# Class 11 Computer Chapter 2: Python Programming

## EXERCISE QUESTIONS - COMPLETE GUIDE

## MULTIPLE CHOICE QUESTIONS

**Q1** **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

✓ **Correct Answer:** c) Check "Add Python to PATH"

**Q2** **A valid variable name in Python is:**

a variable1
b 1 variable
c variable-name
d variable name

✓ **Correct Answer:** a) variable1

**Q3** **Output of the following piece of code is:**

PYTHON

```
Age = 25
print("Age:", age)
```

a Age: 25
b 25
c Age
d age

✓ **Correct Answer:** a) Age: 25

**Q4** **The operator used for exponentiation in Python is:**

a *
b **
c //
d //

✓ **Correct Answer:** b) *

## Q5 A loop used to iterate over a collection such as lists is:

a while
b for
c do-while
d repeat

✓ **Correct Answer:** b) for

## Q6 A range function used to generate a sequence of numbers:

a Generates a list of numbers
b Creates a sequence of numbers
c Calculates the sum of numbers
d Prints a range of numbers

✓ **Correct Answer:** b) Creates a sequence of numbers

## Q7 A keyword used to define a function in Python:

a define
b function
c def
d func

✓ **Correct Answer:** c) def

## Q8 The Output of the following code is:

PYTHON

```
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

✓ **Correct Answer:** b) Warm

**Q9** **The operation used to combine two lists in Python:**

a combine()
b concat()
c +
d merge()

✓ **Correct Answer:** c) +

## Summary of Correct Answers:

| Q.No. | Correct Option |
|---|---|
| 1 | c) Check "Add Python to PATH" |
| 2 | a) variable1 |
| 3 | a) Age: 25 |
| 4 | b) ** |
| 5 | b) for |
| 6 | b) Creates a sequence of numbers |
| 7 | c) def |
| 8 | b) Warm |
| 9 | c) + |

# SHORT ANSWER QUESTIONS

**Q1** **Explain the purpose of using comments in Python code.**

**ANSWER**

## Comments

Lines that are not executed by the Python interpreter. They are used to provide explanations or notes for the code.

- **Single-line comments:** Start with # (e.g., # This is a comment).
- **Multi-line comments:** Use triple quotes """ or ''' (e.g., """This is a multi-line comment""").

### Benefits:

- Improves readability
- Aids collaboration
- Helps debugging

**Q2** **Describe the difference between integer and float data types in Python. Provide an example of each.**

ANSWER

### Difference Between Integer and Float Data Types

| Feature | Integer (int) | Float (float) |
|---|---|---|
| What it is | A whole number (no decimals) | A number with decimal points or scientific notation |
| Examples | 5, -10, 100 | 3.14, -0.5, 5e3 |
| Decimal point? | No | Yes |
| Used for | Exact values like counting items | Measurements, fractions, or precise calculations |
| Precision | Always exact | May have limited precision (due to how computers handle decimals) |
| Type in Python | <class 'int'> | <class 'float'> |

### Examples:

PYTHON

```
# Integer example
age = 25
print(type(age))  # Output: <class 'int'>
# Float example
price = 99.99
print(type(price))  # Output: <class 'float'>
```

**Q3** **Define operator precedence and give an example of an expression where operator precedence affects the result.**

ANSWER

### Operator Precedence

Operator precedence determines the order in which operators are evaluated in an expression. Operators with higher precedence (e.g., *, /) are evaluated before those with lower precedence (e.g., +, -).

### Example:

```python
result = 2 + 3 * 4
print(result)  # Output: 14
Explanation: * has higher precedence than +, so 3 * 4 (12) is evaluated first, then 2 + 12 yields 14.
Using parentheses (2 + 3) * 4 would yield 20, overriding precedence.
```

**Q4** **How does the short-hand if-else statement differ from the regular if-else statement?**

**ANSWER**

### Short-Hand If-Else (Ternary Operator)

A concise, one-line syntax for simple conditional expressions.

### Syntax:

```python
value_if_true if condition else value_if_false
```

### Example:

```python
status = "Adult" if age >= 18 else "Minor"
```

### Regular If-Else

A multi-line block for complex conditions or multiple statements.

### Example:

```python
if age >= 18:
status = "Adult"
else:
status = "Minor"
```

### Difference:

| Feature | Short-hand If-Else | Regular If-Else |
|---|---|---|
| Syntax | One line | Multi-line block |
| Complexity | Limited to single expressions | Supports multiple statements and complex logic |
| Readability | Concise for simple conditions | Clearer for complex logic |

**Q5** **Explain the use of the range() function in a for loop.**

**ANSWER**

The range() function generates a sequence of numbers, commonly used in for loops to iterate a specific number of times.

**Syntax:** range(start, stop, step)

- start is inclusive
- stop is exclusive

### Example:

PYTHON

```
for i in range(1, 5, 1):  # Iterates from 1 to 4
print(i)  # Output: 1, 2, 3, 4
```

### Use:

- Controls loop iterations
- Allows precise indexing or repetition
- Default start is 0, step is 1 if omitted

**Q6** **Explain how default parameters work in Python functions.**

**ANSWER**

Default parameters allow functions to have optional arguments with predefined values, used if no value is provided during the call.

### Example:

```python
def greet(name, message="Hello"):
return f"{message}, {name}!"
print(greet("Shahjee"))        # Output: Hello, Shahjee!
print(greet("Arshad", "Hi"))   # Output: Hi, Arshad!
```

### How They Work:

- If the argument is omitted, the default value is used
- Default parameters must follow non-default parameters in the function definition

**Q7** **Explain why modular programming is useful in Python.**

**ANSWER**

Modular programming involves breaking code into reusable, independent modules or functions.

### Benefits:

- **Reusability:** Modules can be imported and reused across projects (e.g., math module)
- **Maintainability:** Smaller, focused modules are easier to debug and update
- **Organization:** Code is structured, reducing complexity
- **Collaboration:** Teams can work on separate modules

### Example:

Creating a module utils.py with functions and importing it with import utils.

**Q8** **Explain the difference between a class and an object in Python.**

**ANSWER**

| Feature | Class | Object |
|---|---|---|
| What it is | A blueprint or template for creating objects | An instance of a class (a real example) |
| Purpose | Defines what the object will have (variables) and can do (methods) | Uses the blueprint to create a real thing with actual data |
| Analogy | Like a house design on paper | Like the actual house built from that design |
| Example | class Car: defines color, model, speed, etc. | my_car = Car() – this is one real car object |

### Class Example:

```python
class Dog:
def __init__(self, name):
self.name = name
def bark(self):
return f"{self.name} says Woof!"
```

### Object Example:

```python
my_dog = Dog("Buddy")  # Object creation
print(my_dog.bark())   # Output: Buddy says Woof!
```

### Difference:

- A class defines the structure
- An object is a specific instance with actual data
- Multiple objects can be created from one class

## LONG QUESTIONS

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

### (a) (18 / 3 + 4**2) - (2*(7 - 3)) / (97 + 4)

**Note:** (9 7,4) is not a valid operation. It seems like 97 + 4; so this would cause an error in Python. We will solve it by considering 97 + 4.

### Step-by-step:

- 18 / 3 = 6
- 4**2 = 16
- 6 + 16 = 22
- 7 - 3 = 4
- 2 * 4 = 8
- 97 + 4 = 101
- 22 - 8/101

- 8/101 = 0.0792
- 22 - 0.0792 = 21.9208

✓ **Answer:** 21.9208

### (b) (25 + 3*4**2 - 6) / (2**3 + 1) - 7

### Step-by-step:

- 4**2 = 16
- 3 * 16 = 48
- 25 + 48 - 6 = 67
- 2**3 = 8
- 8 + 1 = 9
- 67 / 9 = 7.444
- 7.444 - 7 = 0.444

✓ **Answer:** 0.444 (or exactly 4/9 in fraction form)

### (c) (12 + 6*(5 - 2))**2 / ((4**2 - 7) + 10)

### Step-by-step:

- 5 - 2 = 3
- 6 * 3 = 18
- 12 + 18 = 30
- 30**2 = 900
- 4**2 = 16
- 16 - 7 = 9
- 9 + 10 = 19
- 900 / 19 = 47.368

✓ **Answer:** 47.368

### (d) 45 / (2**2 + 3*4) + 8*(7 - 3)

### Step-by-step:

- 2**2 = 4
- 3 * 4 = 12
- 4 + 12 = 16
- 45 / 16 = 2.8125
- 7 - 3 = 4
- 8 * 4 = 32
- 2.8125 + 32 = 34.8125

✓ **Answer:** 34.8125

## 2 Translating Mathematical Expressions to Python Syntax

### a) Mathematical Expression: 5 × (3 + 2²) over (6 - 2 × 3)

### Python Syntax:

```python
(5 * (3 + 2**2)) / (6 - 2 * 3)
```

### b) Mathematical Expression: 7 + 2²

### Python Syntax:

```python
7 + 2**2
```

## 3 Explain the concept of variables in Python.

**ANSWER**

### Variables in Python

A variable is a named container used to store data in a computer's memory. In Python, variables are created when you assign a value to them using the assignment operator (=).

## Key Concepts:

### 1. Dynamic Typing

Python is dynamically typed, meaning you don't need to declare the data type of a variable. The type is determined automatically based on the value assigned.

```python
x = 10          # x is an integer
x = "Hello"     # x is now a string
```

### 2. Naming Rules

- Must start with a letter (a-z, A-Z) or underscore _
- Can contain letters, digits, and underscores
- Case-sensitive (age and Age are different)
- Cannot use Python keywords (e.g., if, for, while)

### 3. Assignment

```python
name = "John"     # String variable
age = 25          # Integer variable
price = 99.99     # Float variable
is_student = True # Boolean variable
```

### 4. Multiple Assignment

```python
a, b, c = 10, 20, 30     # Assign multiple values
x = y = z = 100          # Assign same value to multiple variables
```

### 5. Variable Scope

- **Local variables:** Defined inside functions
- **Global variables:** Defined outside functions

```python
x = 10  # Global variable
def my_function():
y = 20  # Local variable
print(x)  # Access global variable
```

## Benefits of Variables:

- Store and reuse values
- Make code more readable
- Allow dynamic data manipulation
- Reduce code repetition

**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.**

**ANSWER**

PYTHON

```
# Take input from the user
number = float(input("Enter a number: "))
# Check if the number is positive, negative, or zero
if number > 0:
print("The number is positive.")
elif number < 0:
print("The number is negative.")
else:
print("The number is zero.")
```

## Program Output Examples:

```
Enter a number: 7
The number is positive.
Enter a number: -3
The number is negative.
Enter a number: 0
The number is zero.
```

## Explanation:

- The program takes a number input from the user
- The float() function converts the input to a floating-point number
- The if-elif-else statement checks the number:
- If number > 0, it's positive
- If number < 0, it's negative
- Otherwise (if neither condition is true), it's zero

**5** **Write a Python program using a while loop that prints all the odd numbers between 1 and 100. Also, count and print the total number of odd numbers.**

ANSWER

PYTHON

```
# Initialize variables
number = 1
count = 0
# While loop to print odd numbers between 1 and 100
while number <= 100:
if number % 2 != 0:  # Check if the number is odd
print(number)    # Print the odd number
count += 1       # Increment the count of odd numbers
number += 1          # Move to the next number
# Print the total number of odd numbers
print("Total number of odd numbers:", count)
```

## Program Output (Partial):

```
1
3
5
7
9
11
...
95
97
99
Total number of odd numbers: 50
```

## Explanation:

- **Initialize variables:**
- number = 1 - Start from 1
- count = 0 - Counter for odd numbers
- **While loop condition:** while number <= 100 - Loop continues until number reaches 100
- **Check for odd numbers:** if number % 2 != 0 - Checks if number is not divisible by 2
- **Print and count:** Print the odd number and increment the counter
- **Increment number:** number += 1 - Move to the next number
- **Final output:** Display the total count of odd numbers

### Alternative Solution using range:

```python
# Initialize counter
count = 0
# Using range with step of 2 to get odd numbers
for number in range(1, 100, 2):
print(number)
count += 1
print("Total number of odd numbers:", count)
```

## Chapter Summary - Quick Reference

| Topic | Key Points |
|---|---|
| Python Installation | Check "Add Python to PATH" for command line access |
| Variable Names | Must start with letter or underscore, no spaces, case-sensitive |
| Data Types | int (whole numbers), float (decimal numbers), str (strings), bool (True/False) |
| Operators | + (addition), - (subtraction), * (multiplication), ** (exponentiation), / (division), // (floor division), % (modulus) |
| Operator Precedence | ** → *, /, //, % → +, - |
| Conditional Statements | if, elif, else; ternary operator for concise conditions |
| Loops | for (iteration over sequences), while (condition-based repetition) |
| range() Function | Generates sequence of numbers: range(start, stop, step) |
| Functions | Defined using def keyword; can have default parameters |
| Modular Programming | Breaking code into reusable modules; benefits: reusability, maintainability, organization |
| Classes & Objects | Class = blueprint; Object = instance of a class |
| Comments | Single-line (#), Multi-line (""" or ''') |

*End of Chapter 2 Exercise Questions*