**NAME: UYE TIMILEYIN**

**MATRIC NO: 17/SCI01/081**

**COURSE CODE: CSC 302**

**COURSE TITLE: SURVEY OF PROGRAMMING LANGUAGES**

**QUESTIONS**

1. With examples of different programming languages show how programming languages are organized along the given rubrics:

i. Unstructured, structured, modular, object oriented, aspect oriented, activity oriented and event oriented programming requirement.

ii. Based on domain requirements.

iii. Based on requirements i and ii above.

2. Give brief preview of the evolution of programming languages in a chronological order.

3. Vividly distinguish between modular programming paradigm and object oriented programming paradigm.

**SOLUTION**

**QUESTION 1**

**i. Unstructured programming language:**

In Unstructured Programming, the code is written as a single whole block. The whole program is taken as a single unit. It is harder to do changes in the program. This paradigm was used in earlier versions of **BASIC, COBOL, and FORTRAN.** Unstructured programming languages have a limited number of data types like numbers, arrays,strings. Unstructured programming is a procedural program – the statements are executed in sequence as written(The statements execute in order you write)

In this you have to use goto statements that allow the control to pass. When a goto statement is executed, the sequence continues from the target of the goto. Thus to understand how a program works, you have to pretend to execute it. This means that it is often difficult to understand the logic of such a program

Structured programming languages: Structured programming is a programming paradigm that facilitates the creation of programs with readable code and reusable components. All modern programming languages support structured programming, but the mechanisms of support, like the [syntax](https://whatis.techtarget.com/definition/syntax) of the programming languages, varies.Where modules or elements of code can be reused from a library, it may also be possible to build structured code using modules written in different languages, as long as they can obey a common module interface or application program interface ([API](https://searchapparchitecture.techtarget.com/definition/application-program-interface-API)) specification. However, when modules are reused, it's possible to compromise data security and [governance](https://searchdatamanagement.techtarget.com/definition/data-governance), so it's important to define and enforce a privacy policy controlling the use of modules that bring with them implicit data access rights. Structured programming encourages dividing an application program into a hierarchy of modules or autonomous elements, which may, in turn, contain other such elements. Within each element, code may be further structured using blocks of related logic designed to improve readability and maintainability. These may include case, which tests a variable against a set of values; Repeat, while and for, which construct loops that continue until a condition is met. In all structured programming languages, an unconditional transfer of control, or goto statement, is deprecated and sometimes not even available. Examples of structured programming language includes one of the languages initially used for structured programming include: ALGOL, Pascal, PL/I and Ada, but most new procedural programming languages since that time have included features to encourage structured programming, and sometimes deliberately left out features – notably GOTO

**Modular Programming Language:**

Modular programming is the process of subdividing a computer program into separate sub-programs.

A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system. Similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code so that the code can be reused by other applications.

Modules in modular programming enforce logical boundaries between components and improve maintainability. They are incorporated through interfaces. They are designed in such a way as to minimize dependencies between different modules. Teams can develop modules separately and do not require knowledge of all modules in the system.  
  
Each and every modular application has a version number associated with it. This provides developers flexibility in module maintenance. If any changes have to be applied to a module, only the affected subroutines have to be changed. This makes the program easier to read and understand.  
  
Modular programming has a main module and many auxiliary modules. The main module is compiled as an executable (EXE), which calls the auxiliary module functions. Auxiliary modules exist as separate executable files, which load when the main EXE runs. Each module has a unique name assigned in the PROGRAM statement. Function names across modules should be unique for easy access if functions used by the main module must be exported.  
  
Languages that support the module concept are IBM Assembler, COBOL, RPG, FORTRAN, Morpho, Zonnon and Erlang, among others.  
  
The benefits of using modular programming include:

* Less code has to be written.
* A single procedure can be developed for reuse, eliminating the need to retype the code many times.
* Programs can be designed more easily because a small team deals with only a small part of the entire code.
* Modular programming allows many programmers to collaborate on the same application.
* The code is stored across multiple files.
* Code is short, simple and easy to understand.
* Errors can easily be identified, as they are localized to a subroutine or function.
* The same code can be used in many applications.
* The scoping of variables can easily be controlled.

### Object Oriented Programming language

### : As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function. Principles of OOP

Object-oriented programming is based on the following principles:

* [Encapsulation](https://searchnetworking.techtarget.com/definition/encapsulation). The implementation and state of each object are privately held inside a defined boundary, or class. Other objects do not have access to this class or the authority to make changes but are only able to call a list of public functions, or methods. This characteristic of [data hiding](https://searchsqlserver.techtarget.com/definition/data-hiding) provides greater program security and avoids unintended [data corruption](https://searchsqlserver.techtarget.com/definition/data-corruption).
* [Abstraction](https://whatis.techtarget.com/definition/abstraction). Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary [implementation](https://searchcustomerexperience.techtarget.com/definition/implementation) code. This concept helps developers more easily make changes and additions over time.
* [Inheritance](https://whatis.techtarget.com/definition/inheritance). Relationships and subclasses between objects can be assigned, allowing developers to reuse a common logic while still maintaining a unique hierarchy. This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.
* [Polymorphism](https://whatis.techtarget.com/definition/polymorphism). Objects can take on more than one form depending on the context. The program will determine which meaning or usage is necessary for each execution of that object, cutting down the need to duplicate code.

The example of the most popular OOP languages are:

* [Java](https://www.theserverside.com/definition/Java)
* [JavaScript](https://www.theserverside.com/definition/JavaScript)
* [Python](https://whatis.techtarget.com/definition/Python)
* [C++](https://searchsqlserver.techtarget.com/definition/C)
* [Visual Basic .NET](https://searchwindevelopment.techtarget.com/definition/Visual-Basic-NET)
* [Ruby](https://whatis.techtarget.com/definition/Ruby)
* [Scala](https://searchbusinessanalytics.techtarget.com/definition/Scala-Scalable-Language)
* [PHP](https://whatis.techtarget.com/definition/Personal-Home-Page-PHP)

[OOPSLA](https://searchapparchitecture.techtarget.com/definition/OOPSLA-Object-Oriented-Programming-Systems-Languages-and-Applications) is the annual conference for Object-Oriented Programming Systems, Languages and Applications.

**Aspect Oriented programming language:**

Aspect-Oriented Programming (AOP) is a programming paradigm which complements Object-Oriented Programming (OOP) by separating *concerns* of a software application to improve modularization. The separation of concerns (SoC) aims for making a software easier to maintain by grouping features and behavior into manageable parts which all have a specific purpose and business to take care of.

OOP already allows for modularizing concerns into distinct methods, classes and packages. However, some concerns are difficult to place as they cross the boundaries of classes and even packages. One example for such a *cross-cutting concern* is security: Although the main purpose of a Forum package is to display and manage posts of a forum, it has to implement some kind of security to assert that only moderators can approve or delete posts. And many more packages need a similar functionality for protect the creation, deletion and update of records. AOP enables you to move the security (or any other) aspect into its own package and leave the other objects with clear responsibilities, probably not implementing any security themselves.Aspect-Oriented Programming has been around in other programming languages for quite some time now and sophisticated solutions taking advantage of AOP exist. Flow’s AOP framework allows you to use of the most popular AOP techniques in your own PHP application. In contrast to other approaches it doesn’t require any special PHP extensions or manual compile steps – and it’s a breeze to configure. Aspect is a common feature that's typically scattered across methods, classes, object hierarchies, or even entire object models. It is behavior that looks and smells like it should have structure, but you can't find a way to express this structure in code with traditional object-oriented techniques.

For example, metrics is one common aspect. To generate useful logs from your application, you have to (often liberally) sprinkle informative messages throughout your code. However, metrics is something that your class or object model really shouldn't be concerned about. After all, metrics is irrelevant to your actual application: it doesn't represent a customer or an account, and it doesn't realize a business rule. It's simply orthogonal.

In AOP, a feature like metrics is called a crosscutting concern, as it's a behavior that "cuts" across multiple points in your object models, yet is distinctly different. As a development methodology, AOP recommends that you abstract and encapsulate crosscutting concerns.

Examples of aspect oriented programming language: logging,caching e.t.c

**Activity and Event oriented programming language:**

Event programming is a programming paradigm in which the flow of program execution is determined by events - for example a user action such as a mouse click, key press, or a message from the operating system or another program. An event-driven application is designed to detect events as they occur, and then deal with them using an appropriate event-handling procedure. The idea is an extension of interrupt-driven programming of the kind found in early command-line environments such as DOS, and in embedded systems (where the application is implemented as firmware).

Event-driven programs can be written in any programming language, although some languages(Visual Basic for example) are specifically designed to facilitate event-driven programming, and provide an integrated development environment (IDE) that partially automates the production of code, and provides a comprehensive selection of built-in objects and controls, each of which can respond to a range of events. Virtually all object-oriented and visual languages support event-driven programming. Visual Basic, Visual C++ and Java are examples of such languages.

A visual programming IDE such as VB.Net provides much of the code for detecting events automatically when a new application is created. The programmer can therefore concentrate on issues such as interface design, which involves adding controls such as command buttons, text boxes, and labels to standard forms (a form represents an application's workspace or window). Once the user interface is substantially complete, the programmer can add event-handling code to each control as required.

Many visual programming environments will even provide code templates for event-handlers, so the programmer only needs to provide the code that defines the action the program should take when the event occurs. Each event-handler is usually bound to a specific object or control on a form. Any additional subroutines, methods, or function procedures required are usually placed in a separate code module, and can be called from other parts of the program as and when needed. The example of an event driven programming language includes VBASIC (Visual Basic).

**QUESTION 2**

The following are the chronological order of some popular and/or important programming languages.

**1840 – Analytical Engine Code**  
The Analytical Engine was a theoretical (i.e., never built) mechanical general-purpose computer, created by British mathematician Charles Babbage. The machine was designed to consist of four components: the mill, the store, the reader, and the printer. These components are the essential components of every computer today. The mill was the calculating unit, analogous to the central processing unit (CPU) in a modern computer; the store was where data were held prior to processing, exactly analogous to memory and storage in today’s computers; and the reader and printer were the input and output devices.

**1943 – ENIAC Coding System**  
The ENIAC is regarded as the first electronic general-purpose computer. Both the computer and its coding were created by John von Neumann, John Mauchly, and J. Presper Eckert. It is the first electronic general-purpose digital computer. It was Turing-complete, and able to solve "a large class of numerical problems" through reprogramming. It had a speed on the order of one thousand times faster than that of electro-mechanical machines; this computational power, coupled with general-purpose programmability, excited scientists and industrialists alike. The combination of speed and programmability allowed for thousands more calculations for problems, as ENIAC calculated a trajectory in 30 seconds that took a human 20 hours (allowing one ENIAC hour to displace 2,400 human hours).

**1949 – Brief Code (Later Short Code)**  
Initially proposed by John Mauchly, it was one of the first attempts of an assembly language. **Short Code** was one of the first higher-level languages ever developed for an electronic computer .Unlike machine code, Short Code statements represented mathematic expressions rather than a machine instruction. Also known as an automatic programming, the source code was not compiled but executed through an interpreter to simplify the programming process; the execution time was much slower though

**1954 – Fortran**  
One of the most popular high-level programming languages. It was created by John W. Backus at IBM as an easier alternative to programming in assembly. **Fortran** is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing.  It is a popular language for high-performance computing and is used for programs that benchmark and rank the world's fastest supercomputers.

Fortran encompasses a lineage of versions, each of which evolved to add extensions to the language while usually retaining compatibility with prior versions.

**1958 – LISP**

**LISP**  (historically **LISP**) is a family of programming languageswith a long history and a distinctive, fully parenthesized prefix notation. Originally specified in 1958, Lisp is the second-oldest high-level programming language in widespread use today. Only Fortran is older, by one year. Lisp has changed since its early days, and many dialects have existed over its history. Today, the best-known general-purpose Lisp dialects are Common Lisp, Scheme and Clojure.  It quickly became the favored programming language for artificial intelligence (AI) research. As one of the earliest programming languages, Lisp pioneered many ideas in computer science, including tree data structures, automatic storage management, dynamic typing, conditionals, higher-order functions, recursion, the self-hosting compiler. and the read–eval–print loop.

**1959 – COBOL**  
The name stand for Common Business-Oriented Language, as the language was aimed mainly at banks, financial institutions and companies. is a compiled English-like computer programming language designed for business use. It is imperative, procedural and, since 2002, object-oriented. COBOL is primarily used in business, finance, and administrative systems for companies and governments. COBOL is still widely used in applications deployed on mainframe computers, such as large-scale batch and transaction processing Jobs. But due to its declining popularity and the retirement of experienced COBOL programmers, programs are being migrated to new platforms, rewritten in modern languages or replaced with software packages. Most programming in COBOL is now purely to maintain existing applications, however many large financial institutions were still developing new systems in COBOL in 2006 due to the mainframe processing speed

**1964 – BASIC**  
Beginner’s All-purpose Symbolic Instruction Code, a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use. is a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use. The original version was designed by John G. Kemeny and Thomas E. Kurtz in 1964. They wanted to enable students in fields other than science and mathematics to use computers. At the time, nearly all use of computers required writing custom software, which was something only scientists and mathematicians tended to learn.

**1970 – Pascal**  
Pascal is an influential imperative and procedural programming language, designed in 1968–1969 and published in 1970 by Niklaus Wirth as a small and efficient language intended to encourage good programming practices using structured programming and data structuring.

Pascal enabled defining complex datatypes and building dynamic and recursive data structures such as [lists](https://en.wikipedia.org/wiki/List_(abstract_data_type)), [trees](https://en.wikipedia.org/wiki/Tree_(data_structure)) and [graphs](https://en.wikipedia.org/wiki/Graph_(abstract_data_type)). Pascal has [strong typing](https://en.wikipedia.org/wiki/Strong_typing) on all objects, which means that one type of data cannot be converted or interpreted as another without explicit conversions. Unlike most languages in the C-family, Pascal allows nested procedure definitions to any level of depth, and also allows most kinds of definitions and declarations inside subroutines (procedures and functions). A program is thus syntactically similar to a single procedure or function

**1972 – Smalltalk**  
The language that started to inflate the popularity of object-oriented programming. **Smalltalk** is an object-oriented, dynamically typed reflective programming language. Smalltalk was created as the language underpinning the "new world" of computing exemplified by "human–computer symbiosis" It was designed and created in part for educational use, specifically for constructionist learning.  Smalltalk-like languages are in active development and have gathered loyal communities of users around them.

**1972 – C**  
Created by Dennis Ritchie and Ken Thompson at the AT&T Bell Labs. It’s simplicity and efficiency made it one of the most popular languages around the world. C is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, while a static type system prevents unintended operations. By design, C provides constructs that map efficiently to typical machine instructions and has found lasting use in applications previously coded in assembly language. Such applications include operating systems and various application software for computers, from supercomputers to PLCs and, to embedded systems.

**1972 – SQL**  
Created at IBM, it became the standard for dealing with databases. **Structured Query Language**) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

SQL offers two main advantages over older read–write APIs such as ISAM or VSAM. Firstly, it introduced the concept of accessing many records with one single command. Secondly, it eliminates the need to specify *how* to reach a record, e.g. with or without an index .The scope of SQL includes data query, data manipulation (insert, update and delete), data definition (schema creation and modification), and data access control. Although SQL is essentially a declarative language (4GL), it also includes procedural elements.

**1983 – C++**  
Originally named “C With Classes”, it brought object-orientation to C (which is technically a subset of C++). **C++**  is a high level general purpose programming language created by Bjarne Stroustrope as an extension of the C programming language, or "C with classes ". The language has expanded significantly over time, and modern C++ now has objected-priented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as compiled languages, and many vendors provide C++ compilers.

C++ was designed with a bias toward system programming and embedded, resource-constrained software and large systems, with performance, efficiency, and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications, including, video games, servers (e.g. e-commerce, web search, or SQL servers), and performance-critical applications (e.g. telephone switches or space probes)

**1987 – Perl**  
Perl is a family of high-level, general-purpose, interpreted, dynamic programming languages. Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

**1991 – Python**  
A high-level language that emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C.**Python** is an interpreted,, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

**1995 – Java**  
Java is the most popular object-oriented programming language around, and it was created to have as few implementation dependencies as possible. It’s widely used in commercial and business applications. **Java** is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible. It is intended to let application developers *write once, run anywhere* (WORA) meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but it has fewer low-level facilities than either of them. As of 2019, Java was one of the most popular programming languages in use according to GitHub,particularly for client-server web applications, with a reported 9 million developers

**QUESTION 3**

An object-oriented program usually contains different types of objects, each corresponding to a particular kind of complex data to manage, or perhaps to a real-world object or concept such as a bank account, a hockey player, or a bulldozer.

Modular programming (also called "top-down design" and "stepwise refinement") is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

Differences that I can think of are that you can have more than one objects on a class, where as in modular programming you are supposed to have only 1 module (1 object) for one specific thing. An object oriented program contains different types of objects, each corresponding to a complex real world objects or any complex data or a concept such as a bank customer, a bank account or any departmental store.  
  
Modular Programming (aka 'stepwise refinement' and 'top-down design' paradigm) is a software designing technique that emphasizes separating the functionalities of a program into independent and meaningful modules, such that each module contains everything necessary for executing the one (and only one) aspect of the desired functionality!

**Modular programming** just implies you have these two (or more) modules, but says nothing of how they achieve what they achieve. The modules can use object-oriented approaches or not at all and use procedural C-style programming. The way you described modular programming via classes is just a way of separating modules. You can separate them as classes, or you can separate them as functions across multiple compilation units, for example. It's your choice.

**Object-oriented programming**implies that your program is, well, **oriented towards objects**. It says nothing about modules within your application but demands that logical pieces that represent some ideas within the application are modeled via classes and objects.

As such, the two approaches can be used together, and when you decide to be modular, the object-oriented choice usually imposes on you that these modules are defined via classes and their relationships.