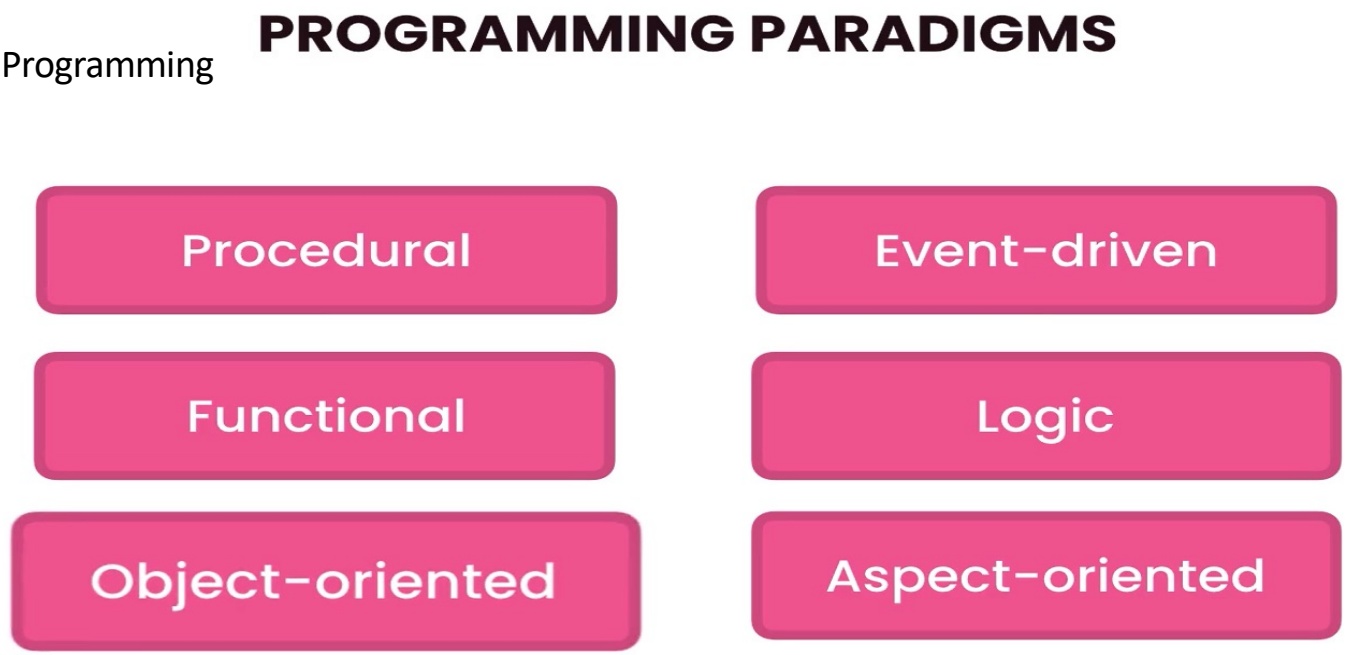
Overview of Programming Paradigms

Rong Li

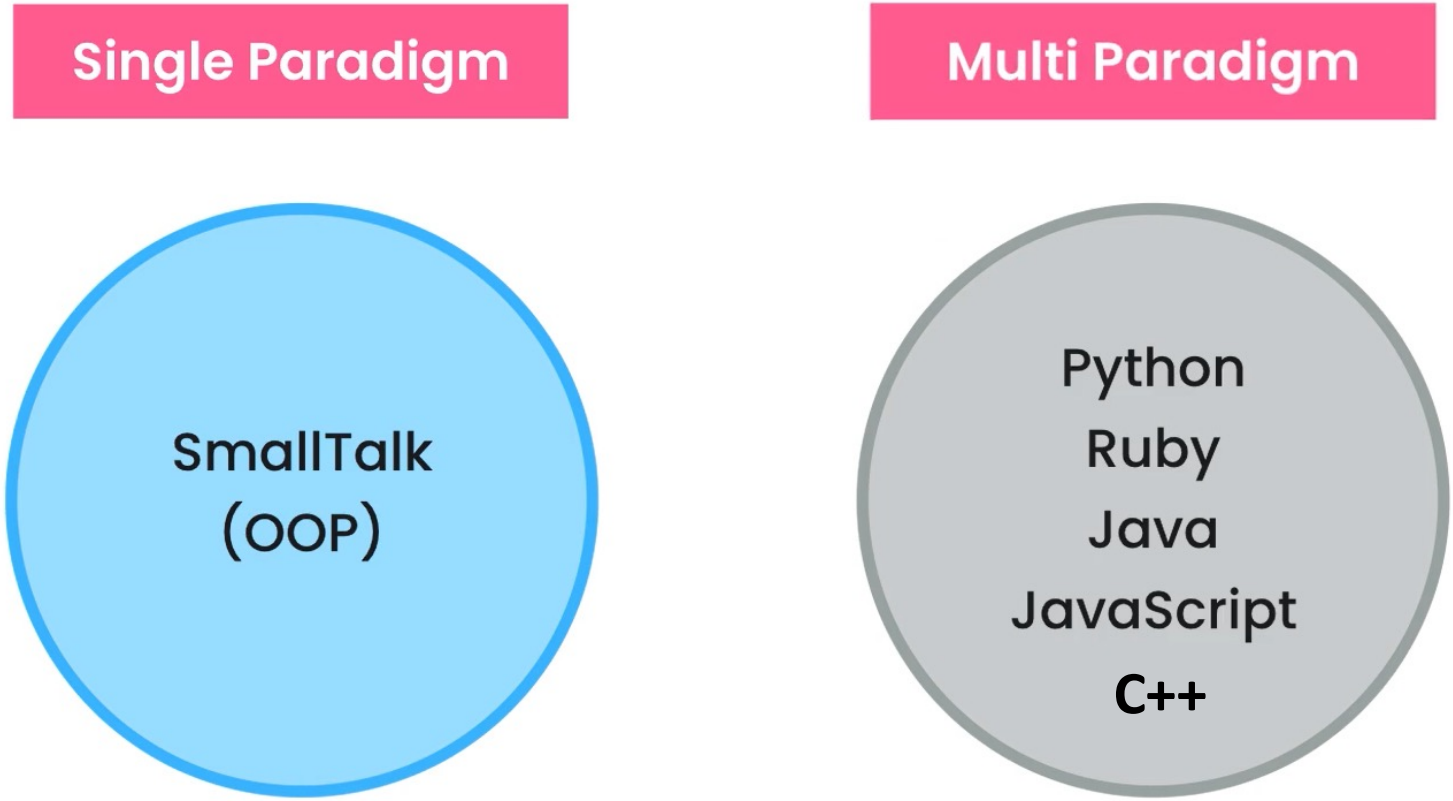
PROGRAMMING PARADIGMS

A paradigm is a model or a framework for describing how a program handles data.

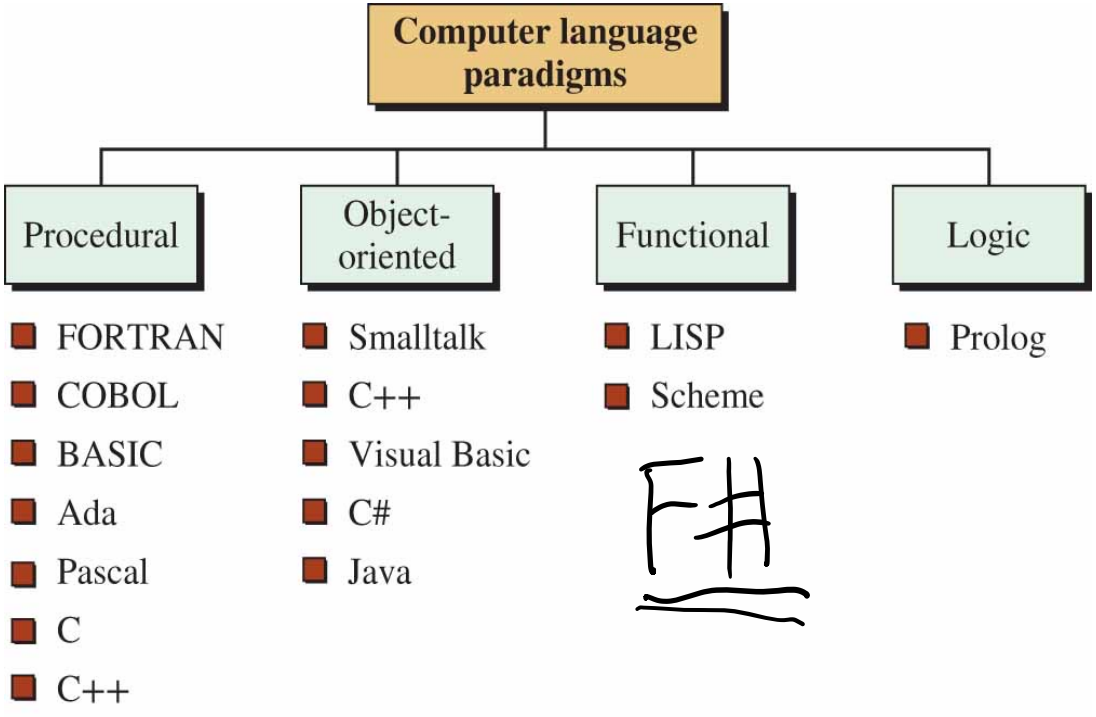
Declarative Programming



PROGRAMMING PARADIGMS



PROGRAMMING PARADIGMS



Access the text alternative for slide images.

In a **procedural** (also called **imperative**) paradigm, a program is a set of commands. The execution of each command changes the state of the memory related to that problem.

Imperative

When you start coding, you most likely write an expression. Then you write another expression followed by another, one after another. Your focus is on solving the problem, and you're being specific about how you solve it. This approach is referred to as an imperative approach.

What are the problems of procedural code? How does object-oriented programming help solve these problems?

Big classes with several unrelated methods focusing on different concerns and responsibilities. These methods often have several parameters. You often see the same group of parameters repeated across these methods. All you see is procedures calling each other passing arguments around.

By applying object-oriented programming techniques, we extract these repetitive parameters and declare them as fields in our classes. Our classes will then encapsulate both the data and the operations on the data (methods). As a result, our methods will have fewer parameters and our code will be cleaner and more reusable.

Object-Oriented programming is about objects. These objects interact with each other to perform various tasks.

Benefits of object-oriented programming:

- Reduce complexity
- Reuse the code

For principles of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Encapsulation is the first principle of object-oriented programming. It suggests that we should bundle the data and operations on the data inside a single unit (class).

Abstraction is the second principle of object-oriented programming. It suggests that we should reduce complexity by hiding the unnecessary implementation details. As a metaphor, think of the remote control of your TV. All the complexity inside the remote control is hidden from you. It's abstracted away. You just work with a simple interface to control your TV. We want our objects to be like our remote controls.

Inheritance is a mechanism for **reusing** code.
So we can implement all these common features once in a class, and then have other classes inherit these features.

Polymorphism means many forms.
It allows us to build applications that can be easily extended.

Refers to the situation where an object can take many different forms. This is a very powerful technique.

Functional paradigm:

In the functional paradigm, a program is a mathematical function. In this context, a function is a black box that maps a list of inputs to a list of outputs.

In this paradigm, we are not using commands and we are not following the memory state. The idea is that we have some primitive function, such as add, subtract, multiply, divide. We also have some primitive functions that create a list or extract the first element or the rest of element from a list. We can write a program or a new function by combining these primitive function.

Functional programming for building up layers of logic using functions or using other functions that can take functions as parameters to build your abstractions and build your more complicated logic.

The primary goal of Functional languages is: To minimize the use of mutable state.

Problem with mutable state:

Whenever you pass a reference to an object between different classes, that all have accessed to that same object at the same time. This can be problematic whenever one component is changing a variable in that object then another component needs to read it.

In a functional programming, everything is really based on immutable data structures.

The more you write your code based on the use of immutable data structures the more stable it is going to be and the more robust your code is going to overall.

Writing code in this way can really improve the quality and stability of the applications that you write.

Makes more sense for applications that require a high level of reliability, or problems that involve messages being passed around and getting transformed along the way.

It helps you write better object-oriented programming and imperative code.

The **logic paradigm** uses a set of facts and a set of rules to answer queries. It is based on formal logic as defined by Greek mathematicians. We first give the program the facts and rules before asking queries.

Both logic and functional paradigm are **Declarative** paradigm**.**

Logic Programming:

A well known slogan (R. Kowalski):

- **Algorithm = Logic + Control**
- Logic: What must be done.
- Control: How the desired solution is found.

Imperative programming must consider both components.
Logic programming only needs to consider the logic one.

Prolog language basically has three different elements:

- **Facts** – The fact is predicate that is true, for example, if we say, “Tom is the son of Jack”, then this is a fact.

- **Rules** – Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as:

- **Questions** – And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

PROGRAMMING PARADIGMS

Question:

What paradigm is the best?

Access the text alternative for slide images.

PROGRAMMING PARADIGMS

No paradigm works best in all situations.

Remember, in software engineering, there is no such thing as the one size fits all.
Every problem is different, there is no such thing as the best language or the best paradigm or the best framework.

It all depends on the problem and more importantly, on the context and budget.

Access the text alternative for slide images.

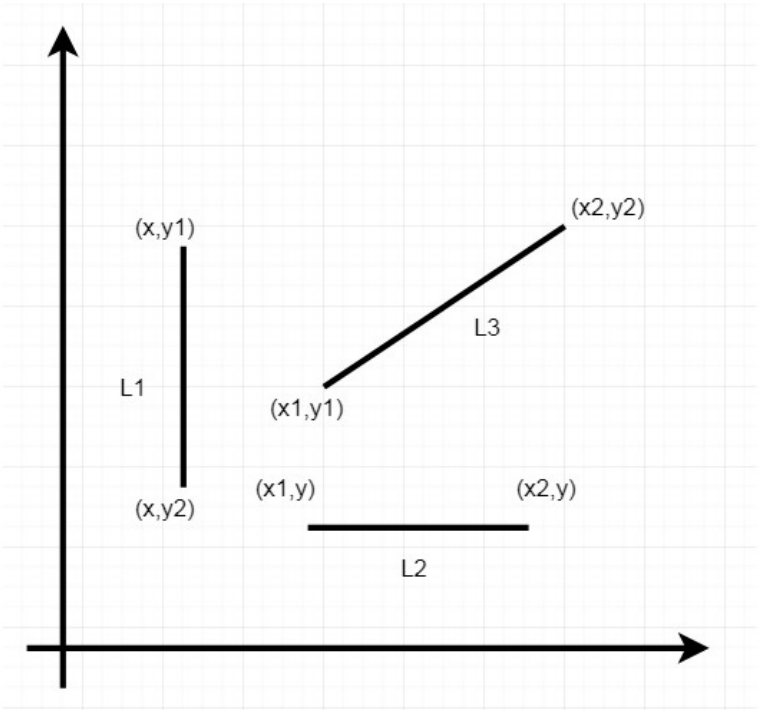
Verify whether a line segment is horizontal, vertical or oblique

Use three different programming paradigms to solve this problem:

- Object-oriented programming C++, Java.
- Functional programming F#
- Logic programming SWI-prolog.

Horizontal and Vertical Line Segments

There are three types of line segments, horizontal, vertical or oblique. This example verifies whether a line segment is horizontal, vertical or oblique.



From this diagram we can understand that:

- For Horizontal lines, the y coordinate values of two endpoints are same.
- For Vertical lines, the x coordinate values of two endpoints are same.
- For Oblique lines, the (x,y) coordinates of two endpoints are different.