1. Software entities should be open for extension, but closed for modification.
2. It’s about **future-proofing your design**
3. Allow adding new behavior *without changing existing code*.
4. Main goal of OCP is to make code easy to extend without breaking existing features.
5. SRP and OCP works together for example :  
   **SRP** → Split report generation process into separate classes.  
   **OCP** → Define interface so you can add new formatter (PDF/Word) types without changing the report generation process.
6. **SRP is about *how you split responsibilities/classes***. For example: In report generator we split processes of fetching reports, processing/formatting reports, save reports and send emails in different classes/responsibilities.
7. **OCP is about *how you design those responsibilities/classes to evolve***.: For example the new added classed for report generator such as formatting reports or save reports inherit the interface which will help us in decide the type of formatting (pdf or wor) we want to perform and by this approach we doesn’t need to make changes in high order report generator function.   
   For real time example go to: <https://github.com/inspiredmonk/SolidPrinciples_SRP/blob/main/SRP/Program.cs>
8. **SRP *does not* say "never change.", It says "only change for *one reason*."**
9. **OCP doesn’t *strictly require* SRP—but in practice, we can’t really do OCP well without first applying SRP.** As both have different principles, but they support each other beautifully
10. **SRP** forces to split unrelated responsibilities and **OCP** Once we’ve split things, we can introduce abstractions to allow extension. Or we can treat **SRP** as foundation and **OCP** could be treated as the next level of flexibility.
11. In *good design*, **SRP almost always comes first**.
12. OCP is great when applied with thought, but it’s not free. Use it **where change is expected**, not everywhere. As it can leads to  
     Class explosion  
     Over-abstraction  
     More files to manage