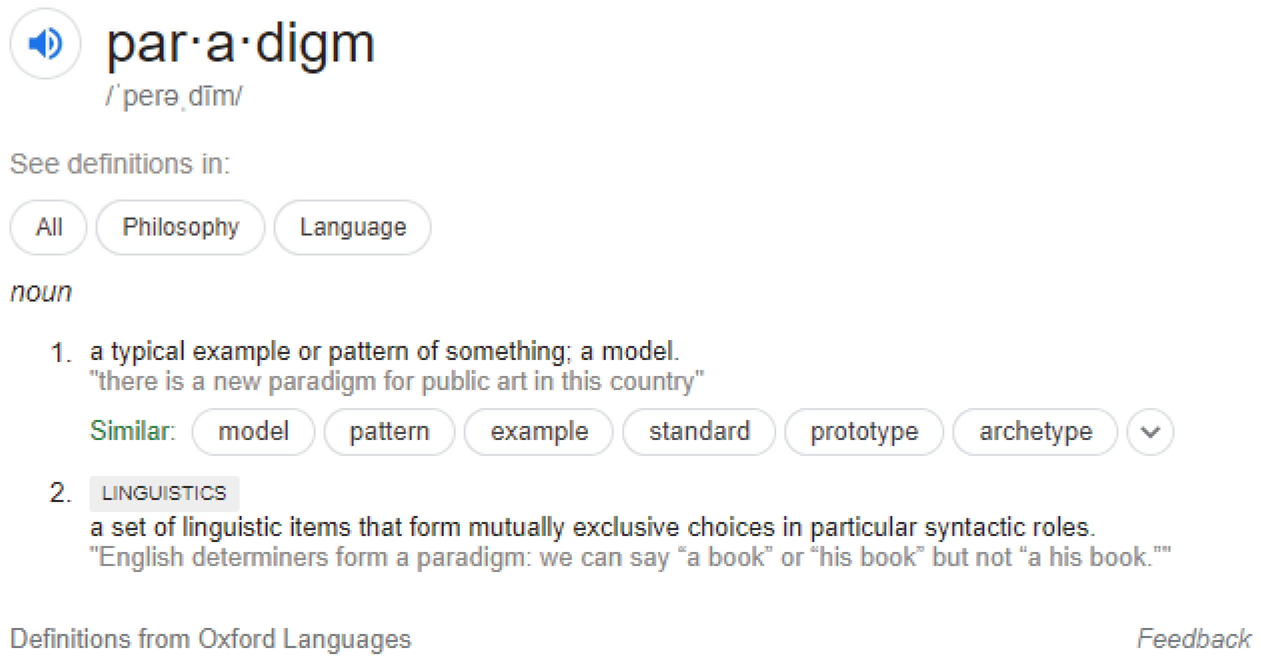
Programming Language Paradigms

What is a paradigm?



par·a·digm

/ˈperəˌdīm/

See definitions in:

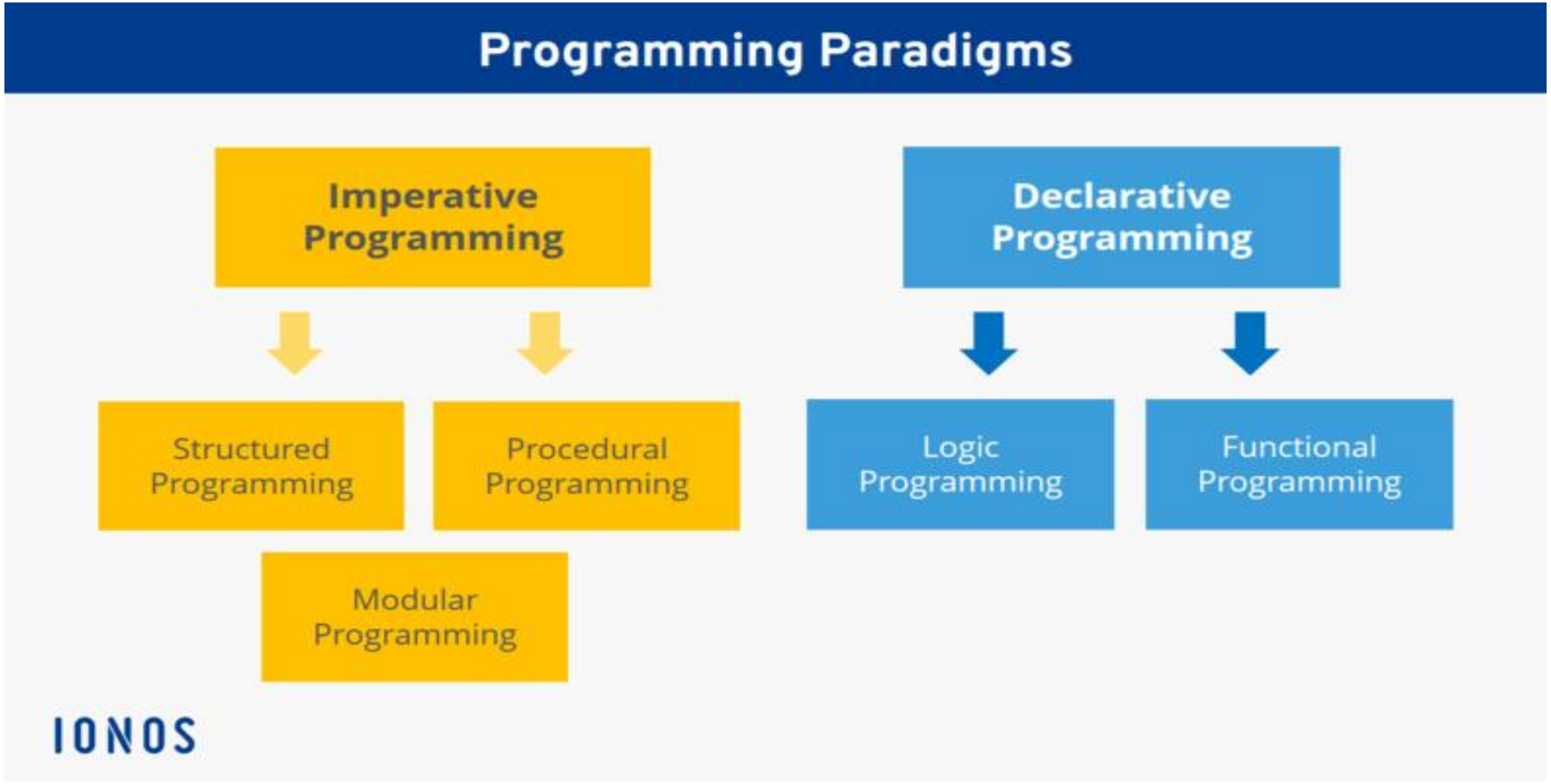
All | Philosophy | Language

noun

1. a typical example or pattern of something; a model.
   "there is a new paradigm for public art in this country"
   Similar: model | pattern | example | standard | prototype | archetype
2. LINGUISTICS
   a set of linguistic items that form mutually exclusive choices in particular syntactic roles.
   "English determiners form a paradigm: we can say “a book” or “his book” but not “a his book.”"

Definitions from Oxford Languages

Feedback



(Adapted from: https://www.ionos.com/digitalguide/websites/web-development/programming-paradigms/)

Imperative Languages (by wiki)

In computer science, **imperative programming** is a programming paradigm that uses statements that change a program's state. In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform. Imperative programming focuses on describing *how* a program operates.

Imperative Languages

Imperative languages specify how a computation is performed by sequences of changes to the random access memory (RAM).

Computations are performed through a guided sequence of steps, where these variables are referred to or changed.

What is imperative programming?

Imperative programming (from Latin imperare = command) is the oldest programming paradigm. A program based on this paradigm is made up of a **clearly-defined sequence of instructions** to a computer.

Therefore, the source code for imperative languages is a series of commands, which specify what the computer has to do – and when – in order to achieve a desired result. Values used in variables are changed at program runtime. To control the commands, **control structures such as loops or branches are integrated into the code**.

Imperative programming languages are very specific, and operation is system-oriented. On the one hand, the code is easy to understand; on the other hand, **many lines of source text** are required to describe what can be achieved with a fraction of the commands using declarative programming languages.

These are the best-known imperative programming languages:

- Fortran
- Java
- Pascal
- ALGOL
- C
- C#
- C++
- Assembler
- BASIC
- COBOL
- Python
- Ruby

Imperative Languages

In **block-structured languages**, the procedure is the principal building block of the program.

Languages like Algol-60, Algol-68, Pascal and C fall in this category.

Object-based paradigms describe languages that employ objects. All data and procedures that apply to the data are captured in a single object.

Examples are packages of Ada, modules of Modula and objects of Smalltalk.

Distributed programming paradigms refer to languages for loosely coupled systems.

In a loosely coupled system, a language need not support memory sharing, thus avoiding some very complicated problems in resolving memory conflicts.

In software terminology, **loosely coupled** refers to software where routines, modules, functions, and similar components are executed only as needed, and do not run at the launch of the software **application** and while it is being used. Web services are a type of software **application** that uses **loose coupling**.

The key **difference between loosely coupled and tightly coupled** system is that **loosely coupled** system has distributed memory, whereas, the **tightly coupled** system has shared memory. **Loosely coupled** is efficient when the tasks running on **different** processors has minimal interaction **between** them.

Declarative Languages

Declarative languages specify what is to be computed. Languages in this paradigm are considered at a higher level than in the imperative languages.

Logic programming is based on a subset of predicate calculus.

Functional languages operate only through functions which return one value given a list of parameters. Lisp is the most popular.

Database languages run on top of a database management system. SQL is a good example of this language.

Declarative vs. imperative programming

Imperative programming languages differ from declarative languages on one fundamental point: imperative programming focuses on the “how”, declarative programming on the “what”.

But what does that mean? Imperative programming languages are composed of **step-by-step instructions** (how) for the computer. They describe explicitly which steps are to be performed in what order to obtain the desired solution at the end. By contrast, in declarative programming, the desired result (what) is described directly. This becomes clearer when using a cooking analogy for illustration: imperative languages provide recipes; declarative languages contribute photos of the finished meal.

In declarative languages, the source code remains very abstract in terms of the specific procedure. To get to the solution, an algorithm is used which automatically identifies and applies appropriate methods. This approach has numerous advantages: Programs can be **written much more quickly**, and applications are also very easy to optimize. If a new method is developed in the future, the abstract instructions in the source code mean that the algorithm can easily utilize the newer method.

Advantages and disadvantages of imperative programming languages

Many programming languages based on the imperative programming paradigm are in use today.

On the one hand, this is because the approach is the **original form of programming**. On the other hand, despite the existence of alternative models, the imperative paradigm still has some practical advantages.

The languages are **relatively easy to learn**, as the code can be read like a step-by-step instruction. Therefore, programmers normally learn an imperative language first as part of their training.

Easy legibility is a crucial factor in day-to-day operations. Ultimately, maintenance and optimization of applications should not be linked to a specific person; different employees should be able to do it without too much difficulty even if they haven’t written the code from scratch themselves.

One disadvantage of procedural programming is that for more complex problems to be solved, the amount of code quickly starts to grow. It remains easy to read but **becomes confusing due to its volume**.

Applications of Programming Languages

- The problem solver is not satisfied.

- There are many problem areas.

- The first computers were invented to help speed up the computations of very complicated mathematical experssions.
    - Design of: FORTRAN, Algol-60, Algol-68.

- Data processing:
  - COBOL
- Artificial Intelligence
  - Lisp and Prolog
- Text processing:
  - SNOBOL
- Systems programming
  - C, Ada, Modula
- General Purpose
  - PL/I (Programming Language One)

Generations of Programming Languages

Or, programming language classification based on “generation”

First Generation:

Machine and Assembly language

Second Generation:

FORTRAN, Algol-60, BASIC, COBOL

Third Generation:

Procedural language such as: PL/I, Pascal, Modula-2, C, Ada

Functional languages such as: Lisp, APL

Logic languages such as: Prolog

Object-oriented languages such as: C++, Objective-C, Smalltalk, Object Pascal, Eiffel, Ada-95 and Java

Fourth Generation Languages

Visual programming: Visual Basic, Delphi, Visual age, PowerBuilder, Visual C++.

Language for database systems: SQL, SQL Server, Access, MySQL, MariaDB.

5th Generation Language:

Prolog, Mercury, OPS5

Evaluation Criteria for Programming Languages

Most likely, you will meet someone who is trying to convince you that a particular language is best for your application. Will you be convinced right away?

What criteria will you use to evaluate the recommended language?

General Criteria:

- Readability - Is it easy to read and understand a program in that language?
- Writability - Is it easy to write programs in that language?
- Reliability - Is the extent to which a program will perform according to its specifications. Does it help to prevent errors?
- Cost - How expensive is it to develop, use and maintain programs in that language?

What is meant by Machine Readability

A language is considered machine-readable if it can be translated efficiently into a form that the computer can execute.

This requires that:

- A translation algorithm exists
- The algorithm is not too complex

What do we mean by Human Readability?

- Generally this requires a programming language to provide enough abstractions to make the algorithms clear to someone who is not familiar with the program’s details.
- As programs gets larger, making a language readable requires that the amount of detail is reduced, so that changes in one part of a program have a limited effect on other parts of the program.

What Contributes to Readability?

- Simplicity
- Control Statements
- Data types and Structures
- Syntax

Simplicity

Programming language with a large number of basic components are harder to learn; most programmers using these languages tend to learn and use subsets of the whole language.

Complex language have multiplicity (more than one way to accomplish an operation).

Overloading operators can reduce the clarity of the program’s meaning

An Example of Multiplicity

- All of the following add one to the variable count in C:

```
count = count + 1;

count += 1;

count++;

++count;
```

Control Statements

- In the 1950s and 1960s, the **goto** was the most common control mechanism in a program; however, it could make programs less readable (even writable).

- The introduction of **while**, **for** and **if-then-else** eliminate the need for **gotos** and led to more readable programs.

Data Types and Structures

A more diverse set of data types and the ability of programmers to create their own increased program readability:

- Boolean make programs more readable:

  Found = 1 vs Found = True

- The use of records or structures to store complex data objects makes programs more readable:

  CHAR[30] NAME(100) INTEGER AGE(100), INTEGER EMP_NUM(100) REAL SALARY(100)

  Array of records vs 4 parallel arrays? Which one is better?

Syntax

Most syntactic features in a programming language can enhance readability:

- Identifier forms - older languaes (like FORTARN) restrict the length of identifiers, which become less meaningful
- Special words - in addition to **while, do and for,** some languages use special words to close structures such as endif and endwhile.
- Form and meaning - In C a static variable within a function and outside a function mean two different things - this is undesirable.

Extensive evaluation criteria:

- **Simplicity -** Is there one or few ways of expressing a concept?
- **Syntax -** Is the syntax clear and understandable?
- **Abstraction -** Is there a way to take virtually any internally consistent theme of a program?
- **Assertions -** Does the language have a facility to state propositions that should hold regardless of the details of implementation.
- **Hierarchical decompositions** - Is there a facility to express a top-down analysis of a programming task?
- **Modular decomposition** - Is there a facility to express program units that may be semi-independently written or executed?
- **Data manipulation** - Is there a facility to carry out primitive operations on the data?
- **Redundancy** - Is there a facility to guide programming by detecting and diagnosing inconsistencies of the programming?

An **assertion** is a statement in the Java^{TM} **programming** language that enables you to test your assumptions about your **program**. For example, if you write a method that calculates the speed of a particle, you might **assert** that the calculated speed is less than the speed of light.

Levels of Programming Languages

Class Activity: Explain in your own words the topic below: Send them via the LMS. Imagine you are explaining to a non computer savvy person.

- Machine language
- Assembly language
- High-level language

Methods of Implementation

Compilation is a method where a high-level language is translated into another implemented language usually assembly or machine language.

1. high-level language -> machine language
2. high-level language -> assembly language -> machine language
3. High-level language -> intermediary language -> assembly language or machine language

Interpretation on the other hand, is a method where we simulate, through a program running on another host computer.

Interpreters read the source program and execute their intended effects directly.

No time is spent during compilation but the execution time is slower.

Activity 1-4

Give the basic difference between a declarative and an imperative language. Classify the following as either imperative or declarative language.

Language	Classification
FORTRAN	
Algol 60	
Lisp	
COBOL	
APL	
SNOBOL	
BASIC	
Jovial	
PL/I	
Prolog	
C++	
Smalltalk	

Thank You