Student Name:

Student ID:

Guideline

Homework for Week 01

Please submit your solutions as a single PDF on Brightspace.

Multiple submissions are allowed before the due time; the last submission will be graded.

Warm up

Consider the following program in Python:

```
numbers = [1, 2, 3, 4, 5, 6]
sum = 0
for number in numbers:
    if number % 3 == 0:
        sum += number
print(sum)
```

1. (points = 5) What will be output if we run this program?

- Answer:

2. (points = 5) What is the major paradigm used in the code? Choose a single answer from (a-e) below: (a) functional (b) object-oriented (c) imperative (d) declarative, and (e) procedural

- Answer:

Functional programming

Lambda functions

In Python, lambda functions are defined using the lambda keyword followed by a comma-separated list of arguments (if any), followed by a colon and an expression. Here's an example:

```
add = lambda x, y: x + y
print(add(9, 3))
```

In this example, we define a lambda function `add` that takes two arguments `x` and `y`, and returns their sum. We then call the `add` function with arguments 9 and 3, which returns the sum 12.

3. (points = 10) Define an lambda function in python that takes a two input numbers and returns their distance. You can use the python function `abs` for the absolute value function. For example, `abs(-4.2)` returns 4.2. Write out the lambda expression. It should start with the key word "lambda".

- Answer:

The `map` function

In Python, `map` is a built-in function that takes two arguments: a function and an iterable. It applies the function to each element of the iterable and returns a new iterable with the results. By "iterable", we mean any object that can be iterated upon, meaning it can be looped over using a for loop. Examples of iterables in Python include lists, tuples, strings, dictionaries, and sets.

Here's an example of using `map` to apply the length function `len` to a list of strings:

```
words = ["apple", "banana", "cherry"]
lengths = map(len, words)
print(list(lengths))
```

In this example, `map` applies the `len` function to each string in the words list, returning an iterable with the lengths of each string. We then convert this iterable to a list using the `list` function and print it. The output will be [5, 6, 6].

The function `map` is often used in conjunction with lambda functions or regular python functions (defined with `def` keyword) to apply more complex operations to an iterable.

4. (points = 10) What is the output of the following code?

- Answer:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(lambda x: x**2, numbers)
print(list(squared_numbers))
```

The `filter` function

The `filter` function takes a function and a sequence of values as arguments and returns a new sequence containing only the values for which the function returns True. Here's an example:

```
numbers = [1, 2, 3, 4, 5]
even_numbers = filter(lambda x: x % 2 == 0, numbers)
print(list(even_numbers))
```

In this example, we use the `filter` function to create a new list containing only the even numbers from the numbers list. We pass a lambda function that returns True if the input value is even as the first

argument to filter, and `numbers` as the second argument. The `filter` function applies the lambda function to each element in the list and returns a new list containing only the even numbers. Note that we need to use the list function to convert the output of filter into a list before printing.

5. (points = 10) What is the output of the following code?

- Answer:

```
def is_even(number):
    return number % 2 == 0
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
e = list(filter(is_even, numbers))
print(e)
```

The `reduce` function

The `reduce` function takes a function and an iterable of values as arguments and returns a single value obtained by performing the specified operation on the values in the iterable. The function is applied cumulatively to the iterable from left to right, reducing it to a single value. Here's an example:

```
from functools import reduce
numbers = [1, 2, 3, 4, 5]
sum = reduce(lambda x, y: x + y, numbers)
print(sum)
```

In this example, we use the `reduce` function to compute the sum of the numbers list. We pass a lambda function that takes two arguments and returns their sum as the first argument to reduce, and numbers as the second argument. The `reduce` function applies the lambda function to the first two elements of the list, then applies it to the result and the next element, and so on, until all elements have been processed and a single value is returned.

6. (points = 10) What is the output of the following code?

- Answer:

```
from functools import reduce
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = filter(lambda x: x % 2 == 0, numbers)
sum = reduce(lambda x, y: x + y, even_numbers)
print(sum)
```

Object-oriented programming

Consider the following code in Java

```
public class Customer {
    private String name;
    private SeniorityLevel level;
    public Customer(String name, SeniorityLevel level) {
        this.name = name;
        this.level = level;
    }
    public String getName() {
        return name;
    }
    public SeniorityLevel getLevel() {
        return level;
    }
    public void setLevel(SeniorityLevel level) {
        this.level = level;
    }
}
enum SeniorityLevel {
    NEW, REGULAR, VIP
}
```

In this example, the Customer class has a constructor that takes a name and a level, and two getter methods for the name and level fields. The level field is of type SeniorityLevel, which is an enum that defines three possible values: NEW, REGULAR, and VIP.

The setLevel method is also provided to allow the level field to be updated if needed.

Note that the SeniorityLevel enum is declared at the bottom of the file and is defined with three values: NEW, REGULAR, and VIP. This allows the level field of the Customer class to be constrained to one of these three values.

Encapsulation

Encapsulation is one of the fundamental principles of object-oriented programming and refers to the practice of hiding the internal details of an object from the rest of the program. In Java, encapsulation is achieved through the use of access modifiers (public, private, protected) to restrict access to the variables and methods of a class.

Suppose you have developed the Customer class, and you want to guarantee this business logic, that (1) a Customer's name cannot be changed after the object is created, and (2) a Customer's seniority level can be changed during the object's lifetime. (Having logic #2 is clearly necessary in practice; having logic 1 can avoid many issues including security breach.)

Now, your co-worker Ji-Ho is working on web development to design a frontend for customers to operate on their accounts. Assume Ji-Ho's code has access to Custom objects.

7. (points = 5) Is there any possibility that Ji-Ho's code can change a customer's name?

8. (points = 5) Is there any possibility that Ji-Ho's code can change a customer's seniority level?

Inheritance and Polymorphism

Consider the Java code below:

```
class Animal {
  String name;
  int age;
  public Animal(String name, int age) {
    this.name = name;
    this.age = age;
  }
  public void speak() {
    System.out.println("The animal makes a sound.");
  }
  public void eat() {
    System.out.println("The animal eats.");
  }
}
class Cat extends Animal {
  boolean isHouseTrained;
  public Cat(String name, int age, boolean isHouseTrained) {
    super(name, age);
    this.isHouseTrained = isHouseTrained;
  }
  public void claw() {
    System.out.println("The cat claws.");
  }
  // Override the speak() method of the superclass
  @Override
  public void speak() {
    System.out.println("The cat meows.");
  }
}
// Main class
public class Main {
  public static void main(String[] args) {
    // Create an instance of the Cat class
    Cat cat1 = new Cat("Whiskers", 5, true);
    // Access properties and methods of the Cat class and Animal class
    System.out.println(cat1.name); // Line A
    cat1.speak(); // Line B
    cat1.eat(); // Line C
    cat1.claw(); // Line D
  }
}
```

9. (points = 5) What is the output at Line A?

- Answer:

10. (points = 5) What is the output at Line B?

- Answer:

11. (points = 5) What is the output at Line C?

- Answer:

12. (points = 5) What is the output at Line D?

- Answer:

Multi-Paradigm Programming Languages

JavaScript

JavaScript is a multi-paradigm programming language. For each piece of JavaScript code below, what is the major paradigm used in the code? Choose a single answer from (a-e): (a) functional (b) object-oriented (c) imperative (d) declarative, and (e) procedural

13. (points = 5)

```
let counter = 0;
for (let i = 0; i < 5; i++) {
    counter += i;
}
console.log(counter); // Output: 10
```

- Answer:

14. (points = 5)

```
const numbers = [1, 2, 3, 4, 5];
const squared = numbers.map(num => num ** 2);
const evenSquared = squared.filter(num => num % 2 === 0);
const sum = evenSquared.reduce((acc, num) => acc + num, 0);
console.log(sum); // Output: 20
```

- Answer:

SQL

SQL usually stands as a pure showcase of declarative programming, although vendors have extended it with procedural elements. For each piece of SQL code below, what is the major paradigm used in the code? Choose from (d-e): (d) declarative, and (e) procedural

15. (points = 5)

```
SELECT *
FROM Customers
WHERE PurchaseDate >= DATEADD(MONTH, -1, GETDATE());
```

- Answer:

16. (points = 5)

```
CREATE PROCEDURE UpdateCustomerEmail
    @customerId INT,
    @newEmail NVARCHAR(255)
AS
BEGIN
    UPDATE Customers
    SET Email = @newEmail
    WHERE CustomerID = @customerId;
END;
```

- Answer: