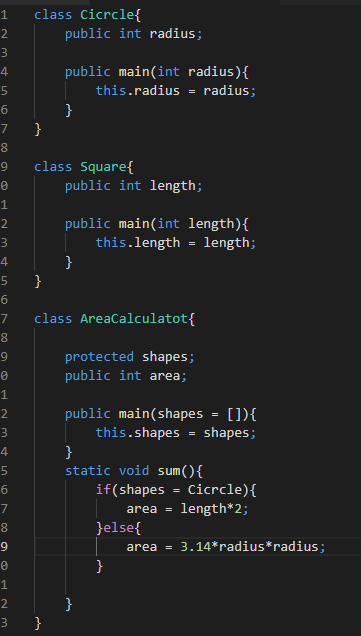
**Basics of S.O.L.I.D Principles, Javascript, Git, and NoSQL**

what are the S.O.L.I.D principles? the SOLID principles are designed to help developers design robust, maintainable applications. we can see five key principles. The SOLID principles were developed by Robert C. Martin in 2000. The broad goal of the SOLID principles is to reduce dependencies so that engineers change one area of software without impacting others.

**What is the Single responsibility principle?**

In this principle A class should have one and only one reason to change, meaning that a class should have only one job. This principle states that if we have 2 reasons to change for a class, we have to split the functionality into two classes. Each class will handle only one responsibility and if in the future we need to make one change we are going to make it in the class which handles it. For example, consider an application that takes a collection of shapes (circles, squares) and calculates the sum of the area of all the shapes in the collection. Then we can create separate classes for the square and circle to get the length and radius. Also, we can create a separate class for area calculation.



**What is the open-close principle?**

The Open-Closed Principle (OCP) states that software entities (classes, modules, methods, etc.) should be open for extension, but closed for modification. A module will be said to be **open** if it is still available for extension. For example, it should be possible to add fields to the data structures it contains, or new elements to the set of functions it performs. A module will be said to be **closed** if it is available for use by other modules. This assumes that the module has been given a well-defined, stable description.

**What is the Liskov substitution principle?**

This means that every subclass or derived class should be substitutable for their base or parent class. Simply put, if class *A* is a subtype of class *B*, then we should be able to replace objects of *B*with objects of *A*(i.e., objects of type A may substitute objects of type B)without changing the behavior (correctness, functionality, etc.) of our program.

If your code follows Liskov’s Substitutional Principle, you will have the below benefits:

1) Code Reusability  
2) Easier Maintenance  
3) Reduced Coupling

**What is the interface segregation principle?**

The Interface Segregation Principle (ISP) states thata client should not be exposed to methods it doesn’t need. Declaring methods in an interface that the client doesn’t need pollutes the interface and leads to a “bulky” or “fat” interface.

**What is the dependency inversion principle?**

The general idea of this principle is as simple as it is important: High-level modules, which provide complex logic, should be easily reusable and unaffected by changes in low-level modules, which provide utility features. To achieve that, you need to introduce an abstraction that decouples the high-level and low-level modules from each other.

**JavaScript**

JavaScript programs run using a single thread. **JavaScript** is a lightweight, cross-platform, interpreted scripting language. It is well-known for the development of web pages, many non-browser environments also use it. Here are a few things that we can do with JavaScript:

* JavaScript was created in the first place for DOM manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.
* Functions in JS are objects. They may have properties and methods just like another object. They can be passed as arguments in other functions.
* Can handle date and time.
* Performs Form Validation although the forms are created using HTML.
* No compiler is needed.

**‘this’ keyword**

In JavaScript, the ‘this’ keyword refers to an object. Which object depends on how ‘this’ is being invoked (used or called). The ‘this’ keyword refers to different objects depending on how it is used:

Alone, ‘this’ refers to the global object. In a function, ‘this’ refers to the global object. In a function, in strict mode, ‘this’ is undefined.

**Strict notation**

Strict mode makes it easier to write “secure” JavaScript.Strict mode changes previously accepted “bad syntax” into real errors. As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable. In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties. In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

**Closure**

example:-

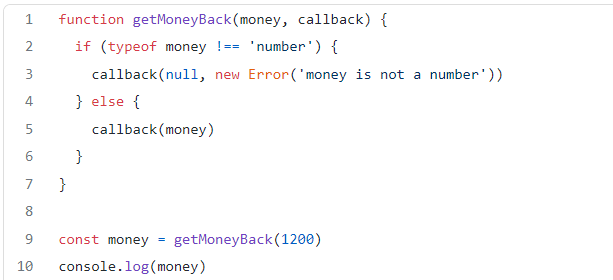
const add = (function () {  
let counter = 0;  
return function () {counter += 1; return counter}  
})();

add();  
add();  
add();

The variable ‘add’ is assigned to the return value of a self-invoking function. The self-invoking function only runs once. It sets the counter to zero (0) and returns a function expression. This way add becomes a function. The “wonderful” part is that it can access the counter in the parent scope. This is called a JavaScript **closure.** It makes it possible for a function to have “**private**” variables. The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

**Callback and promises**

A key difference between the two is when using the callback approach, we’d normally just pass a callback into a function that would then get called upon completion in order to get the result of something. In promises, however, you attach callbacks on the returned promise object.



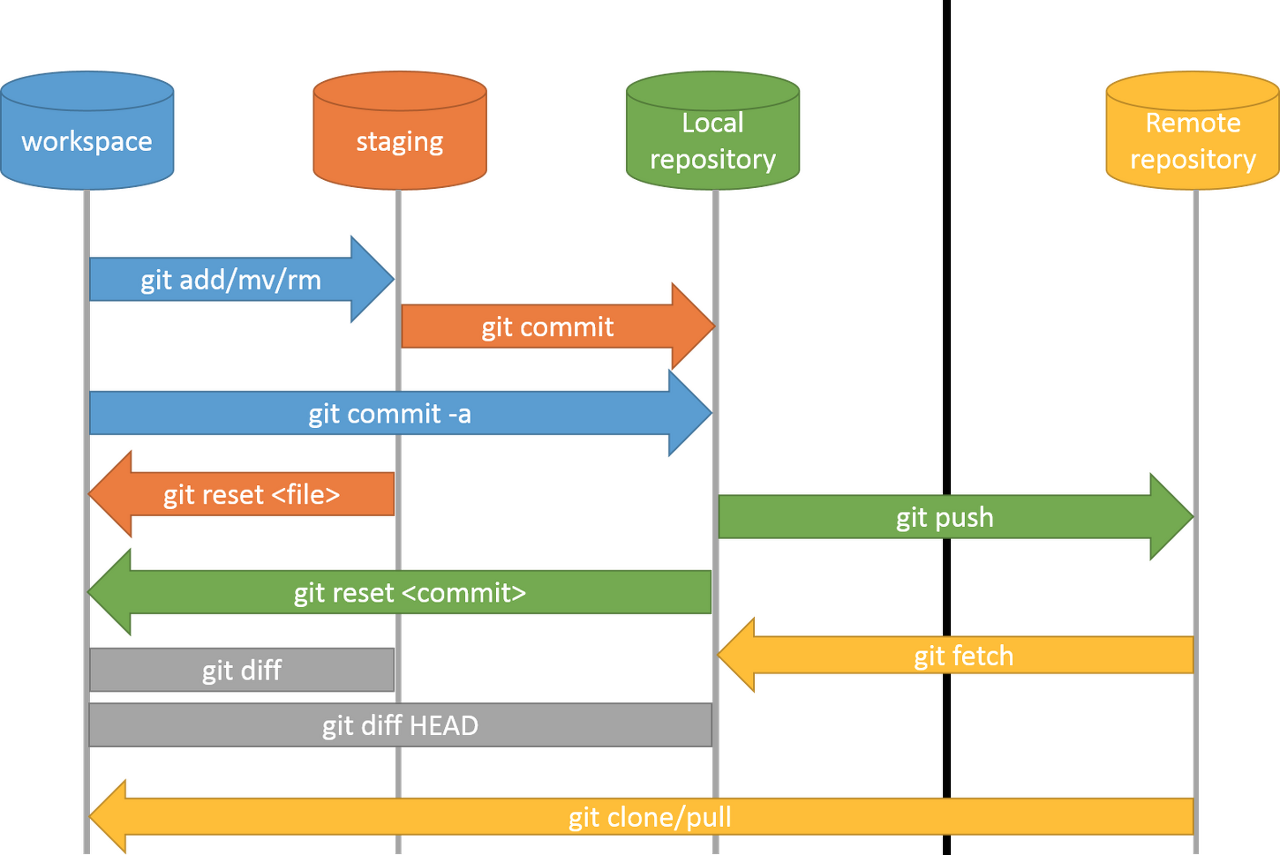
The promise constructor in JavaScript defines several static methods that can be used to retrieve one or more results from promises: *Promise.all , Promise.race, Promise.allSettled, Promise.any*.

**Git**

Git is the most commonly used version control system. Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. Git also makes collaboration easier, allowing changes by multiple people to all to be merged into one source.

**Git Repositories**

A Git **repository** (or **repo** for short) contains all of the project files and the entire revision history. You’ll take an ordinary folder of files (such as a website’s root folder), and tell Git to make it a repository. This creates a **.git**subfolder, which contains all of the Git metadata for tracking changes.



**Git clone:** Git clone is a command for downloading existing source code from a remote repository.

**Git branch:** Branches are highly important in the git world. By using branches, several developers are able to work in parallel on the same project simultaneously.

**Git status:** The Git status command gives us all the necessary information about the current branch.

**Git commit**: This is maybe the most-used command of Git. Once we reach a certain point in development, we want to save our changes.

**Git push:**After committing your changes, the next thing you want to do is send your changes to the remote server. Git push uploads your commits to the remote repository.

**NoSQL**

NoSQL databases are purpose-built for specific data models and have flexible schemas for building modern applications. NoSQL databases are widely recognized for their ease of development, functionality, and performance at scale.

Flexibility: NoSQL databases generally provide flexible schemas that enable faster and more iterative development. The flexible data model makes NoSQL databases ideal for semi-structured and unstructured data.

Scalability: NoSQL databases are generally designed to scale out by using distributed clusters of hardware instead of scaling up by adding expensive and robust servers. Some cloud providers handle these operations behind the scenes as a fully managed service.

examples for NoSQL databases: MongoDB, CouchDB, HBase, Riak, Cassandra.



https://medium.com/@samudithahasantha12/basics-of-s-o-l-i-d-principles-javascript-git-and-nosql-179667548472