**NAME: GABRIEL-OHANU VICTOR**

**MATRIC NUMBER: 17/SCI01/035**

**COURSE CODE: CSC302**

1

Unstructured programming language - COBOL, FORTRAN, BASIC, MUMPS, FOCAL

Structured programming language **-** C, C++, PHP, ALGOL, ADA

Procedural programming language **-** FORTRAN, BASIC, C, Java, Pascal

Modular programming language **-** COBOL, Morpho, RPG, Zonnon, Erlang

Object-oriented programming language **-** Python, Java, C++, Ruby, Emerald

Aspect oriented language **-** Aspect Java, C++, C#, Smalltalk, Aspect C

Event oriented programming **-** Visual Basic, Visual C++, Java

II.

General purpose Domain – PASCAL, DELPH, JAVA, C, PYTHON

Business Domain – COBOL, BPEL, PYTHON, PERL, RUBY

Artificial intelligence Domain – LISP, PYTHON, R, PROLOG, LDT

Web programming Domain – JAVA, C#, PHP, HTML, JAVASCRIPT

Mobile programming Domain – JAVA, C#, PASCAL, PHP, PYTHON

Embedded device Domain – PYTHON, JAVASCRIPT, C, C++, ADA

Scientific Domain – FORTRAN, ALGOL, MATLAB, R

III.

|  |  |  |
| --- | --- | --- |
| **PROGRAMMING** **LANGUAGE** | **DOMAIN** **REQUEST** | **CLASSIFICATION** **OF** **PROGRAMMING** **LANGUAGE** |
| Java | General Use | Object Oriented Programming Language |
| C | General Use | Structured Programming Language |
| Visual Basic Net | Local Applications | Event Oriented Programming Language |
| PAP | Web Applications | Structured Programming Language |
| Ada | General Use | Structured Programming Language |
| Objective-C | Apple IOS devices | Object Oriented Programming Language |
| Erlang | Telecom, e-commerce, etc | Modular Programming Language |
| MUMPS | Scientific Application | Unstructured Programming Language |
| FOCAL | Digital equipment Corporation | Unstructured Programming Language |
| SQL | Database Application | Structured Programming Language |
| Latex | Documentation | Structured Programming Language |
| BPEL | Business Application | Structured Programming Language |
| HTML | Web Application | Structured Programming Language |
| Python | General Use | Object Oriented Programming |
| COBOL | Business Application | Unstructured Programming Language |
| PHP | Web Application | Structured Programming Language |

2.

|  |  |  |
| --- | --- | --- |
| **YEAR** | **PROGRAMMING** **LANGUAGE** | **BRIEF** **HISTORY** |
| 1840 | Analytical Engine Code | The Analytical Engine was a theoretical (i.e., never built) mechanical general-purpose computer, created by British mathematician Charles Babbage. Ada Lovelace came across the idea, and created some code for the Analytical Engine. That’s why she’s considered the first programmer ever. |
| 1943 | ENIAC Code 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. |
| 1949 | Brief Code (Later Short Code) | Initially proposed by John Mauchly, it was one of the first attempts of an assembly language. |
| 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. |
| 1958  | LISP | Created by John McCarthy, one of the pioneers of AI as well. |
| 1959 | COBOL | The name stands for COmmon Business-Oriented Language, as the language was aimed mainly at banks, financial institutions and companies. |
| 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. |
| 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. |
| 1972 | Smalltalk | The language that started to inflate the popularity of object-oriented programming. |
| 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. |
| 1972 | SQL | Created at IBM, it became the standard for dealing with databases. |
| 1983 | C++ | Originally named “C With Classes”, it brought object-orientation to C (which is technically a subset of C++). |
| 1987 | Perl | Perl is a family of high-level, general-purpose, interpreted, dynamic programming languages. |
| 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. |
| 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. |

3.

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.

The relationship between objects and modules is very confusing. Perhaps the best way to answer that question is to look at it historically.

Modules predate objects. Modules made their debut in 1968 when Larry Constantine organized the National Symposium on Modular Programming. Precursors to objects were already around at that time, but modern OOP didn't gain popularity until the early 90's.

Modular programming was devised in a predominately procedural world. Modules drew a box around a set of procedures and required that box to be independently deployable. It was an organization scheme for keeping procedures neat and tidy. Different languages might offer different mechanisms for implementing that scheme.

Metrics became a big part of the modular worldview. *Modularity* is defined as a function of *coupling* and *cohesion*, where both of those terms are also metrics. A project can be objectively evaluated to see how modular it is. A number of tools were developed for analyzing code and coming up with a set of numeric values based on different measures of *software quality*.

Objects form a box around a set of procedures, which qualifies them as modules. But they do much more than that, and some languages use the term module to refer to even higher levels of code organization.

Modular programming doesn't really live an independent existence these days. Modular concepts are baked into nearly every language and developer tool, and there's really no consistency between each of them. Every language is a reimagination of the modular programming tenets, and ultimately it's up to the programmer to use the tools available to create modular code.