OOP 2022/2023

Section A – OOP through Python (answer 2 of 3)

1. (a) Name of operator that can perform multiple tasks, how does it know which to perform? Give an example.

→ Operator overloading is when a single operator can perform multiple tasks. Python knows which operation to perform depending on the operands. For example – the '+' operator with numbers performs addition but with strings performs string concatenation. When '+' operator is used with a list it performs list concatenation.

```
result1 = 5+3  # ouput 8
result2 = "Hello" + "world"  # Helloworld
result3 = [1,2,3] + [4,5,6]  # [1,2,3,4,5,6]
```

(b) Analyze code and answer the questions →

```
my_str = input("Input a string")
index_int = 0
result_str = ''
while index_int <(len(my_str) - 1):  # Line 1
    if my_str[index_int] > my_str[index_int+1]:
        result_str = result_str + my_str[index_int]
    else:
        result_str = result_str * 2  # Line 2
    index_int += 1
print(result_str)  # Line 3
```

(i) Output of Line 3 with input 'abc'.

→ Output with Line 3 with input 'abc' → would be null or empty (the result string is empty)

(ii) What output of Line 3 with input 'cba'.

→ The output of Line 3 with input 'cba' is cb.

(iii) What output of Line 3 with input 'cab'

→ The output of Line 3 with input 'cab' is cc.

(iv) What happens if Line 2 is removed.

→ If Line 2 is removed the while loop would not increment to the len(my-str)-1 value. This would cause an endless loop and the program would crash.

(v) What happens if Line 1 is modified to while index-int < len (my-str):?

→ If the while loop was modified and its condition was now len (my_str) → the program would show an error for the string index being out of range.

A string comprises of characters that are incremented from 0 upto the length of the string - 1. This is why when the condition is changed to be the length of the input string - the program shows an error - specifically the index error.

(c) Point class is as follows →

```
class Point(object):
    def __init__(self, x-param = 0.0, y-param = 0.0):
        self.x = x-param
        self.y = y-param
```

(i) Create method to return distance between two points →

→

```
class Point:
    def __init__(# class from before):
        ——

    def distanceXY (self, other_point):
        distance = math.sqrt ((self.x - other_point.x)**2
                   + (self.y - other_point.y) ** 2 ))
        return distance
```

(ii) Define new method for Point class called '__eq__' which should overload the "==" operator and allow comparison of 2 instances of Point. Instance is equal if x and y values are same.

→ Code is as follows

```
class Point
    def __init__():
        # also method from before
    def distanceXY():
        # method from before
    def comparePoints(self, other_point)
        if ((self.x == other_point.x) & (self.y == other_point.y))
            return true
        else:
            return false
point1 = Point(1, 2)
point2 = Point(1,2)
print ( point1.comparePoints(point2))
                    ↳ # prints true
```

2. (a) Output of print statement below? Explain why?

```
my_list = [4, 7, 1, 2]
my_list = my_list.sort()
print(my_list)
```

→ The print statement above would return 'None'. The reason is because sort() as a function modifies the original list (but does not return a new sorted list. Since the sort method did not return anything, it modifies the list in place and returns None. Since that value gets assigned to my_list = my_list when printed returns 'None'.

(b) Analyze Python code →

```
class Person(object):
    def __init__(self, name, age):
        self.__name = name
        self.__age = age
    def __str__(self):
        return_str = "\n Name : " + str(self.__name)
        return_str += "\n Age : " + str(self.__age)
        return return_str
```

Create another class to inherit from Person. Add instance attribute to your subclass that only applies to subclass. Add a __str__ method to your subclass that reuses the __str__ method of Person. Create an instance of your subclass and print it.

→

```
class Employee(Person):
    def __init__(self, name, age, employee_id):
        super().__init__(name, age)
        self.employee_id = employee_id
    def __str__(self):
        return super().__str__() + "\n Employee ID: " + str(self.employee_id)
employee = Employee("John", 30, "E123")
print(employee)
```

(c) (i) Explain in detail what kind of relationship is modelled by a composition or aggregation relationship.

→ Composition is a concept that models a has-a relationship. It enables creating complex types by combining objects of other types. It also enables the reuse of code without having to inherit. Aggregation is a weak form of composition. The main difference is that both

objects are independent of each other. Both have association relationships with objects. Aggregation models a 'has a' relationship. Composition models a 'part of' relationship.

(ii) Explain difference between aggregation and composition in Python classes.

→ (mainly explained in last question).

- Composition represents a strong relationships where objects are tightly coupled together.
- Aggregation represents a weaker relationship where components objects can exist independently and may be shared among multiple aggregates

(iii) Write python examples for cases in which one is better.

→ Example of composition:

```
class Salary:
    def __init__(self, pay, bonus):
        self.pay = pay
        self.bonus = bonus
    def annual_salary(this):
        return (12 * self.pay) + self.bonus

class Employee:
    def __init__(self, name, age, pay, bonus):
        self.name = name
        self.age = age
        self.object_salary = Salary(pay, bonus)
    def total_salary(self):
        return self.obj_salary.annual_salary()
emp = Employee('max', 25, 15,000, 10000)
print(emp.total_salary())
```

In this example Employee owns the Salary object.

Aggregation on next page →

Example of Aggregation →

```
class Salary:
    def __init__(self, pay, bonus):
        self.pay = pay
        self.bonus = bonus
    def annual_salary(self):
        return (self.pay * 12) + self.bonus
class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.obj_salary = salary
    def total_salary(self):
        return self.obj_salary.annual_salary()
Salary = Salary(15,000, 10,000)
emp = Employee('max', 25, salary)
print(emp.total_salary())
```

In this example salary is created independtly and Employee class aggregates Salary objects.

Section B - OOP Through Java (Answer 2 of 3)

5. (a) Explain the terms →

→ • Jar Files - JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc) into one file to distribute application software or libraries on the Java platform.

• Path - PATH is an environment variable that is used to find and locate binary files like 'java' and 'javac' and to locate needed executables from the command line. or

- Classpath - Classpath is an environment variable that is used by the application ClassLoader or system to locate and load the compiled Java stored in '.class' file.
- JDK - Java Development Kit (JDK) is a software cross-platformed ~~software~~ software development environment that offers a collection of tools and libraries necessary for Java-based development.
- Packages - A mechanism in Java to encapsulate a group of classes, sub packages and interfaces. Used for providing controlled access and preventing naming conflicts.

(b) How does a complied Java program run on multiple OSs?

→ Java is a platform independent programming language. It can run ~~of~~ on multiple OS without any modification. When a java file is ~~compl~~ compiled, the compiler creates a .class bytecode file that can run on any os that has JVM or Java Virtual Machine installed.

(c) Difference between overloading and overriding a method?

→ They both are concepts or types of polymorphism. In method overloading more than one method shares the same method name with a different signature in the class. In this case the same method can have different parameters to allow different ~~results~~ results - even when their names are the same. In method overriding, the derived class provides the specific implementation of the method that is already provided by the base class or parent class - without changing the method class.

(d) What is constructor chaining? Where is this ~~was~~ used?

→ Constructor chaining is the process of calling one constructor from another constructor with respect to current object. We use constructor chaining when we want to perform

to perform multiple tasks in a single constructor rather than creating code for each task in a single constructor. Thus we create a separate constructor for each task and make their chain which makes the program more readable. It can be done from the same class using 'this()' keyword or can be done from base class by using 'super()' keyword.

(e) Write code for function - public int sortaSum (int a, int b) which returns the sum of a and b. But if the result is between 10-19, it returns 20.

→
```
public class Main {
    public static void main (String [] args) {
        System.out.println ("Hello World");
        int result = sortaSum (4, 5);
        int result2 = sortaSum (4, 10);
        System.out.println (result); // result = 9
        System.out.println (result2); // result2 = 20
    }
    public static int sortaSum (int a, int b) {
        int sum = a+b;
        if (sum >= 10 && sum =<19) {
            return 20;
        }
        else {
            return sum;
        }
    }
}
```

(f) Write code for function public float average (float [] nums) that returns sum of parameter. Use Java 'for each'.

→ Code on next page.

↳ enhanced for loop to iterate over elements in an array or object.

```
public class main {
    public static void main (String[] args) {
        float [] nums = {1.5f, 2.5f, 3.5f, 4.5f, 5.5f };
        float result = average (nums);
        System.out.println ("Average : " + result);
    }
    public static float average (float[] nums) {
        float sum = 0;
        for (float num : nums ) {
            sum += num;
        }
        return sum;
    }
}
```

6. (a) Write an implementation for the map function →

```
float map (float a, float b, float c, float d, float e);
```

That maps the parameter a from the range b to c onto the range d to e. Such that when function is called as such →

```
println (map (3.0, 10, 100, 200);
```

The result is 130.

→ An implementation would look as follows →

```
public class main {
    public static void main (String[] args) {
        float result = map (3.0, 10, 100, 200);
        System.out.println (result);
    }
    public static float map (float a, float b, float c, float d, float e ){
        float result = (a-b) * (e-d) / (c-b) + d;
        return result;
    }
}
```

(b) Code for visualization of the waveform →

```
public class Audio3 extends PoAApplet {
    Minim m;
    AudioInput ai;
    AudioPlayer ap;
    AudioBuffer ab;
    FFT fft;
    public void settings () {
        fullScreen (P3D, SPAN);
    }
    public void setup() {
        m = new Minim (this);
        ai = m.getLineIn (Minim.MONO, width, 44100, 16);
        ab = ai.mix;
    }
    float [] lerpedBuffer;
    float lerpedAverage = 0;
    ....
}
```

(i) Explain the significance of the parameters to the call m.getLineIn.

→ m.getLineIn method is used to obtain an input audio stream from an external audio source, such as microphone or line input. The parameters are as follows →

- Minim.MONO - specifies the type of audio input is desired. In this case - monophonic audio input.
- width - specifies the desired buffer for the audio input. The amount of data that is captured and processed.
- '44100' - Specifies the desired sample rate for the audio input.
- 'ab' - Specifies the target buffer where audio data will be stored.

- '16' - Specifies the number of sample buffers used for mixing.

(ii) Write code to assign elements of lerpedBuffer and draw visualization (waveform).

→ setup.....

```
public void draw() {
    background(0);
    fft.forward(ai.mix);
    for (int i=0; i<lerpedBufferLength; i++) {          fft.specSize()-1);
        float lerpedValue = map(i, 0, lerpedBufferLength-1, 0, ^
        lerpedBuffer[i] = fft.getBand((int)lerpedValue);
    }
    float sum = 0;
    for (float value : lerpedBuffer) {
        sum += value;
    }
    lerpedAverage = sum / lerpedBufferLength;
    strokeWeight(2);
    beginShape();
    for (int i=0; i<lerpedBufferLength; i++) {
        float lerpedValue = lerpedBuffer[i] * height/2;
        float x = map(i, 0, lerpedBufferLength, 0, width);
        float y = map(lerpedValue, -1, 1, height, 0);
        stroke(map(i, 0, lerpedBufferLength, 0, 255), 255, 255);
        vertex(x, y);
    }
    endShape();
```

(iii) What is involved in modifying program to identify music notes?

→ The process is as follows →

- Perform Frequency Analysis - Use techniques like FFT (Fast Fouier Transform) which is used to analyze audio input and determine frequencies present in audio signals.

- Identify dominant frequencies - Analyze the FFT data to identify the dominant frequencies or peaks in the frequency spectrum.
- Map frequencies to Musical notes - map the dominant frequencies to their corresponding musical notes.
- Display data - We can use visualizations such as the waveform to display them visually on the screen.

To do the following we also use the mimim audio library to read audio or microphone input.