# 1. How does Object Oriented Programming differ from Process Oriented Programming?

**Object Orientated programming**

Object-oriented Programming is a programming language that uses classes and objects to create models based on the real-world environment. An Object-oriented Programming application may use a collection of objects which will pass messages when called upon to request a specific service or information. Objects can pass, receive messages, or process information as data.

When using Object-Oriented Programming, new objects inherit characteristics from existing ones, making it easy to maintain and modify existing code. Thus, development time is cut down considerably, and adjusting the program is more leisurely.

It is also easy to develop an object-oriented program using Object-oriented Programming, and other developers can easily understand the program. Developers can tell from well-commented objects and classes what process the program developer followed. For novice programmers, it can also make adding new features to the program more accessible.

The difference between Object Oriented Programming and Process Oriented Programming is as follows.

- Procedural Programming also uses different methods throughout the code than Object-oriented Programming.
- For example, Object-oriented Programming uses methods, whereas Procedural Programming uses procedures.
- Object-oriented Programming uses objects, whereas Procedural Programming uses records.
- Object-oriented Programming uses classes, whereas Procedural Programming uses modules and Object-oriented Programming uses messages, whereas Procedural Programming uses procedure calls.
- In addition, Object-oriented Programming uses data fields, whereas Procedural Programming uses procedures.

**Process Oriented (Procedural) Programming**

Procedural programming, also known as inline programming, follows a top-down programming approach. Classes and objects are used in object-oriented programming. In contrast, procedural programming tackles problems from the top of the code down to the bottom.

Unlike object-orientated programming, procedural programming has the obvious drawback of requiring the developer to edit every line of code that corresponds to the original change. This occurs when a program starts with a problem and breaks it down into smaller sub-

problems or sub-procedures. As part of the functional decomposition process, these sub-procedures are continually broken down until they are simple enough to be solved.

Finding and modifying all related elements in a program becomes increasingly difficult as more changes are needed. A program might start with a variable set to 1, for example. Those sub-procedures will also need to be edited if they depend on that variable equalling 1 to function correctly.

Code Comparison

Two codes are shown below, and the first program uses object-orientated programming and the second process-orientated programming. They both print out the same output.

Joe Cool is a 4th year Computer Science student.

```
# PROCEDURAL PROGRAMMING EXAMPLE

studentName = "Joe Cool"
year = 4
course = "Computer Science"


def print_Info(name, year, course):
    print(f"{name} is a {year}th year {course} student.")


print_Info(studentName, year, course)
```

```
# OOP EXAMPLE
# ————————————
class Student:
    def __init__(self, name, year, course):
        self.name = name
        self.year = year
        self.course = course

    def print_info(self):
        print(f"{self.name} is a {self.year}th year {self.course}
student.")


# instantiating Student class objects
student1 = Student("Joe Cool", 4, "Computer Science")

# calling method
student1.print_info()
```

## 2. What is Polymorphism in OOP?

Polymorphism is an ability (in OOP) to use a common interface for multiple forms (data types). Polymorphism is a fundamental concept in programming. Suppose we want to tell a user the sounds an animal can make. There are multiple options (cow-moo, dog-bark, pig-oink). However, we could use the same methods to make these sounds.

It refers to using a single type of entity (method, operator, or object) to represent different types in different scenarios. A simple example is the + operator, which is used extensively in python. For integer data types, the + operator is used to perform arithmetic addition operation

```
sum = 1 + 2
print(sum)

# OUTPUT: 3
```

Similarly, for strings, the + operator is used to perform concatenation

```
word1 = "Code"
word2 = "First"
word3 = "Girls"
print(word1 + " " + word2 + " " + word3)

# OUTPUT: Code First Girls
```

Like in other programming languages, the child classes in Python also inherit methods and attributes from the parent class. We can redefine certain methods and attributes specifically to fit the child class, known as **Method Overriding**.

Polymorphism allows us to access these overridden methods and attributes with the same name as the parent class.

The code below shows Polymorphism being implemented with a function. The program uses the same sound() method. However, their functions are different. To use Polymorphism, we created a common interface called the make_sound() function that takes any object and calls the object's sound() method. Thus, it runs effectively when we pass the african_lions, nubian and duroc objects in the make_sound() function.

```
class Donkey:
    def facts(self):
        print("A donkey is stronger than a horse of the dame size")

    def category(self):
        print("Herbivores")

    def sound(self):
        print("Bray")
```

```
class Pig:
    def facts(self):
        print("Pigs love belly rubs!")

    def category(self):
        print("Omnivores")

    def sound(self):
        print("Oink(Grunt)")


# common interface
def make_sound(animals):
    animals.sound()


# instantiating objects
african_lions = Lion()
nubian = Donkey()
duroc = Pig()

# passing the object
make_sound(african_lions)
make_sound(nubian)
make_sound(duroc)

# OUTPUT: Roar
# Bray
# Oink(Grunt)
```

## 3. What's inheritance in OOP?

Inheritance is a powerful feature in object-oriented programming. It refers to defining a new class with little or no modification to an existing class. Inheritance enables us to define a class that takes all the functionality from a parent class and allows us to add more. The new class is called the derived (or child) class, and the one it inherits is called the base (or parent) class.

The parent and child class is said to be an **"is a"** relationship where the child **is a** specialised version of the parent class.

Inheritance syntax

```
Class BaseClass:
    {Body}
Class DerivedClass(BaseClass):
    {Body}
```

A derived class inherits features from the base class, where new features can be added. This results in the reusability of code.

**The benefits of inheritance are:**

- It represents real-world relationships well.
- It provides the reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

The following example demonstrates how a child can inherit the method of its parent class. The defined Employee class inherits from the Person class.

```
class Person:
    def __init__(self, job):
        self.job = job

    def message(self):
        print("Hi, I am " + self.job)


class Employee(Person):
    pass

    employee = Person('Python Developer')
    employee.message()

#OUTPUT: Hi, I am Python Developer
```

If you look at the code of our Employee class, you can see that we haven't defined any attributes or methods in this class. Inheriting these methods means using them as if they were defined in the Employee class. As the class Employee is a subclass of Person, it inherits, in this case, both the method __init__ and message(). When we create an instance of Employee, the __init__ function will also create a name attribute. We can apply the message() method to the Employee object employee, as seen in the output from the code above.

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?

- Assuming each Employee can vote for 1 funniest person and the program will finally pick the top 3 funniest people in the office.

- Create a model class for Employee. This class should contain class variables such as employee id, name, Voted employee and Vote count. Voted Employee is again an instance to the Employee class, and it is used to contain the object of which person an employee is voting. Vote count is used later to store the number of votes each Employee received for being the funniest

- The program should create and populate the list of employees. This will include employee id and name of the Employee

- After creating all the employee details, the program will get the vote of each Employee. The votedEmployee field will be set for each Employee.

- Since votedEmployee is an instance to Employee itself, the voteCount will be set for the votedEmployee object.

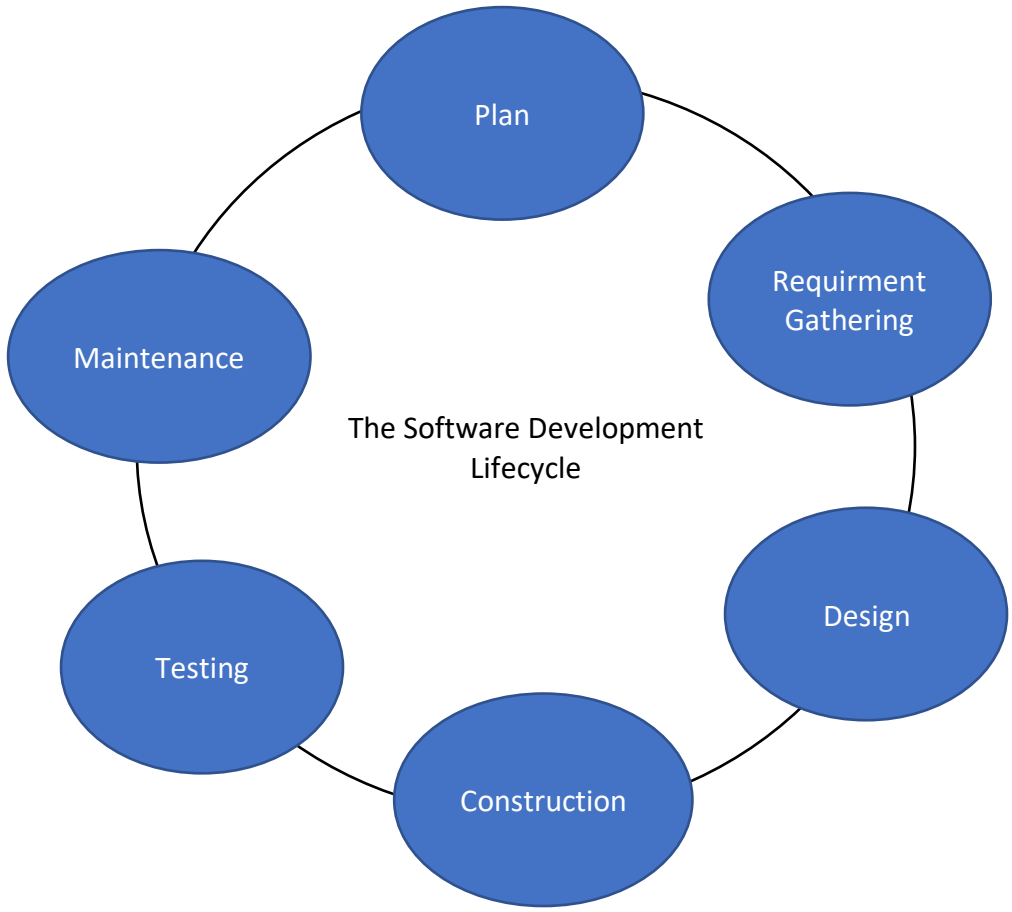
- Finally, the program will iterate through the list of employees and find the employees who have the top 3 vote count. To do this, the program will store the first 3 employee ids and vote count in a temp list of objects, navigate through the list and update the list as it finds an employee with a higher vote count.

- The top 3 funniest people chosen based on votes must be displayed to the user.

## 5. What's the Software Development Cycle?

The Software Development Lifecycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. SDLC works by lowering the cost of software development while simultaneously improving quality and shortening production time. SDLC achieves these divergent goals by following a plan that removes the typical pitfalls of software development projects. That plan starts by evaluating existing systems for deficiencies.

Next, it defines the requirements of the new system. It then creates the software through the planning, requirement gathering, design, construction, testing, and maintenance stages. By anticipating costly mistakes like failing to ask the end-user or client for feedback, SLDC can eliminate redundant rework and after-the-fact fixes.



In detail, the SDLC methodology focuses on the following phases of software development:

- Plan

"What do we want?" In this stage of the SDLC, the team determines the cost and resources required for implementing the analysed requirements. It also details the risks involved and provides sub-plans for softening those risks.

In other words, the team should determine the project's feasibility and how they can implement the project successfully with the lowest risk in mind.

- Requirements gathering

The product's features and how they should work are specified as part of the requirement-gathering process. Additionally, you will need to understand the processes involved in implementing the system and be able to communicate the potential implications of the design with the client. Requirements gathering is, therefore, integral to any software development project. In this part, we need to determine the project's risks. We communicate with the customer, and find out if something goes wrong with the project, what would be the risk of it? If one part of the project doesn't work, what would be the risk of it? It is essential to measure the risk because, with that, we can define the cost of the project

- Design

It is crucial to plan the system's design after gathering requirements and planning the project's logistics. As part of this process, the developer needs to carefully consider how each requirement will be incorporated into the system's architecture at both the high-level and low-level and the User Interface (UI). Architectural design is the first step in system design. This involves identifying the system's major components (front-end and back-end) and how they are interconnected and communicate with the user. To manage the front-end and back-end interactions, consider implementing the Model/View/Controller (MVC) framework.

- Construction

When the design phase is completed, the implementation of the system proceeds using the tools and procedures selected during the design phase. The development will be followed by the test-driven development (TDD) technique. During this phase of software development, the system is transformed into a functional application that can be implemented in real life.

- Testing

Software testing is an essential part of the software development process, where an application or software product is tested for functionality by evaluating and verifying that it performs as intended. Testing involves the evaluation of one or more properties of a software or system carried out by executing manual or automated tools. The purpose of software

testing is to detect bugs or errors in the software as early as possible to be corrected before the software product is released. The properly tested software is reliable, secure, and high performing, resulting in time savings, cost savings, and customer satisfaction.

- Maintenance

In the maintenance state, we integrate, deploy, and finally review. Maintenance is where we find bugs, fix them, look for essential requirements, and add changes. This is the long-term maintenance phase of our software. So, develop, evolve until we have a stable product that meets our needs, and then maintain it for as long as it has a useful life, after which we phase out or retire the software.

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

**Agile Methodology**

Agile is an iterative based software development approach where software is broken down into several iterations or sprints. Every iteration has phases like the waterfall model requirements gathering, design, development, testing and maintenance. So, the duration of each iteration is generally 2-8 weeks. So, in agile, the application is released for the first time with high-priority features in its first iteration. After release, the end user or customers give feedback about the application's performance, and the necessary changes are made to the application and some new features. The application is released, which is the second iteration. Therefore, this entire process is repeated until the desired software is achieved.

The main idea is that as the project moves through the phases on a macro level, the team adds value and reduces risk to the project. Value in an Agile project equates to working, production-quality code that meets the current requirements of the project. The more working code the team has delivered, the greater the value of the project and the lower the risk of successful delivery.

The agile approach isn't as intuitive and straightforward as opposed to the linear approach of the waterfall methodology. One thing that separates Agile from other approaches to software development is the focus on the people doing the work and how they work together. Solutions evolve through collaboration between self-organising cross-functional teams utilising the appropriate practices for their context. This methodology has been implemented using various agile frameworks, such as Scrum.

**Waterfall Methodology**

The waterfall approach to building software has been around for a long time and is simple and intuitive. The waterfall model is a sequential model that follows a top-down approach and has various phases starting with requirement gathering and analysis, software design implementation, verification, and maintenance. The main outline looks like an outline for approaching just about any project and consists of these parts:

- There is an idea for a project
- Initial requirements are gathered
- The next iteration is planned
- The features scheduled for that iteration are developed and tested
- The expected features for that iteration are delivered
- Wash, rinse, and repeat steps 2-5 until the project is ready to ship

Steps 2-5 are repeated until the project is ready to ship. Each time step 2 is repeated, the team re-evaluates the overall requirements and ensures that the project is still on the course relative to the current situation. This process rests on the premise that all requirements cannot be known upfront and that even the known upfront requirements are subject to change at any point.

Despite its inflexible nature, the waterfall approach is well-suited for specific projects. Generally, those projects involve systems that put people's lives on the line, such as avionics or medical equipment systems. These types of systems must get it right the first time and be verifiable against a stable specification because the price of a mistake is very high. This process is great if you have a project where all the requirements are known upfront, and you know they won't change.

## 7. What is a reduced function used for?

The reduce() function in python takes a pre-defined function and applies it to all the elements in an iterable (e.g., list, tuple, dictionary, etc.) and computes a single-valued result. This single-valued output results from applying the reduce function on the iterable passed as an argument; only a single integer, string, or boolean is returned. Like the map() and filter()functions, the reduce() function receives two arguments, a function and an iterable. However, it doesn't return another iterable; instead, it returns a single value. This function is defined in the "functools" module.

The syntax of the reduce function is:

functools.reduce(function, iterable)

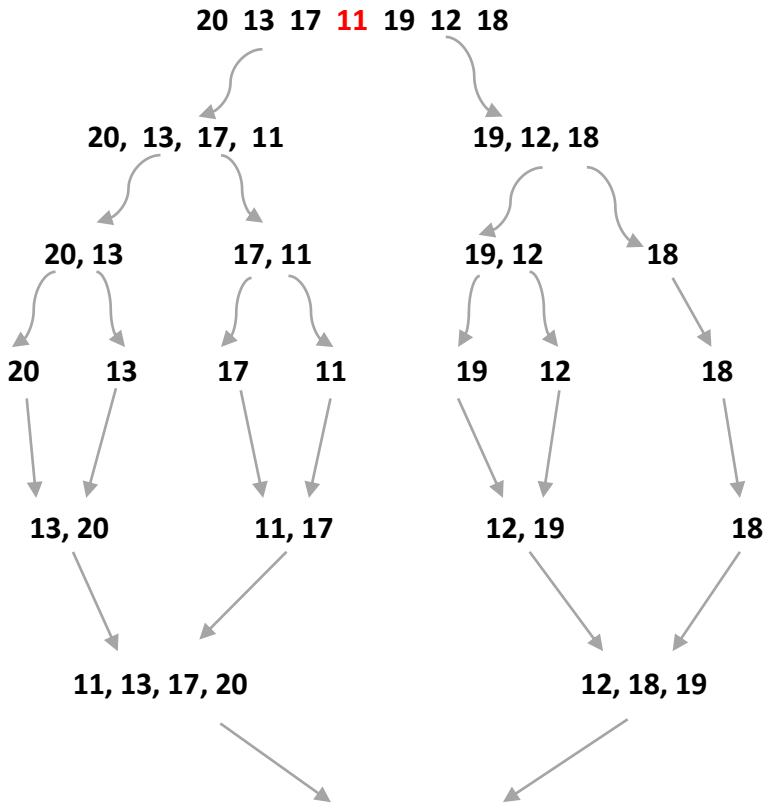
```
# importing functools for reduce()
import functools

# initializing list
nums = [5, 3, 2]

# using reduce to compute the multiple of the list
answer = functools.reduce(lambda a, b: a * b, nums)
print(f"The multiple of the list elements is: {answer}")

# OUTPUT: The sum of the list elements is: 30
```

## 8. How does merge sort work

The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner. The list is repeatedly divided into two until all the elements are separated individually. Pairs of elements are then compared, placed into order, and combined. The process is then repeated until the list is recompiled.

An array of Size 'N' is divided into two parts 'N/2' size of each. then those arrays are further divided till we reach a single element. The base case here is reaching one single element. When the base case is hit, we start merging the left part and the right part and we get a sorted array at the end. Merge sort repeatedly breaks down an array into several subarrays until each subarray consists of a single element and merging those subarrays in a manner that results in a sorted array.

**Merge sort algorithm flow**



**11, 12, 13, 17, 18, 19, 20**

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** allow you to declare a function that behaves like an iterator, i.e. it can be used in a for loop. Python provides a generator to create your own iterator function. A generator is a special type of function which does not return a single value, instead, it returns an iterator object with a sequence of values. In a generator function, a yield statement is used rather than a return statement.

```
# A generator function that yields 5 for first time,
# 10 second time and 15 third time
def generator():
    yield 5
    yield 10
    yield 15


# Driver code to check above generator function
for value in generator():
    print(value)

# OUTPUT: 5 10 15
```

**Use-cases**

Let's say we have to fetch 100K records of item numbers from an inventory. The beauty of this function is it creates a generator, it does not hold entire records in memory and yield one item number at a time whereas iterators load entire 100K records of item number in a memory and then iterate the process. This way it improves the performance efficiently.

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

A decorator takes in a function, add new functionality to an existing object without modifying its structure. Decorators are usually called before the definition of a function you want to decorate. You can pass the function as a parameter to another function. You can return the function from a function.

Python considers functions to be first-class citizens. In other words, they can be assigned to variables, used as arguments in functions, and returned from functions just like any other object. To make use of those last two points, decorators must take advantage of them. It is a function that takes another function as an argument and returns another function, or it is a function that wraps another function. They can, therefore, be used to modify the behaviour of the function passed as an argument without changing it explicitly. It is still possible to use the original function in its original form, but the modified version is also available.

This simple example demonstrates how you can create a decorator function passed to another function as an argument without directly modifying the original. In this example, the original function course() prints a course message – " I am currently enrolled in a python course." The decorator function calls the course() function with the action function used as an argument.

```
def course(func):
    func()
    print("I am currently enrolled in a python course.")


def action():
    print("Python is a fun programming language.")


course(action)

# OUTPUT: Python is a fun programming language.
# I am currently enrolled in a python course.
```