OOPS & Python Programming (4351108) - Winter 2023 Solution

Milav Dabgar

December 06, 2023

Question 1(a) [3 marks]

List any 6 applications of Python programming language.

Solution

Table 1. Python Applications

Application Area	Description
Web Development	Django, Flask frameworks
Data Science	Analysis and visualization
Machine Learning	AI model development
Desktop Applications	GUI using Tkinter, PyQt
Game Development	Pygame library
Automation	Scripting and testing

Mnemonic

"Web Data Machine Desktop Game Auto"

Question 1(b) [4 marks]

List any 8 features of Python programming language.

Solution

Table 2. Python Features

Feature	Description
Simple Syntax	Easy to read and write
Interpreted	No compilation needed
Object-Oriented	Supports OOP concepts
Dynamic Typing	Variables don't need type declaration
Cross-Platform	Runs on multiple OS
Large Libraries	Rich standard library
Open Source	Free to use and modify
Interactive	REPL environment

Mnemonic

"Simple Interpreted Object Dynamic Cross Large Open Interactive"

Question 1(c) [7 marks]

Explain working of for and while loops in Python.

Solution

For Loop:
- **Iteration**: Repeats over sequences (lists, strings, ranges)
- **Syntax**: `for variable in sequence:`
- **Automatic**: Handles iteration automatically

While Loop:
- **Condition-based**: Continues while condition is true
- **Manual control**: Programmer controls iteration
- **Risk**: Can create infinite loops if condition never becomes false

Loop Logic Diagram:

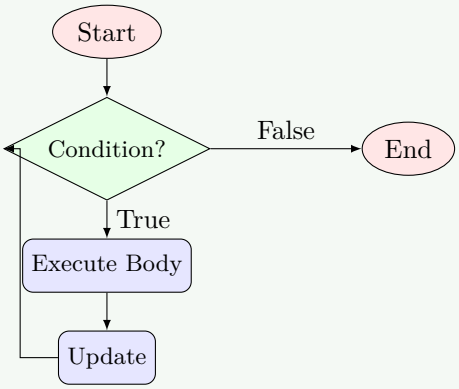

Figure 1. General Loop Flow

Code Example:

```
# For loop
for i in range(5):
    print(i)

# While loop
i = 0
while i < 5:
    print(i)
    i += 1
```

Mnemonic

"For Automatic, While Manual"

Question 1(c OR) [7 marks]

Explain working of break continue and pass statements in Python.

Solution

Break Statement:

- **Exit**: Terminates the entire loop
- **Usage**: When specific condition is met
- **Effect**: Control moves to next statement after loop

Continue Statement:

- **Skip**: Skips current iteration only
- **Usage**: Skip specific values in iteration
- **Effect**: Moves to next iteration

Pass Statement:

- **Placeholder**: Does nothing, syntactic placeholder
- **Usage**: When syntax requires statement but no action needed
- **Effect**: No operation performed

Control Flow Visualization:

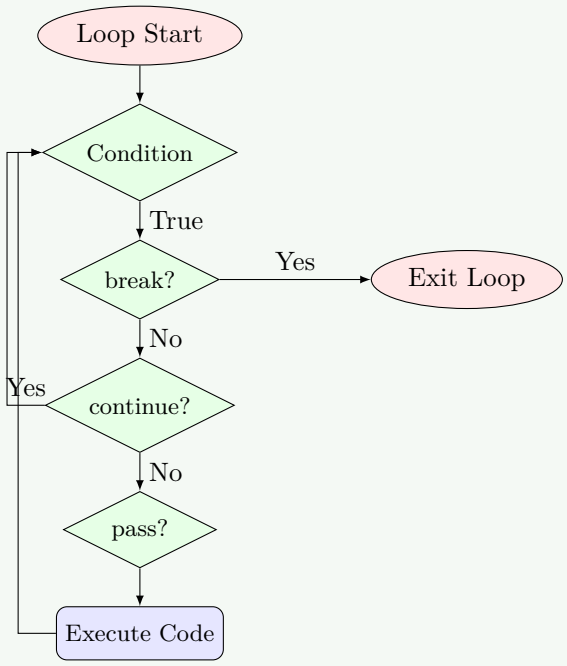

Figure 2. Loop Control Statements

Code Examples:

```python
# Break
for i in range(10):
    if i == 5: break
    print(i)  # 0,1,2,3,4

# Continue
for i in range(5):
    if i == 2: continue
    print(i)  # 0,1,3,4

# Pass
if True: pass  # placeholder
```

Mnemonic

"Break Exits, Continue Skips, Pass Waits"

Question 2(a) [3 marks]

Develop a Python program to increment each element of list by one.

Solution

Code:

```python
# Method 1 - Using for loop
numbers = [1, 2, 3, 4, 5]
for i in range(len(numbers)):
    numbers[i] += 1
print(numbers)

# Method 2 - List comprehension
numbers = [1, 2, 3, 4, 5]
result = [x + 1 for x in numbers]
print(result)
```

Mnemonic

"Loop Index or Comprehension"

Question 2(b) [4 marks]

Develop a Python program to read three numbers from the user and find the average of the numbers.

Solution

Code:

```python
# Input three numbers
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))

# Calculate average
average = (num1 + num2 + num3) / 3

# Display result
print(f"Average is: {average}")
```

Key Points:

- **Input**: Use `float()` for decimal numbers
- **Formula**: Sum divided by count
- **Output**: Use f-string for formatting

Mnemonic

"Input Float, Sum Divide, Format Output"

Question 2(c) [7 marks]

Explain Python's list data type in detail.

Solution

List Characteristics:

- **Ordered**: Elements maintain sequence
- **Mutable**: Can be modified after creation
- **Heterogeneous**: Can store different data types
- **Indexed**: Access elements using index (0-based)

List Operations Table:

Table 3. List Operations

Operation	Syntax	Description
Creation	`list = [1,2,3]`	Create new list
Access	`list[0]`	Get element by index
Append	`list.append(4)`	Add element at end
Insert	`list.insert(1,5)`	Add at specific position
Remove	`list.remove(2)`	Remove first occurrence
Pop	`list.pop()`	Remove and return last
Slice	`list[1:3]`	Get sublist

List Structure Diagram:

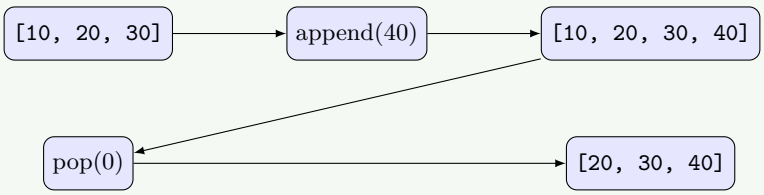

Figure 3. List Mutation

Code Example:

```
# List creation and operations
fruits = ['apple', 'banana', 'orange']
fruits.append('mango')
fruits.insert(1, 'grape')
print(fruits[0])  # apple
print(len(fruits))  # 5
```

Mnemonic

"Ordered Mutable Heterogeneous Indexed"

Question 2(a OR) [3 marks]

Develop a Python program to find sum of all elements in a list using for loop.

Solution

Code:

```
# Method 1 - Traditional for loop
numbers = [10, 20, 30, 40, 50]
total = 0
for num in numbers:
    total += num
```

```python
print(f"Sum is: {total}")

# Method 2 - Using range and index
numbers = [10, 20, 30, 40, 50]
total = 0
for i in range(len(numbers)):
    total += numbers[i]
print(f"Sum is: {total}")
```

Mnemonic

"Initialize Zero, Loop Add, Print Total"

Question 2(b OR) [4 marks]

Develop a Python program to get input from user for principal, rate and no of years then calculate and display simple interest from that.

Solution

Code:

```python
# Get input from user
principal = float(input("Enter principal amount: "))
rate = float(input("Enter rate of interest: "))
time = float(input("Enter time in years: "))

# Calculate simple interest
simple_interest = (principal * rate * time) / 100

# Display results
print(f"Principal: {principal}")
print(f"Rate: {rate}%")
print(f"Time: {time} years")
print(f"Simple Interest: {simple_interest}")
print(f"Total Amount: {principal + simple_interest}")
```

Formula:

- **Simple Interest** = (P × R × T) / 100
- **Total Amount** = Principal + Simple Interest

Mnemonic

"Principal Rate Time, Multiply Divide Hundred"

Question 2(c OR) [7 marks]

Explain Python's tuple data type in detail.

Solution

Tuple Characteristics:

- **Ordered**: Elements maintain sequence
- **Immutable**: Cannot be modified after creation
- **Heterogeneous**: Can store different data types

- **Indexed**: Access using index (0-based)

Tuple Operations Table:

Table 4. Tuple Operations

Operation	Syntax	Description
Creation	`tuple = (1,2,3)`	Create new tuple
Access	`tuple[0]`	Get element by index
Count	`tuple.count(2)`	Count occurrences
Index	`tuple.index(3)`	Find first index
Slice	`tuple[1:3]`	Get sub-tuple
Length	`len(tuple)`	Get tuple size
Concatenate	`tuple1 + tuple2`	Join tuples

Tuple Comparison Diagram:

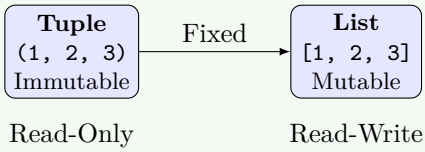

Figure 4. Tuple vs List

Code Example:

```python
# Tuple creation and operations
coordinates = (10, 20, 30)
print(coordinates[0])  # 10
print(len(coordinates))  # 3
x, y, z = coordinates  # tuple unpacking
new_tuple = coordinates + (40, 50)
```

Mnemonic

"Ordered Immutable Heterogeneous Indexed"

Question 3(a) [3 marks]

Explain any 3 random module methods.

Solution

Table 5. Random Module Methods

Method	Syntax	Description
random()	`random.random()`	Float between 0.0 to 1.0
randint()	`random.randint(a,b)`	Integer between a and b
choice()	`random.choice(list)`	Random element from sequence

Code Example:

```python
import random
print(random.random())        # 0.7234567
print(random.randint(1, 10))  # 7
print(random.choice(['r','g','b'])) # g
```

Mnemonic

"Random Float, Randint Integer, Choice Select"

Question 3(b) [4 marks]

Develop a Python program that asks the user for a string and prints out the location of each 'a' in the string.

Solution

Code:

```
# Get string from user
text = input("Enter a string: ")

# Find all positions of 'a'
positions = []
for i in range(len(text)):
    if text[i].lower() == 'a':
        positions.append(i)

# Display results
if positions:
    print(f"Letter 'a' found at positions: {positions}")
else:
    print("Letter 'a' not found in the string")
```

Key Points:

- **Case-insensitive**: Use `.lower()` to find both 'a' and 'A'
- **Index tracking**: Use range or enumerate
- **Output format**: Clear position indication

Mnemonic

"Loop Index Check Append Print"

Question 3(c) [7 marks]

Explain Python's string data type in detail.

Solution

String Characteristics:

- **Immutable**: Cannot be changed after creation
- **Sequence**: Ordered collection of characters
- **Indexed**: Access characters using index
- **Unicode**: Supports all languages and symbols

String Methods Table:

Table 6. String Methods

Method	Example	Description
upper()	`"s".upper()`	Convert to uppercase
lower()	`"S".lower()`	Convert to lowercase
strip()	`" s ".strip()`	Remove whitespace
split()	`"a,b".split(",")`	Split into list
replace()	`"s".replace("s","x")`	Replace substring
find()	`"s".find("e")`	Find substring index

String Structure Diagram:

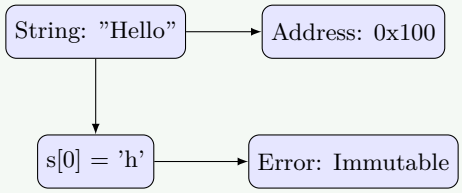

Figure 5. String Immutability

Code Example:

```
name = "Python Programming"
print(name[0])      # P
print(name[0:6])    # Python
message = f"I love {name}"
```

Mnemonic

"Immutable Sequence Indexed Unicode"

Question 3(a OR) [3 marks]

Explain any 3 math module methods.

Solution

Table 7. Math Module Methods

Method	Syntax	Description
sqrt()	`math.sqrt(16)`	Square root calculation
pow()	`math.pow(2,3)`	Power calculation
ceil()	`math.ceil(4.3)`	Round up to integer

Code Example:

```
import math
print(math.sqrt(25))    # 5.0
print(math.pow(2, 3))   # 8.0
print(math.ceil(4.2))   # 5
```

Mnemonic

"Square Root, Power Up, Ceiling Round"

Question 3(b OR) [4 marks]

Develop a Python program to get a string from the user and count total no. of Vowels present in that string.

Solution

Code:

```python
# Get string from user
text = input("Enter a string: ")

# Define vowels
vowels = "aeiouAEIOU"

# Count vowels
vowel_count = 0
for char in text:
    if char in vowels:
        vowel_count += 1

# Display result
print(f"Total vowels in '{text}': {vowel_count}")
```

Mnemonic

"Define Vowels, Loop Check, Count Increment"

Question 3(c OR) [7 marks]

Explain Python's set data type in detail.

Solution

Set Characteristics:

- **Unordered**: No fixed sequence of elements
- **Mutable**: Can add/remove elements
- **Unique**: No duplicate elements allowed
- **Iterable**: Can loop through elements

Set Operations Table:

Table 8. Set Operations

Operation	Syntax	Description
Creation	`set = {1,2,3}`	Create new set
Add	`set.add(4)`	Add single element
Remove	`set.remove(2)`	Remove element
Union	`s1 | s2`	Combine sets
Intersection	`s1 & s2`	Common elements
Difference	`s1 - s2`	Elements in s1 only

Set Venn Diagram:

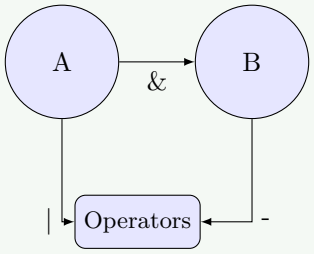

Figure 6. Set Logic

Code Example:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
print(A | B)    # Union: {1,2,3,4,5,6}
print(A & B)    # Intersection: {3,4}
```

Mnemonic

"Unordered Mutable Unique Iterable"

Question 4(a) [3 marks]

What is the class in Python. How is it different from an object?

Solution

Class vs Object comparison:

Table 9. Class vs Object

Aspect	Class	Object
Definition	Blueprint/Template	Instance of Class
Memory	No memory allocated	Memory allocated
Existence	Logical entity	Physical entity
Creation	`class` keyword	Constructor

Relationship Diagram:

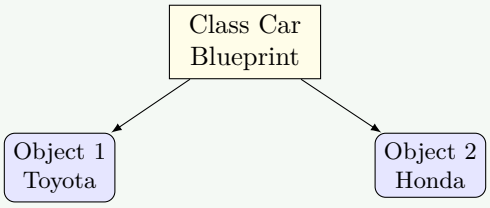

Figure 7. Class to Objects

Mnemonic

"Class Blueprint, Object Instance"

Question 4(b) [4 marks]

Explain any four methods of dictionary data type of Python.

Solution

Dictionary Methods Table:

Table 10. Dictionary Methods

Method	Syntax	Description
keys()	`d.keys()`	Get all keys
values()	`d.values()`	Get all values
items()	`d.items()`	Get key-value pairs
get()	`d.get('k')`	Get value safely

Code Example:

```
student = {'name': 'John', 'grade': 'A'}
print(student.keys())    # ['name', 'grade']
print(student.values())  # ['John', 'A']
print(student.get('age')) # None (no error)
```

Mnemonic

"Keys Values Items Get"

Question 4(c) [7 marks]

Develop a Python program that defines a user-defined module for performing some tasks. Import this module and use its functions.

Solution

Module Structure:

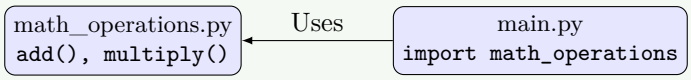

Figure 8. Module Import

Module (math_operations.py):

```
def add(a, b):
    return a + b

def multiply(a, b):
    return a * b

PI = 3.14159
```

Main Program (main.py):

```
import math_operations as mo

res1 = mo.add(5, 3)
res2 = mo.multiply(4, 6)

```

```
print(f"Addition: {res1}")
print(f"Multiplication: {res2}")
print(f"PI: {mo.PI}")
```

Key Points:

- **Module creation**: Separate .py file with functions
- **Import**: `import module` or `from module import func`
- **Usage**: `module.function()`

Mnemonic

"Create Import Use"

Question 4(a OR) [3 marks]

Define types of methods available in Python classes.

Solution

Table 11. Method Types

Type	Decorator	First Argument
Instance	None	`self`
Class	`@classmethod`	`cls`
Static	`@staticmethod`	None

Example:

```
class Demo:
    def inst(self): pass
    @classmethod
    def cls_m(cls): pass
    @staticmethod
    def stat(): pass
```

Mnemonic

"Instance Self, Class Cls, Static None"

Question 4(b OR) [4 marks]

Explain any four methods of string data type of Python.

Solution

String Methods: Checks Counts

Table 12. String Check Methods

Method	Syntax	Description
startswith	`s.startswith('a')`	Starts with substring?
endswith	`s.endswith('z')`	Ends with substring?
isdigit	`s.isdigit()`	All digits?
count	`s.count('a')`	Count occurrences

Code Example:

```
s = "Hello World 123"
print(s.startswith("He")) # True
print(s.endswith("23"))   # True
print("123".isdigit())    # True
print(s.count("l"))       # 3
```

Mnemonic

"Start End Digit Count"

Question 4(c OR) [7 marks]

Develop a Python program to find factorial of a number using recursive user defined function.

Solution

Code:

```
def factorial(n):
    # Base case
    if n == 0 or n == 1:
        return 1
    # Recursive case
    else:
        return n * factorial(n - 1)

try:
    num = int(input("Enter a number: "))
    if num < 0:
        print("Negative number not allowed")
    else:
        print(f"Factorial of {num} is {factorial(num)}")
except ValueError:
    print("Invalid input")
```

Recursion Stack Visualization:

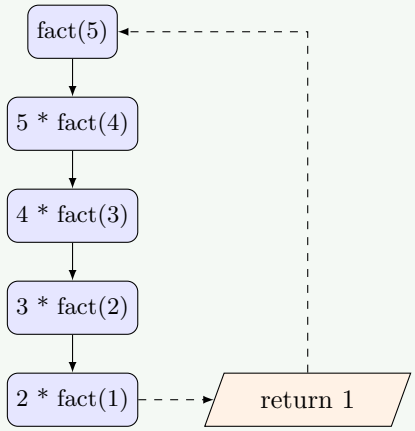

Figure 9. Recursive Calls

Mnemonic

"Base Stop, Recursive Call, Error Check"

Question 5(a) [3 marks]

Develop a python program to Implement single inheritance.

Solution

Code:

```
class Animal:
    def speak(self): print("Animal Speak")

class Dog(Animal):
    def bark(self): print("Dog Bark")

d = Dog()
d.speak()  # Inherited
d.bark()   # Own
```

Inheritance Diagram:

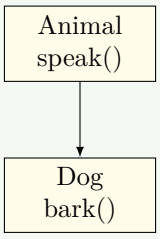

Figure 10. Single Inheritance

Mnemonic

"Parent Child Inherit Override"

Question 5(b) [4 marks]

Explain the significance of constructors in Python classes.

Solution

Constructor Significance Table:

Table 13. Constructor Features

Aspect	Description
Initialization	Called automatically on creation
Setup	Set initial attribute values
Memory	Allocate memory for object
Validation	Validate inputs

Lifecycle Flow:

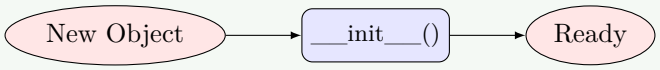

Figure 11. Constructor Flow

Mnemonic

"Initialize Setup Memory Validate"

Question 5(c) [7 marks]

Develop a Python program to demonstrate method overriding using inheritance.

Solution

Code:

```
class Shape:
    def area(self): print("Shape Area")

class Circle(Shape):
    def area(self): print("Circle Area")

class Rect(Shape):
    def area(self): print("Rect Area")

shapes = [Circle(), Rect()]
for s in shapes:
    s.area()
```

Method Overriding Hierarchy:

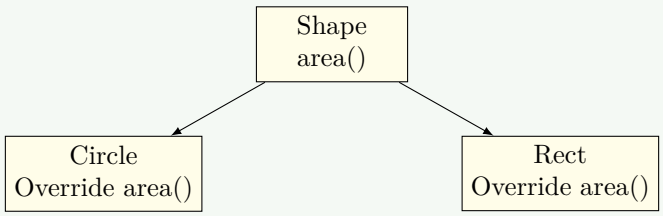

Figure 12. Polymorphism

Key Points:
- **Same Name**: Method name same in parent/child
- **Different Logic**: Child provides specific implementation
- **Runtime Decision**: Object type determines method

Mnemonic

"Same Name Different Logic Runtime Decision"

Question 5(a OR) [3 marks]

Explain concept of data encapsulation in Python.

Solution

Comparison:

Table 14. Encapsulation

Aspect	Description
Definition	Bundling data & methods
Data Hiding	Private attributes (___var)
Access	Public methods (getters/setters)

Encapsulation Diagram:

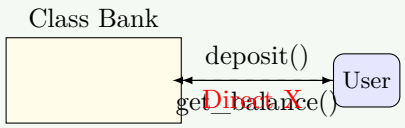

Figure 13. Secure Access

Mnemonic

"Bundle Data Hide Interface"

Question 5(b OR) [4 marks]

Explain concept of abstract classes in Python.

Solution

Abstract Class Properties:
- **Definition**: Class that cannot be instantiated
- **Abstract Methods**: Methods without implementation
- **Implementation**: Child MUST implement these methods
- **Module**: Uses `abc` module

Abstraction Logic:

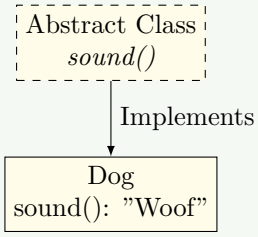

Figure 14. Abstract Base Class

Mnemonic

"Cannot Instantiate Force Implementation Common Interface"

Question 5(c OR) [7 marks]

Develop a python program to Implement multiple inheritance.

Solution

Code:

```
class Father:
    def work(self): print("Father Engineer")

class Mother:
    def work(self): print("Mother Doctor")

class Child(Father, Mother):
    pass

c = Child()
c.work() # Father (MRO)
print(Child.__mro__)
```

Multiple Inheritance Structure:

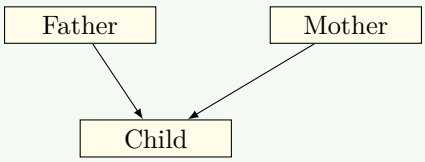

Figure 15. Multiple Parents

Key Points:

- **Structure**: Child inherits from >1 parent
- **MRO**: Method Resolution Order determines priority
- **Diamond Problem**: Solved by C3 linearization

Mnemonic

"Multiple Parents MRO Constructor Diamond"