Design principles

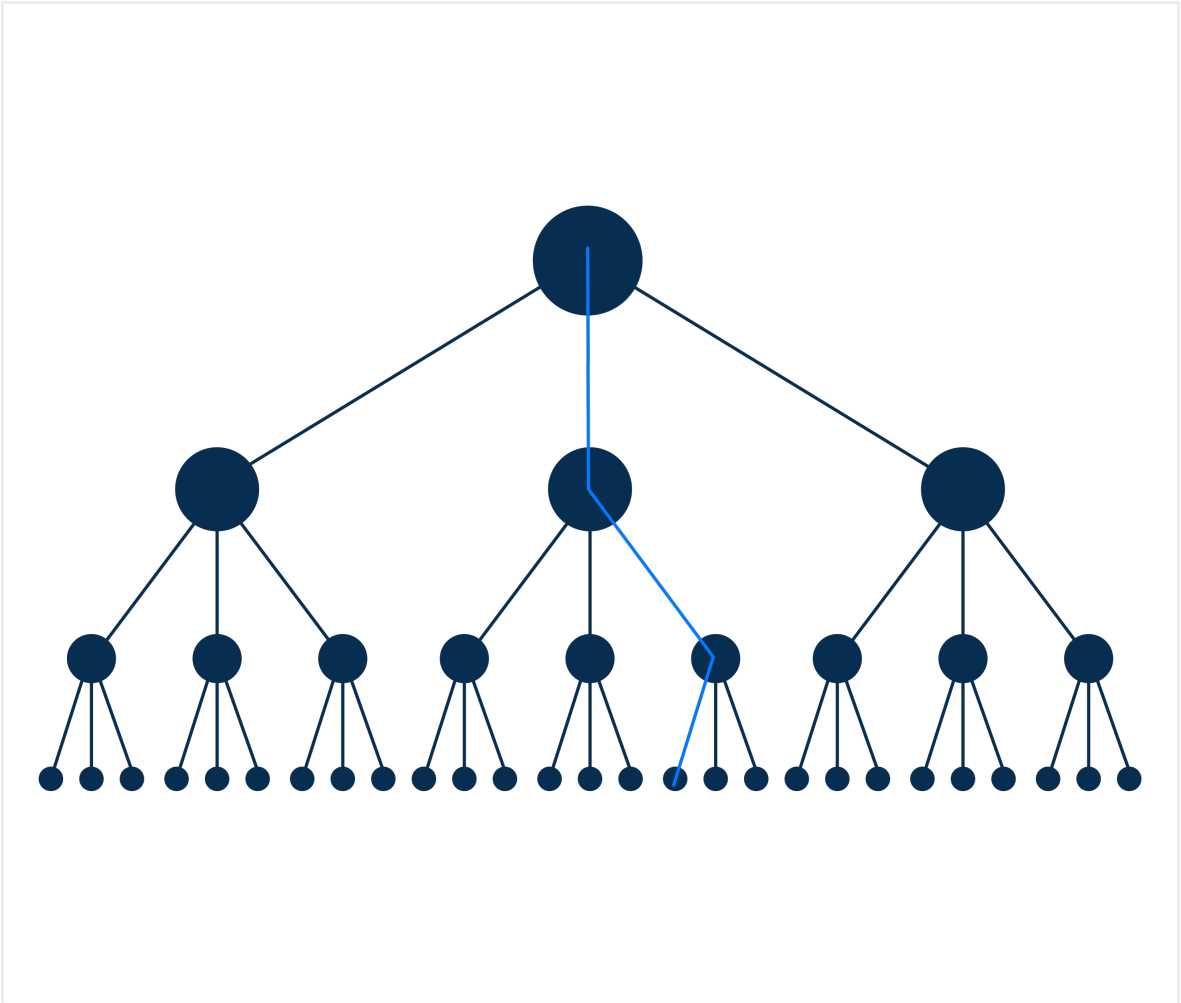
A program that passes all the tests might still have other problems:

- It can have a bad design, so no one uses it
- We cannot extend any part of it and add new features, because it's hard to understand how it works

Program design

All programs manipulate data, and although all programs are different, let's think of them as pipelines with data: a program receives some input, processes it, and produces some output.

The data goes from one function to another, and so on, and in the end, we get the result. It seems like we can control the data, but if there is an obstacle in our way, we are facing a complex problem. Moving only by straight paths doesn't save us from the growing complexity.



We keep the diversity of data paths but organize the code differently. We can say that we have another **design** for the program.

The design of a program is the way to organize the code structure to achieve its primary goals.

What we surely can do is follow guidelines or principles to make the design more effective and clear.

Design principles

Design principles are rules that guide you to better decisions for the design of your program. You can refer to them when you want to add or update any part of the code.

- Don't Repeat Yourself (**DRY**)
- You Ain't Gonna Need It (**YAGNI**)
- Keep It Simple, Stupid (**KISS**)

As you can guess, the *DRY* principle means, it's better to reuse code instead of copying it from one place to another. *YAGNI* is the principle of doing only the work that you need and not doing anything else. *KISS* stands for making the code simpler for better understanding.

Not all principles are easy to learn. Some of them, like **GRASP** (General Responsibility Assignment Software Patterns), are a whole ecosystem with many design patterns included and mostly adhere to object-oriented programming.

The other principles do not involve any additional knowledge, but it takes time to understand their meaning and significance. Let's look at the five software design principles hiding in the acronym — **SOLID.**

SOLID
There's no unanimous design principle to follow when designing your program.

Because as your program grows, it becomes more complex.

You will need to familiarise yourself with different approaches to help manage your program through different kinds of complexities. To make the path to our goals smooth, we can rely on the *SOLID* design principles.

Each letter in SOLID refers to a distinct design principle:

- **S**ingle Responsibility Principle (*SRP*)
- **O**pen-Closed Principle (*OCP*)
- **L**iskov Substitution Principle (*LSP*)
- **I**nterface Segregation Principle (*ISP*)

- **D**ependency Inversion Principle (*DIP*)

Each principle is independent of others, but applying them together works as a synergy for your design.

SOLID principles help you to organize your code in a way that any other developer familiar with these principles can use and extend it.

Testing Python Code 101 with PyTest and PyCharm