# **S.O.L.I.D – The First Five Principles of OOP**

## Five OOP Principles

**S –** Single-Responsibility Principle

**O –** Open-Closed Principle

**L –** Liskov Substitution Principle

**I –** Interface Segregation Principle

**D –** Dependency Inversion Principle

## Single-Responsibility Principle

* Every function should only have a single reason to change. ***A function should only have a single job to do***

x = int(input('Input Length: '))

y = int(input('Input Width: '))

z = int(input('Input Height: '))

def calculateVolume(length, width, height):

volume = length \* width \* height

return 'Volume: ' + str(volume)

## Open-Closed Principle

* Objects should be open for extension, but closed for modification. ***A class should be extendable without modifying the class itself*** 
  + Makes your code easier to update, test, and understand.
    - *Imagine you have a box of crayons. You want to add new colors to your collection, but you don't want to mess up or change the crayons already inside the box. Instead, you just add new crayons* ***without modifying the old ones****.*
  + **WHY:** Makes code easier to update, test, and understand
* **Open** means you can add new things, like new features or behaviors, without any trouble.
* **Closed** means you shouldn't change the existing code to make those new features work.
* For example:
  + A drawing app supports circles and squares. If you want to add triangles, you **don't modify the circle and square code**. You just add a new triangle class that follows the same rules as the others.
  + This way, your old shapes still work, and your app now supports triangles too!

from abc import ABC, abstractmethod

# Define the ShapeInterface

class ShapeInterface(ABC):

@abstractmethod

def area(self):

pass

# Custom exception for invalid shapes

class AreaCalculatorInvalidShapeException(Exception):

pass

# **AreaCalculator** **class**

class AreaCalculator:

def \_\_init\_\_(self, shapes):

self.shapes = shapes

def sum(self):

total\_area = 0

for shape in self.shapes:

if isinstance(shape, ShapeInterface):

total\_area += shape.area()

else:

raise AreaCalculatorInvalidShapeException("Invalid shape provided.")

return total\_area

# Example Shapes

class Circle(ShapeInterface):

def \_\_init\_\_(self, radius):

self.radius = radius

def area(self):

return 3.14159 \* (self.radius \*\* 2)

class Rectangle(ShapeInterface):

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def area(self):

return self.width \* self.height

# Main code for testing

if \_\_name\_\_ == "\_\_main\_\_":

shapes = [

Circle(3),

Rectangle(4, 5)

]

calculator = AreaCalculator(shapes)

print("Total Area:", calculator.sum())

## Liskov Substitution Principle

*Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.* **Every subclass or derived class should be substitutable for their parent or base class.**

* + If you have a group of things (like classes or objects) that all follow the same rules (like a base class or interface), adding a new thing to the group shouldn’t break those rules.
* **WHY**: Makes programs easier to understand and less likely to have bugs.

## Integration Segregation Principle

Every function should

## Dependency Inversion Principle

Every function should

Resources

1. [DigitalOcean - SOLID: The First 5 Principles of Object Oriented Design](https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design)
2. [Code Sandbox](https://codesandbox.io/)