Assignment for

CSC 302

(Survey of programming)

by

Adesina Alameen Boluwatife

18/SCI01/099

COMPUTER SCIENCE

To be submitted to

Dr Aweh

 Programming languages and their organization along the following rubrics

a) Unstructured programming languages

Unstructured Programming is historically the earliest type of programming that was capable of creating Turing-complete algorithms. As it was the earliest, it had its own set of advantages and disadvantages. Eventually, unstructured programming morphed and evolved into structured programming, which was easier to use. Examples include: early versions of BASIC (such as MSX BASIC and GW-BASIC), JOSS, FOCAL, MUMPS, TELCOMP, COBOL, machine-level code, early assembler systems (without procedural metaoperators), assembler debuggers and some scripting languages such as MS-DOS batch file language.

b) Structured programming languages

 Structured programming language allows a programmer to code a program by diving the whole program into smaller units or modules. This makes it easier to code, as the programmer can work on one segment of the code at a time. This also allows the programmer to check the module individually, before combining it with the program. Hence, it becomes easier to modify and debug, as the programmer can check and modify a single module, while leaving the rest of the program as is. Examples include: C, C+, C++, C#, Java, PERL, Ruby, PHP, ALGOL, Pascal, PL/I and Ada

c)Modular programming languages

 Modular programming, in the form of subsystems (particularly for I/O) and software libraries, dates to early software systems, where it was used for [code reuse](https://en.wikipedia.org/wiki/Code_reuse). Modular programming per se, with a goal of modularity, developed in the late 1960s and 1970s, as a larger-scale analog of the concept of [structured programming](https://en.wikipedia.org/wiki/Structured_programming) (1960s). The term "modular programming" dates at least to the National Symposium on Modular Programming, organized at the Information and Systems Institute in July 1968 by [Larry Constantine](https://en.wikipedia.org/wiki/Larry_Constantine); other key concepts were [information hiding](https://en.wikipedia.org/wiki/Information_hiding)(1972) and [separation of concerns](https://en.wikipedia.org/wiki/Separation_of_concerns) (SoC, 1974).

Modules were not included in the original specification for [ALGOL 68](https://en.wikipedia.org/wiki/ALGOL_68) (1968), but were included as extensions in early implementations, [ALGOL 68-R](https://en.wikipedia.org/wiki/ALGOL_68-R) (1970) and [ALGOL 68C](https://en.wikipedia.org/wiki/ALGOL_68C) (1970), and later formalized.[[6]](https://en.wikipedia.org/wiki/Modular_programming#cite_note-7) One of the first languages designed from the start for modular programming was the short-lived [Modula](https://en.wikipedia.org/wiki/Modula) (1975), by [Niklaus Wirth](https://en.wikipedia.org/wiki/Niklaus_Wirth). Another early modular language was [Mesa](https://en.wikipedia.org/wiki/Mesa_%28programming_language%29) (1970s), by [Xerox PARC](https://en.wikipedia.org/wiki/Xerox_PARC), and Wirth drew on Mesa as well as the original Modula in its successor, [Modula-2](https://en.wikipedia.org/wiki/Modula-2) (1978), which influenced later languages, particularly through its successor, [Modula-3](https://en.wikipedia.org/wiki/Modula-3) (1980s). Modula's use of dot-[qualified names](https://en.wikipedia.org/wiki/Qualified_name), like M.a to refer to object a from module M, coincides with notation to access a field of a record (and similarly for attributes or methods of objects), and is now widespread, seen in C#, Dart, Go, Java, and Python, among others. Modular programming became widespread from the 1980s: the original [Pascal](https://en.wikipedia.org/wiki/Pascal_%28programming_language%29) language (1970) did not include modules, but later versions, notably [UCSD Pascal](https://en.wikipedia.org/wiki/UCSD_Pascal) (1978) and [Turbo Pascal](https://en.wikipedia.org/wiki/Turbo_Pascal) (1983) included them in the form of "units", as did the Pascal-influenced [Ada](https://en.wikipedia.org/wiki/Ada_%28programming_language%29) (1980). The Extended Pascal ISO 10206:1990 standard kept closer to Modula2 in its modular support. [Standard ML](https://en.wikipedia.org/wiki/Standard_ML) (1984)[[7]](https://en.wikipedia.org/wiki/Modular_programming#cite_note-8) has one of the most complete module systems, including [functors](https://en.wikipedia.org/wiki/Standard_ML%22%20%5Cl%20%22Module_system%22%20%5Co%20%22Standard%20ML) (parameterized modules) to map between modules.

In the 1980s and 1990s, modular programming was overshadowed by and often conflated with [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), particularly due to the popularity of C++ and Java. For example, the C family of languages had support for objects and classes in C++ (originally [C with Classes](https://en.wikipedia.org/wiki/C_with_Classes), 1980) and Objective-C (1983), only supporting modules 30 years or more later. Java (1995) supports modules in the form of packages, though the primary unit of code organization is a class. However, Python (1991) prominently used both modules and objects from the start, using modules as the primary unit of code organization and "packages" as a larger-scale unit; and [Perl 5](https://en.wikipedia.org/wiki/Perl_5) (1994) includes support for both modules and objects, with a vast array of modules being available from [CPAN](https://en.wikipedia.org/wiki/CPAN) (1993).

Modular programming is now widespread, and found in virtually all major languages developed since the 1990s. The relative importance of modules varies between languages, and in class-based object-oriented languages there is still overlap and confusion with classes as a unit of organization and encapsulation, but these are both well-established as distinct concepts.

d) Object Oriented programming languages

 In the early and mid-1990s object-oriented programming developed as the dominant programming [paradigm](https://en.wikipedia.org/wiki/Paradigm) when programming languages supporting the techniques became widely available. These included Visual [FoxPro](https://en.wikipedia.org/wiki/FoxPro) 3.0, [C++](https://en.wikipedia.org/wiki/C%2B%2B), and [Delphi](https://en.wikipedia.org/wiki/Embarcadero_Delphi). Its dominance was further enhanced by the rising popularity of [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface), which rely heavily upon object-oriented programming techniques. An example of a closely related dynamic GUI library and OOP language can be found in the [Cocoa](https://en.wikipedia.org/wiki/Cocoa_%28software%29) frameworks on [Mac OS X](https://en.wikipedia.org/wiki/Mac_OS_X), written in [Objective-C](https://en.wikipedia.org/wiki/Objective-C), an object-oriented, dynamic messaging extension to C based on Smalltalk. OOP toolkits also enhanced the popularity of [event-driven programming](https://en.wikipedia.org/wiki/Event-driven_programming) (although this concept is not limited to OOP). Examples include: [Simula](https://en.wikipedia.org/wiki/Simula) (1967) is generally accepted as being the first language with the primary features of an object-oriented language. It was created for making [simulation programs](https://en.wikipedia.org/wiki/Computer_simulation), in which what came to be called objects were the most important information representation. [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk) (1972 to 1980) is another early example, and the one with which much of the theory of OOP was developed. Concerning the degree of object orientation, the following distinctions can be made:

* Languages called "pure" OO languages, because everything in them is treated consistently as an object, from primitives such as characters and punctuation, all the way up to whole classes, prototypes, blocks, modules, etc. They were designed specifically to facilitate, even enforce, OO methods. Examples: [Python](https://en.wikipedia.org/wiki/Python_%28programming_language%29), [Ruby](https://en.wikipedia.org/wiki/Ruby_%28programming_language%29), [Scala](https://en.wikipedia.org/wiki/Scala_%28programming_language%29), [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk), [Eiffel](https://en.wikipedia.org/wiki/Eiffel_%28programming_language%29), [Emerald](https://en.wikipedia.org/wiki/Emerald_%28programming_language%29),[[26]](https://en.wikipedia.org/wiki/Object-oriented_programming#cite_note-26) [JADE](https://en.wikipedia.org/wiki/JADE_%28programming_language%29), [Self](https://en.wikipedia.org/wiki/Self_%28programming_language%29).
* Languages designed mainly for OO programming, but with some procedural elements. Examples: [Java](https://en.wikipedia.org/wiki/Java_%28programming_language%29), [C++](https://en.wikipedia.org/wiki/C%2B%2B), [C#](https://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29), [Delphi/Object Pascal](https://en.wikipedia.org/wiki/Object_Pascal), [VB.NET](https://en.wikipedia.org/wiki/VB.NET).
* Languages that are historically [procedural languages](https://en.wikipedia.org/wiki/Procedural_programming), but have been extended with some OO features. Examples: [PHP](https://en.wikipedia.org/wiki/PHP), [Perl](https://en.wikipedia.org/wiki/Perl), [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic) (derived from BASIC), [MATLAB](https://en.wikipedia.org/wiki/MATLAB), [COBOL 2002](https://en.wikipedia.org/wiki/COBOL_2002), [Fortran 2003](https://en.wikipedia.org/wiki/Fortran_2003), [ABAP](https://en.wikipedia.org/wiki/ABAP), [Ada 95](https://en.wikipedia.org/wiki/Ada_%28programming_language%29), [Pascal](https://en.wikipedia.org/wiki/Pascal_%28programming_language%29).
* Languages with most of the features of objects (classes, methods, inheritance), but in a distinctly original form. Examples: [Oberon](https://en.wikipedia.org/wiki/Oberon_%28programming_language%29) (Oberon-1 or Oberon-2).
* Languages with [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type) support which may be used to resemble OO programming, but without all features of object-orientation. This includes [object-*based*](https://en.wikipedia.org/wiki/Object-based) and [prototype-based](https://en.wikipedia.org/wiki/Prototype-based_programming) languages. Examples: [JavaScript](https://en.wikipedia.org/wiki/JavaScript), [Lua](https://en.wikipedia.org/wiki/Lua_%28programming_language%29), [Modula-2](https://en.wikipedia.org/wiki/Modula-2), [CLU](https://en.wikipedia.org/wiki/CLU_%28programming_language%29).
* Chameleon languages that support multiple paradigms, including OO. [Tcl](https://en.wikipedia.org/wiki/Tcl%22%20%5Co%20%22Tcl) stands out among these for TclOO, a hybrid object system that supports both [prototype-based programming](https://en.wikipedia.org/wiki/Prototype-based_programming) and class-based OO.

e) Aspect oriented programming languages

 In [computing](https://en.wikipedia.org/wiki/Computing), aspect-oriented programming (AOP) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) that aims to increase [modularity](https://en.wikipedia.org/wiki/Modularity_%28programming%29) by allowing the [separation of](https://en.wikipedia.org/wiki/Separation_of_concerns) [cross-cutting concerns](https://en.wikipedia.org/wiki/Cross-cutting_concern). It does so by adding additional behavior to existing code (an [advice](https://en.wikipedia.org/wiki/Advice_%28programming%29)) *without* modifying the code itself, instead separately specifying which code is modified via a "[pointcut](https://en.wikipedia.org/wiki/Pointcut)" specification, such as "log all function calls when the function's name begins with 'set'". This allows behaviors that are not central to the [business logic](https://en.wikipedia.org/wiki/Business_logic) (such as logging) to be added to a program without cluttering the code, core to the functionality. AOP forms a basis for [aspect-oriented software development](https://en.wikipedia.org/wiki/Aspect-oriented_software_development).

AOP includes programming methods and tools that support the modularization of concerns at the level of the source code, while "aspect-oriented software development" refers to a whole engineering discipline.

Aspect-oriented programming entails breaking down program logic into distinct parts (so-called *concerns*, cohesive areas of functionality). Nearly all programming paradigms support some level of grouping and [encapsulation](https://en.wikipedia.org/wiki/Encapsulation_%28computer_science%29) of concerns into separate, independent entities by providing abstractions (e.g., functions, procedures, modules, classes, methods) that can be used for implementing, abstracting and composing these concerns. Some concerns "cut across" multiple abstractions in a program, and defy these forms of implementation. These concerns are called *cross-cutting concerns* or horizontal concerns.

AOP has several direct antecedents A1 and A2:[[1]](https://en.wikipedia.org/wiki/Aspect-oriented_programming#cite_note-1) [reflection](https://en.wikipedia.org/wiki/Reflection_%28computer_programming%29) and [metaobject protocols](https://en.wikipedia.org/wiki/Metaobject), [subject-oriented programming](https://en.wikipedia.org/wiki/Subject-oriented_programming), Composition Filters and Adaptive Programming.[[2]](https://en.wikipedia.org/wiki/Aspect-oriented_programming#cite_note-2)

[Gregor Kiczales](https://en.wikipedia.org/wiki/Gregor_Kiczales) and colleagues at [Xerox PARC](https://en.wikipedia.org/wiki/Xerox_PARC) developed the explicit concept of AOP, and followed this with the [AspectJ](https://en.wikipedia.org/wiki/AspectJ) AOP extension to Java. IBM's research team pursued a tool approach over a language design approach and in 2001 proposed [Hyper/J](https://en.wikipedia.org/wiki/Hyper/J) and the [Concern Manipulation Environment](https://en.wikipedia.org/wiki/Concern_Manipulation_Environment), which have not seen wide usage.

The examples in this article use AspectJ.

The [Microsoft Transaction Server](https://en.wikipedia.org/wiki/Microsoft_Transaction_Server) is considered to be the first major application of AOP followed by [Enterprise JavaBeans](https://en.wikipedia.org/wiki/Enterprise_JavaBeans).

f) Event oriented programming requirement

Event-driven 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.

CLASSIFICATION OF PROGRAMMING LANGUAGES BASED ON DOMAIN REQUIREMENTS

Scientific applications

–  typically require simple data structures but a large number of floating-point operations

–  typical data structures: arrays and matrices; typical control constructs: selection and loop

–  earliest high-level language: FORTRAN other language: ALGOL 60

 Business applications

–  requires preparation of detailed, complex reports and precise ways of handling decimal numbers and character data

– Commercial data processing was one of the earliest commercial applications of computers.

Grace Murray Hopper et. al. at Univac developed FLOWMATIC, an English-like language for business applications.

– The U.S. Defense Dept. sponsored the effort to develop COBOL (Common Business-Oriented Language), which was standardized in 1960, revised in 1961 & 1962, re-standardized in 1968, 1974, and 1984.

–  most popular language: COBOL

Systems programming applications

–  systems software is used almost continuously and so a language for systems programming must have fast execution.

–  a language used for systems programming must have low-level features to access external devices.

–  popular language: C (almost the whole of UNIX is written in C)

Artificial intelligence applications

– requires symbolic rather than numeric processing – typical data structure: linked list
– popular languages: LISP, PROLOG

Special-purpose languages

–  a variety of application-specific languages (or software tools); example: RPG (business report generator), GPSS (systems simulation)

–  many computer scientists do not consider these to be programming languages

Web Software (Eclectic collection of languages)

– Markup (e.g., HTML) – used for annotating a document in a manner that can be distinguished from the text.

– Scripting (e.g., PHP) - the language that enable the script to run these commands and typically include control structures such as if-then-else and while-do.

– General-purpose (e.g., Java) – can be used for a wide range of programming jobs.

ORGANIZATION OF PROGRAMMING LANGUAGES ALOG THE RUBICS OF

 I AND II

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S/N | Programming Language | Date | Classification | Author | Paradigm | Application |
| 1 | Java | 1991 | Compiled | Sun micro systems | OOP | Web Application, Embedded system, General Purpose |
| 2 | C | 1972 | Compiled | Dennis Richie | Structured PL | Embedded system, Hardware drivers |
| 3 | Python | 1989 | Interpreted | Guido Van Rossum | OOP | Web applications, AI |
| 4 | C++ | 1983 | Compiled | Bjarne Stroustrup | OOP | Local application, Web Services, Propietory Services |
| 5 | VB.Net | 2007 | Compiled | Microsoft | Event Oriented | Web application, Local application |
| 6 | C# | 2000 | Compiled  | Microsoft | OOP | Web application, Local application |
| 7 | PHP | 1995 | Hybrid | Rasmus Lerdorf | Scripting Language | Web application |
| 8 | JavaScript | 1995 | Compiled | Brendan Eich | Scripting Language | Web application |
| 9 | SQL | 1986 | Interpreted | Donald D. Chamberlin and Raymond F. Boyce | Multi paradigmDeclarative | Database Queries |
| 10 | ADA | 1980 | Compiled | Jean Ichbiah and S. Tucker Taft | Structured | General purpose |
| 11 | Objective-C | 2014 | Compiled | Apple | OOP | Apple IOS devices |
| S/N | Programming Language | Date | Classification | Author | Paradigm | Application |
| 12 | Objective-Pascal/Delphi | 1960 | Hybrid | Apple | OOP | Local Application |
| 13 | Chapel | 2009 | Interpreted | David Callahan, Hans Zima, Brad Chamberlain, John Plevyak | OOP | General Purpose |

EVOLUTION OF PROGRAMMING LANGUAGES

(IN CHRONOLOGICAL ORDER)

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 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.

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.

Programming language evolution continues, in both industry and research. Some of the recent trends have included:

* Increasing support for [functional programming](https://en.wikipedia.org/wiki/Functional_programming) in mainstream languages used commercially, including [pure functional programming](https://en.wikipedia.org/wiki/Purely_functional_programming) for making code easier to reason about and easier to parallelize (at both micro- and macro- levels)
* Constructs to support [concurrent](https://en.wikipedia.org/wiki/Concurrent_computing) and [distributed](https://en.wikipedia.org/wiki/Distributed_computing) programming.
* Mechanisms for adding security and reliability [verification](https://en.wikipedia.org/wiki/Software_verification) to the language: extended static checking, [dependent typing](https://en.wikipedia.org/wiki/Dependent_typing), information flow control, static [thread safety](https://en.wikipedia.org/wiki/Thread_safety).
* Alternative mechanisms for composability and modularity: [mixins](https://en.wikipedia.org/wiki/Mixin), [traits](https://en.wikipedia.org/wiki/Trait_%28computer_programming%29), [type classes](https://en.wikipedia.org/wiki/Typeclass), [delegates](https://en.wikipedia.org/wiki/Delegation_%28programming%29), [aspects](https://en.wikipedia.org/wiki/Aspect-oriented_programming).
* Component-oriented software development.
* [Metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming), [reflection](https://en.wikipedia.org/wiki/Reflection_%28computer_science%29) or access to the [abstract syntax tree](https://en.wikipedia.org/wiki/Abstract_syntax_tree)
	+ AOP or [Aspect Oriented Programming](https://en.wikipedia.org/wiki/Aspect_Oriented_Programming) allowing developers to insert code in another module or class at "join points"
	+ [Domain specific languages](https://en.wikipedia.org/wiki/Domain_specific_language) and [code generation](https://en.wikipedia.org/wiki/Automatic_programming)
		- XML for graphical interface ([XUL](https://en.wikipedia.org/wiki/XUL), [XAML](https://en.wikipedia.org/wiki/Extensible_Application_Markup_Language))
* Increased interest in distribution and mobility.
* Integration with databases, including [XML](https://en.wikipedia.org/wiki/XML) and [relational databases](https://en.wikipedia.org/wiki/Relational_database).
* [Open source](https://en.wikipedia.org/wiki/Open-source_software) as a developmental philosophy for languages, including the GNU Compiler Collection and languages such as [Python](https://en.wikipedia.org/wiki/Python_%28programming_language%29), [Ruby](https://en.wikipedia.org/wiki/Ruby_%28programming_language%29), and [Scala](https://en.wikipedia.org/wiki/Scala_%28programming_language%29).
* Massively parallel languages for coding 2000 processor GPU graphics processing units and supercomputer arrays including [OpenCL](https://en.wikipedia.org/wiki/OpenCL)
* Early research into (as-yet-unimplementable) [quantum computing](https://en.wikipedia.org/wiki/Quantum_computing) programming languages
* More interest in [visual programming languages](https://en.wikipedia.org/wiki/Visual_programming_language) like [Scratch](https://en.wikipedia.org/wiki/Scratch_%28programming_language%29)

Some notable languages developed during this period include: [[15]](https://en.wikipedia.org/wiki/History_of_programming_languages#cite_note-TIOBE_Index-15)[[16]](https://en.wikipedia.org/wiki/History_of_programming_languages#cite_note-GitHub's_Octoverse_2018-16)

|  |  |
| --- | --- |
| * 2000 – [ActionScript](https://en.wikipedia.org/wiki/ActionScript)
* 2001 – [C#](https://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29)
* 2001 – [D](https://en.wikipedia.org/wiki/D_%28programming_language%29)
* 2002 – [Scratch](https://en.wikipedia.org/wiki/Scratch_%28programming_language%29)
* 2003 – [Groovy](https://en.wikipedia.org/wiki/Groovy_%28programming_language%29)
* 2003 – [Scala](https://en.wikipedia.org/wiki/Scala_%28programming_language%29)
* 2005 – [F#](https://en.wikipedia.org/wiki/F_Sharp_%28programming_language%29)
* 2006 – [PowerShell](https://en.wikipedia.org/wiki/Windows_PowerShell)
* 2007 – [Clojure](https://en.wikipedia.org/wiki/Clojure)
 | * 2009 – [Go](https://en.wikipedia.org/wiki/Go_%28programming_language%29)
* 2010 – [Rust](https://en.wikipedia.org/wiki/Rust_%28programming_language%29)
* 2011 – [Dart](https://en.wikipedia.org/wiki/Dart_%28programming_language%29)
* 2011 – [Kotlin](https://en.wikipedia.org/wiki/Kotlin_%28programming_language%29)
* 2011 – [Elixir](https://en.wikipedia.org/wiki/Elixir_%28programming_language%29)
* 2012 – [Julia](https://en.wikipedia.org/wiki/Julia_%28programming_language%29)
* 2012 - [TypeScript](https://en.wikipedia.org/wiki/TypeScript)
* 2014 – [Swift](https://en.wikipedia.org/wiki/Swift_%28programming_language%29)
 |

Other new programming languages include [Red](https://en.wikipedia.org/wiki/Red_%28programming_language%29), [Crystal](https://en.wikipedia.org/wiki/Crystal_%28programming_language%29), [Hack](https://en.wikipedia.org/wiki/Hack_%28programming_language%29), [Reason](https://en.wikipedia.org/wiki/Reason_%28programming_language%29) and [Ballerina](https://en.wikipedia.org/wiki/Ballerina_%28programming_language%29).

Differences between modular programming and OOP

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!

 [Object Oriented](https://wiki.c2.com/?ObjectOriented) programs are usually modular. [Modular Programming](https://wiki.c2.com/?ModularProgramming) does not have to be [ObjectOriented](https://wiki.c2.com/?ObjectOriented). The difference is that "merely" Modular code does not need to be associated with data (at least the way [ObjectOriented](https://wiki.c2.com/?ObjectOriented)code is structurally associated with data.)

References

<http://www.differencebetween.info/difference-between-structured-and-unstructured-programming-language>

https://en.wikipedia.org/wiki/Modular\_programming

<https://en.wikipedia.org/wiki/Object-oriented_programming#History>

<https://en.wikipedia.org/wiki/Aspect-oriented_programming>

<http://www.technologyuk.net/computing/software-development/software-design/event-driven-programming.shtml>

<https://en.wikipedia.org/wiki/History_of_programming_languages>

<https://www.programminglogic.com/chronological-order-of-popular-programming-languages/>

https://www.quora.com/What-is-the-difference-between-modular-programming-and-OOP