# Advanced Python Programming (4321602) - Summer 2024 Solution

Milav Dabgar

June 21, 2024

## Question 1

## Question 1(a) [3 marks]

**Give the difference between Tuple and List in python.**

---

**Solution**

**Comparison:**

| Feature | Tuple | List |
|---|---|---|
| **Mutability** | Immutable (cannot be changed) | Mutable (can be changed) |
| **Syntax** | Created using () | Created using [] |
| **Performance** | Faster | Slower |
| **Methods** | Limited methods (count, index) | Many methods (append, remove, etc.) |

- **Memory efficient**: Tuples use less memory than lists
- **Use case**: Tuples for fixed data, lists for dynamic data

**Mnemonic**

Tuples are Tight, Lists are Loose

---

## Question 1(b) [4 marks]

**Define Set and how is it created in python?**

---

**Solution**

**Set** is an unordered collection of unique elements in Python.
**Creating Sets:**

```python
# Empty set
my_set = set()

# Set with elements
fruits = {"apple", "banana", "orange"}

# From list
numbers = set([1, 2, 3, 4])

```

- **Unique elements**: No duplicates allowed

---

- **Unordered**: Elements have no specific order
- **Operations**: Union, intersection, difference supported

> **Mnemonic**
>
> Sets are Special - Unique and Unordered

# Question 1(c) [7 marks]

**What is Dictionary in Python? Write a program to concatenate two dictionary into new one.**

> **Solution**
>
> **Dictionary** is an ordered collection of key-value pairs in Python.
>
> **Program:**
>
> ```python
> # Two dictionaries
> dict1 = {1: 10, 2: 20}
> dict2 = {3: 30, 4: 40}
>
> # Method 1: Using update()
> result1 = dict1.copy()
> result1.update(dict2)
>
> # Method 2: Using ** operator
> result2 = {**dict1, **dict2}
>
> print("Result:", result2)
> # Output: {1: 10, 2: 20, 3: 30, 4: 40}
>
> ```
>
> - **Key-value pairs**: Each element has a key and value
> - **Mutable**: Can be modified after creation
> - **Fast access**: O(1) average time complexity
>
> > **Mnemonic**
> >
> > Dictionaries are Dynamic Key-Value stores

# Question 1(c) OR [7 marks]

**What is a list in python? Write a program that finds maximum and minimum numbers from a list.**

> **Solution**
>
> **List** is an ordered, mutable collection of elements in Python.
>
> **Program:**
>
> ```python
> # Input list
> numbers = [45, 12, 78, 23, 56, 89, 34]
>
> # Find maximum and minimum
> maximum = max(numbers)
> minimum = min(numbers)
> ```

```
7
8    print(f"Maximum: {maximum}")
9    print(f"Minimum: {minimum}")
10
11   # Manual method
12   max_val = numbers[0]
13   min_val = numbers[0]
14   for num in numbers:
15       if num > max_val:
16           max_val = num
17       if num < min_val:
18           min_val = num
19
```

- **Ordered**: Elements maintain insertion order
- **Indexing**: Accessed using index [0, 1, 2...]
- **Built-in functions**: min(), max(), len() available

**Mnemonic**

Lists are Linear and Indexed

# Question 2

## Question 2(a) [3 marks]

**Explain Nested Tuple with example.**

**Solution**

**Nested Tuple** is a tuple containing other tuples as elements.
**Example:**

```
1    # Nested tuple
2    student_data = (
3        ("John", 85, "A"),
4        ("Alice", 92, "A+"),
5        ("Bob", 78, "B")
6    )
7
8    # Accessing elements
9    print(student_data[0][1])   # Output: 85
10   print(student_data[1][0])   # Output: Alice
11
```

- **Multi-dimensional**: Tuples within tuples
- **Indexing**: Use multiple indices [i][j]
- **Immutable**: Cannot change nested elements

**Mnemonic**

Nested means Tuples inside Tuples

## Question 2(b) [4 marks]

**What is random module? Explain with example.**

> **Solution**
>
> **Random module** generates random numbers and performs random operations.
> **Example:**
>
> ```python
> import random
>
> # Random integer
> num = random.randint(1, 10)
> print(f"Random number: {num}")
>
> # Random choice from list
> colors = ["red", "blue", "green"]
> choice = random.choice(colors)
> print(f"Random color: {choice}")
>
> # Random float
> decimal = random.random()
> print(f"Random decimal: {decimal}")
>
> ```
>
> - **Import required**: import random
> - **Various functions**: randint(), choice(), random()
> - **Useful for**: Games, simulations, testing
>
> > **Mnemonic**
> >
> > Random makes things Unpredictable

# Question 2(c) [7 marks]

**Explain different ways of importing package. Give one example of it.**

> **Solution**
>
> **Import Methods:**
>
> | Method | Syntax | Usage |
> |---|---|---|
> | **Normal import** | `import package` | package.function() |
> | **From import** | `from package import function` | function() |
> | **Import all** | `from package import *` | function() |
> | **Alias import** | `import package as alias` | alias.function() |
>
> **Example:**
>
> ```python
> # Normal import
> import math
> result1 = math.sqrt(16)
>
> # From import
> from math import sqrt
> result2 = sqrt(16)
>
> # Import with alias
> import math as m
> result3 = m.sqrt(16)
>
> # Import all (not recommended)
> ```

```
14  from math import *
15  result4 = sqrt(16)
16
```

- **Namespace**: Normal import keeps separate namespace
- **Direct access**: From import allows direct function call
- **Alias**: Shorter names for convenience

> **Mnemonic**
>
> Import methods: Normal, From, All, Alias

## Question 2(a) OR [3 marks]

**Write down the properties of dictionary in python.**

> **Solution**
>
> **Dictionary Properties:**
>
> | Property | Description |
> |---|---|
> | **Ordered** | Maintains insertion order (Python 3.7+) |
> | **Mutable** | Can be modified after creation |
> | **Key-unique** | No duplicate keys allowed |
> | **Heterogeneous** | Keys and values can be different types |
>
> - **Fast access**: O(1) average lookup time
> - **Dynamic size**: Can grow or shrink
> - **Key restrictions**: Keys must be immutable
>
> > **Mnemonic**
> >
> > Dictionaries are Ordered, Mutable, Unique, Heterogeneous

## Question 2(b) OR [4 marks]

**What is the dir() function in python. Explain with example.**

> **Solution**
>
> **dir() function** returns all attributes and methods of an object.
> **Example:**
>
> ```python
> 1   # List all attributes of string
> 2   text = "hello"
> 3   attributes = dir(text)
> 4   print(attributes[:5])  # First 5 attributes
> 5
> 6   # Check available methods
> 7   print("upper" in dir(text))  # True
> 8
> 9   # For modules
> 10  import math
> ```

```
11   math_methods = dir(math)
12   print("sqrt" in math_methods)  # True
13
14   # For custom objects
15   class MyClass:
16       def my_method(self):
17           pass
18
19   obj = MyClass()
20   print(dir(obj))
21
```

- **Introspection**: Examines object properties
- **Debugging**: Helps find available methods
- **All objects**: Works with any Python object

> **Mnemonic**
>
> dir() shows Directory of object attributes

# Question 2(c) OR [7 marks]

**Write a program to define module to find sum of two numbers. Import module to another program.**

> **Solution**
>
> **Module file (calculator.py):**
>
> ```python
> 1   # calculator.py
> 2   def add_numbers(a, b):
> 3       """Function to add two numbers"""
> 4       return a + b
> 5
> 6   def multiply_numbers(a, b):
> 7       """Function to multiply two numbers"""
> 8       return a * b
> 9
> 10  def get_sum(num1, num2):
> 11      """Alternative sum function"""
> 12      result = num1 + num2
> 13      return result
> 14
> ```
>
> **Main program (main.py):**
>
> ```python
> 1   # main.py
> 2   import calculator
> 3
> 4   # Using the module
> 5   result1 = calculator.add_numbers(10, 20)
> 6   print(f"Sum: {result1}")
> 7
> 8   # From import
> 9   from calculator import get_sum
> 10  result2 = get_sum(15, 25)
> 11  print(f"Sum using from import: {result2}")
> 12
> ```

- **Module creation**: Save functions in .py file
- **Import**: Use import statement to access
- **Code reusability**: Use same module in multiple programs

> **Mnemonic**
>
> Modules make code Reusable and Organized

## Question 3

## Question 3(a) [3 marks]

**What is Runtime error and Logical error. Explain with example.**

> **Solution**
>
> Comparison:
>
> | Error Type | Definition | Example |
> |---|---|---|
> | **Runtime Error** | Occurs during program execution | Division by zero, file not found |
> | **Logical Error** | Program runs but gives wrong output | Wrong formula, incorrect condition |
>
> Examples:
>
> ```
> # Runtime Error
> x = 10
> y = 0
> result = x / y  # ZeroDivisionError
>
> # Logical Error
> def calculate_area(radius):
>     return 3.14 * radius  # Should be radius * radius
>
> ```
>
> > **Mnemonic**
> >
> > Runtime Crashes, Logical Confuses

## Question 3(b) [4 marks]

**Write points on Except and explaining it.**

> **Solution**
>
> **Except clause** handles specific exceptions in try-except block.
> **Key Points:**

| Feature | Description |
|---|---|
| **Syntax** | `except ExceptionType:` |
| **Multiple** | Can have multiple except blocks |
| **Generic** | `except:` catches all exceptions |
| **Variable** | `except Exception as e:` stores error |

```
1   try:
2       number = int(input("Enter number: "))
3       result = 10 / number
4   except ValueError:
5       print("Invalid input")
6   except ZeroDivisionError:
7       print("Cannot divide by zero")
8   except Exception as e:
9       print(f"Error: {e}")
10
```

**Mnemonic**

Except Catches and Handles errors

## Question 3(c) [7 marks]

**Write a program to catch Divide by zero Exception. Also use finally block.**

**Solution**

**Program:**

```
1   def safe_division():
2       try:
3           # Get input from user
4           numerator = float(input("Enter numerator: "))
5           denominator = float(input("Enter denominator: "))
6
7           # Perform division
8           result = numerator / denominator
9           print(f"Result: {numerator} / {denominator} = {result}")
10
11      except ZeroDivisionError:
12          print("Error: Cannot divide by zero!")
13          print("Please enter a non-zero denominator")
14
15      except ValueError:
16          print("Error: Please enter valid numbers only")
17
18      except Exception as e:
19          print(f"Unexpected error occurred: {e}")
20
21      finally:
22          print("Division operation completed")
23          print("Thank you for using the calculator")
24
25  # Call the function
26  safe_division()
```

```
27
```

- **Try block**: Contains risky code
- **Except**: Handles ZeroDivisionError specifically
- **Finally**: Always executes regardless of exception

> **Mnemonic**
>
> Try risky code, Except handles errors, Finally always runs

## Question 3(a) OR [3 marks]

**What are the built-in exceptions and gives its types.**

> **Solution**
>
> **Built-in Exception Types:**
>
> | Type | Description | Example |
> |------|-------------|---------|
> | **ValueError** | Invalid value for operation | `int("abc")` |
> | **TypeError** | Wrong data type | `"5" + 5` |
> | **IndexError** | Index out of range | `list[10]` |
> | **KeyError** | Key not found | `dict["missing"]` |
> | **FileNotFoundError** | File doesn't exist | `open("no.txt")` |
>
> > **Mnemonic**
> >
> > Value, Type, Index, Key, File - common error types

## Question 3(b) OR [4 marks]

**Explain Syntax error and how do we identify it? Give an example.**

> **Solution**
>
> **Syntax Error** occurs when Python cannot parse the code due to incorrect syntax.
> **Identification:**
>
> | Method | Description |
> |--------|-------------|
> | **Python interpreter** | Shows error message with line number |
> | **IDE highlighting** | Code editors highlight syntax errors |
> | **Error message** | Points to exact location of error |
>
> **Examples:**
>
> ```python
> # Missing colon
> if x > 5
>     print("Greater")  # SyntaxError
>
> # Unmatched parentheses
> print("Hello"  # SyntaxError
>
> ```

```
8   # Incorrect indentation
9   def my_function():
10  print("Hello")  # IndentationError
11
```

> **Mnemonic**
>
> Syntax errors Stop code from Starting

## Question 3(c) OR [7 marks]

**What is Exception handling in python? Explain with proper example.**

> **Solution**
>
> **Exception Handling** is a mechanism to handle runtime errors gracefully without crashing the program.
> **Program:**
>
> ```python
> 1   def file_processor():
> 2       filename = None
> 3       try:
> 4           filename = input("Enter filename: ")
> 5           with open(filename, 'r') as file:
> 6               content = file.read()
> 7               numbers = [int(x) for x in content.split()]
> 8               average = sum(numbers) / len(numbers)
> 9               print(f"Average: {average}")
> 10
> 11      except FileNotFoundError:
> 12          print(f"Error: File '{filename}' not found")
> 13
> 14      except ValueError:
> 15          print("Error: File contains non-numeric data")
> 16
> 17      except ZeroDivisionError:
> 18          print("Error: No numbers found in file")
> 19
> 20      except Exception as e:
> 21          print(f"Unexpected error: {e}")
> 22
> 23      else:
> 24          print("File processed successfully")
> 25
> 26      finally:
> 27          print("File processing operation completed")
> 28
> 29  # Run the function
> 30  file_processor()
> 31
> ```
>
> > **Mnemonic**
> >
> > Try-Except-Else-Finally: Complete error handling

## Question 4

# Question 4(a) [3 marks]

**What kind of different operations we can perform in a file?**

**Solution**

**File Operations:**

| Operation | Description | Method |
|---|---|---|
| **Read** | Read file content | `read()`, `readline()` |
| **Write** | Write data to file | `write()` |
| **Append** | Add data to end | mode 'a' |
| **Create** | Create new file | mode 'w', 'x' |
| **Delete** | Remove file | `os.remove()` |
| **Seek** | Move file pointer | `seek()` |

**Mnemonic**

Read, Write, Append, Create, Delete, Seek

# Question 4(b) [4 marks]

**Give list of file modes. Write Description of any four mode.**

**Solution**

**File Modes:**

| Mode | Description | Purpose |
|---|---|---|
| **'r'** | Read mode | Read existing file (default) |
| **'w'** | Write mode | Create/Overwrite file |
| **'a'** | Append mode | Add to end of file |
| **'x'** | Exclusive | Create new, fail if exists |
| **'b'** | Binary mode | Binary files |
| **'+'** | Read+Write | Update mode |

**Mnemonic**

Read, Write, Append, eXclusive - main file modes

# Question 4(c) [7 marks]

**Write a program to sort all the words in a file and put it in list.**

**Solution**

**Program:**

```
1  def sort_words_from_file():
2      try:
```

```
3          # Input filename
4          filename = input("Enter filename: ")
5
6          # Read file content
7          with open(filename, 'r') as file:
8              content = file.read()
9
10         # Split into words and clean them
11         words = content.lower().split()
12
13         # Remove punctuation and empty strings
14         import string
15         clean_words = []
16         for word in words:
17             clean_word = word.translate(str.maketrans('', '', string.punctuation))
18             if clean_word:  # Add only non-empty words
19                 clean_words.append(clean_word)
20
21         # Sort the words
22         sorted_words = sorted(clean_words)
23
24         # Display results
25         print("Sorted words:")
26         print(sorted_words)
27
28         # Save to new file
29         with open('sorted_words.txt', 'w') as output_file:
30             for word in sorted_words:
31                 output_file.write(word + '\n')
32
33         print(f"Total words: {len(sorted_words)}")
34
35     except FileNotFoundError:
36         print("Error: File not found")
37     except Exception as e:
38         print(f"Error: {e}")
39
40 sort_words_from_file()
41
```

### Mnemonic

Read, Split, Clean, Sort, Save

## Question 4(a) OR [3 marks]

**What is file handling? List files handling operation and explain it.**

### Solution

**File Handling** is the process of working with files to store and retrieve data permanently.
**File Handling Operations:**

| Operation | Function | Description |
|-----------|----------|-------------|
| **Open** | `open()` | Opens file in specified mode |
| **Read** | `read()` | Reads data from file |
| **Write** | `write()` | Writes data to file |
| **Close** | `close()` | Closes file |
| **Seek** | `seek()` | Moves file pointer |
| **Tell** | `tell()` | Returns current position |

**Mnemonic**

Open, Read, Write, Close - basic file cycle

## Question 4(b) OR [4 marks]

**Explain load() method with example.**

**Solution**

**load() method** is used to deserialize data from a file (usually with pickle module).
**Example:**

```python
import pickle

# First, save some data
data_to_save = {'name': 'John', 'scores': [85, 92, 78]}
with open('data.pkl', 'wb') as file:
    pickle.dump(data_to_save, file)

# Load data from file
with open('data.pkl', 'rb') as file:
    loaded_data = pickle.load(file)

print("Loaded data:", loaded_data)
```

- **Deserialization**: Converts file data back to Python objects
- **Binary mode**: Use 'rb' mode for pickle files

**Mnemonic**

load() brings file data back to Python objects

## Question 4(c) OR [7 marks]

**Write a program that inputs a text file. The program should print all of the unique words in the file in alphabetical order.**

**Solution**

**Program:**

```python
def find_unique_words():
```

```
 2      try:
 3          # Get filename
 4          filename = input("Enter text filename: ")
 5
 6          # Read file content
 7          with open(filename, 'r', encoding='utf-8') as file:
 8              content = file.read().lower()
 9
10          # Clean and extract words
11          import re
12          words = re.findall(r'\b[a-zA-Z]+\b', content)
13
14          # Create set (unique) and sort
15          unique_words = sorted(list(set(words)))
16
17          # Display results
18          print("\nUnique words in alphabetical order:")
19          for i, word in enumerate(unique_words, 1):
20              print(f"{i:3d}. {word}")
21
22          print(f"\nTotal unique words: {len(unique_words)}")
23
24          # Save results
25          with open('unique_words.txt', 'w') as f:
26              for word in unique_words:
27                  f.write(word + '\n')
28
29      except FileNotFoundError:
30          print(f"Error: File '{filename}' not found")
31      except Exception as e:
32          print(f"Error: {e}")
33
34  find_unique_words()
35
```

**Mnemonic**

Read, Extract, Unique, Sort, Display

# Question 5

## Question 5(a) [3 marks]

**Explain the use of the following turtle function with an appropriate example. (a) turn() (b) move().**

**Solution**

**Note**: Standard turtle corresponds to `left/right` for turn and `forward/backward` for move.
**Functions:**

| Function | Purpose | Example |
|----------|---------|---------|
| **Turn** | Change direction | `turtle.left(90)` |
| **Move** | Change position | `turtle.forward(100)` |

```
1  import turtle
```

```
2  t = turtle.Turtle()
3  t.forward(100)  # Move
4  t.left(90)      # Turn
5
```

**Mnemonic**

Turn changes direction, Move changes position

# Question 5(b) [4 marks]

**Explain the various inbuilt methods to change the direction of the turtle.**

**Solution**

**Direction Methods:**

| Method | Description | Example |
|---|---|---|
| **left(deg)** | Turn left (counter-clockwise) | `t.left(90)` |
| **right(deg)** | Turn right (clockwise) | `t.right(45)` |
| **setheading(deg)** | Set absolute angle | `t.setheading(0)` |
| **towards(x,y)** | Angle towards point | `t.towards(0,0)` |

**Mnemonic**

Left-Right relative, Heading absolute, Towards calculates

# Question 5(c) [7 marks]

**Write a program to draw square, rectangle and circle using turtle.**

**Solution**

**Program:**

```
1   import turtle
2
3   def draw_shapes():
4       t = turtle.Turtle()
5       t.speed(3)
6
7       # Draw Square
8       t.penup(); t.goto(-200, 50); t.pendown()
9       t.write("Square")
10      for _ in range(4):
11          t.forward(80)
12          t.right(90)
13
14      # Draw Rectangle
15      t.penup(); t.goto(0, 50); t.pendown()
16      t.write("Rectangle")
17      for _ in range(2):
18          t.forward(120)  # Length
```

```
19          t.right(90)
20          t.forward(60)    # Width
21          t.right(90)
22
23      # Draw Circle
24      t.penup(); t.goto(200, 50); t.pendown()
25      t.write("Circle")
26      t.circle(40)
27
28      turtle.done()
29
30  draw_shapes()
31
```

**Mnemonic**

Square: 4 equal sides, Rectangle: 2 pairs, Circle: radius method

## Question 5(a) OR [3 marks]

**What are the various types of pen command in turtle? Explain them all.**

**Solution**

**Pen Commands:**

| Command | Purpose |
|---|---|
| **penup()** | Lift pen (stop drawing) |
| **pendown()** | Lower pen (start drawing) |
| **pensize(w)** | Set line thickness |
| **pencolor(c)** | Set line color |
| **fillcolor(c)** | Set fill color |
| **begin_fill()** | Start filling shape |
| **end_fill()** | Stop filling shape |

**Mnemonic**

Up-Down controls drawing, Size-Color controls appearance

## Question 5(b) OR [4 marks]

**Draw circle and star shapes using turtle and fill them with red color.**

**Solution**

**Program:**

```
1  import turtle
2
3  t = turtle.Turtle()
4  t.color("red", "red")   # Pen and Fill color
5
```

```
 6   # Filled Circle
 7   t.begin_fill()
 8   t.circle(50)
 9   t.end_fill()
10
11   t.penup(); t.forward(150); t.pendown()
12
13   # Filled Star
14   t.begin_fill()
15   for _ in range(5):
16       t.forward(100)
17       t.right(144)
18   t.end_fill()
19
20   turtle.done()
21
```

**Mnemonic**

Begin fill, Draw shape, End fill

## Question 5(c) OR [7 marks]

**Write a program to draw Indian Flag using turtle.**

**Solution**

**Indian Flag Program:**

```
 1   import turtle
 2
 3   def draw_rect(color, x, y, width, height):
 4       t.penup(); t.goto(x, y); t.pendown()
 5       t.color(color)
 6       t.begin_fill()
 7       for _ in range(2):
 8           t.forward(width); t.right(90)
 9           t.forward(height); t.right(90)
10       t.end_fill()
11
12   t = turtle.Turtle()
13   t.speed(5)
14   width = 300
15   height = 60
16
17   # Draw Stripes
18   draw_rect("orange", -150, 150, width, height)
19   draw_rect("white", -150, 90, width, height)
20   draw_rect("green", -150, 30, width, height)
21
22   # Draw Chakra
23   t.penup()
24   t.goto(0, 60)   # Center of white stripe
25   t.pendown()
26   t.color("navy")
27   t.circle(30)    # Outer circle
28
29   # Spokes
```

```
30  for i in range(24):
31      t.penup(); t.goto(0, 90); t.pendown()
32      t.setheading(i * 15)
33      t.forward(30)
34
35  t.hideturtle()
36  turtle.done()
37
```

### Mnemonic

Saffron-White-Green stripes with 24-spoke Chakra