 2, 3>
169. How is the second element of a tuple accessed?
   a. tuple[2]   b. tuple(1)
   c. tuple[1]   d. tuple.get(2)
170. What is the output of my_tuple[3] if my_tuple = (1, 2, 3, "Hello", 4.5)?
   a. 3   b. 4.5   c. Hello   d. Error
171. Which of the following operations cannot be performed on a Python tuple?
   a. Indexing   b. Slicing
   c. Modifying an item   d. Getting length
172. What does my_tuple[-1] return?
   a. Last element   b. First element
   c. Middle element   d. Error
173. Given a tuple my_tuple = (10, 20, 30), what does my_tuple[-1] return?
   a. 10   b. 20   c. 30   d. IndexError
174. Which function returns the number of items in a tuple?
   a. count()   b. len()   c. length()   d. size()
175. What does len((1, 2, 3, "Hello", 4.5)) return?
   a. 4   b. 5   c. 6   d. 3
176. Which programming approach specifically focuses on dividing a program into reusable, independent components?
   a. Modular Programming   b. Looping
   c. Procedural Programming   d. Object-Oriented Programming
177. What is the file extension of Python modules?
   a. .java   b. .py   c. .txt   d. .mod
178. A programming technique in which programs are written based on objects is called:
   a. Procedural programming   b. Object-oriented programming
   c. Linear programming   d. Object-Oriented Procedure
179. OOP stands for:
   a. Object-Oriented Programming
   b. Object-Organized Programming
   c. Only-Oriented Python
   d. Logical programming
180. The blueprint or template used to create objects with specific attributes and behaviors in OOP is called:
   a. Method   b. Object   c. Class   d. Module
181. Which keyword is used to define a class in Python?
   a. define   b. object   c. class   d. template
182. Which of the following is the correct way to define a class in Python?
   a. Class ToyCar:   b. class ToyCar()
   c. class ToyCar:   d. def ToyCar:

**183. Which of the following is used to access an attribute of an object?**
a. object(attribute) b. object->attribute
c. object.attribute d. object:attribute

**184. How would you define a color attribute for an object of a ToyCar class?**
a. color = "" b. self.color = color
c. ToyCar.color = color d. def color(self)

**185. What is the term for functions defined inside a class?**
a. Method b. Action c. Process d. Instance

**186. Which method is automatically called when a new object of a class is created?**
a. describe() b. __init__() c. main() d. create()

**187. Which of the following refers the object of the class within its methods?**
a. this b. cls c. self d. init

**188. What does exception handling help achieve in Python programs?**
a. Code reuse b. Fix syntax errors
c. Handle runtime errors d. Slow down execution

**189. Which block is used to test a piece of code for errors in Python?**
a. test b. catch c. try d. check

**190. Which block is used to handle the error in Python?**
a. except b. error c. else d. handle

**191. Which exception is raised when dividing a number by zero in Python?**
a. ValueError b. TypeError
c. ZeroDivisionError d. ArithmeticError

**192. Which function is used to open a file in Python?**
a. file() b. open() c. read() d. load()

**193. Before a file can be used by a program, it must be________.**
a. Formatted b. Encrypted c. Closed d. Opened

**194. Which method is used to read the entire contents of a file?**
a. readall() b. readline() c. read() d. readfile()

**195. What is the correct mode to open a file for reading?**
a. "r" b. "w" c. "a" d. "x"

**196. What happens when you open a file in "w" mode?**
a. It appends data b. It reads the file
c. It overwrites the file d. It deletes the file

**197. The ________ statement in Python automatically closes the file after the indented block of code executes.**
a. open b. if c. with d. def

**198. When a file is opened in this mode, any new data will overwrite the existing content.**
a. append mode b. Output mode
c. Write mode d. Read-only mode

**199. Which method is used to write data into a file in Python?**
a. writeline() b. print()
c. write() d. put()

**200. When a file is opened in this mode, data will be written at the end of the file's existing contents.**
a. append mode b. Output mode
c. Backup mode d. Read-only mode

**201. Which Python module is commonly used for unit testing?**
a. pytest b. unittest
c. debugger d. testpy

**202. Which type of testing ensures that new changes do not break existing functionality?**
a. Unit Testing b. Regression Testing
c. Speed Testing d. Performance Testing

**203. Which debugging tool is built into Python?**
a. gdb b. pdb c. jdb d. rdb



**204. Which type of testing checks individual components of the code such as functions or classes in isolation?**
a. Integration Testing b. Regression Testing
c. Unit Testing d. Functional Testing

**205. Which testing type checks the software from the user's perspective?**
a. Integration Testing b. Debug Testing
c. Unit Testing d. Functional Testing

**206. Which of the following is not a type of testing?**
a. Regression Testing b. Print Testing
c. Unit Testing d. Functional Testing

**Answers**

| | | | | | |
|---|---|---|---|---|---|
| 1. c | 2. d | 3. a | 4. d | 5. c | 6. a |
| 7. c | 8. c | 9. b | 10. a | 11. a | 12. c |
| 13. c | 14. c | 15. b | 16. c | 17. d | 18. b |
| 19. a | 20. d | 21. b | 22. b | 23. b | 24. c |
| 25. d | 26. b | 27. c | 28. b | 29. b | 30. c |
| 31. c | 32. d | 33. d | 34. c | 35. b | 36. b |
| 37. b | 38. b | 39. c | 40. a | 41. b | 42. b |
| 43. c | 44. c | 45. c | 46. d | 47. d | 48. c |
| 49. d | 50. b | 51. c | 52. b | 53. d | 54. c |
| 55. b | 56. c | 57. c | 58. b | 59. b | 60. a |
| 61. d | 62. c | 63. d | 64. c | 65. a | 66. a |
| 67. b | 68. b | 69. a | 70. d | 71. c | 72. b |
| 73. b | 74. b | 75. c | 76. c | 77. d | 78. b |
| 79. c | 80. c | 81. b | 82. b | 83. a | 84. c |
| 85. b | 86. c | 87. b | 88. c | 89. c | 90. b |
| 91. d | 92. b | 93. d | 94. b | 95. c | 96. c |
| 97. d | 98. b | 99. b | 100.b | 101.c | 102.a |
| 103.c | 104.b | 105.b | 106.b | 107.a | 108.d |
| 109.c | 110.b | 111.b | 112.b | 113.b | 114.a |
| 115.d | 116.b | 117.b | 118.b | 119.c | 120.a |
| 121.a | 122.a | 123.b | 124.b | 125.b | 126.d |
| 127.b | 128.d | 129.a | 130.a | 131.c | 132.a |
| 133.c | 134.c | 135.b | 136.a | 137.a | 138.c |
| 139.c | 140.c | 141.b | 142.a | 143.b | 144.b |
| 145.c | 146.d | 147.c | 148.d | 149.b | 150.c |
| 151.b | 152.b | 153.c | 154.d | 155.d | 156.a |
| 157.b | 158.b | 159.c | 160.a | 161.c | 162.c |
| 163.c | 164.b | 165.a | 166.c | 167.c | 168.c |
| 169.c | 170.c | 171.c | 172.a | 173.c | 174.c |
| 175.b | 176.a | 177.b | 178.b | 179.b | 180.c |
| 181.c | 182.c | 183.c | 184.b | 185.a | 186.b |
| 187.c | 188.c | 189.c | 190.a | 191.c | 192.b |
| 193.d | 194.c | 195.a | 196.c | 197.c | 198.c |
| 199.c | 200.a | 201.b | 202.b | 203.b | 204.c |
| 205.d | 206.b | | | | |