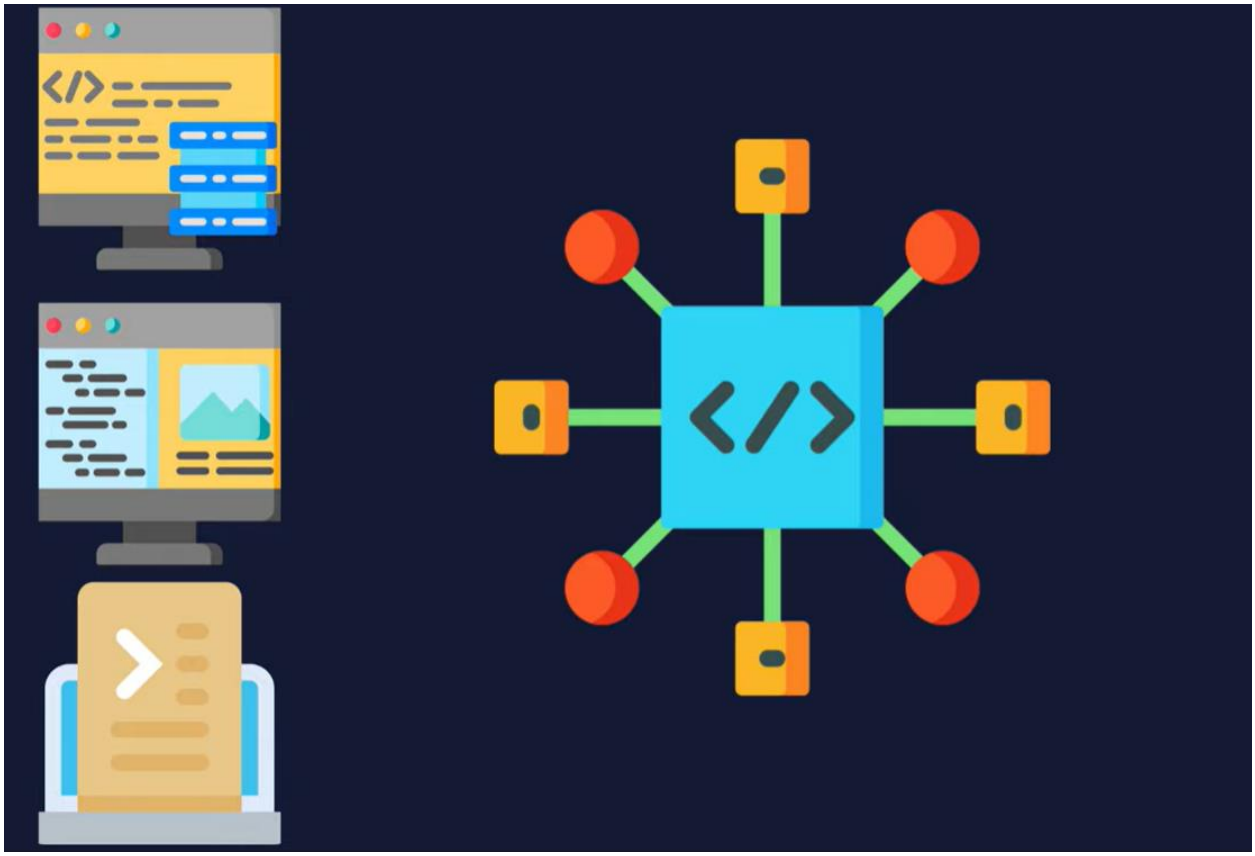


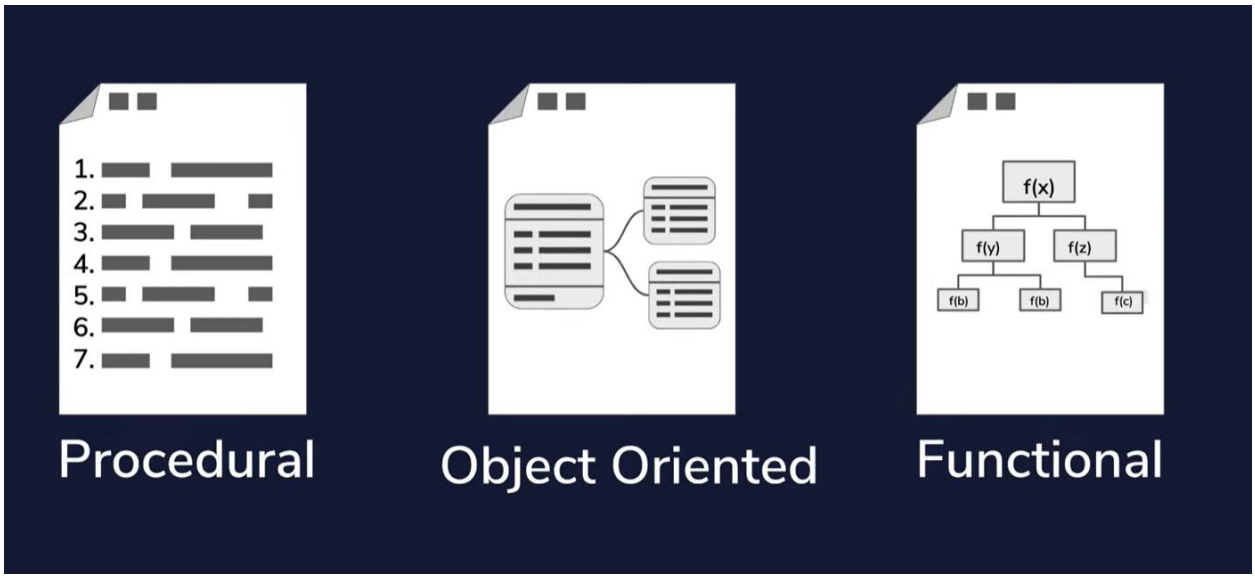
A programming paradigm is a specific style of organizing and writing programs, with the goal of increase organization, less codes and better code maintainability.

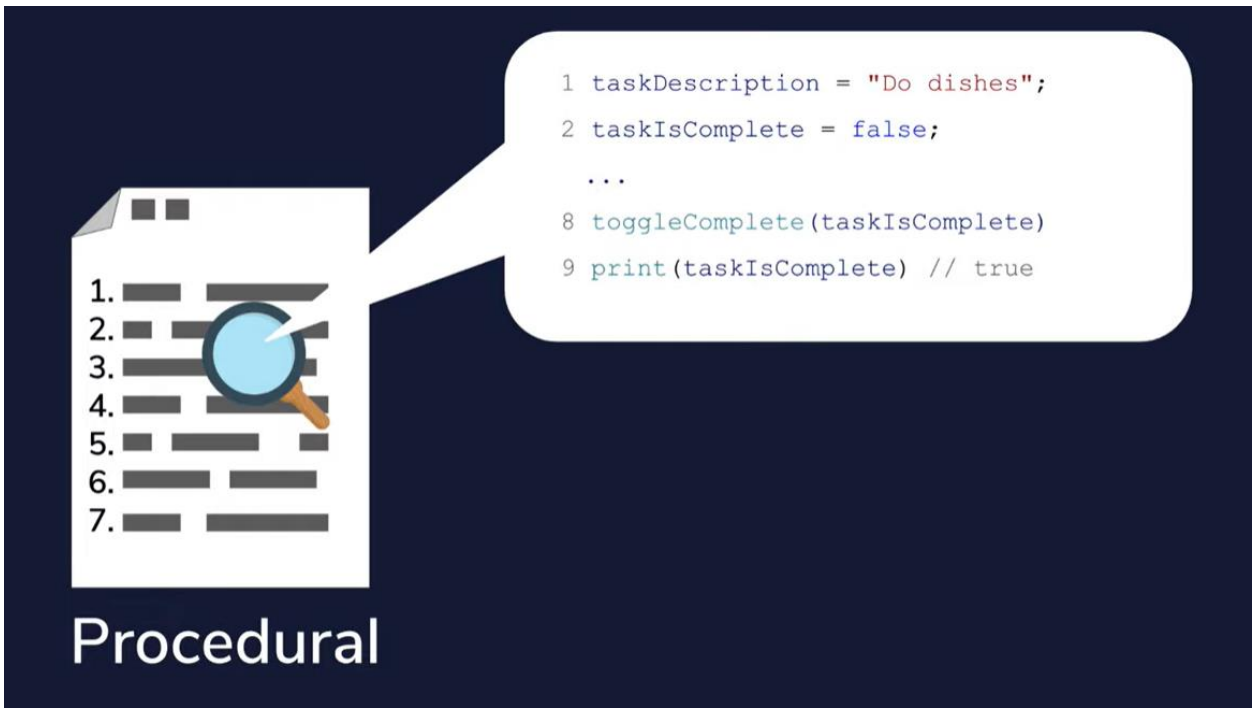
We are going to discuss here, object orientation.

All programs have Data and Behavior.

Data: What the program knows.

Behavior: What the program can do with the data.

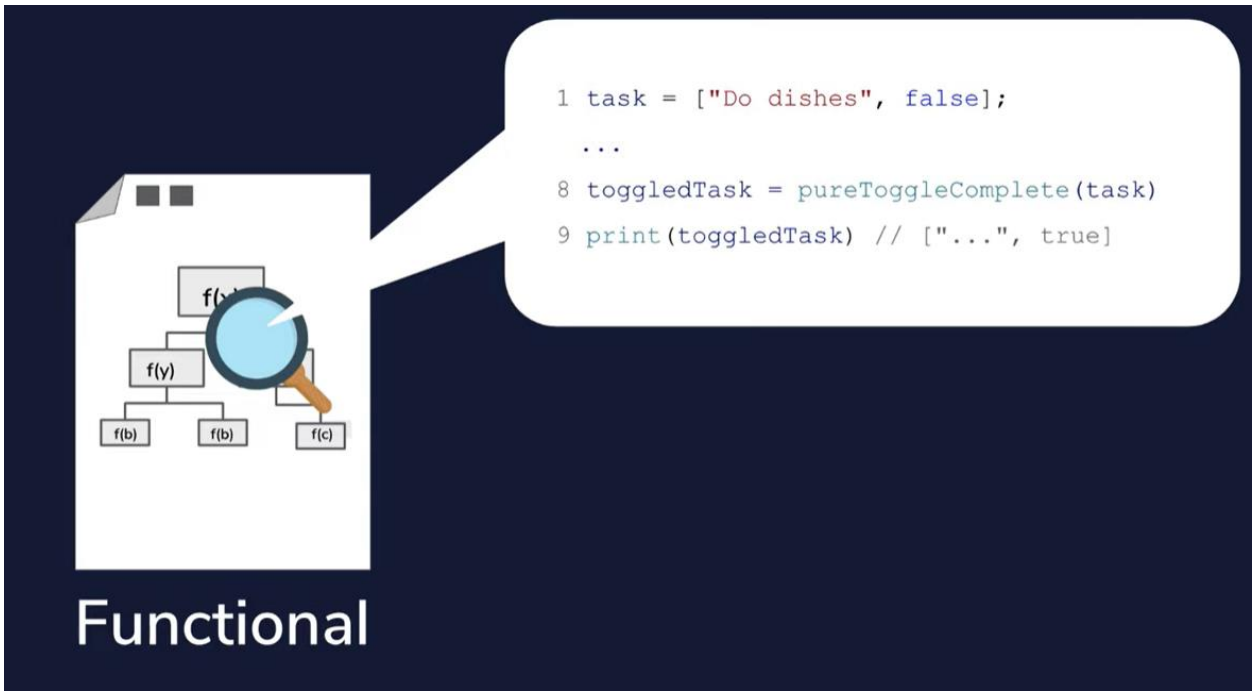




In procedural programming, the data and the behaviors are spread out into the same set of step by step instructions.

When we need to use the behavior, in this case toggling a todo class, we can manipulate the data directly, thus changing the overall state of our application.

In procedural programming, the order of the code matters very much and the data is mutated directly as the program carries out its behavior.



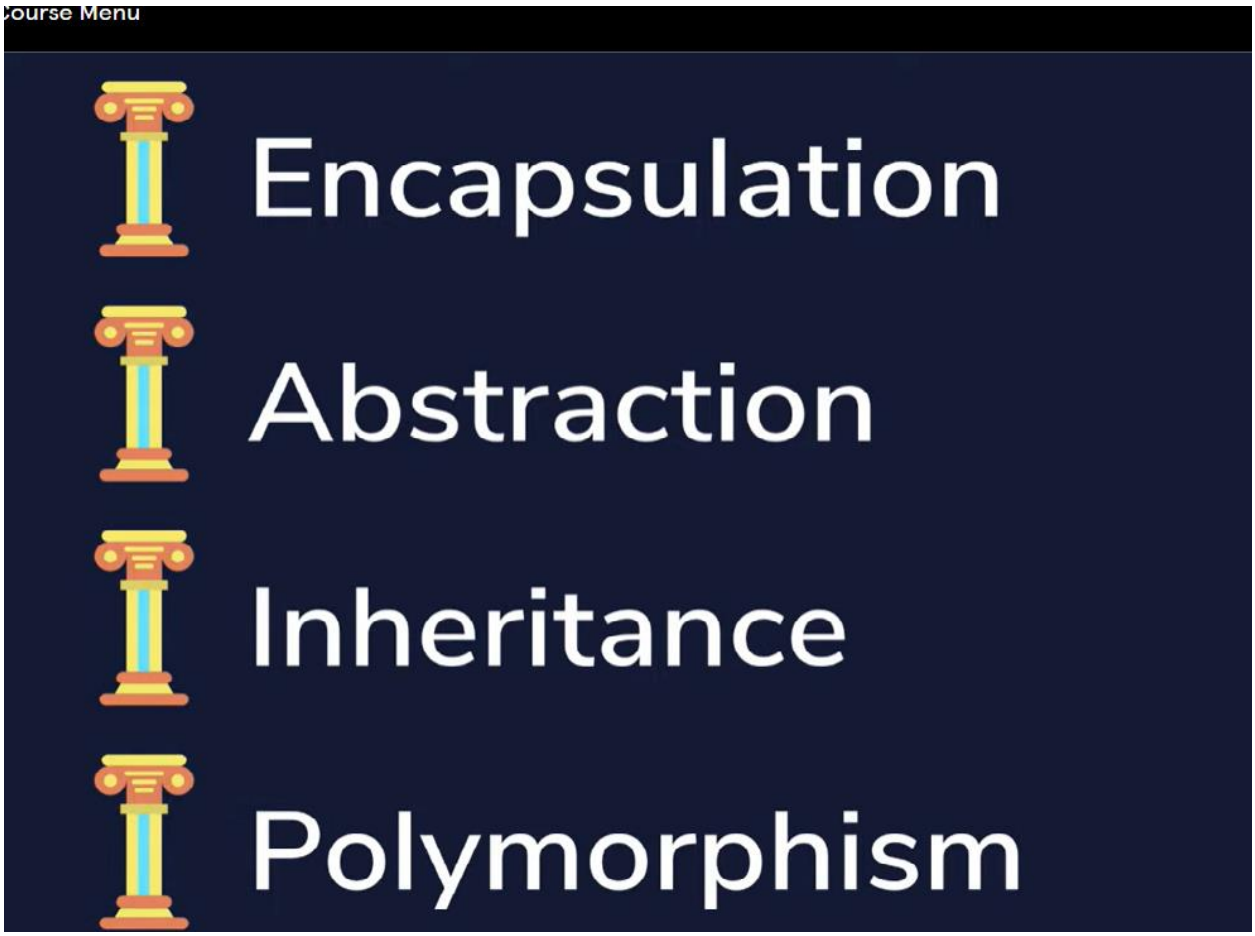
In Functional programming, the data and the behavior are kept separately. When we use the behavior, instead of mutating the data directly, our behavior will be abstracted into a pure function which takes our data in as an input and returns and updated copy of our data as an output.

In functional programming we never mutate the data directly, instead we rely in pure functions to return a new copy of the updated data.

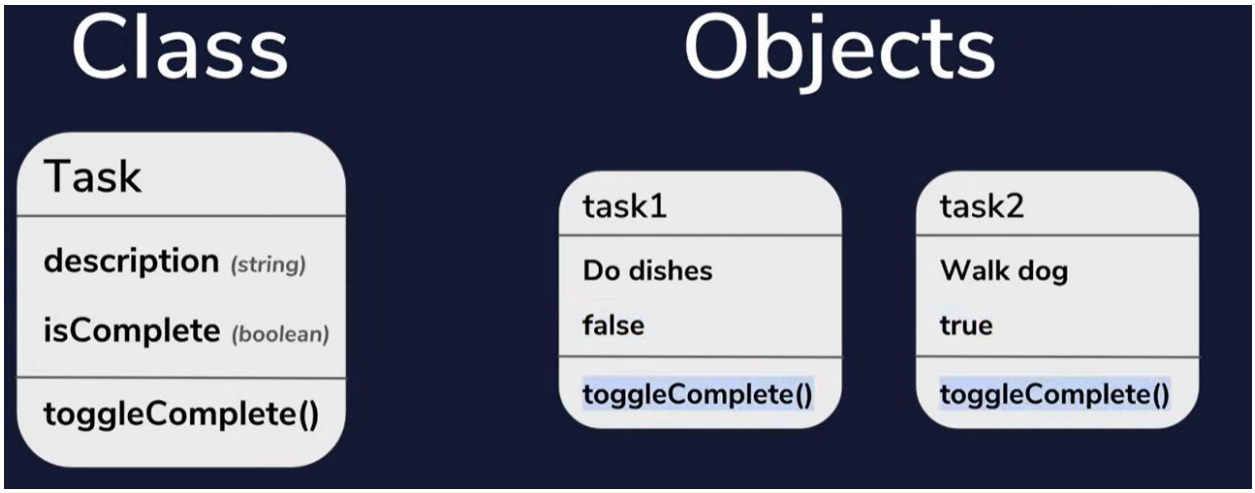
In object-oriented programming, the data and the behavior are groped together in objects. An object combines all of the attributes or data as well as all of the methods or behavior, into an easy to use container.

When we want to access the data or use the behaviors, we do so through the object.

4 PILARS OF OBJECT ORIENTATION:

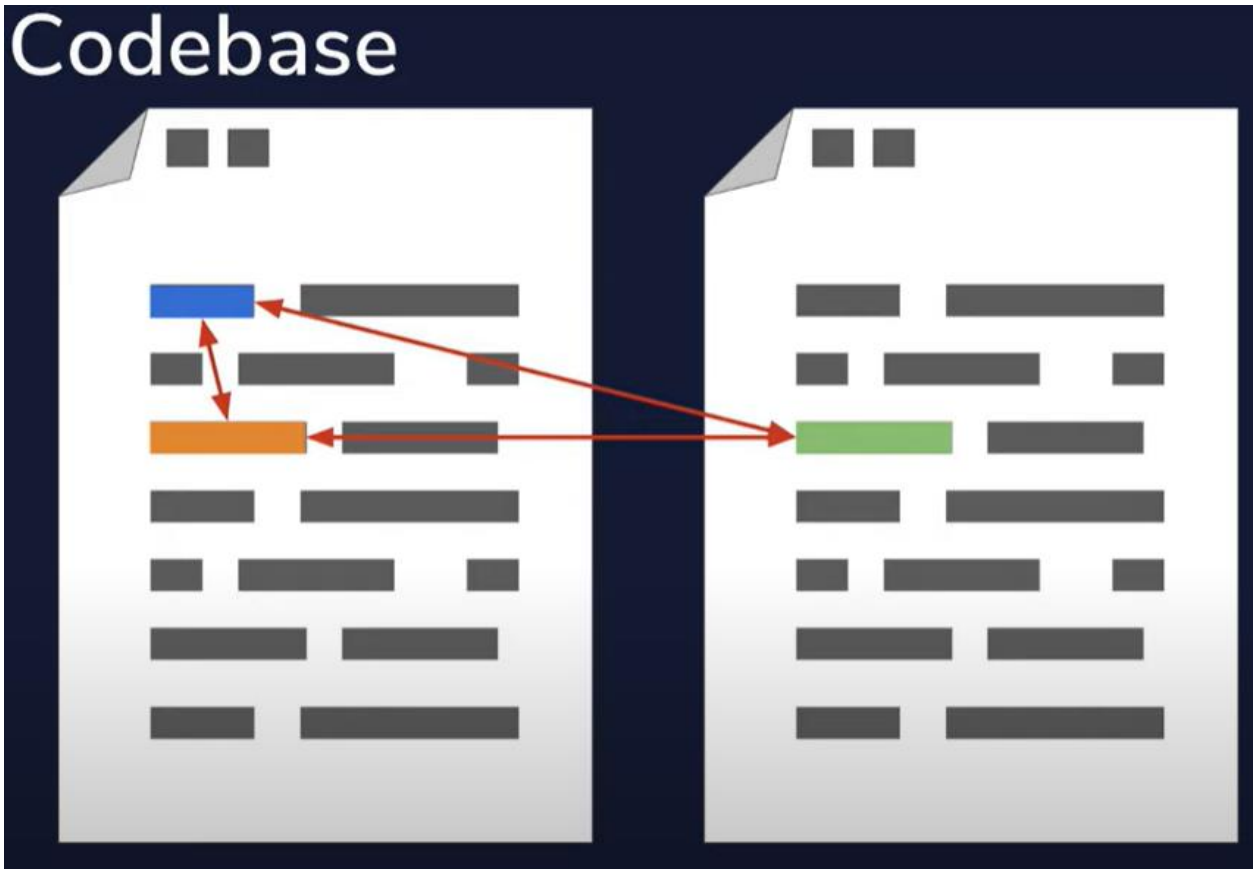


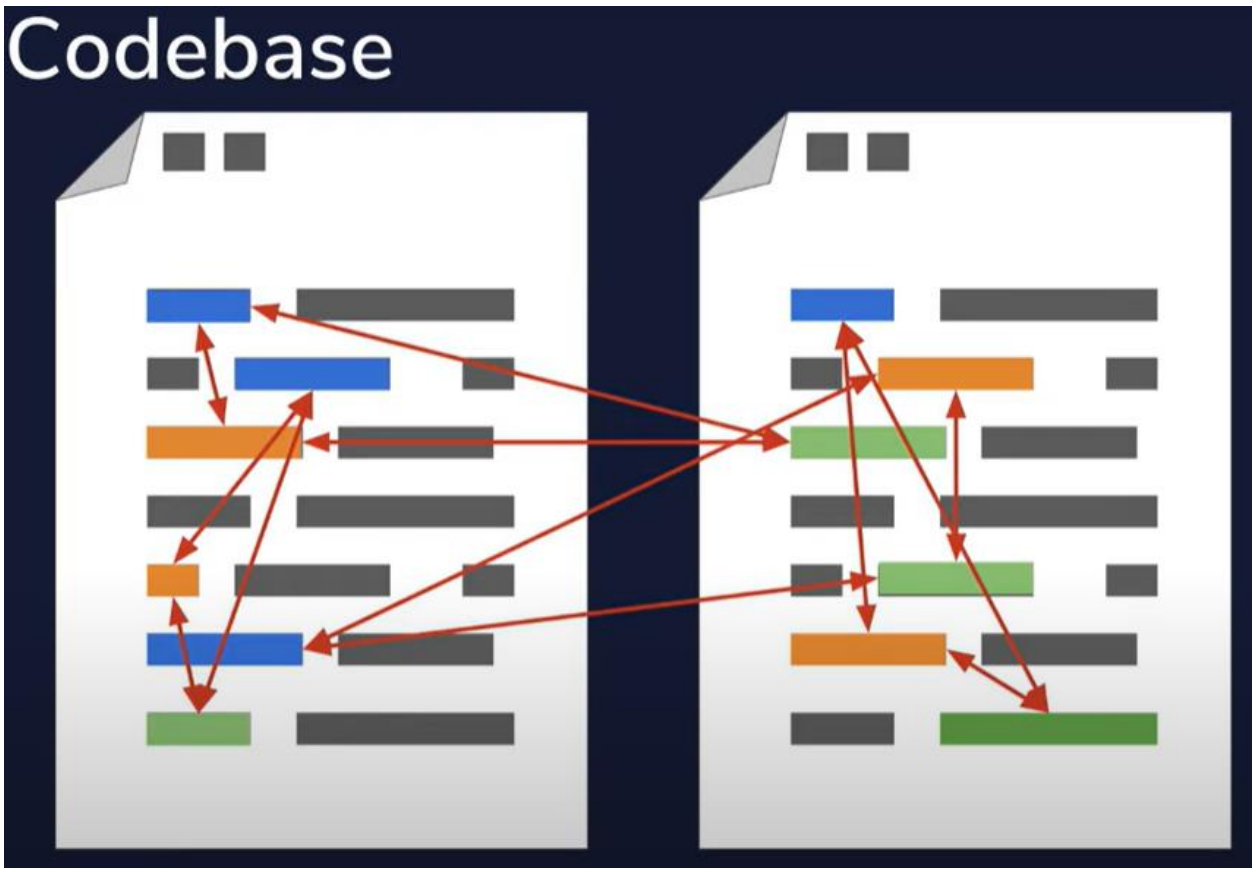
ENCAPSULATION:



Refers to an object's ability to bundle together related data and behavior and limit data scope. It prevents code from outside the object from seeing how it works and modifying it in undesirable ways.

Without encapsulation, you will have the data and behavior distributed in different sections of the program, creating spaghetti code.





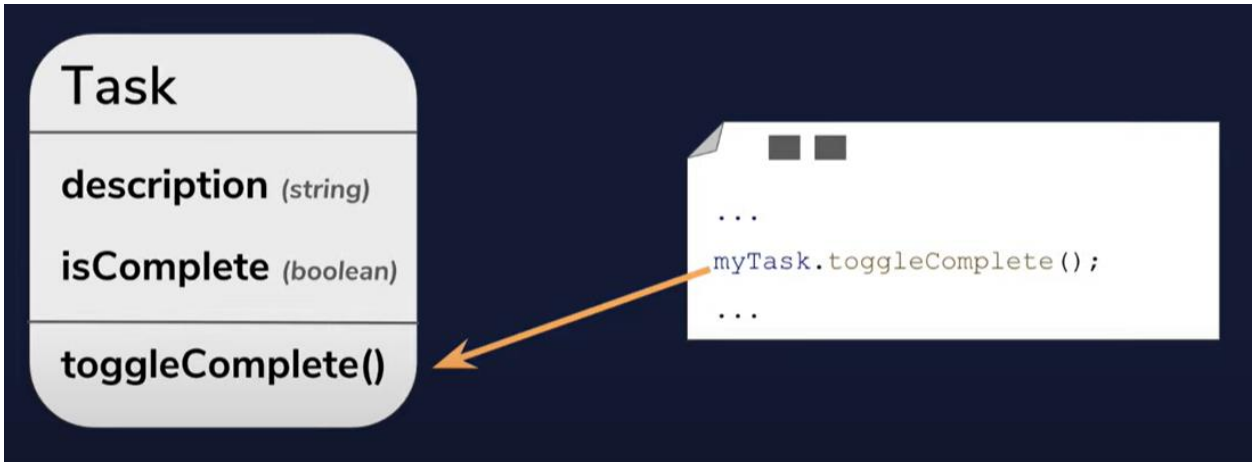
Encapsulation bundles data and behavior together that should be together, making it easy to be used in other parts of the program.

ABSTRACTION:

Ability to hide complex logic from the user making the code easier to use in other places.

Email is a good example of abstraction.

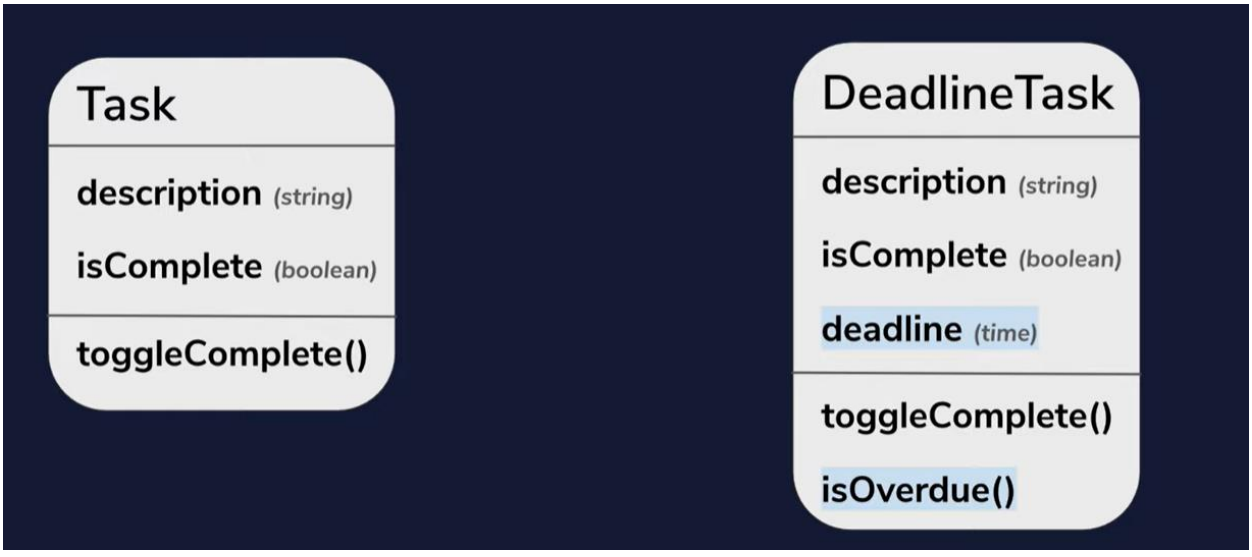
For example, we just need to call the class method toggleComplete() to toggle a class and use it and we don't need to know how the method was implemented.

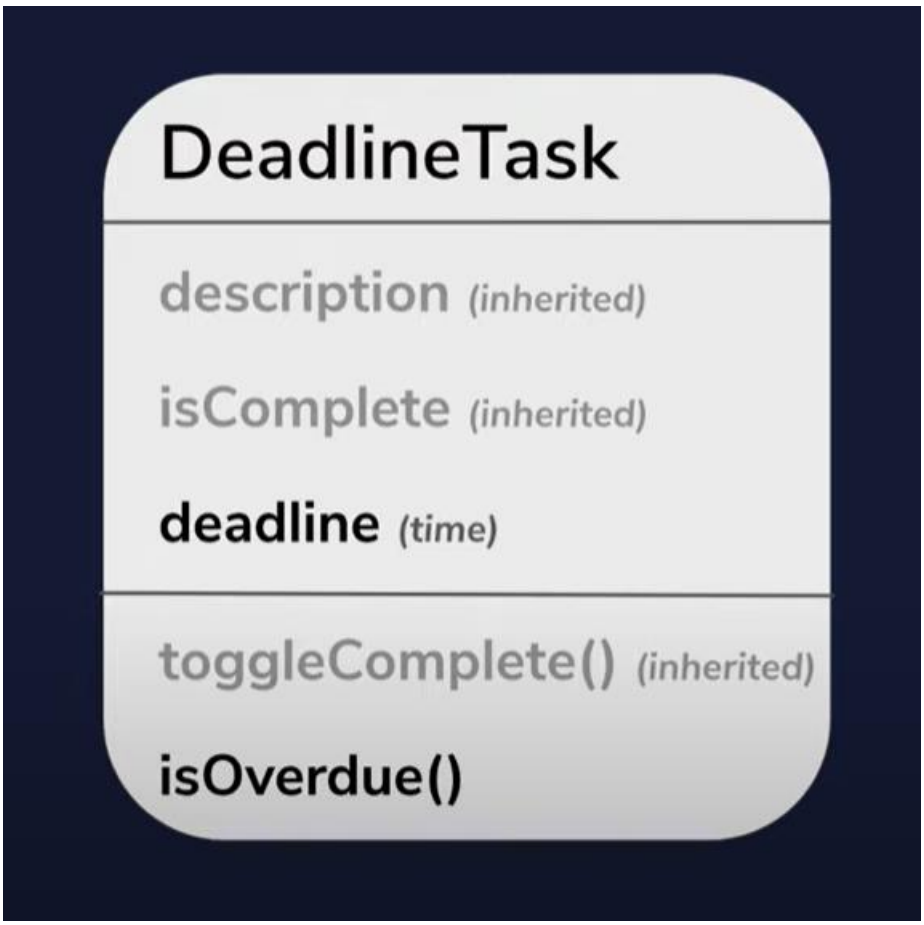


INHERITANCE:

Allows a class to inherit all of the functionality of another class.

This allows us to share code between multiple classes.





POLYMORPHISM

Ability of the objects to have different forms.

In the example, any code that is designed to work with a task object is also going to work with a deadline task object.