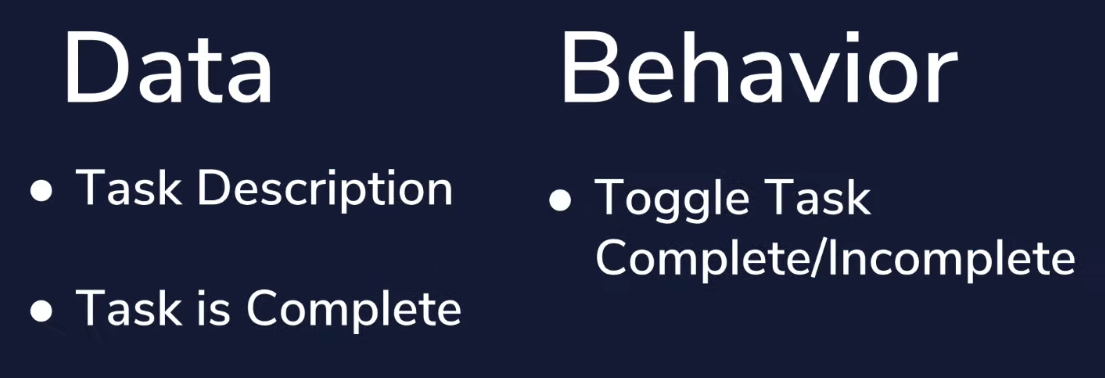
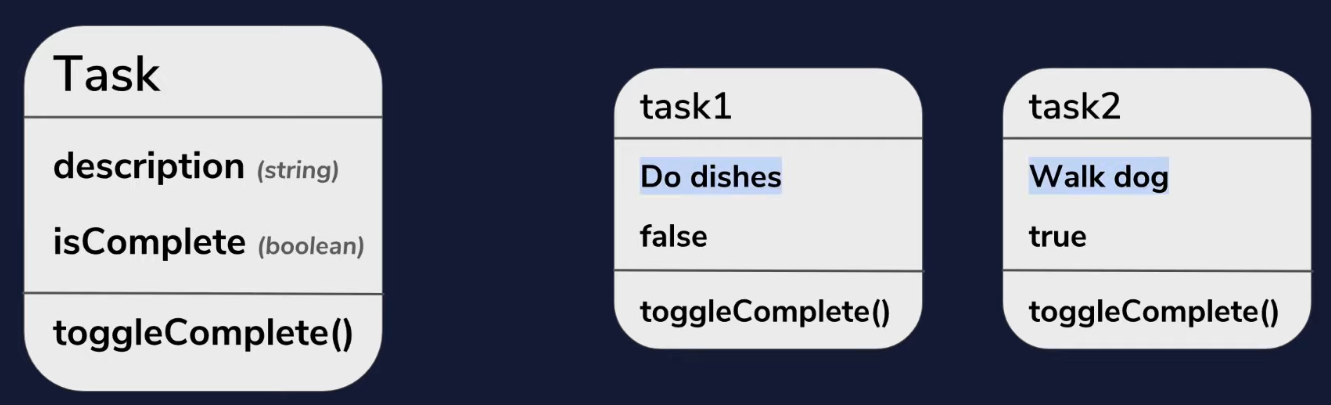
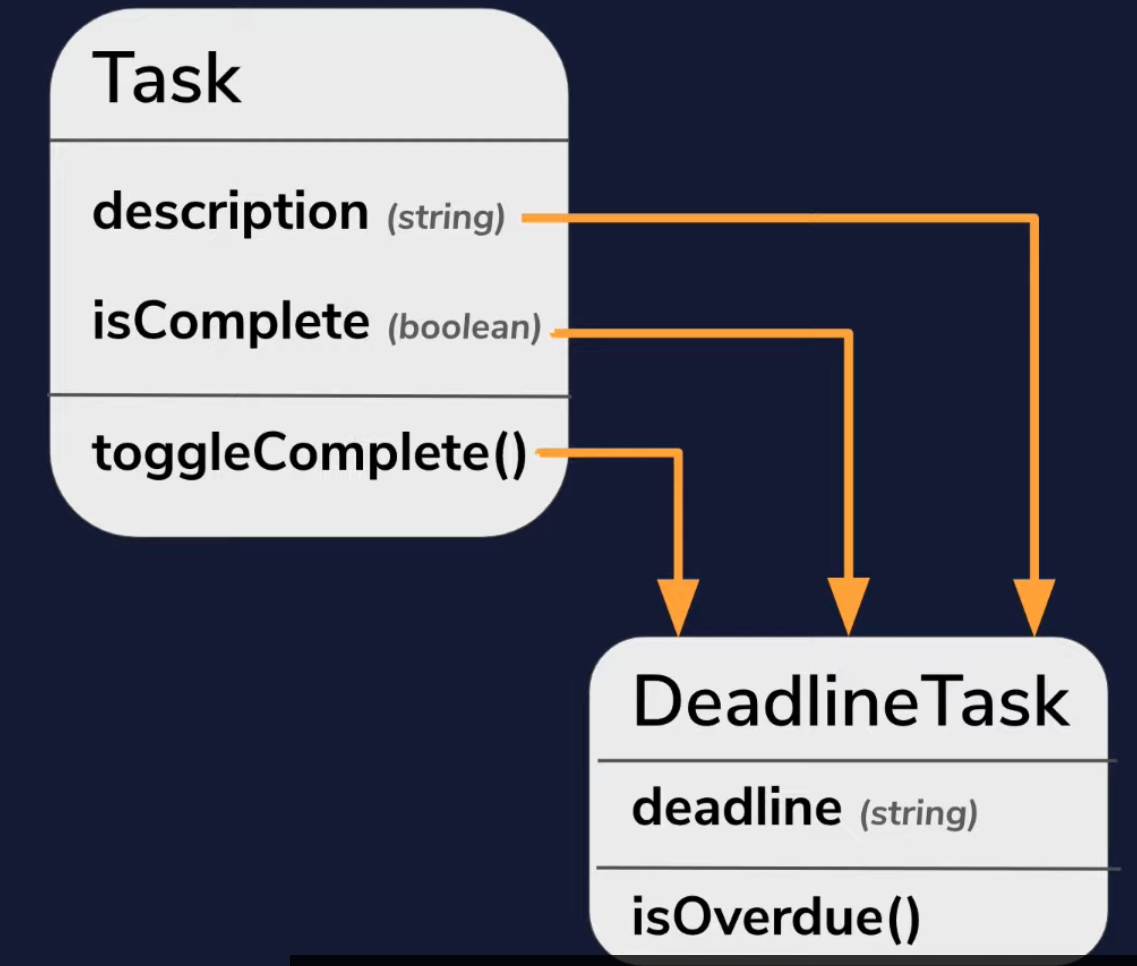
**Why Object-Oriented Programming?**

- In the beginning, computer programs were simple and straightforward lines of code. As they evolved, however, programming paradigms developed as a way to tackle increasing complex problems  
***Programming Paradigm*** - A specific style of organizing programs to increase organization, reduce bugs, and improve code maintainability  
- All programs have ***data*** (what the program knows) and ***behavior*** (what the program can do with that data)  
  
***Procedural Programming*** – Another programming paradigm where the data and the behaviors are spread out into the same set of step-by-step instructions  
- When we need to use a behavior we can manipulate data directly and change it’s state  
- Order of code matters very much and data is mutated directly as program carries out its function  
- Basic programming, beginning/starting out  
***Functional Programming***  - Another programming paradigm where the data and the behavior are kept separate   
- When we want to use a behavior, the data is fed to it as an input, a function runs and provides us with an updated copy of the data as an output  
- Putting repeated actions/tasks into a function (*def function\_name():*) and calling it in code  
***Object-Oriented Programming*** – Data and behaviors are grouped together into objects  
- An object is a special entity in our code that combines all of the data and attributes as well as all of the behaviors into a container   
- When we want to use a behavior we access it through the object   
- Essentially like using VM’s or Docker Containers/Kubernetes Clusters to keep each function in it’s own fully formed, self-contained container and calling when needed

**Four Pillars of OOP:**

**1.** **Encapsulation**- Combine all data and behavior into a single class and define each task as a self-contained object that does a specific function  
- Prevents code from outside of object from seeing how it works and/or modifying its behavior  


**2. Abstraction**- Hides complex code from the user and makes it easier to use code in other places  
- Sending email only requires opening up email client and hitting send, all knowledge about servers, networking, protocols, etc is performed on the backend and not seen by end user

**3. Inheritance**- Allows a class to inherit all of the functionality of another class  
- This allows us to share code between multiple classes and reduce amount of code overall  
- Works best in situations where an object requires all the attributes of another one but also adds other specific requirements of its own  


**4. Polymorphism**- Word means *having multiple forms*  
- When an object inherits attributes from another object it also inherits its class type (Task = Task Type, Deadline Task = DeadlineTask Type and Task Type  
- Any code designed to work with Task object will also work with DeadlineTask object (because it is also a Task Type object)  
