CENG204 - Programming Languages Concepts

Asst. Prof. Dr. Emre ŞATIR

Lecture 2
Preliminaries (Part 1)

Lecture 2 Topics

- What is a Programming Language?
- Classification of Programming Languages
  - Classification According to Application Areas
  - Classification According to Their Level
  - Classification According to Design Methodologies (Paradigms)

What is a Programming Language?

What is a Programming Language?

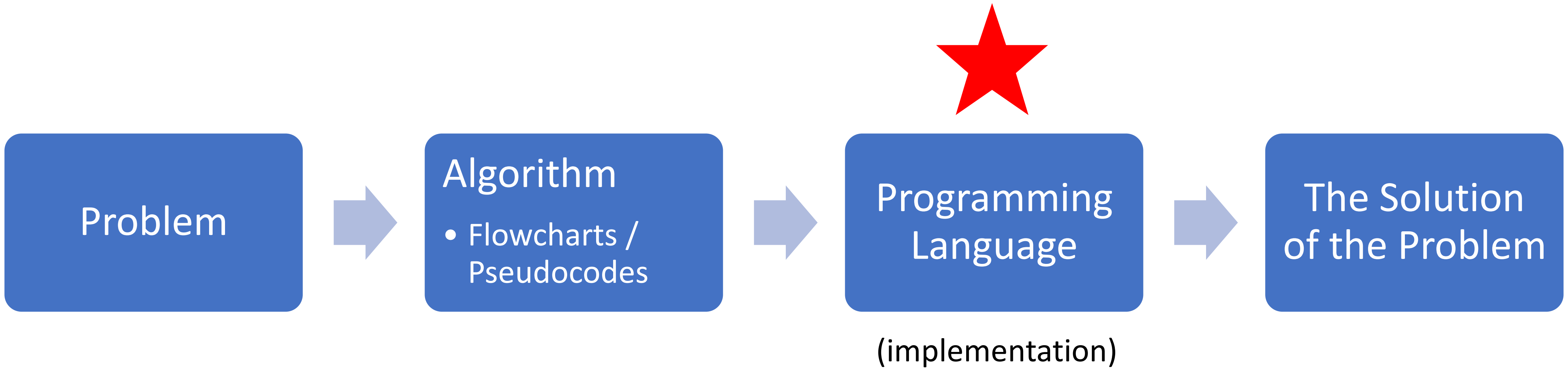
- Definition of the term "language" from the “Theory of Computation (ToC)” course:
    - A language is a “set” of strings (can be finite or infinite).
    - In that course (ToC), languages are associated with <u>problems</u>.

- A “natural language” is any language that has evolved <u>naturally</u> in humans through use and repetition without conscious planning or premeditation as contrasted with an artificial language or computer code (Turkish, English, German, etc.).

What is a Programming Language?

- A programming language is a computer language that is used by programmers (developers) to <u>communicate with computers</u>.
- It is a set of instructions written in any specific language (C, C++, Java, Python) to <u>perform a specific task</u>.
- Programming languages have <u>rules and symbols</u> that can be understood by <u>both</u> humans and computers.

The Place of a Programming Language in Solving the Problem



Classification of Programming Languages

Classification of Programming Languages

- Classification According to **Application Areas**
- Classification According to Their **Level**
- Classification According to **Design Methodologies (Paradigms)**

Classification of PLs According to Application Areas

Programming Domains

Computers have been applied to a myriad of different areas, from controlling nuclear power plants to providing video games in mobile phones.

Because of this great diversity in computer use, programming languages with very different goals have been developed.

In this section, we briefly discuss a few of the areas of computer applications and their associated languages.

Programming Domains

- **Scientific and Engineering Applications**
  - Large numbers of floating-point computations; use of arrays and matrices.
  - Fortran (**FOR**mula **TRAN**slating)

- **Business Applications**
  - Data creation, data processing, storing and accessing data (e.g., database applications).
  - Produce reports, use decimal numbers and characters.
  - COBOL (**CO**mmon **B**usiness **O**riented **L**anguage)

- **Artificial Intelligence**
  - Computer applications characterized by the use of “symbolic” rather than “numeric” computations.
  - LISP (**LIS**t **P**rocessing Language), PROLOG (**PRO**gramming **LOG**ic)

Programming Domains

- **Systems Programming**
  - System programs are software that manages the <u>hardware</u> of the computer.
  - Operating systems, compilers, interpreters, …
  - Systems programs vs application software (Systems programming vs application programming).
  - <u>Need efficiency</u> because of continuous use.
  - Assembler, C.

- **Simulation Software**
  - Simula 67, GPSS (**G**eneral **P**urpose **S**imulation **S**ystem).

- **Web Software**
  - Eclectic collection of languages: Markup (HTML), Scripting (PHP, JavaScript), General-Purpose (Java)

- **General-Propose Programming Languages**
  - Pascal, Basic, C, Java, C#.

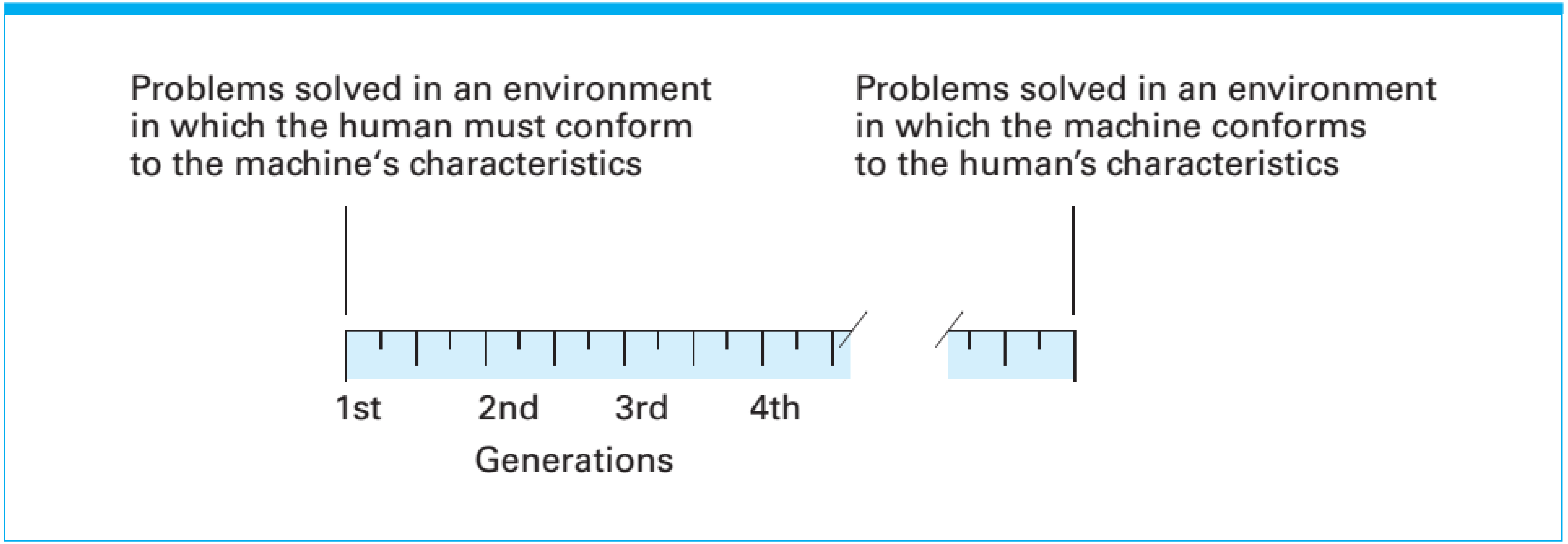
Classification of PLs According to Their Level

Classification of Programming Languages According to Their Level

- The concept of the level of a programming language should be understood as the degree of abstraction of that programming language from computer hardware and the degree of proximity to human perception.
- As a programming language approaches human perception, its level increases.
- The closer the programming language is to the computer hardware, the lower it is.
- The higher the level of the language, the easier it is for the programmer to learn this language and produce software with it.

Classification of Programming Languages According to Their Level

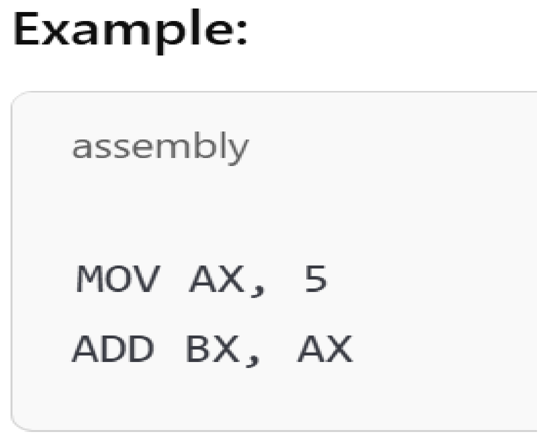
Generations of programming languages


Problems solved in an environment in which the human must conform to the machine's characteristics
Problems solved in an environment in which the machine conforms to the human's characteristics
1st
2nd
3rd
4th
Generations

Generations of Programming Languages

- **1st Generation (1GL) – Machine Language**
  - The lowest-level language, consisting of pure binary code (0s and 1s) executed directly by the CPU.
  - Example: 10110100 11000011 (specific to a processor).

- **2nd Generation (2GL) – Assembly Language**
  - Uses “mnemonics” (human-readable representations of machine code) instead of binary code, making programming slightly easier.
  - Requires an **assembler** to convert code into machine language.

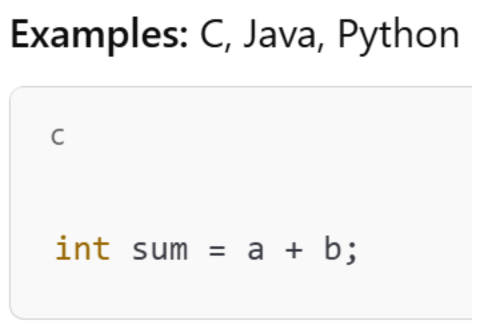


Example:

```assembly
MOV AX, 5
ADD BX, AX
```

Generations of Programming Languages

- **3rd Generation (3GL) – High-Level Languages**
  - High-level procedural and structured languages that abstract away hardware details, closer to human language.
  - Requires a **compiler** or **interpreter** to <u>convert</u> into machine code.

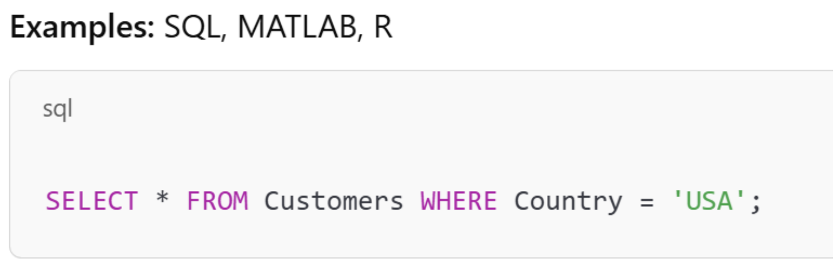


Examples: C, Java, Python

c

```c
int sum = a + b;
```

- **4th Generation (4GL) – Declarative & Domain-Specific Languages**
  - Focuses on "what to do" rather than "how to do it”.
  - Problem-oriented languages focused on specific domains with even higher abstraction.
  - Often used in database querying, report generation, and scripting.



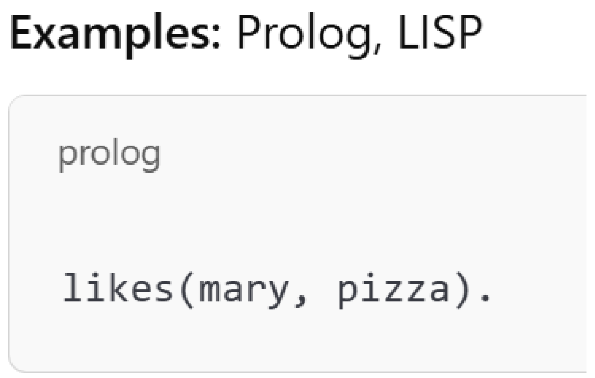
Examples: SQL, MATLAB, R

sql

```sql
SELECT * FROM Customers WHERE Country = 'USA';
```

Generations of Programming Languages

- **5th Generation (5GL) – AI and Logic-Based Languages**
  - Declarative languages where programmers specify what the program should accomplish rather than how.
  - Designed for problem-solving using constraints and logic.
  - Used in artificial intelligence and expert systems.



Examples: Prolog, LISP

prolog

```
likes(mary, pizza).
```

- *** Each generation improved **abstraction** (brought programming closer to natural human expression and further from the machine's native language), making programming more **accessible** and **productive** for humans.

Classification of Programming Languages According to Their Level

- **Machine Language:** It is a “primitive” level programming language. It consists of only 0's and 1's. It is the fastest and most efficient language as the commands run directly. These languages are machine dependent.

- **Low-Level Programming Languages:** They are commonly referred to as “assembly languages” or “symbolic machine languages”. They work fast as the conversion to machine language is done with very little loss. They are machine dependent languages, too.

Classification of Programming Languages According to Their Level

- **Intermediate Level Programming Languages:** They are relatively fast compared to high-level languages. (C)

- **High-Level Programming Languages:** They run slower, but it is easier to produce software using them. For example, Java in this category is a machine independent language.

- **Very High-Level Programming Languages (Visual Languages):** There are different platforms and virtual machines in the middleware. They are quite slow but easy to program especially with visual elements. They are usually domain-specific languages, limited to a very specific application, purpose, or type of task. (SAP - Abap)

Classification of PLs According to Design Methodologies (Paradigms)

Classification of Programming Languages According to Design Methodologies

- This is also called classification according to “the programming **paradigm**”.
- A “**paradigm**” is a conceptual diagram that represents the collective decisions of a group and shows the group's perspective on the subject.
- Different paradigms bring different programming styles and affect the way the programmer looks at algorithms.

Classification of Programming Languages According to Design Methodologies

- **Imperative Programming** (*Structured Programming, *Procedural Programming, *Modular Programming)
- **Object Oriented Programming**
- **Declarative Programming / Logical Programming**
- **Functional Programming**

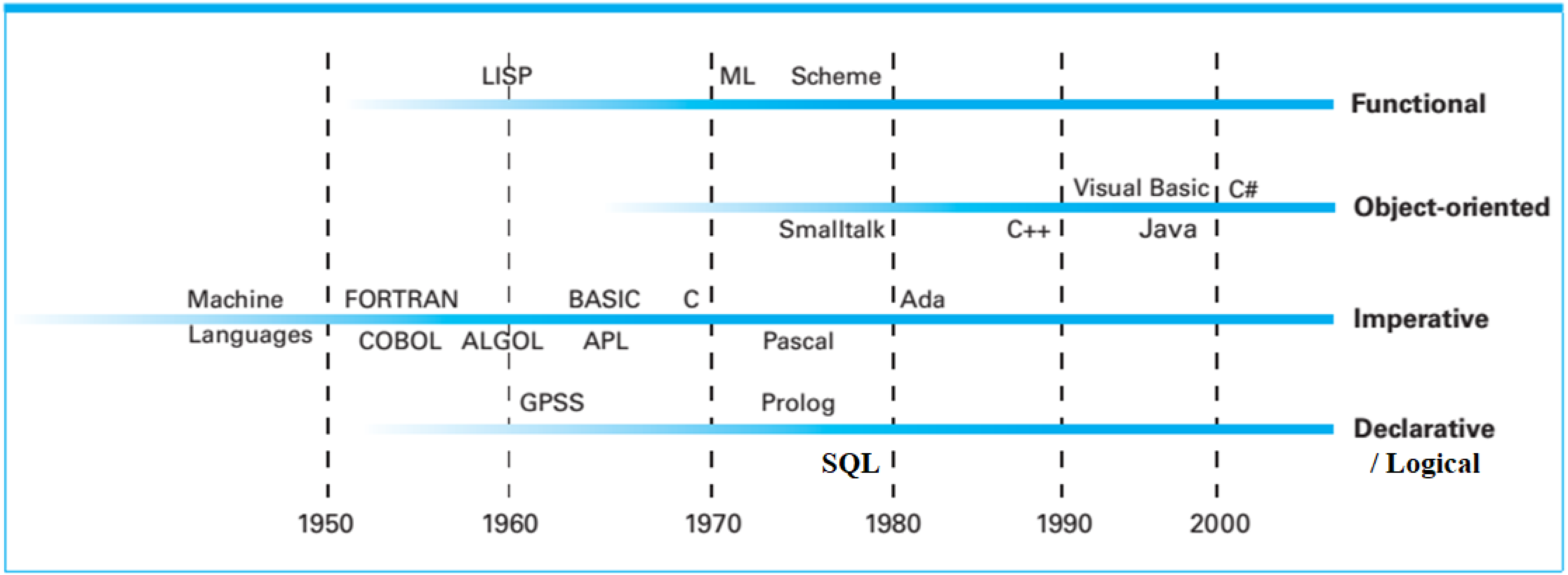
+ Scripting Languages
+ Markup Languages

*** It is not necessary to have a one-to-one relationship between programming languages and paradigms. For example, C++ supports the development of both imperative and object-oriented programs.

Classification of Programming Languages According to Design Methodologies

- If the design method and language paradigm used in the development of a software system is the same, software development will be much easier.
- Therefore, it is very important to choose the software paradigm and the language correctly.
- Also, because there are intense similarities between languages in a paradigm group; learning one member of a paradigm group makes it much easier to learn other languages in the group.

The evolution of programming paradigms



Imperative Programming

- **Imperative programming** is one of the major programming paradigms, where you describe a series of commands (or statements) for the computer to execute (so you pretty much order it to take specific actions, hence the name "imperative").
- Central features are **variables**, **assignment statements** and **iteration**.
- In the good old days, when programming was broadly in assembly, code would have tons of **GOTO**s. Even higher-level languages like FORTRAN and BASIC began using the same primitives. In this programming paradigm, the entire program is a single algorithm or complete functionality written linearly - *step-by-step*. This is "imperative style".
- Understand that one can really write totally bad imperative work even in modern C language as well, but it is quite easy to organize code in higher level languages.

Imperative Programming

- In this paradigm, a program is viewed as a series of operations.
- Statements in the program communicate with each other through variables.
- Execution of each statement causes the value of one or more of the locations in memory to be changed.
- An example assembly program that demonstrates the addition of two numbers:

```
MOV ax, [0x00B00]
MOV bx, [0x00B02]
ADD ax,bx
```

Imperative Programming (Assembly Example)

```
...
...
print_loop:
...
...
inc byte [num]
cmp byte [num], 11      ; Check if num > 10
jl print_loop           ; If not, continue loop
...
...
```

Imperative Programming (C Example)

```c
#include <stdio.h>

int main()
{
    int num = 1;

print_loop:
    printf("Number: %d\n", num);
    num++;
    if (num <= 10)
        goto print_loop;

    return 0;
}
```

Imperative Programming - Structured Programming

- **Structured Programming** is any programming when functionality is divided into units like “for loop”, “while loop”, “if... then” etc. **block** structure. Structured Programming emerged to remove the reliance on the GOTO statement.
- C is an “Imperative” and “Structured” programming language.

Imperative Programming - Procedural Programming

- When higher level languages begun to get richer, one realized that all units of work should be broken into smaller tractable parts - that is when **functions** came into existence and programming became a hierarchy of functions and many at lower level could be re-used.
- **Procedural Programming** is a specific type (or subset) of Imperative Programming, where you use procedures (i.e., functions) to describe the commands the computer should perform.
- C is also a “Procedural” programming language.
- Most programming languages included in Imperative Paradigm support Structured Programming and Procedural Programming. But a language can be imperative without being structured or procedural. An example is **pure Assembly language**.

Imperative Programming - Modular Programming

- In **Modular Programming**, one can create a physical form of package - i.e. a "chunk of code" that can be shipped; which are fairly general purpose and re-usable. This is called modules of elements compiled together.
- This concept refers to structuring the program into **independent** and **reusable** **modules**.
- Basically, structured code where functions (or procedures) dominate over data is called procedural whereas class and object based representation is called object oriented. Both by definition are also modular.

Object Oriented Programming

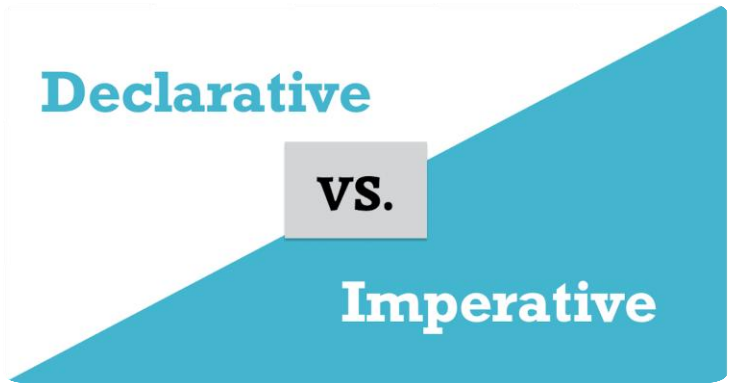
- The concept of object-oriented programming had its roots in SIMULA 67 but was not fully developed until the evolution of Smalltalk resulted in Smalltalk 80 (in 1980).
- Although SIMULA 67 had classes, the members defined inside them were <u>not</u> hidden from outside code (information hiding is <u>not</u> supported).

Object Oriented Programming

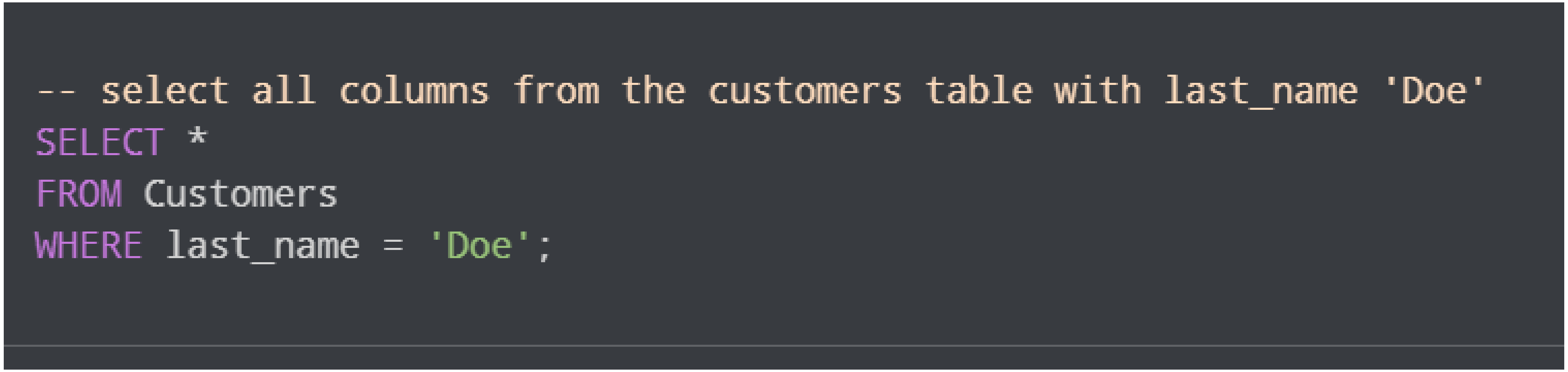
- Indeed, some consider Smalltalk to be the base model for a purely object-oriented programming language.
- A language that is object oriented must provide support for three key language features: "abstract data types", "inheritance", and "dynamic binding".
- Although Smalltalk never became as widely used as many other languages, support for object-oriented programming is now part of most popular imperative languages, including C++, Java, and C#.

Declarative Programming

- In contrast to the imperative paradigm is the declarative paradigm, which asks a programmer to <u>describe the problem</u> to be solved rather than an algorithm to be followed.
- More precisely, a declarative programming system applies a preestablished general-purpose problem-solving algorithm to solve problems presented to it.
- In such an environment the task of a programmer becomes that of <u>developing a precise statement of the problem</u> rather than of describing an algorithm for solving the problem.



Declarative Programming



```
-- select all columns from the customers table with last_name 'Doe'
SELECT *
FROM Customers
WHERE last_name = 'Doe';
```

Declarative Programming / Logical Programming

- A tremendous boost was given to the declarative paradigm with the discovery that the subject of “formal logic” within mathematics provides a simple problem-solving algorithm suitable for use in a general-purpose declarative programming system.
- The result has been increased attention to the declarative paradigm and the emergence of logic(al) programming.

Logical Programming

- In the logic(al) programming paradigm, programming is seen as specifying what is wanted to be done rather than how to do something (declerative programming).
- Languages that support the logic(al) programming paradigm work by checking for the existence of a certain condition and performing an appropriate action if that condition is met.
- During the execution of a logic-based language, statements are not processed sequentially, unlike the way an imperative language works. (“Rule-based”, rules are specified in no particular order)
- The most well-known example of languages in this model is the “Prolog” programming language.
- The Prolog language is suitable for human-machine interaction and responds to queries posed to it using a special way of reasoning.

A Prolog Example

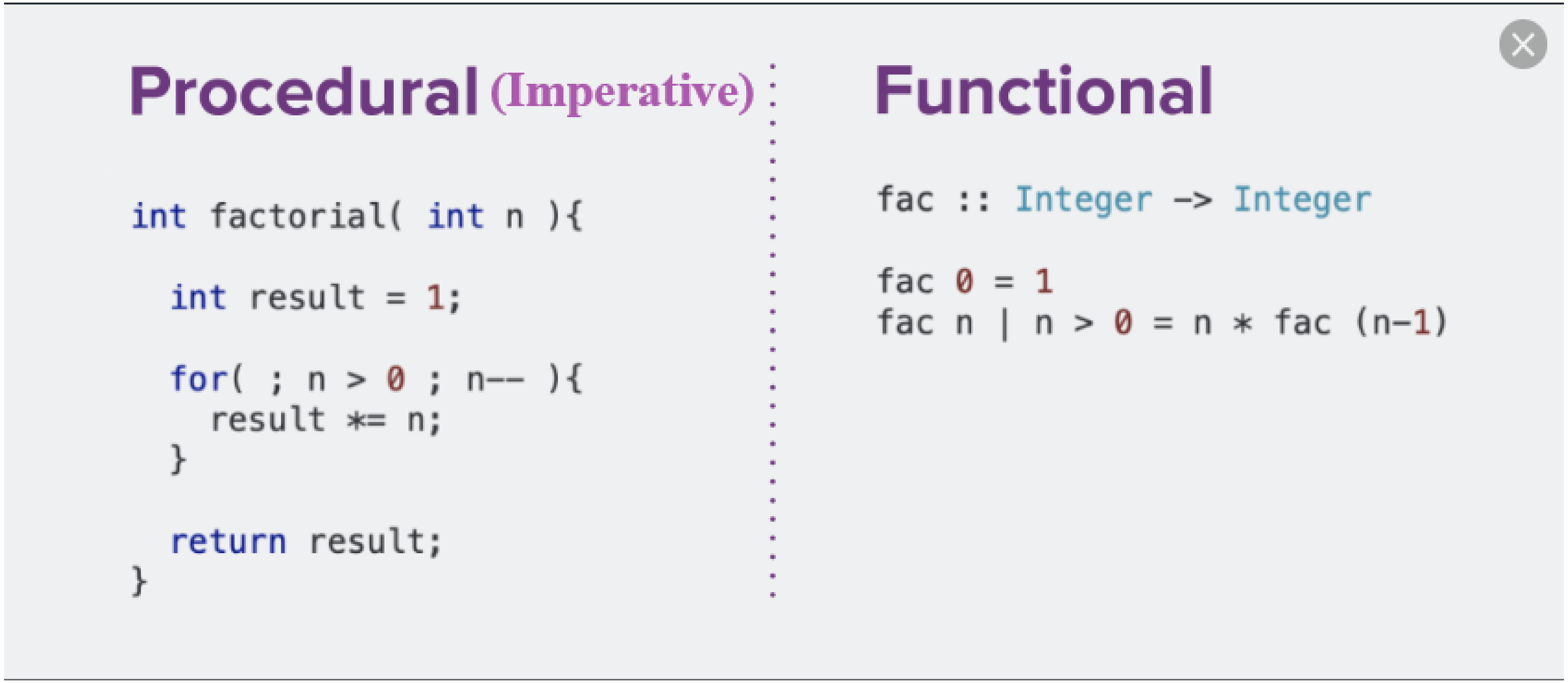
```
% Family members
mother(ayse, mehmet).
mother(ayse, fatma).
father(ali, mehmet).
father(ali, fatma).

% Sibling definition
sibling(X, Y) :- mother(M, X), mother(M, Y), father(F, X),
father(F, Y).

% Queries
sibling(mehmet, fatma).
```

→ true

Functional Programming

- The first functional programming language was invented to provide language features for **list processing** (LISP), the need for which grew out of the first applications in the area of artificial intelligence (AI).
- The data and the functional transformations that will be applied to the data to obtain the result form the basis of this paradigm (Main means of making computations is by applying functions to given parameters).
- ML (MetaLanguage), Scheme, Haskell and F# (a .NET language) are other important functional languages.

Functional Programming



Procedural **(Imperative)**

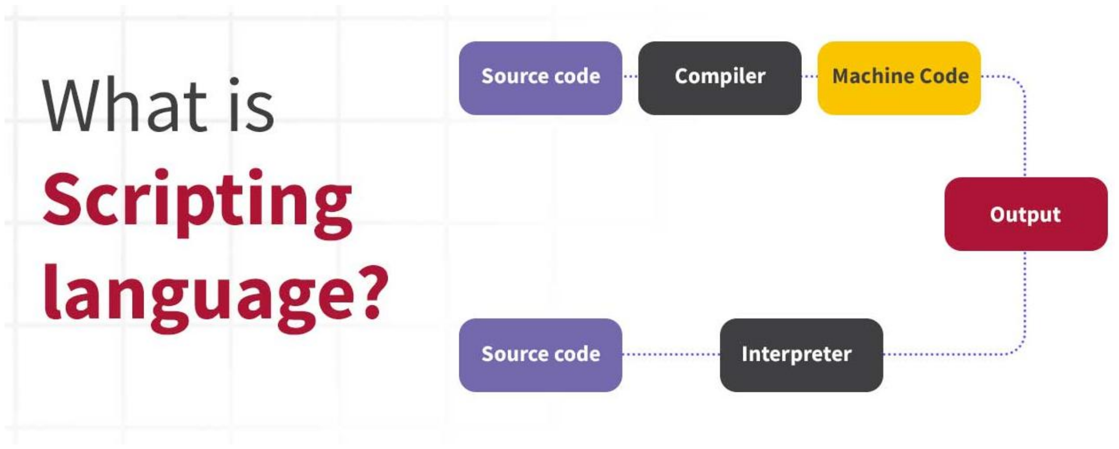
```
int factorial( int n ){

  int result = 1;

  for( ; n > 0 ; n-- ){
    result *= n;
  }

  return result;
}
```

Functional

```
fac :: Integer -> Integer

fac 0 = 1
fac n | n > 0 = n * fac (n-1)
```

Scripting Languages

- Some authors refer to scripting languages as a <u>separate category</u> of programming languages.
- However, languages in this category are bound together more by their <u>implementation method</u>, partial or full **interpretation**, than by a common language design.
- The languages that are typically called scripting languages, among them Perl, JavaScript, PHP, Ruby and Lua are <u>imperative languages</u> in every sense.
- Also, Python is a relatively recent object-oriented scripting language.



Scripting Languages

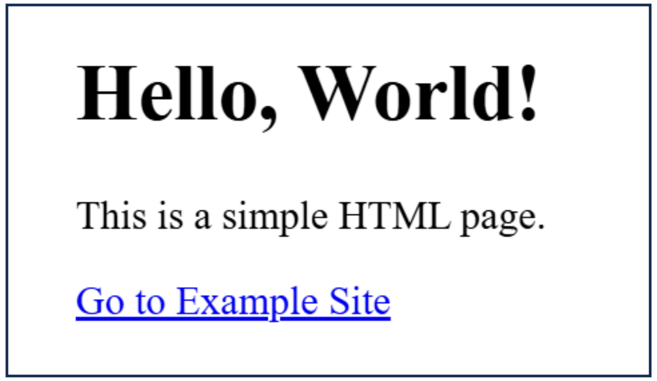
- In English, “script” usually means a text or written instructions.
- In computer science, the term “scripting language” is used specifically for languages that facilitate the execution of scripts. This usage comes from the idea that scripts execute a series of instructions step by step, similar to theater or movie scripts.
- Script languages are defined as languages that are usually run by an “interpreter” and do not require compilation. These languages are widely used in areas such as system administration, automation and web development.

Markup Languages

- Markup languages extended to support some programming.
- Markup languages are not programming languages. For instance, HTML, the most widely used markup language, is used to specify the layout of information in Web documents. (Some programming capability has crept into some extensions to HTML and XML).
- However, in recent years, a new category of languages has emerged, the **markup-programming hybrid languages**.
- A markup-programming hybrid language is a markup language in which some of the elements can specify programming actions, such as control flow and computation.
- Among these are “eXtensible Stylesheet Language Transformations (XSLT)” and the “Java Server Pages Standard Tag Library (JSTL)”.

An HTML Example

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple HTML Example</title>
</head>
<body>
    <h1>Hello, World!</h1>
    <p>This is a simple HTML page.</p>
    <a href="https://www.example.com">Go to Example Site</a>
</body>
</html>
```



Hello, World!

This is a simple HTML page.

Go to Example Site

An XML Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book>
        <title>Example Book</title>
        <author>John Doe</author>
        <year>2024</year>
        <price>19.99</price>
    </book>
</bookstore>
```