**Emma Gill THEORY QUESTIONS ASSIGNMENT**

Software Stream

**Maximum** 

**score: 100**

KEY NOTES

• This assignment to be completed at student’s own pace and submitted before given deadline.

• There are 10 questions in total and each question is marked on a scale 1 to 10. The maximum possible grade for this assignment is 100 points.

• Students are welcome to use any online or written resources to answer these questions.

• The answers need to be explained clearly and illustrated with relevant examples where necessary. Your examples can include code snippets, diagrams or any other evidence-based representation of your answer.

| **Theory questions** | **10 point each** |
| --- | --- |

**1. How does Object Oriented Programming differ from Procedural Orientated Programming?**

A program is the communication between humans and the computer, instructing the computer what we would like it to do to solve problems. The way programs are structured and approached vary and these are known as programming paradigms.

Object-orientated programming (OOP) and procedural programming (PP) are two key paradigms.

**Procedural Programming (PP)**

*Examples of PP languages*

PP was seen in early programs, examples of procedural languages are Fortran, Cobol, C, Algogl, Basic (including BBC basic which was my first introduction to coding) and Pascal**.**

*Overview of PP*

Core Concepts = calling procedure

In PP, sometimes known as inline or imperative programming, the computer is given a list of instructions to execute step by step( a top down approach). Procedural programming focuses on procedures/functions required to execute what it is doing.

PP is more concerned with procedure rather than data, data and procedures are seen as separate structures.

A procedural program relies on procedures, routines and sub-routines.

**Object-Orientated Programming (OOP)**

*Examples of OOP languages:*

Examples of OOPs are C#, Python, C++, Java, Ruby, PHP, Scala and Perl.

*Overview of OOP:*

Core Concepts = encapsulation, polymorphism, abstraction, inheritance

In OOP objects are used to implement actions, objects are the basic units of a program.

Building blocks of OOP = objects

Unlike PP**,** in OOP data and methods can be lumped together (encapsulated) into their own units which are called objects.

Within OOP classes may be used as a blueprint for creating objects. This means that attributes and methods can be grouped together in a logical way, this means they are easier to build upon and be reused.

**Differences**

**Process**

OOP - bottom up

PP - top-down

**Small units used**

OOP - smallest entities are objects

PP - program divided into procedures or functions

**Data Control**

**Code Reusability**

PP - the code is not reusable

OOP -the code is reusable

**Security**

OOP is considered more secure.

In OOP this is because abstraction hides details from the user. Data is attached to the functions which use it, therefore more secure.

In PP there is no abstraction, the data is globally available within the program (not hidden)

**Size:**

PP - used for medium sized programs

OOP - used for large and complex programs

**2. What's polymorphism in OOP?**

Poly = many, lots, multiple

Morphism = thing/entity/item/element/form

Therefore we could think of polymorphism as meaning “many forms”.

In computer programming polymorphism describes the program's ability to interpret different forms based on their class or data type. This enables the computer to do the same thing in different ways.

An example of this is when a function behaves differently depending on what is passed into it as arguments. For example, in Python, print(“hello” \* 5) returns hellohellohellohellohello whereas print(5 \* 5) returns 25. We can see that the function print behaves differently depending on what its arguments are. This is because the print function exists as different versions of itself in the same class. All the versions are called print but have different parameters.

This process is called overloading, when two or more methods in the same class have the same name, in this example print, but have different parameters.

Another example of polymorphism is a process called overriding. Overriding is a process that occurs when a method has the same name and parameters in a child and parent class. But the inherited method is overridden in the child class meaning the method is implemented in a specific way to better suit its inherited class.

**3. What's inheritance in OOP?**

This is the process of creating new classes or subclasses from an existing class. The original class is called a parent class and the new classes are called child, sub-class or inherited classes.

The inherited class inherits the same methods and attributes of the parent class as well as its own characteristics.

Inheritance reduces duplicated code.

**4. If you had to make a program that could vote for the top three funniest people in the office, how would you do that? How would you make it possible to vote on those people?**

1. SQL - Create database in SQL to store the collected data (votes) in tables
2. Python - Users sign in system on Python ( security measure to stop duplicate votes)
3. Python - Collect votes(input) from users
4. Connect Python to SQL - inputted data stored into tables on SQL - using my.sql
5. Use python to query data in SQL and generate top three scorers

5**. What's the software development cycle?**

The software development cycle (SDC), or software development life cycle, is a model to show the stages included in developing software applications.

The life cycle process is divided into between 5 and 8 steps. The idea is that the SDC is used to outline all the tasks needed at each stage to help increase efficiency and improve the development process. It can be used to measure progress and identify inefficiencies.

As it is a cycle, at the end the loop can begin again

The steps are:

* Plan

This stage can include: costing, creating timetables, creating teams, creating leadership structure, defining the project. This stage can be focussed on the cost element of the project

* Requirements

Considered part of the planning this stage includes communications between IT teams and business to clarify the needs of the project. All the information is gathered from the customer and the business side of the team and hopefully any ambiguities are resolved. A core understanding of the needs of the project from everyone's angle are gathered, customer, business team, IT team etc.

* Design

As it is now clear what the requirements of the project are, the project is designed. There are Design patterns that already exist and can be used.

* Build /implement

This is the software building phase of the project. This could be in sprints if working in an Agile way or in linear waterfall manner.

* Test

The testing phase is important, there are many factors to be considered and tested before the project is ready to be used. In essence the software needs to be bug free, stable and it meets the customers needs.

Some areas to be considered are: Security testing, Functional testing - unit testing, User experience testing, Compatibility testing and Performance and load testing

* Deploy

This is the phase where whatever has been developed is delivered to the user. Deployment can be handled manually or automated. Automated deployment is considered a more efficient method. It is common for new software to be released in stages to check it all works before full release

* maintain

The maintenance phase is a continuous phase this is where software is built upon and maintained, for example: updated, security features updated

**6. What's the difference between agile and waterfall?**

Agile and waterfall are methodologies for approaching the process of software development.

In waterfall, once each phase of the software development is completed the next step begins, like the water flowing over the edge of a waterfall, one thing leads to another.

In the Agile method the whole project is divided into smaller chunks and each of these chunks are worked on in small amounts of time from one to three weeks - these are called sprints.

**7. What is a reduce function used for?**

A reduce function in Python is used on an iterable to reduce it into a single cumulative value without using a for loop.

Examples of how reduce() is used in practice are:

* Calculating the cumulative sum of numbers in an iterable
* Multiplying all the values in an iterable to find the product.
* Finding the minimum and/or maximum value in an iterable
* Finding out if all the items in an iterable are True.
* Finding out if at least one item in an iterable is True

**8. How does merge sort work**

Merge sort is a sorting algorithm. It is based on the divide and conquer paradigm.

The item to be sorted is halved.

The halves are halved again and again until there are groups of two separate elements remaining in each half

The groups of two elements that are left are then sorted.

These groups are merged one by one until a sorted half is created.

The two halves are merged in a sorted way.

In python this is implemented using recursion and iterators.

9**. Generators - Generator functions allow you to declare a function that behaves like an iterator, i.e. it can be used in a for loop. What is the use case?**

Generator functions could be used to iterate through infinite or very large iterable objects. This is because using yield(a generator) would use less memory than an iterator.Because the yield statement generates its value each time it is called and stores it, instead of an iterator which evaluates every iteration and stores them in memory.

Also, because the yield statement waits and remembers the previous value until it is called again it would not carry on and on in the case of an infinite object.

**10. Decorators - A page for useful (or potentially abusive?) decorator ideas. What is the return type of the decorator?**

Definition: A decorator takes another function as an argument, does something, then returns another function, without altering the source code.

This can be useful if you want to do the same thing to many functions and using decorators makes the code look neater which can make it more readable.

Useful decorators:

| **Decorator Idea** | **Ideas/examples of uses** | **Return Type** |
| --- | --- | --- |
| Show the actual time of every function used | Monetary transactions | string |
| Print output | Debugging, checking code | string |
| Show the location | IP address | float |
| Access counter | Shows how many times a function has been called | int |
| Sorting | Sorting iterables | interable |
| Formatting text | Display text in particular way | string |
| Frequently used input | Something that you type all the time | string |
| Timer | How long something takes to execute eg a whole, program | float |
| Repeat | A decorator that repeats an operation, might be useful for debugging | Function output |
| Measure memory | Measures how much memory a function uses | float |
| login | The decorator log into an external API automatically |  |
| Password store | A decorator to securely access stored passwords to login |  |