Security in functional programming: Functional programming is better at preventing future exposures to bugs and is easier to test than the object-oriented way. While it’s easy to believe that your software is secured when it is first created, changes and time can introduce new vulnerabilities.

The way functional programming works makes it easier to prevent the introduction of new vulnerabilities in the particular section of the software once it’s written, reviewed, and approved for production. The code may need to change structurally, but the movement of code won’t break it in the same capacity or severity as introducing changes to an object or moving it.

Object-oriented works like a web and can grow into a monolithic piece of code if it is unmonitored. Functional programming, however, works in smaller pieces of code by design, with distinct reasons for existing.

It’s good to also note that functional programming and object-oriented programming need to be used exclusively from each other. Functional programming ideas can exist inside objects, increasing the robustness of the object itself. Object size management is out of scope for this piece but is something that you should be aware of when creating extensions. Functional programming can provide entryways into testing code and keep your objects in check, making it easier to read the code and debug when your objects get too big.

* Solid principle:
* The Single Responsibility Principle: The Single Responsibility Principle states that **a class should do one thing and therefore it should have only a single reason to change.**
* The Open-Closed Principle: The Open-Closed Principle requires that **classes should be open for extension and closed to modification.**
* The Liskov Substitution Principle: The Liskov Substitution Principle states that subclasses should be substitutable for their base classes.
* The Interface Segregation Principle: Segregation means keeping things separated, and the Interface Segregation Principle is about separating the interfaces.
* The Dependency Inversion Principle: The Dependency Inversion principle states that our classes should depend upon interfaces or abstract classes instead of concrete classes and functions..

Code injection: Code Injection is the general term for attack types which consist of injecting code that is then interpreted/executed by the application. This type of attack exploits poor handling of untrusted data. These types of attacks are usually made possible due to a lack of proper input/output data validation

Jupiterlab: JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning.