BSCCS2005: Practice Assignment with Solutions
Week 1

1. Which of the following statements is/are true about the *type of a variable*?

[MSQ:2points]

- ◯ In Java, the type of a variable depends on the value assigned to it.
- ✓ In Python, the type of a variable depends on the value assigned to it.
- ◯ In Python, the type of a variable can be determined by its data type at compile time itself.
- ✓ In Java, the type of a variable can be determined by its data type at compile time itself.

2. Which of the following statements is/are true about *dynamic lookup*?

[MCQ : 2 points]

- ◯ Dynamic lookup means different objects respond in the same way to the same message.
- √ Dynamic lookup means different objects respond in different ways to the same message.
- ◯ Dynamic lookup means the same object responds in different ways to the same message.
- ◯ Dynamic look up hides internal details of an object.

Solution: Dynamic lookup is a feature of object-Oriented programming languages. It says that different objects respond to a same message in different ways.

3. Which of the following features of an object-Oriented programming language hides implementation details?

[MCQ : 2 points]

- ○ Dynamic lookup
- [✓ Abstraction]
- ○ Subtyping
- ○ Inheritance

Solution: Abstraction is a feature of object-oriented programming languages that hides implementation details.

4. Suppose ‘a’ and ‘b’ are two objects such that object ‘a’ has all the functionalities of object ‘b’ along with some extra functionalities. Then, which of the following statements is/are true?

[MCQ : 2 points]

- ◯ We can make object ‘a’ as a subtype of object ‘b’, and use object ‘b’ wherever object ‘a’ is required.
- ◯ We can make object ‘b’ as a subtype of object ‘a’, and use object ‘b’ wherever object ‘a’ is required.
- ✓ We can make object ‘a’ as a subtype of object ‘b’ and use object ‘a’ wherever object ‘b’ is required.
- ◯ We can make object ‘b’ as a subtype of object ‘a’, and use object ‘a’ wherever object ‘b’ is required.

5. Which among the following statements is/are true about writing programs in low level languages and high level languages?

[MSQ: 2points]

- ✓ Writing a program in a low level language is error-prone.
- ◯ A program written in a low level language is more readable and easy to maintain.
- ◯ A programmer has more control over how a code is mapped to the machine architecture when the code is written in a high level language.
- ✓ Language translators like compilers and interpreters are required to convert a program in a high level language into a low level language.

Solution: The correct statements are as follows:

1. Writing a program in low level language is error-prone.
2. Considering the factors readability and maintainability of the code, high level languages are more effective.
3. Writing code in high level language is less efficient since the programmer has limited control over how the code is mapped to the machine architecture.
4. Language translators like compilers and interpreters are required to convert a program in a high level language into a low level language.

6. Which among the following statements is/are true about imperative and declarative programming style? [MSQ: 2points]

   ◯ An imperative program focuses on what the output is and how the output is related to the input, whereas a declarative program focuses on how to obtain the output.

   √ An imperative program describes the steps of instructions to produce the output, whereas a declarative program describes what the output should be.

   √ An imperative program depends on a large set of instructions that manipulate the intermediate variables, whereas a declarative program may totally avoid using intermediate variables.

   ◯ Imperative programs use an inductive structure and allows defining functions in terms of smaller functions. Thus, imperative programs are more readable as compared to declarative programs.

Solution: The correct statements are as follows:

1. A declarative program focus on "what is the output? and how it is related to input?", whereas an imperative program focus on "how to get the output?"
2. An imperative program describes the steps of instructions to produce the output, whereas a declarative program describes what computation should produce.
3. An imperative program depends on large set of instructions that manipulates the intermediate variables, whereas a declarative program may totally avoid using intermediate variables,
4. Declarative programs uses an inductive structure and allows defining functions in terms of smaller functions compare to the imperative programs. Thus, the functions in imperative programs are more readable (or understandable).

7. Which of the following statements is/are true about *variables*?

[MSQ: 2points]

- ◯ Any variable can be accessed if it is on the stack.
- √ A variable should be in the correct scope in order to be accessible.
- ◯ Statically and dynamically created data is stored on the stack.
- √ During the lifetime of a variable, there may be times when it is not in scope.

Solution:

- variable can be out of scope even if it is in stack.
- **variable must be in its scope in order to access it**.
- variables which outlives the activation record are stored in heap.
- **If the variable is stored in heap, its scope outlives the function in which it is called, but the scope is within the function in which it is defined.**

8. Let **A** be an object of type **Animal** and **B** be an object of type **Bird**.

[MSQ: 2points]

```
class Animal:
    def __init__(self, L):
        self.Legs = L
    def walk(self):
      print("Animal walks with", self.Legs, "legs")
class Bird(Animal):
    def __init__(self, L, W):
        self.Legs = L
        self.Wings = W
    def fly(self):
        print("Bird flies")
```

Which of the following options is/are NOT valid?

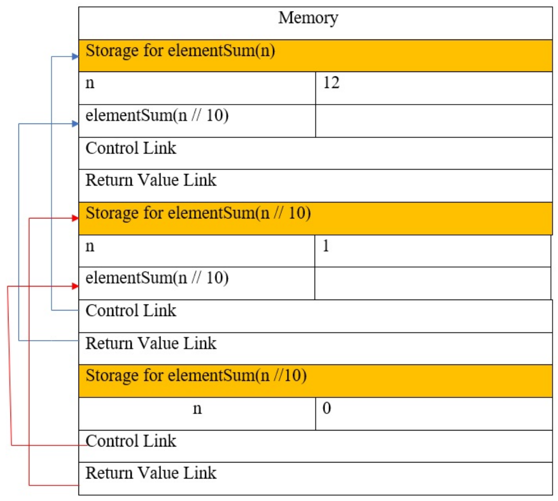
- ○ **B**.fly( )
- ✓ **A**.fly( )
- ○ **A**.walk( )
- ○ **B**.walk( )

9. Choose the correct memory mapping for the following pseudocode:

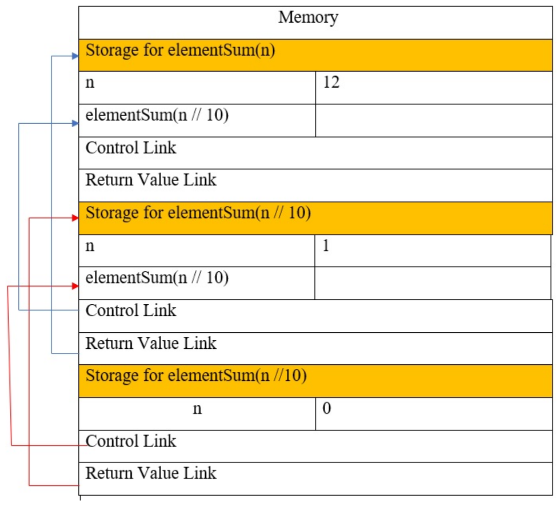
[MCQ: 2points]

```
def elementSum(n):
    if(n == 0):
        return 0
    else:
        return n % 10 + elementSum(n//10)
elementSum(12)
```

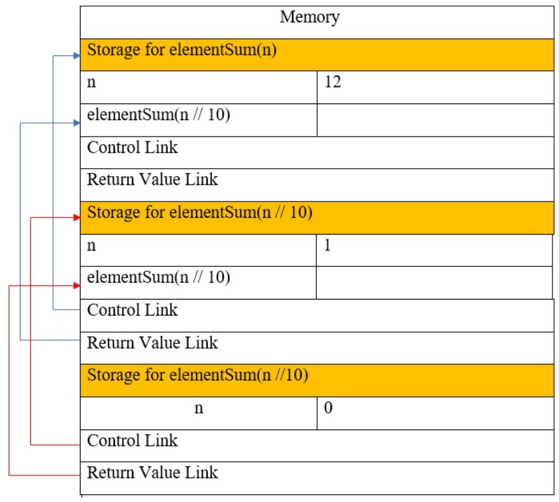
◯ Option 1



◯ Option 2

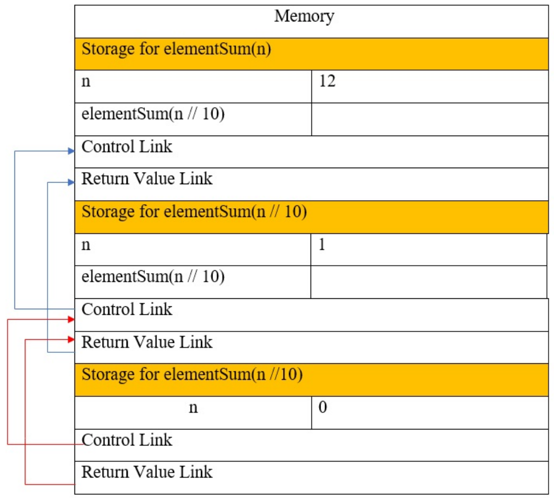


○ Option 3



Memory	
Storage for elementSum(n)	
n	12
elementSum(n // 10)	
Control Link	
Return Value Link	
Storage for elementSum(n // 10)	
n	1
elementSum(n // 10)	
Control Link	
Return Value Link	
Storage for elementSum(n //10)	
n	0
Control Link	
Return Value Link	

○ Option 4



Memory	
Storage for elementSum(n)	
n	12
elementSum(n // 10)	
Control Link	
Return Value Link	
Storage for elementSum(n // 10)	
n	1
elementSum(n // 10)	
Control Link	
Return Value Link	
Storage for elementSum(n //10)	
n	0
Control Link	
Return Value Link	

Solution: Control link points to the previous activation record. The return value link tells where to store the result.

10. Consider the following Python code.

[MCQ: 2points]

```
def area(radius):
    return(22/7*radius*radius)

def area(breadth,length):
    return(length*breadth)

print(area(7))
print(area(7,10))
```

What will be the output of the code?

- √ The code generates an error
- ○ 154,70
- ○ 70,70
- ○ 154,154

Solution: This example demonstrates that Python does not support method overloading by default.

11. From among the following, choose the INCORRECT characteristic/s about Abstract Data Types.

[MSQ: 2points]

- ○ Fixed interface
- ○ Structured data collection
- ✓ Fixed implementation
- ○ Implementation can be changed but should not affect interface

Solution: Abstract Data Types are structured collections with fixed interfaces. Changing implementation of Abstract Data Types should not affect interface

12. Let **A** be an object of **Vehicle**, **B** be an object of **Twowheeler**. What is the output of **B**.display( )? [MCQ: 2points]

```
class Vehicle:
    def display(self):
      print("Every vehicle has wheels")
class Twowheeler(Vehicle):
    def display(self):
        print("Every Two wheeler has two wheels")
```

- ○ Error
- ○ Every vehicle has wheels
  Every Two wheeler has two wheels
- ○ Every vehicle has wheels
- ✓ Every Two wheeler has two wheels

Solution:

- **B** is subtype of **A**.
- Both **A**, **B** have display( ) method.
- **B** is calling display( ) method, so the display( ) method of **B** gets executed.