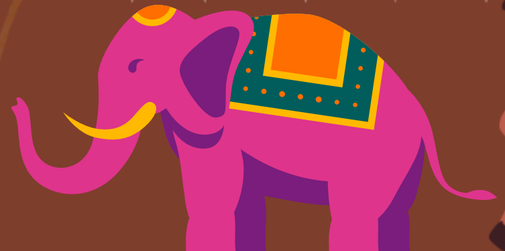
VENKATESH MOGILI

https://vmtraining.netlify.app


సంక్రాంతి
శుభాకాంక్షలు

Happy makara Sankranti



SOLID PRINCIPLES

అంటే ఏంటి?

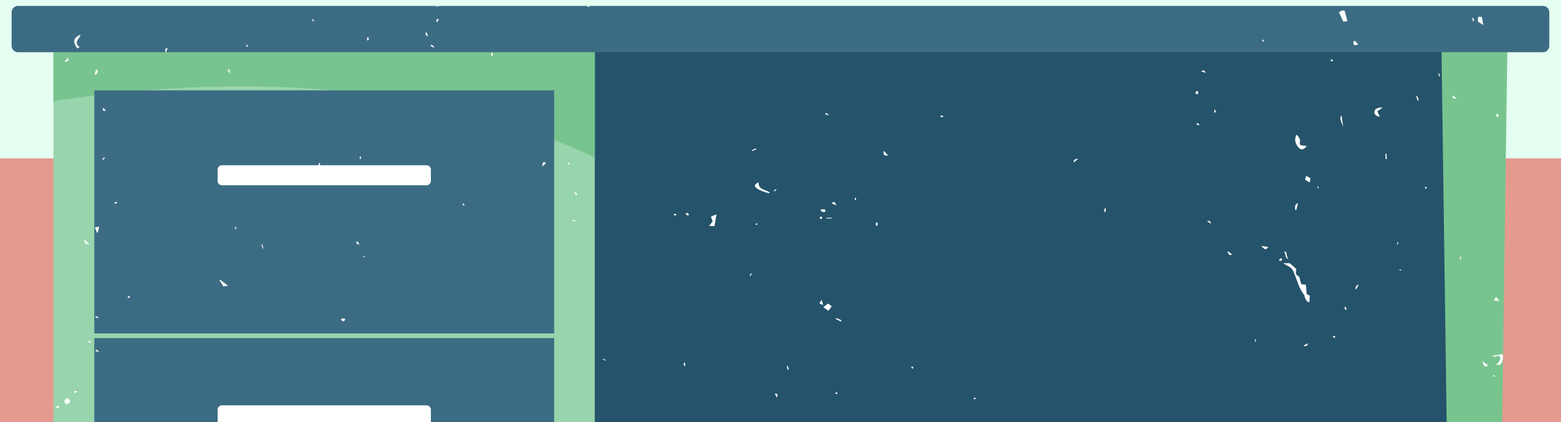
VENKATESH MOGILI

SUBSCRIBE

LET'S GET STARTED

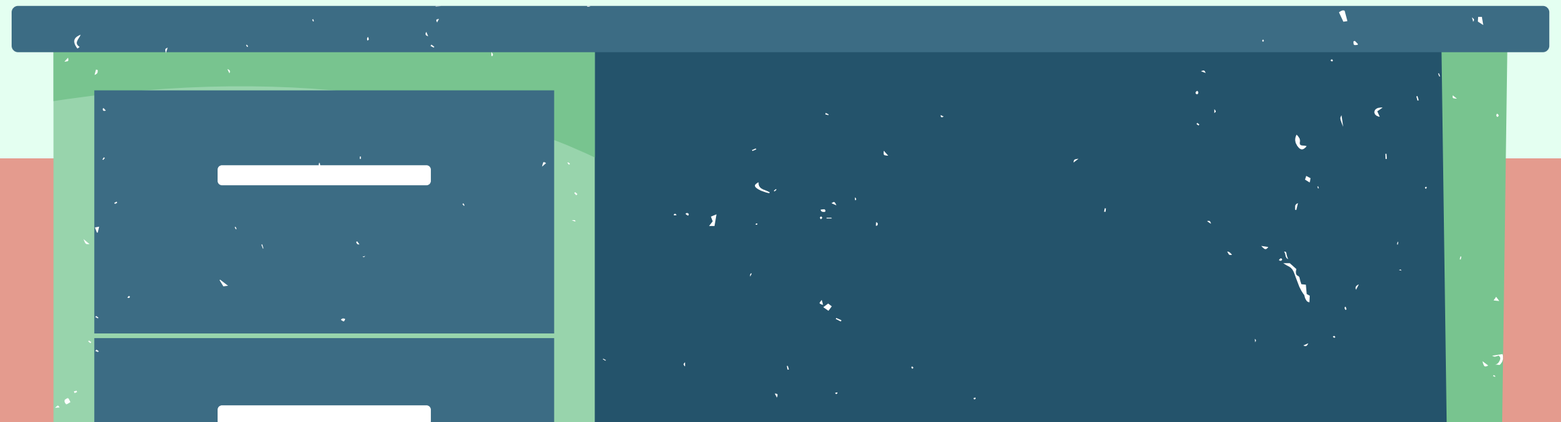
SOLID

SOLID IS AN ACRONYM FOR FIVE SEPARATE DESIGN PRINCIPLES.



SOLID

WE USE SOLID PRINCIPLES IRRESPECTIVE OF THE LANGUAGE FOR BETTER CODING PURPOSE.

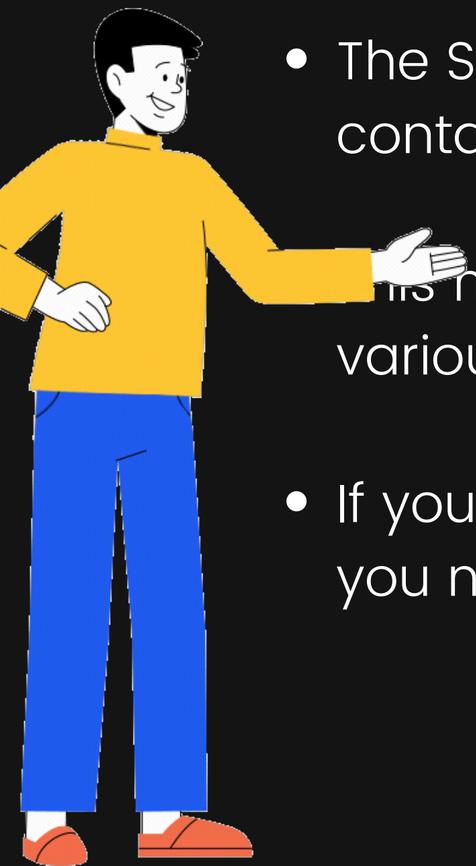


S

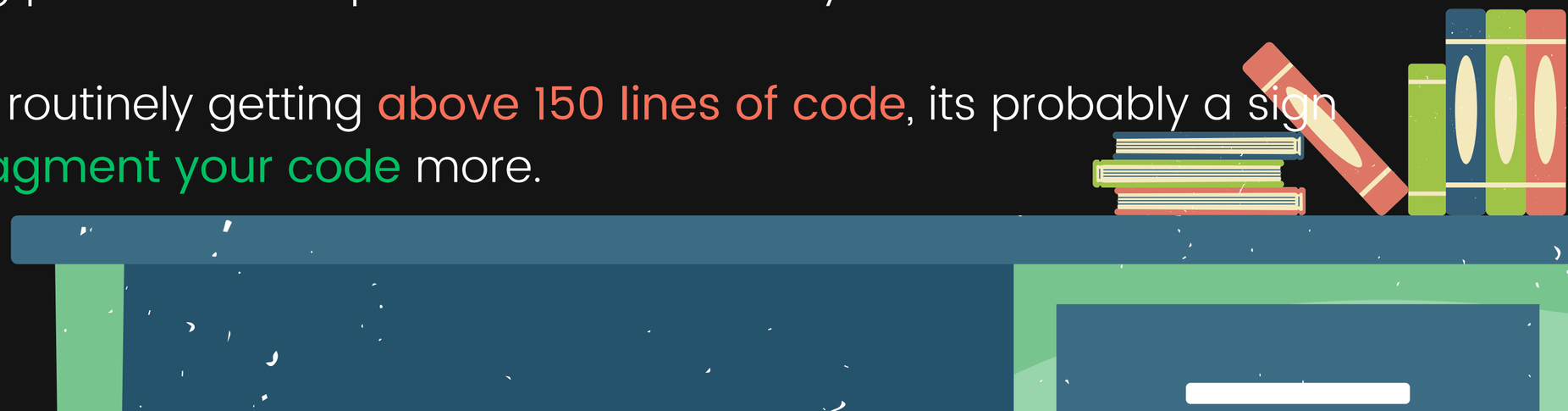
SINGLE RESPONSIBILITY PRINCIPLE (SRP)

Every
function/class/component
should do exactly one thing.

Single Responsibility Principle (SRP)



- The SRP encourages us to fragment our code, from monolithic files containing thousands of lines, into dozens of smaller 50-100 line files.
- This makes it much more maintainable, as it becomes easy to see the various moving parts to some particular functionality.
- If your files are routinely getting above 150 lines of code, its probably a sign you need to fragment your code more.



SRP

PRACTICAL EXAMPLE

`<TodosPage />` component shouldn't care where the todos come from, or in what format they are displayed.

<TodosPage /> should only care about it actually showing the user our list of todos.

So, we should probably break this component into two.

SRP

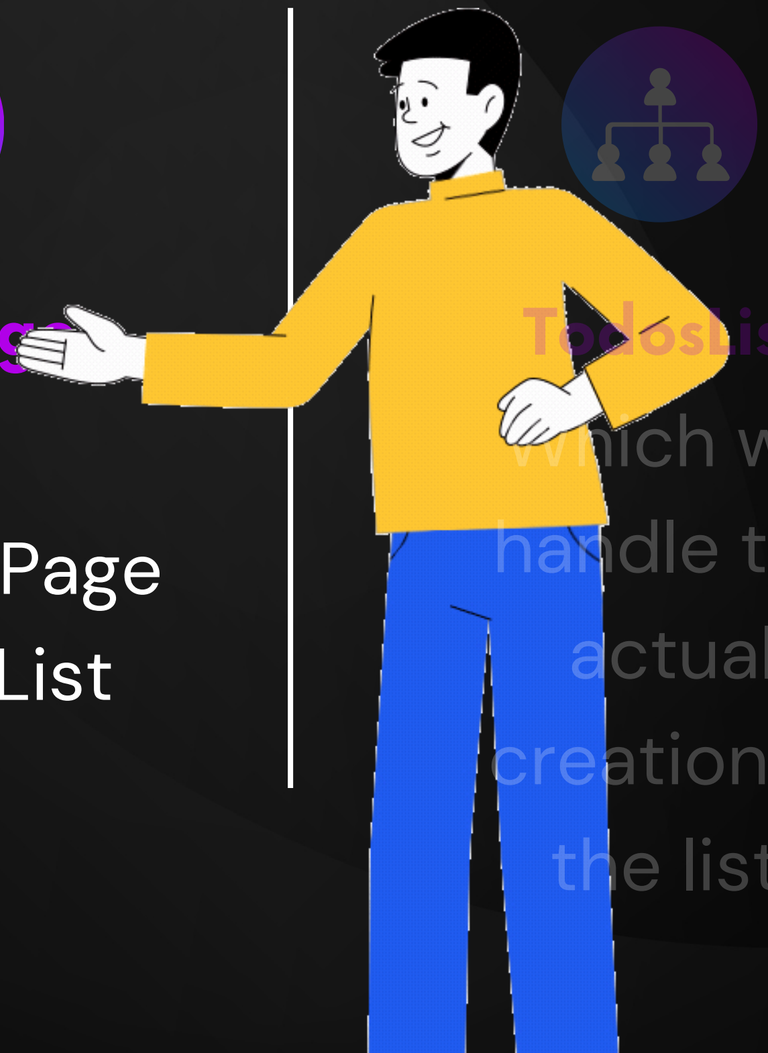
TodosPage

Which will show all the user todos.

TodosList

Which will handle the actual creation of the list.

SRP



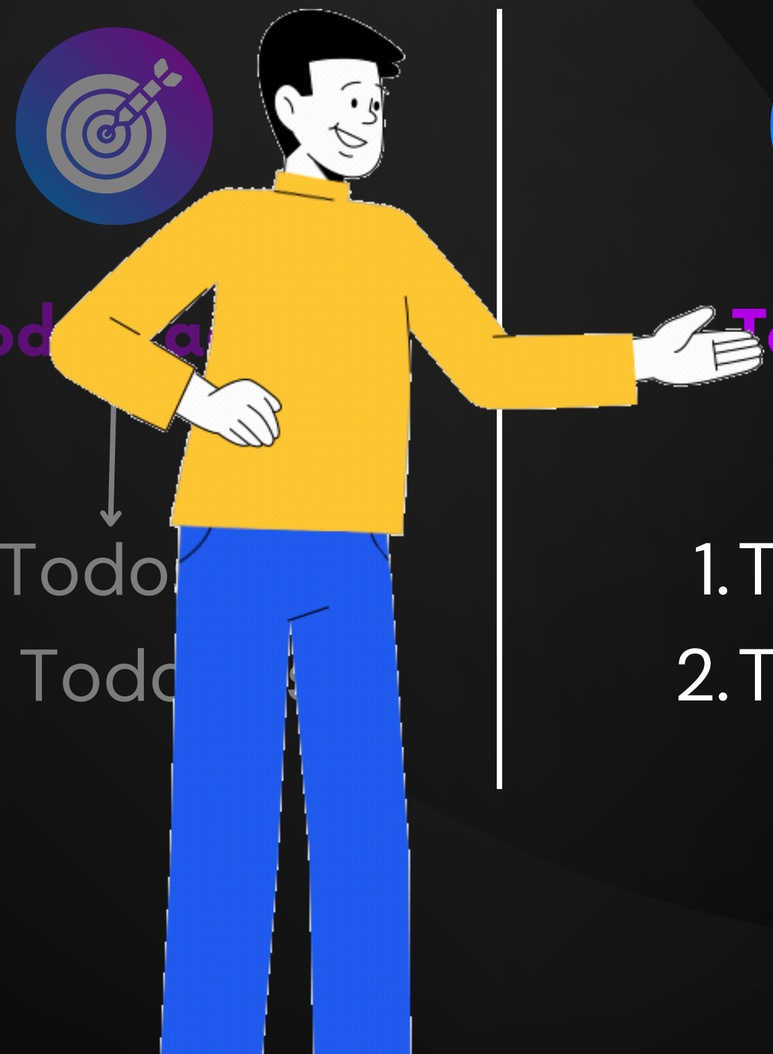
TodosPage

1. TodosPage
2. TodosList

TodosList

which will handle the actual creation of the list.

SRP



Tod a

1. Todo
2. Todo

TodosList

1. TodosList
2. TodoItem

SRP

Move the fetching logic into a separate function called fetchTodos.

3

Finally, We have 4 components:

TodosPage

TodosList

TodoItem

fetchTodos

This makes our codebase far more **modular and easier to maintain,** as each component only deals with one thing.

SUMMARY

IN THIS LESSON, WE HAVE LEARNED:

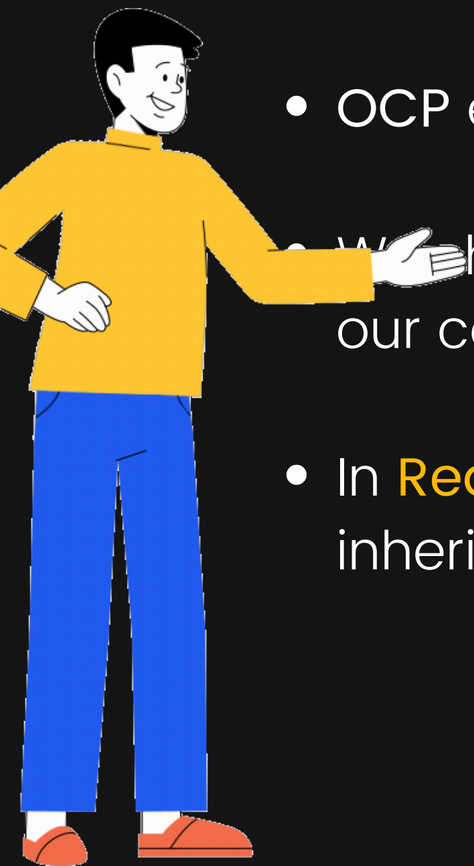
1. Each component should have a single responsibility.
2. Each component should be small, reusable, understandable and easy to maintain.
3. Each component should have a single reason to change.

O

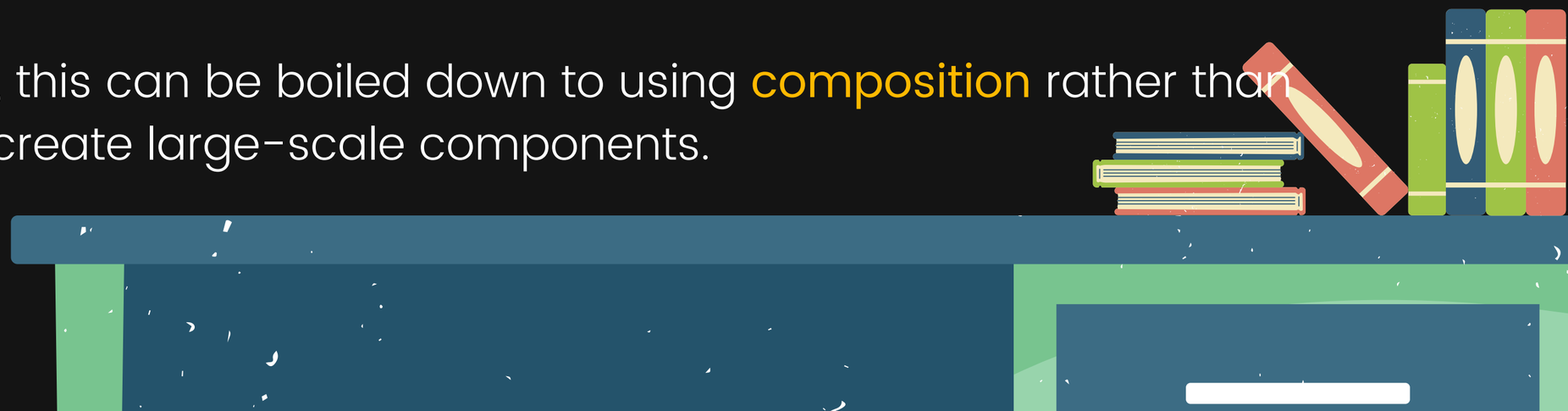
OPEN/CLOSED PRINCIPLE (OCP)

Make big components from lots of smaller ones.

Open/Closed Principle (OCP)

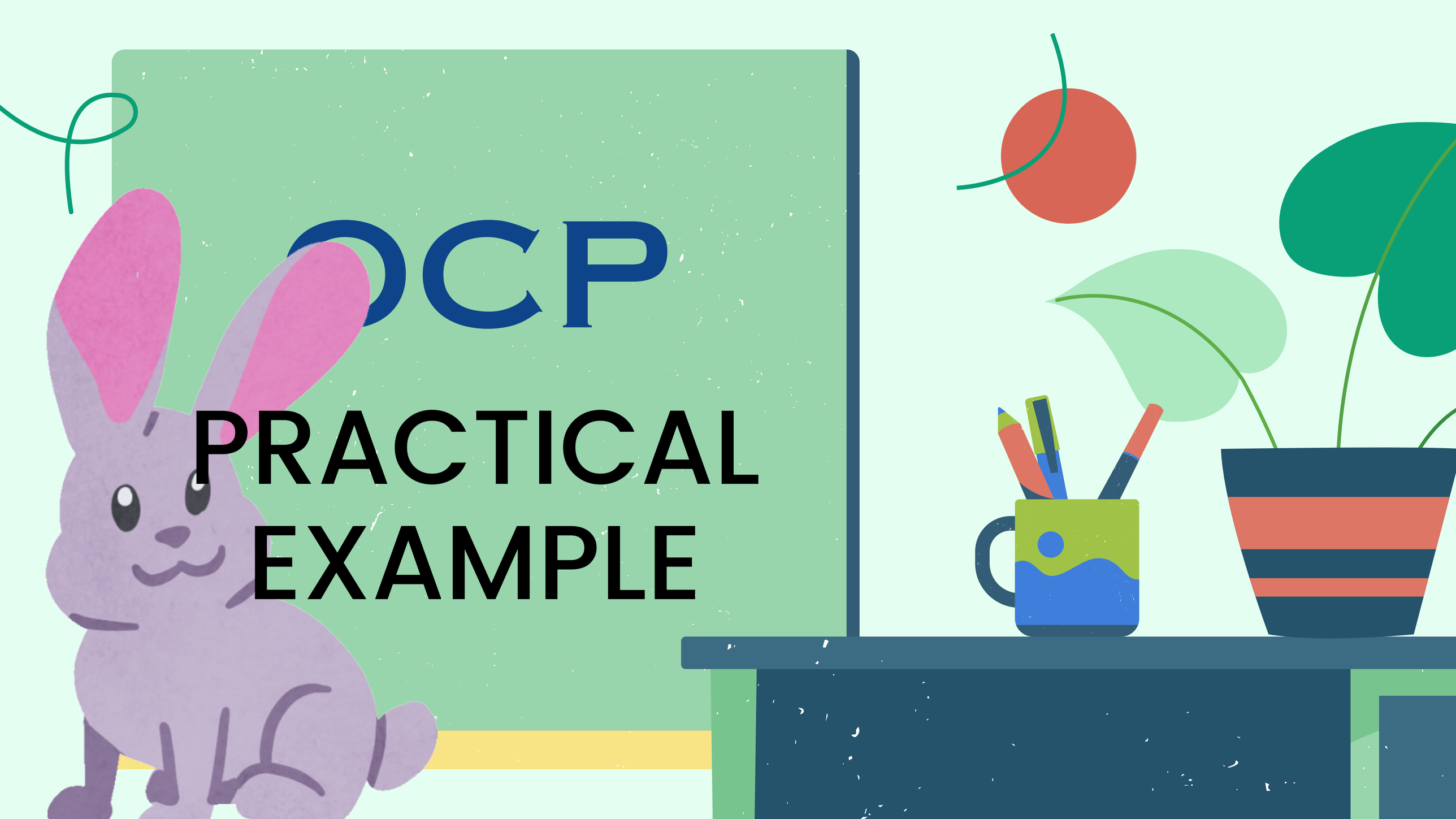


- OCP explains that we should make our code extensible.
- We should be able to add new features without having to rewrite parts of our codebase.
- In React terms, this can be boiled down to using composition rather than inheritance to create large-scale components.



OCP

PRACTICAL EXAMPLE



We should always build big components from smaller ones, using props and special properties like props.children to build complexity when required.

props - Properties

Use composition to make large components, rather than extending from other components.

Composition: the way in which the parts of something are arranged.

SUMMARY

IN THIS LESSON, WE HAVE LEARNED:

1. "Tight-coupling" means that one component relies heavily on another, such that any change in one could break the other.
2. This goes against the idea of modularity in code, and so we usually want to avoid this wherever possible.

L

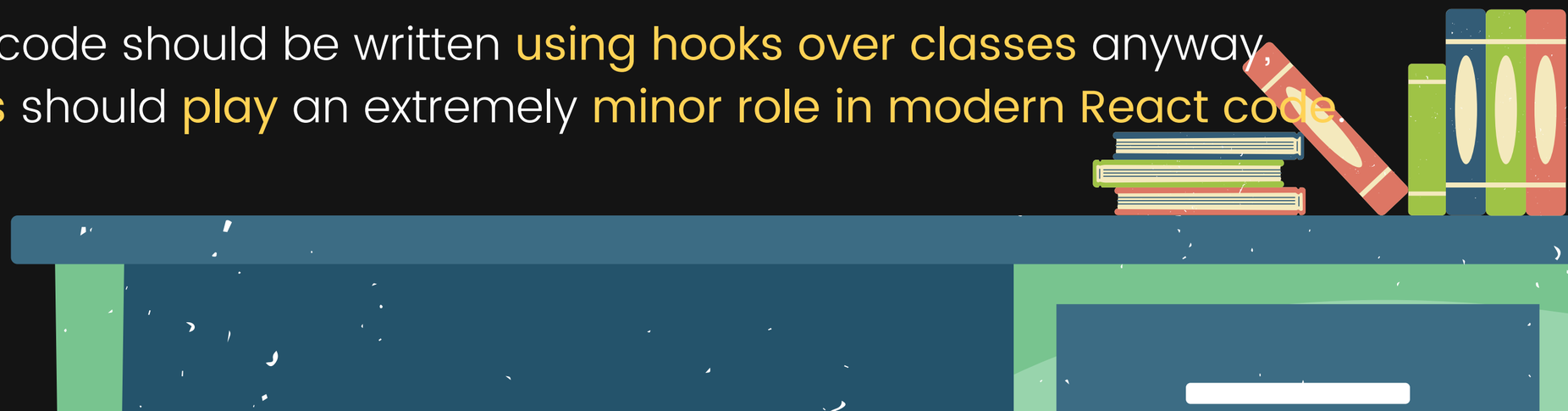
LISKOV SUBSTITUTION PRINCIPLE (LSP)

This is certainly one of the less relevant principles to be applied to frontend design.

We just don't really use this in React.

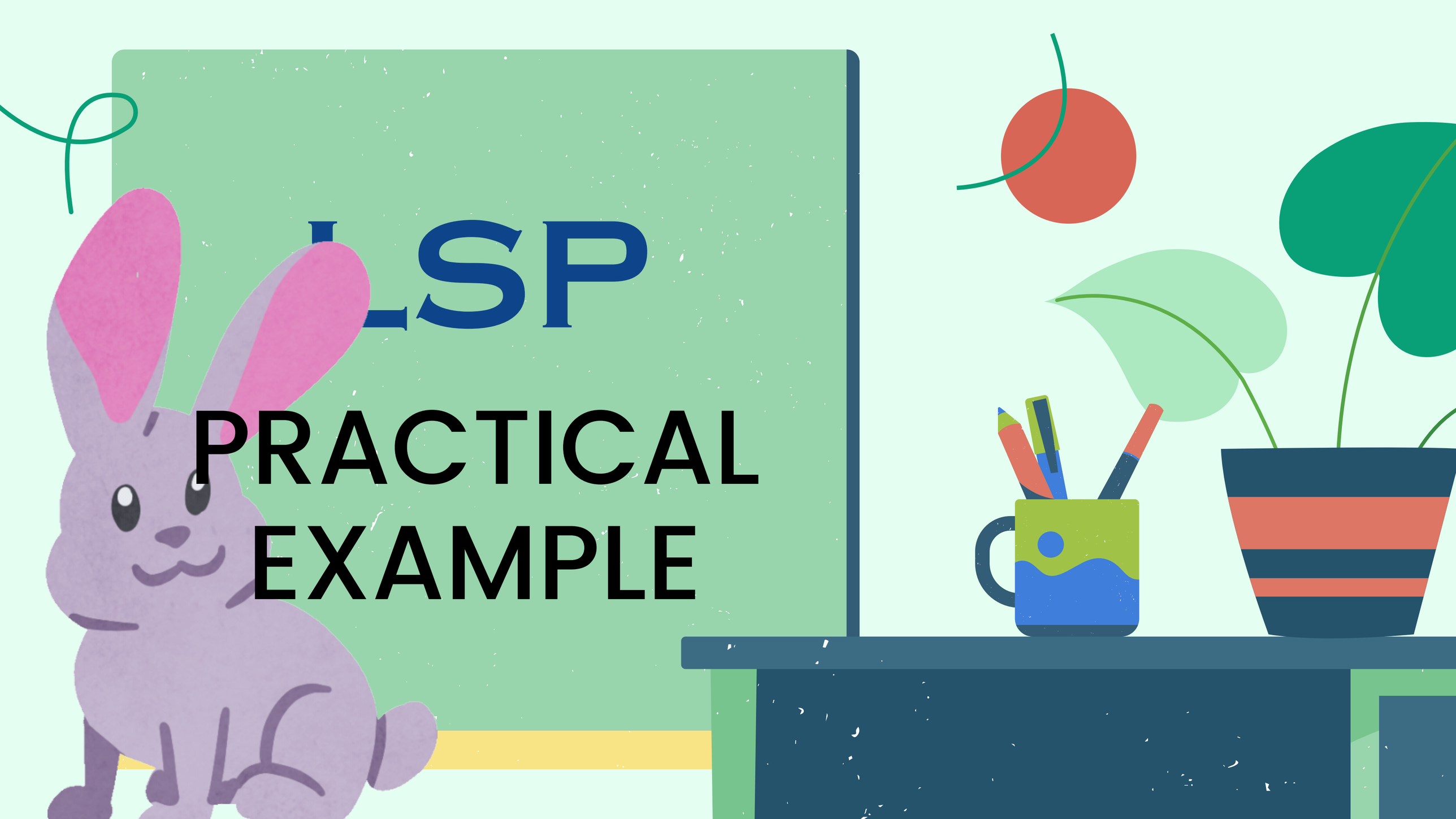
Liskov Substitution Principle (LSP)

- The LSP states that any subclass should be substitutable for its base class.
- This means that if B extends A, then we should be able to use B everywhere we use A without altering any functionality.
- Nowadays, all code should be written using hooks over classes anyway, and so classes should play an extremely minor role in modern React code.



LSP

PRACTICAL EXAMPLE



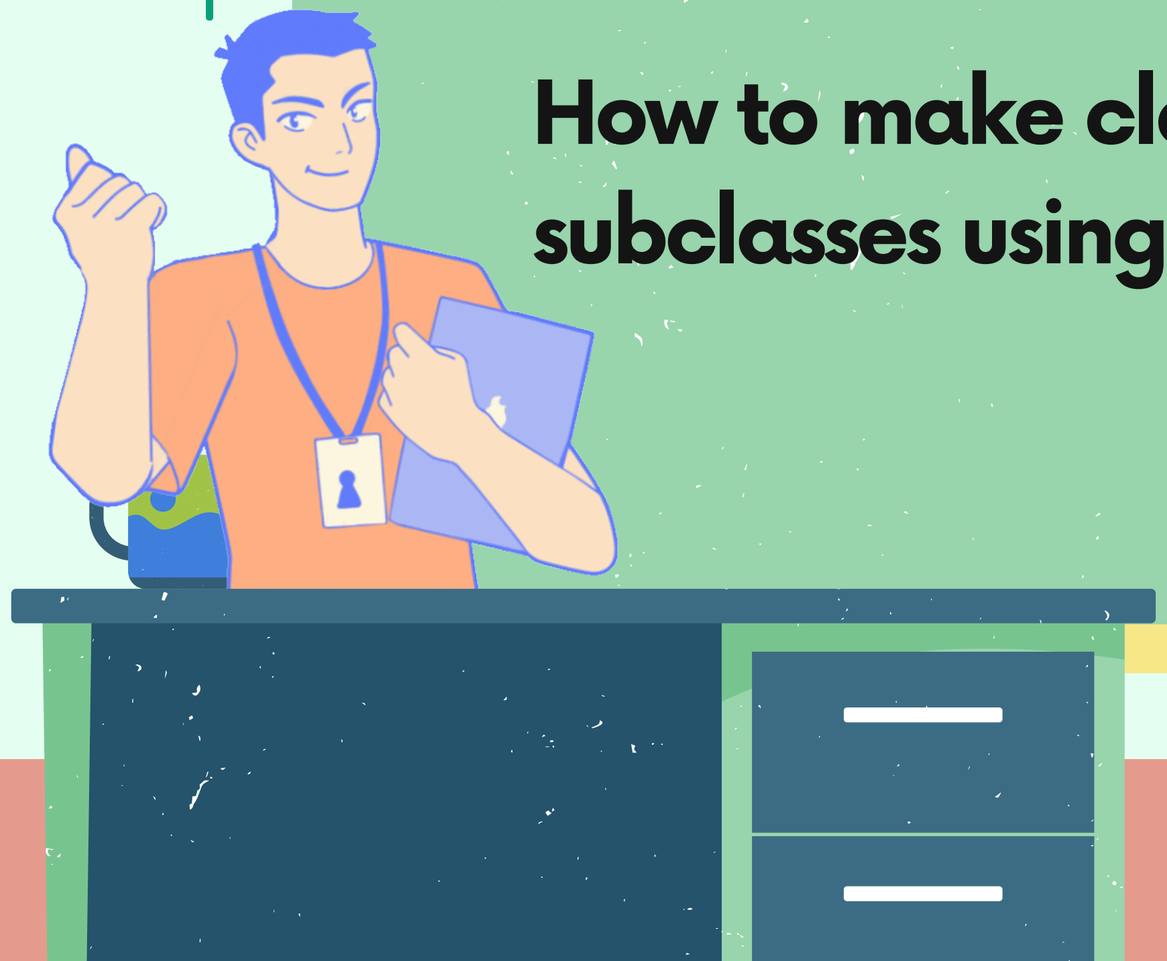
If you're extending from a class, make sure the base class is as generic as needed to be a true representation of every subclass.

Extend: When we extend something, we will inherit all it‘s properties and methods.

SUMMARY

IN THIS LESSON, WE HAVE LEARNED:

How to make classes substitutable for subclasses using LSP Principle.



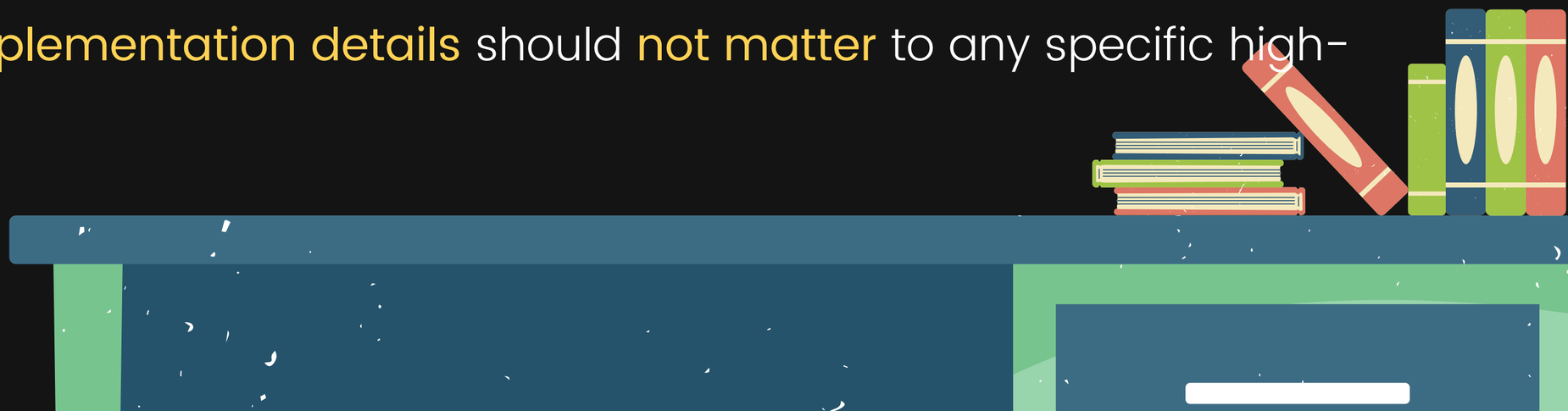
I

INTERFACE SEGREGATION PRINCIPLE (ISP)

Only pass a component props it needs.

Interface Segregation Principle (ISP)

- The ISP states that people should not be forced to rely upon interfaces that they don't use.
- Give components only what they need.
- This means implementation details should not matter to any specific high-level function.



An implementation detail is some specific way a task is accomplished.

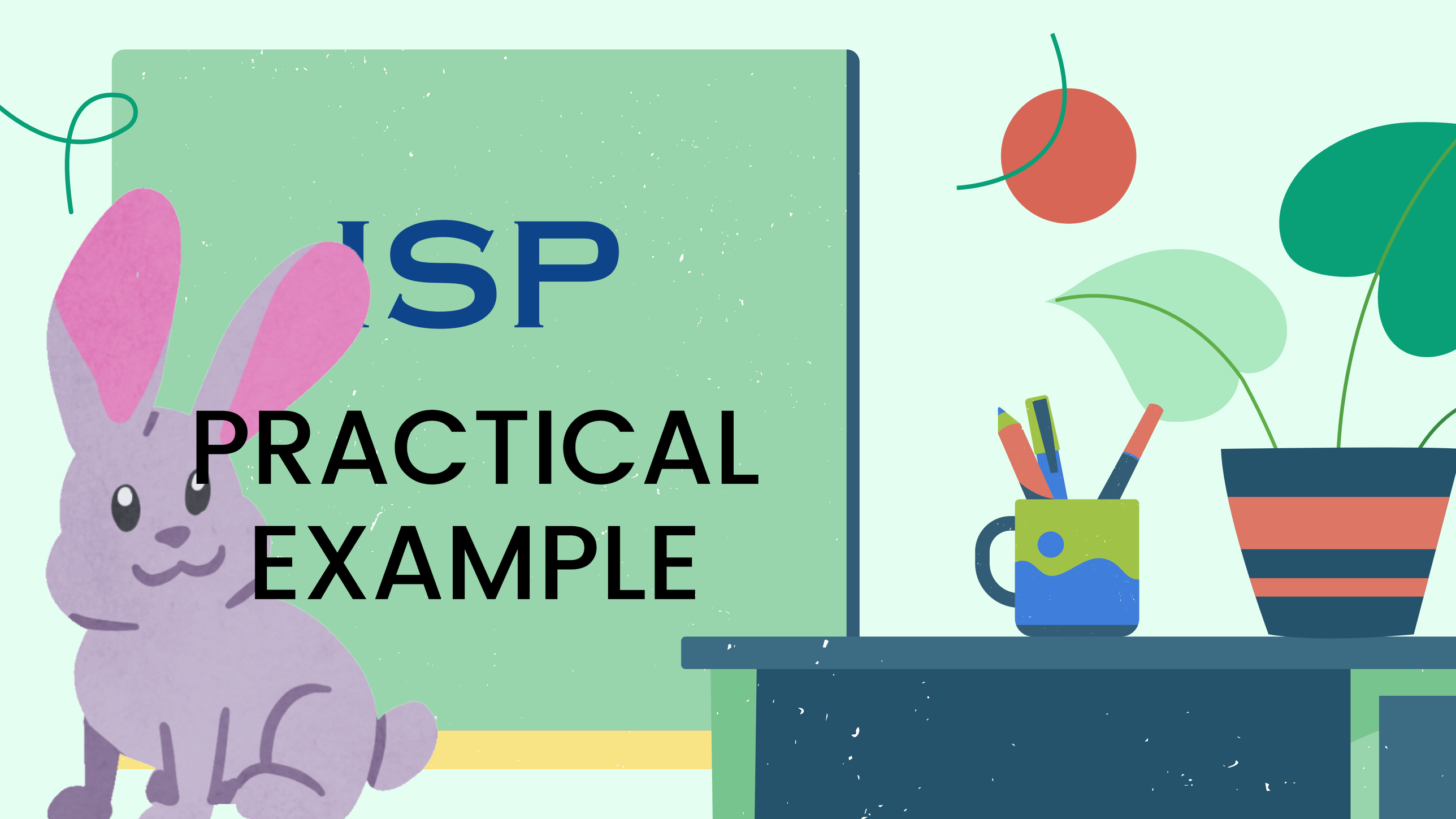
For instance, "getting todos from the API" is a task we can accomplish by using the axios library (an implementation detail).

Another task might be "storing users in a database".

Our choice of database is an implementation detail for this task.

JSP

PRACTICAL EXAMPLE



Destructure out the needed props for a component if possible.

This way, the component does not rely on the details in its parent component.

SUMMARY

IN THIS LESSON, WE HAVE LEARNED:

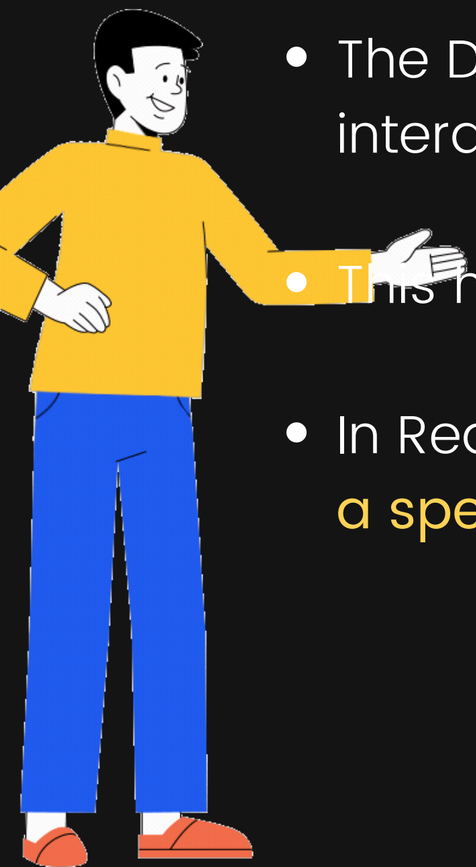
How to destructure and use the only needed props instead of passing the whole object using ISP Principle.

D

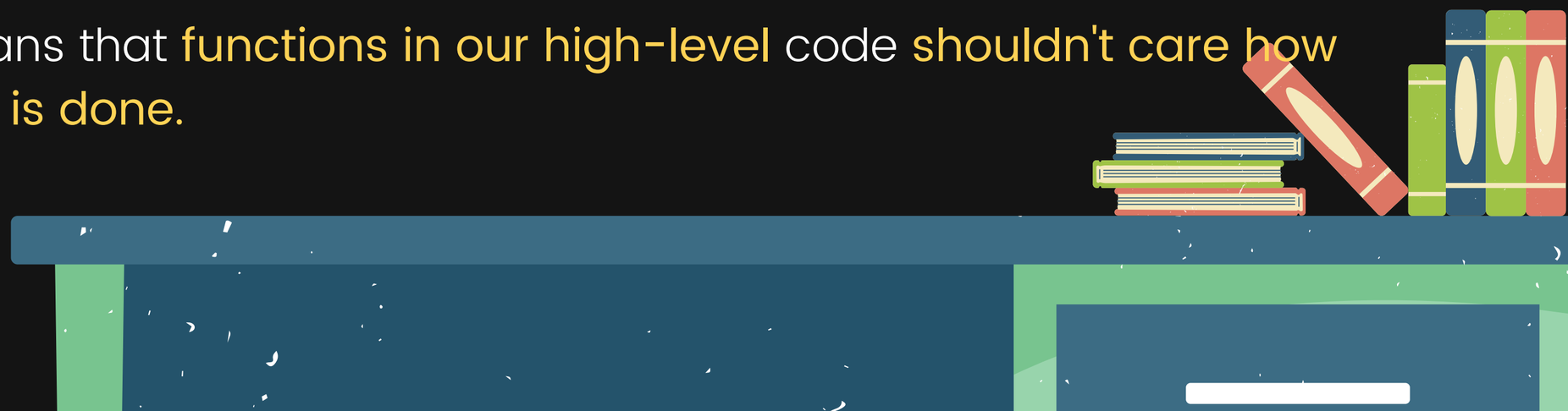
DEPENDENCY INVERSION PRINCIPLE (DIP)

Always have high-level code interface with an abstraction, rather than an implementation detail.

Dependency Inversion Principle (DIP)

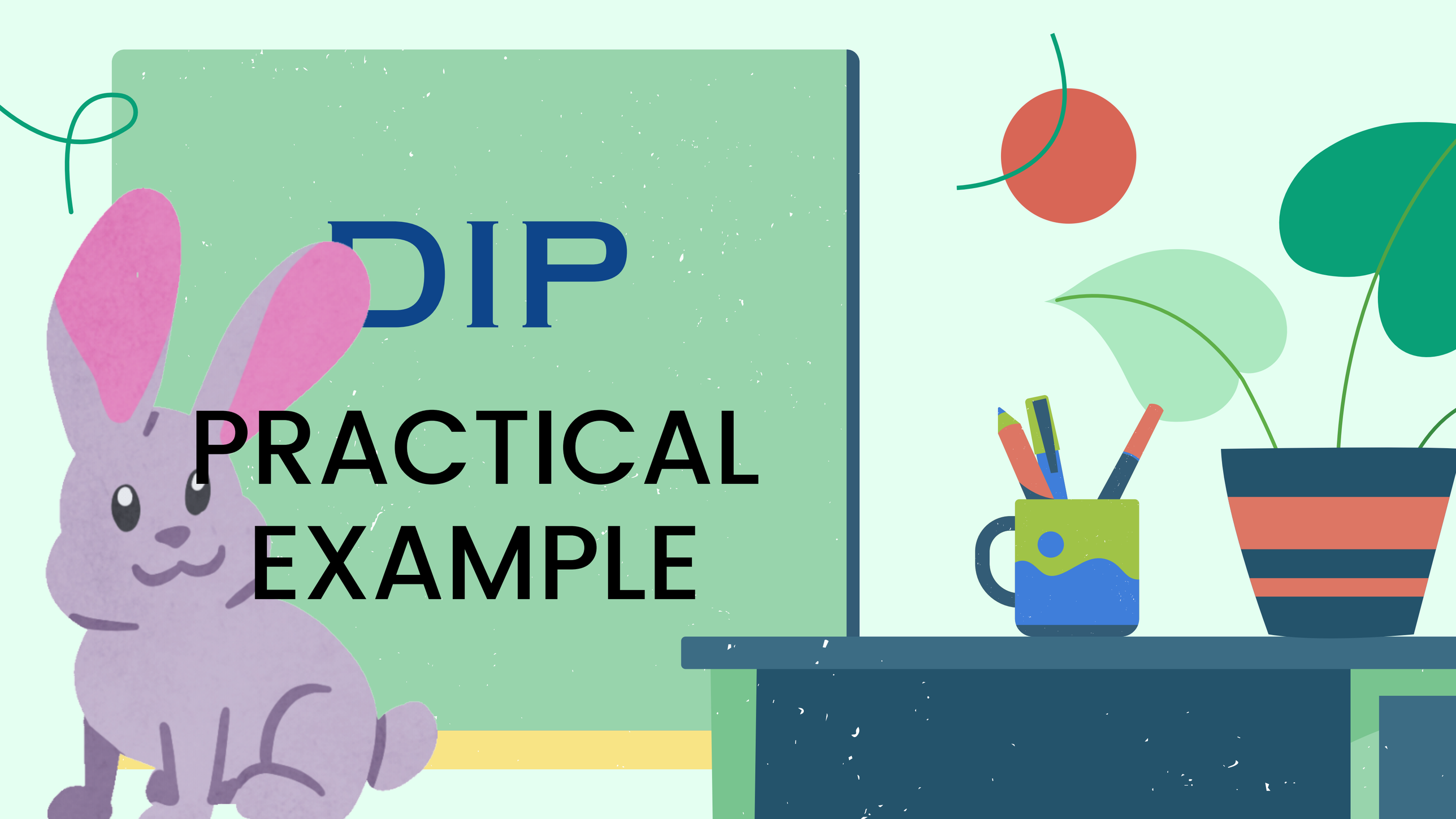


- The DIP tells us that we should "hide the wiring behind the wall" by always interacting with low-level details via abstractions.
- This has strong ties to the SRP and the ISP detailed above.
- In React, It means that functions in our high-level code shouldn't care how a specific task is done.



DIP

PRACTICAL EXAMPLE



Does <TodosPage /> care how or where those todos come from?

NO!


We changed an implementation detail (how we did a specific task), however we didn't have to alter anything in `<TodosPage />`.

These are guidelines; use them to the extent that they have a positive impact on your code.

SUMMARY

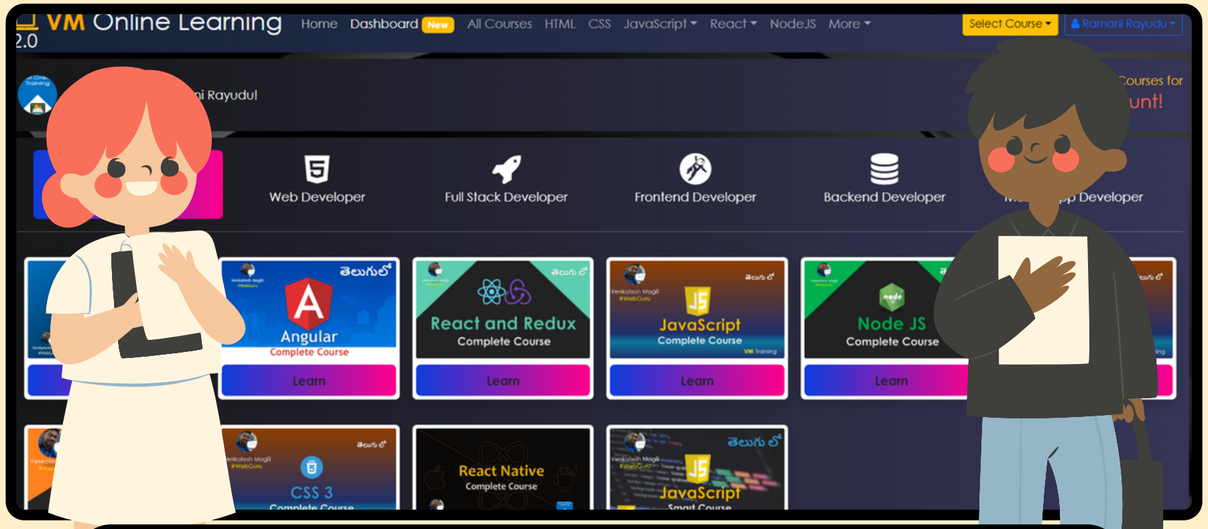
IN THIS LESSON, WE HAVE LEARNED:

1. DIP is combination of SRP and ISP Principles.
2. We can able to change the implementation details whatever extent that we want without changing the high level code.
3. It's not mandatory to use these principles everywhere, We can use them based on our needs.

QUICK RECAP

IN THIS VIDEO, WE HAVE LEARNED:

1. What are SOLID Principles with definitions.
2. How they used in practical projects in each specific scenario.
3. We have learned the real time examples with React code.



VM ONLINE TRAINING

https://vmtraining.netlify.app

Happy makara Sankranti

Share

COMMENT BELOW

SUBSCRIBE


THANK
YOU
FOR
WATCHING

VENKATESH MOGILI

SUBSCRIBE