OOPS & Python Programming (4351108) - Winter 2024 Solution

Milav Dabgar

November 25, 2024

Question 1(a) [3 marks]

List out features of python programming language.

Solution

Table 1. Features of Python

Feature	Description
Simple & Easy	Clean, readable syntax
Free & Open Source	No cost, community driven
Cross-platform	Runs on Windows, Linux, Mac
Interpreted	No compilation needed
Object-Oriented	Supports classes and objects
Large Libraries	Rich standard library

Mnemonic

"Simple Free Cross Interpreted Object Large"

Question 1(b) [4 marks]

Write applications of python programming language.

Solution

Table 2. Python Applications

Application Area	Examples
Web Development	Django, Flask frameworks
Data Science	NumPy, Pandas, Matplotlib
Machine Learning	TensorFlow, Scikit-learn
Desktop GUI	Tkinter, PyQt applications
Game Development	Pygame library
Automation	Scripting and testing

Mnemonic

"Web Data Machine Desktop Game Auto"

Question 1(c) [7 marks]

Explain various datatypes in python.

Solution

Data Type Hierarchy:

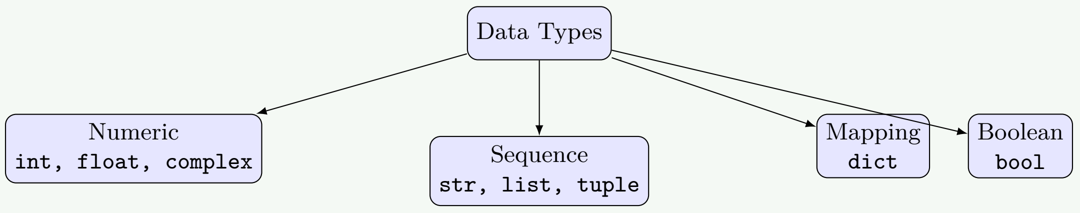


Figure 1. Python Data Types

Table 3. Python Data Types

Data Type	Example	Description
int	`x = 5`	Whole numbers
float	`y = 3.14`	Decimal numbers
str	`name = "John"`	Text data
bool	`flag = True`	True/False values
list	`[1, 2, 3]`	Ordered, mutable
tuple	`(1, 2, 3)`	Ordered, immutable
dict	`{"a": 1}`	Key-value pairs
set	`{1, 2, 3}`	Unique elements

Code Example:

```python
# Numeric types
age = 25          # int
price = 99.99     # float

# Text type
name = "Python"   # str

# Boolean type
is_valid = True   # bool

# Collection types
numbers = [1, 2, 3]         # list
coordinates = (10, 20)      # tuple
student = {"name": "John"} # dict
unique_ids = {1, 2, 3}      # set
```

Mnemonic

"Integer Float String Boolean List Tuple Dict Set"

Question 1(c OR) [7 marks]

Explain arithmetic, assignment, and identity operators with example.

Solution

Arithmetic Operators:

Table 4. Arithmetic Operators

Op	Name	Example
`+`	Addition	`5 + 3 = 8`
`-`	Subtraction	`5 - 3 = 2`
`*`	Multiplication	`5 * 3 = 15`
`/`	Division	`10 / 3 = 3.33`
`//`	Floor Div	`10 // 3 = 3`
`%`	Modulus	`10 % 3 = 1`
`**`	Exponent	`2 ** 3 = 8`

Assignment Operators:

Table 5. Assignment Operators

Op	Example	Equivalent
`=`	`x = 5`	Assign value
`+=`	`x += 3`	`x = x + 3`
`-=`	`x -= 2`	`x = x - 2`
`*=`	`x *= 4`	`x = x * 4`

Identity Operators:

Table 6. Identity Operators

Op	Purpose	Example
`is`	Same object	`x is y`
`is not`	Different object	`x is not y`

Code Example:

```python
# Arithmetic
a = 10 + 5    # 15
b = 10 // 3   # 3

# Assignment
x = 5
x += 3        # x becomes 8

# Identity
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(list1 is list2)      # False
print(list1 is not list2)  # True
```

Mnemonic

"Add Assign Identity"

Question 2(a) [3 marks]

Which of the following identifier names are invalid? (i) Total Marks (ii) Total_Marks (iii)

total-Marks (iv) Hundred$ (v) _Percentage (vi) True

Solution

Table 7. Identifier Validity

Identifier	Status	Reason
Total Marks	**Invalid**	Contains space
Total_Marks	Valid	Underscore allowed
total-Marks	**Invalid**	Hyphen not allowed
Hundred$	**Invalid**	$ symbol not allowed
_Percentage	Valid	Can start with underscore
True	**Invalid**	Reserved keyword

Invalid identifiers: Total Marks, total-Marks, Hundred$, True

Mnemonic

"Space Hyphen Dollar Keyword = Invalid"

Question 2(b) [4 marks]

Write a program to find a maximum number among the given three 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: "))

# Find maximum using if-elif-else
if num1 >= num2 and num1 >= num3:
    maximum = num1
elif num2 >= num1 and num2 >= num3:
    maximum = num2
else:
    maximum = num3

# Display result
print(f"Maximum number is: {maximum}")
```

Alternative using max() function:

```python
num1, num2, num3 = map(float, input("Enter 3 numbers: ").split())
maximum = max(num1, num2, num3)
print(f"Maximum: {maximum}")
```

Mnemonic

"Input Compare Display"

Question 2(c) [7 marks]

Explain dictionaries in Python. Write statements to add, modify, and delete elements in a dictionary.

Solution

Dictionary Definition: A dictionary is a collection of key-value pairs that is ordered, changeable, and does not allow duplicate keys.

Dictionary Operations:

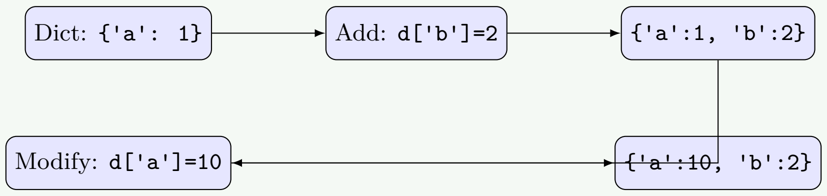


Figure 2. Dictionary Operations

Table 8. Dictionary Methods

Operation	Syntax	Example
Create	`dict = {}`	`student = {}`
Add	`d[k] = v`	`student['name'] = 'John'`
Modify	`d[k] = new`	`student['name'] = 'Jane'`
Delete	`del d[k]`	`del student['name']`
Access	`d[k]`	`print(student['name'])`

Code Example:

```python
# Create empty dictionary
student = {}

# Add elements
student['name'] = 'John'
student['age'] = 20

# Modify element
student['age'] = 21

# Delete element
del student['name']

# Display dictionary
print(student)  # Output: {'age': 21}
```

Mnemonic

"Key-Value Ordered Changeable Unique"

Question 2(a OR) [3 marks]

Write a program to display the following pattern.

Solution

Pattern:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Code:

```python
# Pattern program
for i in range(1, 6):
    for j in range(1, i + 1):
        print(j, end=" ")
    print()  # New line after each row
```

Mnemonic

"Outer Row Inner Column Print"

Question 2(b OR) [4 marks]

Write a program to find the sum of digits of an integer number, input by the user.

Solution

Code:

```python
# Input number from user
number = int(input("Enter a number: "))
original_number = number
sum_digits = 0

# Extract and sum digits
while number > 0:
    digit = number % 10    # Get last digit
    sum_digits += digit    # Add to sum
    number = number // 10  # Remove last digit

# Display result
print(f"Sum of digits of {original_number} is: {sum_digits}")
```

Alternative Method:

```python
number = input("Enter number: ")
sum_digits = sum(int(digit) for digit in number)
print(f"Sum of digits: {sum_digits}")
```

Mnemonic

"Input Extract Sum Display"

Question 2(c OR) [7 marks]

Explain slicing and concatenation operation on list.

Solution

List Slicing: Extracting portion of list using `[start:stop:step]` syntax.
Slicing Visualization:

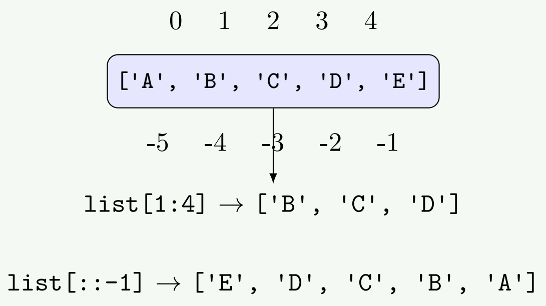

0 1 2 3 4
['A', 'B', 'C', 'D', 'E']
-5 -4 -3 -2 -1
list[1:4] → ['B', 'C', 'D']
list[::-1] → ['E', 'D', 'C', 'B', 'A']

Figure 3. List Indexing & Slicing

Operations Table:

Table 9. List Operations

Syntax	Description	Example
`l[start:stop]`	Elements from start to stop-1	`nums[1:4]`
`l[:stop]`	From beginning	`nums[:3]`
`l[::step]`	With step	`nums[::2]`
`l1 + l2`	Concatenation	`[1]+[2]`

Code Example:

```python
list1 = [1, 2, 3, 4, 5]
list2 = [6, 7, 8]

# Slicing
print(list1[1:4])    # [2, 3, 4]
print(list1[::-1])   # [5, 4, 3, 2, 1]

# Concatenation
result = list1 + list2  # [1, 2, 3... 8]
list1.extend(list2)     # Modifies list1
```

Mnemonic

"Slice Extract Concat Join"

Question 3(a) [3 marks]

Define a list in Python. Write name of the function used to add an element to the end of a list.

Solution

List Definition: A **list** is an ordered collection of items that is changeable and allows duplicate values.
Properties:

- **Ordered**: Items have defined order
- **Changeable**: Can modify after creation
- **Duplicates**: Allows duplicate values

Function to add element: `append()`

Example:

```python
fruits = ['apple', 'banana']
fruits.append('orange')
print(fruits)  # ['apple', 'banana', 'orange']
```

Mnemonic

"List Append End"

Question 3(b) [4 marks]

Define a tuple in Python. Write statement to access last element of a tuple.

Solution

Tuple Definition: A **tuple** is an ordered collection of items that is unchangeable and allows duplicate values.
Accessing Last Element:

```python
my_tuple = (10, 20, 30, 40, 50)

# Method 1: Negative index
last = my_tuple[-1]  # 50

# Method 2: Lens
last = my_tuple[len(my_tuple) - 1] # 50
```

Mnemonic

"Tuple Unchangeable Negative Index"

Question 3(c) [7 marks]

Write statements for following set operations: create empty set, add an element to a set, remove an element from set, Union of two sets, Intersection of two sets, Difference between two sets and symmetric difference between two sets.

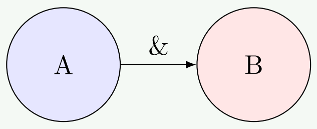
Solution

Set Operations Table:

Table 10. Set Operations

Operation	Method/Op	Example
Create Empty	`set()`	`s = set()`
Add	`add()`	`s.add(5)`
Remove	`remove()`	`s.remove(5)`
Union	`union() |`	`A | B`
Intersection	`intersection() &`	`A & B`
Difference	`difference() -`	`A - B`
Symm. Diff	`sym_diff() ^`	`A ^ B`

Set Venn Diagram:



Union: All Intersect: Common

Figure 4. Set Relations

Code Example:

```python
s = set()
s.add(10)
s.remove(10)

A = {1, 2, 3}
B = {3, 4, 5}
print(A | B)  # {1, 2, 3, 4, 5}
print(A & B)  # {3}
print(A - B)  # {1, 2}
print(A ^ B)  # {1, 2, 4, 5}
```

Mnemonic

"Create Add Remove Union Intersect Differ Symmetric"

Question 3(a OR) [3 marks]

Define a string in Python. Using example illustrate (i) How to create a string. (ii) Accessing individual characters using indexing.

Solution

String Definition: A string is a sequence of characters enclosed in quotes (single, double, or triple).
String Structure:

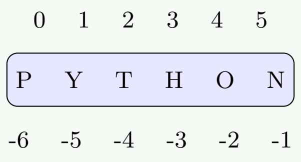


Figure 5. String Indexing

Code:

```python
# Creation
s1 = 'Hello'
s2 = "World"

# Accessing
word = "PYTHON"
print(word[0])   # P
print(word[-1])  # N
```

Mnemonic

"String Quotes Index Access"

Question 3(b OR) [4 marks]

Explain list traversing using for loop and while loop.

Solution

List Traversing means visiting each element of list one by one.
Comparison:

Table 11. Loops Comparison

For Loop	While Loop
Simpler syntax	More control
Best for fixed iterations	Best for condition-based

Code Example:

```python
nums = [10, 20, 30]

# For Loop
for x in nums:
    print(x)

# While Loop
i = 0
while i < len(nums):
    print(nums[i])
    i += 1
```

Mnemonic

"For Simple While Control"

Question 3(c OR) [7 marks]

Write a program to create a dictionary with the roll number, name, and marks of n students and display the names of students who have scored marks above 75.

Solution

Code:

```python
# Input number of students
n = int(input("Enter number of students: "))
students = {}

# Input data
for i in range(n):
    print(f"\nStudent {i + 1}:")
    roll = int(input("Roll: "))
    name = input("Name: ")
    marks = float(input("Marks: "))

    students[roll] = {'name': name, 'marks': marks}

# Display high performers
print("\nStudents with marks > 75:")
found = False
for roll, data in students.items():
    if data['marks'] > 75:
        print(f"Name: {data['name']}, Marks: {data['marks']}")
        found = True

if not found:
    print("None found")
```

Mnemonic

"Input Store Filter Display"

Question 4(a) [3 marks]

Write any three functions available in random module. Write syntax and example of each function.

Solution

Table 12. Random Functions

Function	Description	Example
`random()`	Float 0.0 to 1.0	`0.75`
`randint(a,b)`	Integer a to b	`5`
`choice(seq)`	Random element	`'red'`

Code:

```python
import random
print(random.random())
print(random.randint(1, 10))
print(random.choice(['a', 'b', 'c']))
```

Mnemonic

"Random Randint Choice"

Question 4(b) [4 marks]

Write the advantages of functions.

Solution

Advantages:
- **Code Reusability**: Write once, use multiple times.
- **Modularity**: Break complex problems into smaller parts.
- **Debugging**: Easier to isolate and fix errors.

- **Readability**: Code is more organized.

Concept Map:

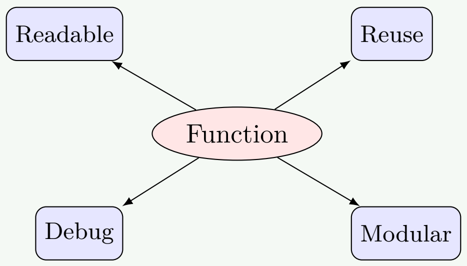


Figure 6. Function Advantages

Mnemonic

"Reuse Modular Debug Read Maintain Avoid"

Question 4(c) [7 marks]

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

Solution

Code:

```python
text = input("Enter a string: ")
positions = []

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

# Display
if positions:
    print(f"'a' found at indices: {positions}")
    for pos in positions:
        print(f"Index {pos}: '{text[pos]}'")
else:
    print("'a' not found")
```

Mnemonic

"Input Loop Check Store Display"

Question 4(a OR) [3 marks]

Explain local and global variables.

Solution

Scope Comparison:

Table 13. Variable Scopes

Type	Scope	Access
Local	Inside function	Function only
Global	Entire program	Everywhere

Scope Visualization:

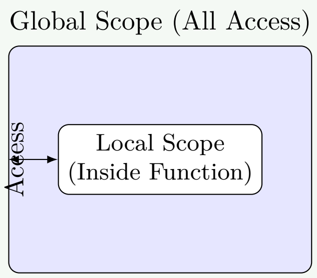


Figure 7. Variable Scope

Code:

```python
g = 10  # Global

def func():
    l = 5    # Local
    print(g) # Access Global
    # global g; g = 20 # To modify
```

Mnemonic

"Local Inside Global Everywhere"

Question 4(b OR) [4 marks]

Explain creation and use of user defined function with example.

Solution

Function Syntax:

```python
def function_name(params):
    """Docstring"""
    # Body
    return value
```

Components: 1. **def**: Keyword 2. **Name**: Identifier 3. **Parameters**: Inputs 4. **Return**: Output
Example:

```python
def greet(name):
    return f"Hello {name}"

msg = greet("John")
print(msg)
```

Mnemonic

"Define Call Return Parameter"

Question 4(c OR) [7 marks]

Write a program to create a user defined function calcFact() to calculate and display the factorial of a number passed as an argument.

Solution

Code:

```python
def calcFact(n):
    if n < 0:
        return "Undefined"
    elif n == 0 or n == 1:
        return 1
    else:
        fact = 1
        for i in range(2, n + 1):
            fact *= i
        return fact

# Logic
num = int(input("Enter number: "))
print(f"Factorial of {num} is {calcFact(num)}")
```

Recursive Visual:

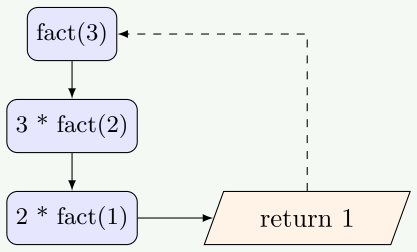


Figure 8. Recursion Stack

Mnemonic

"Define Check Loop Multiply Return"

Question 5(a) [3 marks]

Give difference between class and object.

Solution

Comparison:

Table 14. Class vs Object

Feature	Class	Object
Definition	Blueprint	Instance
Memory	Not allocated	Allocated
Keyword	`class`	Constructor call

Analogy:

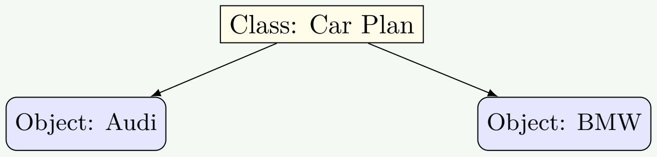


Figure 9. Blueprint vs Instances

Mnemonic

"Class Blueprint Object Instance"

Question 5(b) [4 marks]

State the purpose of a constructor in a class.

Solution

Purpose:
- **Initialize**: Set initial state of object.
- **Automatic**: Called when object is created.
- **Memory**: Allocates required memory.

Lifecycle:

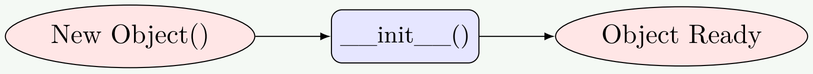


Figure 10. Constructor Flow

Code:

```python
class Demo:
    def __init__(self, val):
        self.val = val

obj = Demo(10) # Calls __init__
```

Mnemonic

"Initialize Automatic Memory Default"

Question 5(c) [7 marks]

Write a program to create a class "Student" with attributes such as name, roll number, and marks. Implement method to display student information. Create object of the student class and show how to use method.

Solution

Code:

```python
class Student:
    def __init__(self, name, roll, marks):
        self.name = name
        self.roll = roll
        self.marks = marks

    def display_info(self):
        print("-" * 20)
        print(f"Name: {self.name}")
        print(f"Roll: {self.roll}")
        print(f"Marks: {self.marks}")
        print("-" * 20)

# Create objects
s1 = Student("John", 101, 85)
s2 = Student("Alice", 102, 90)

# Use method
s1.display_info()
s2.display_info()
```

Output:

```
--------------------
Name: John
Roll: 101
Marks: 85
--------------------
```

Mnemonic

"Class Attributes Constructor Methods Objects"

Question 5(a OR) [3 marks]

State the purpose of encapsulation.

Solution

Encapsulation is bundling data and methods, and restricting direct access to data.
Purpose:

- **Data Hiding**: Protects internal state.
- **Security**: Prevents accidental modification.
- **Controlled Access**: Use getters/setters.

Code:

```python
class Bank:
    def __init__(self):
        self.__bal = 0  # Private

    def deposit(self, amt):
        self.__bal += amt
```

Mnemonic

"Hide Protect Control Secure Modular"

Question 5(b OR) [4 marks]

Explain multilevel inheritance.

Solution

Definition: Chain of inheritance (A ← B ← C).
Diagram:

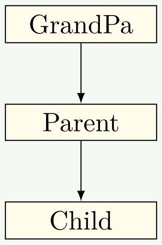


Figure 11. Multilevel Inheritance

Code:

```python
class A: pass
class B(A): pass
class C(B): pass

obj = C() # Has features of A, B, C
```

Mnemonic

"Chain Inherit Level Access"

Question 5(c OR) [7 marks]

Write a Python program to demonstrate working of hybrid inheritance.

Solution

Hybrid Inheritance: Combination of multiple inheritance types (e.g., Diamond problem).
Diagram:

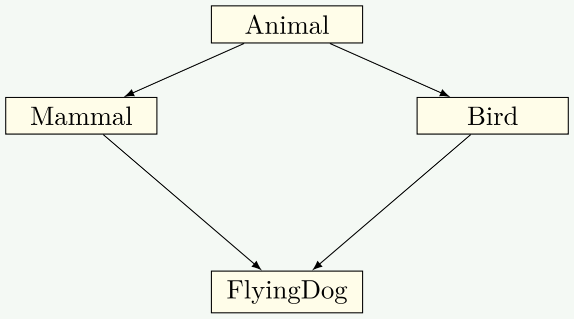


Figure 12. Hybrid Inheritance

Code:

```python
class Animal:
    def __init__(self): print("Animal")

class Mammal(Animal):
    def feed(self): print("Milk")

class Bird(Animal):
    def fly(self): print("Flying")

class FlyingDog(Mammal, Bird):
    def bark(self): print("Bark")

# Object
fd = FlyingDog()
fd.feed()  # Mammal
fd.fly()   # Bird
fd.bark()  # Own
```

Mnemonic

"Hybrid Multiple Single Multilevel Combined"