**What is Cron Job?**

A **cron job** is a Linux command used for scheduling tasks to be executed sometime in the future. This is normally used to schedule a job that is executed periodically – for example, to send out a notice every morning. Some scripts, such as Drupal and WHMCS may require you to set up cron jobs to perform certain functions.

For most cron jobs, there are three components present:

1. The script that is to be called or executed.
2. The command that executes the script on a reoccurring basis. This is typically set in [cPanel](https://www.hivelocity.net/blog/cpanel-what-you-need-to-know/).
3. The action or output of the script, which depends on what the script being called does. Frequently, scripts called as cron jobs modify files or databases. However, they can perform other tasks that do not modify data on the server as well, such as sending out email notifications.

**SOLID Principle in Programming: Understand With Real Life Examples**

In software development, Object-Oriented Design plays a crucial role when it comes to writing flexible, scalable, maintainable, and reusable code. There are so many benefits of using OOD but every developer should also have the knowledge of the SOLID principle for good object-oriented design in programming. The SOLID principle was introduced by Robert C. Martin, also known as Uncle Bob and it is a coding standard in programming. This principle is an acronym of the five principles which is given below…

* Single Responsibility Principle (SRP)
* Open/Closed Principle
* Liskov’s Substitution Principle (LSP)
* Interface Segregation Principle (ISP)
* Dependency Inversion Principle (DIP)

The SOLID principle helps in reducing tight coupling. Tight coupling means a group of classes are highly dependent on one another which you should avoid in your code. Opposite of tight coupling is loose coupling and your code is considered as a good code when it has loosely-coupled classes. Loosely coupled classes minimize changes in your code, helps in making code more reusable, maintainable, flexible and stable. Now let’s discuss one by one these principles…

**1. Single Responsibility Principle:**

This principle states that “a class should have only one reason to change” which means every class should have a single responsibility or single job or single purpose. Take the example of developing software. The task is divided into different members doing different things as front-end designers do design, the tester does testing and backend developer takes care of backend development part then we can say that everyone has a single job or responsibility.

Most of the time it happens that when programmers have to add features or new behavior they implement everything into the existing class which is completely wrong. It makes their code lengthy, complex and consumes time when later something needs to be modified. Use layers in your application and break God classes into smaller classes or modules.

**2. Open/Closed Principle:**

This principle states that “software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification” which means you should be able to extend a class behavior, without modifying it.

Suppose developer A needs to release an update for a library or framework and developer B wants some modification or add some feature on that then developer B is allowed to extend the existing class created by developer A but developer B is not supposed to modify the class directly. Using this principle separates the existing code from the modified code so it provides better stability, maintainability and minimizes changes as in your code.

**3. Liskov’s Substitution Principle:**

The principle was introduced by Barbara Liskov in 1987 and according to this principle “Derived or child classes must be substitutable for their base or parent classes“. This principle ensures that any class that is the child of a parent class should be usable in place of its parent without any unexpected behavior.

You can understand it in a way that a farmer’s son should inherit farming skills from his father and should be able to replace his father if needed. If the son wants to become a farmer then he can replace his father but if he wants to become a cricketer then definitely the son can’t replace his father even though they both belong to the same family hierarchy.

One of the classic examples of this principle is a rectangle having four sides. A rectangle’s height can be any value and width can be any value. A square is a rectangle with equal width and height. So we can say that we can extend the properties of the rectangle class into square class. In order to do that you need to swap the child (square) class with parent (rectangle) class to fit the definition of a square having four equal sides but a derived class does not affect the behavior of the parent class so if you will do that it will violate the Liskov Substitution Principle. Check the link Liskov Substitution Principle for better understanding.

**4. Interface Segregation Principle:**

This principle is the first principle that applies to Interfaces instead of classes in SOLID and it is similar to the single responsibility principle. It states that “do not force any client to implement an interface which is irrelevant to them“. Here your main goal is to focus on avoiding fat interface and give preference to many small client-specific interfaces. You should prefer many client interfaces rather than one general interface and each interface should have a specific responsibility.

Suppose if you enter a restaurant and you are pure vegetarian. The waiter in that restaurant gave you the menu card which includes vegetarian items, non-vegetarian items, drinks, and sweets. In this case, as a customer, you should have a menu card which includes only vegetarian items, not everything which you don’t eat in your food. Here the menu should be different for different types of customers. The common or general menu card for everyone can be divided into multiple cards instead of just one. Using this principle helps in reducing the side effects and frequency of required changes.

**5. Dependency Inversion Principle:**

Before we discuss this topic keep in mind that Dependency Inversion and Dependency Injection both are different concepts. Most of the people get confused about it and consider both are the same. Now two key points are here to keep in mind about this principle

* High-level modules/classes should not depend on low-level modules/classes. Both should depend upon abstractions.
* Abstractions should not depend upon details. Details should depend upon abstractions.

The above lines simply state that if a high module or class will be dependent more on low-level modules or class then your code would have tight coupling and if you will try to make a change in one class it can break another class which is risky at the production level. So always try to make classes loosely coupled as much as you can and you can achieve this through abstraction. The main motive of this principle is decoupling the dependencies so if class A changes the class B doesn’t need to care or know about the changes.

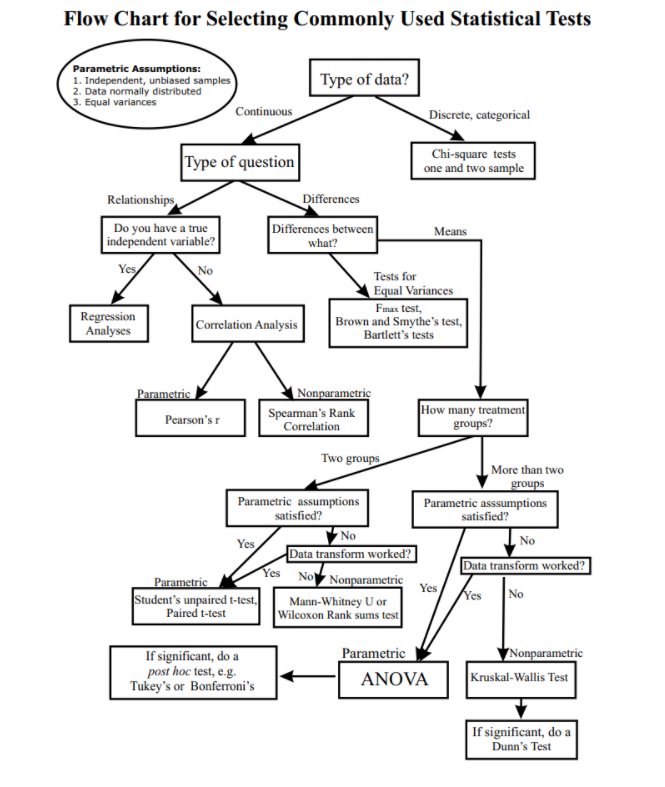
You can consider the real-life example of a TV remote battery. Your remote needs a battery but it’s not dependent on the battery brand. You can use any XYZ brand that you want and it will work. So we can say that the TV remote is loosely coupled with the brand name. Dependency Inversion makes your code more reusable.

**What Is the Central Limit Theorem (CLT)?**

In probability theory, the central limit theorem (CLT) states that the distribution of a sample variable approximates a normal distribution (i.e., a “bell curve”) as the sample size becomes larger, assuming that all samples are identical in size, and regardless of the population's actual distribution shape.

Put another way, CLT is a statistical premise that, given a sufficiently large sample size from a population with a finite level of variance, the mean of all sampled variables from the same population will be approximately equal to the mean of the whole population. Furthermore, these samples approximate a normal distribution, with their variances being approximately equal to the variance of the population as the sample size gets larger, according to the law of large numbers.

The central limit theorem is often used in conjunction with the law of large numbers, which states that the average of the sample means and standard deviations will come closer to equaling the population mean and standard deviation as the sample size grows, which is extremely useful in accurately predicting the characteristics of populations.



**What Is Hypothesis Testing?**

Hypothesis testing is an act in statistics whereby an analyst tests an assumption regarding a population parameter. The methodology employed by the analyst depends on the nature of the data used and the reason for the analysis.

Hypothesis testing is used to assess the plausibility of a hypothesis by using sample data. Such data may come from a larger population, or from a data-generating process. The word "population" will be used for both of these cases in the following descriptions.

**How Hypothesis Testing Works**

In hypothesis testing, an analyst tests a statistical sample, with the goal of providing evidence on the plausibility of the null hypothesis.

Statistical analysts test a hypothesis by measuring and examining a random sample of the population being analyzed. All analysts use a random population sample to test two different hypotheses: the null hypothesis and the alternative hypothesis.

The null hypothesis is usually a hypothesis of equality between population parameters; e.g., a null hypothesis may state that the population mean return is equal to zero. The alternative hypothesis is effectively the opposite of a null hypothesis (e.g., the population mean return is not equal to zero). Thus, they are mutually exclusive, and only one can be true. However, one of the two hypotheses will always be true.

**4 Steps of Hypothesis Testing**

All hypotheses are tested using a four-step process:

1. The first step is for the analyst to state the two hypotheses so that only one can be right.
2. The next step is to formulate an analysis plan, which outlines how the data will be evaluated.
3. The third step is to carry out the plan and physically analyze the sample data.
4. The fourth and final step is to analyze the results and either reject the null hypothesis, or state that the null hypothesis is plausible, given the data.

**Real-World Example of Hypothesis Testing**

If, for example, a person wants to test that a penny has exactly a 50% chance of landing on heads, the null hypothesis would be that 50% is correct, and the alternative hypothesis would be that 50% is not correct.

Mathematically, the null hypothesis would be represented as Ho: P = 0.5. The alternative hypothesis would be denoted as "Ha" and be identical to the null hypothesis, except with the equal sign struck-through, meaning that it does not equal 50%.

A random sample of 100 coin flips is taken, and the null hypothesis is then tested. If it is found that the 100 coin flips were distributed as 40 heads and 60 tails, the analyst would assume that a penny does not have a 50% chance of landing on heads and would reject the null hypothesis and accept the alternative hypothesis.

If, on the other hand, there were 48 heads and 52 tails, then it is plausible that the coin could be fair and still produce such a result. In cases such as this where the null hypothesis is "accepted," the analyst states that the difference between the expected results (50 heads and 50 tails) and the observed results (48 heads and 52 tails) is "explainable by chance alone."

**How Much Data Is Created Every Day in 2022?**

In 2020, people created 1.7 MB of data every second.

By 2022, 70% of the globe’s GDP will have undergone digitization.

In 2021, 68% of Instagram users view photos from brands.

By 2025, 200+ zettabytes of data will be in cloud storage around the globe.

In 2020, users sent around 500,000 Tweets per day.

By the end of 2020, 44 zettabytes will make up the entire digital universe.

Every day, 306.4 billion emails are sent, and 500 million Tweets are made.

**RESEARCH LABS**

Departmental research labs and groups are listed below in alphabetical order.

**Advancing Machine and Human Reasoning (AMHR) Lab**

AMHR is a cross-disciplinary lab dedicated to answering the following guiding research questions: How can artificial intelligence make people better reasoners? How can we create better artificially intelligent reasoners? And how can we advance our knowledge of logic and other cognitive-level reasoning processes in order to produce better conclusions, justifications, and arguments? We're devoted to not only creating smarter AI, but ensuring that these advances help improve, rather than replace, human reasoners. We believe that one day, we can do this with advances in computational, logical, rational, and justifiable reasoning / argumentation.

**ARGUS Cybersecurity Lab**

The focus of this lab is on the defense aspect of the cyber space, and the philosophy is to start from real problems, and create solutions that last. Research attempts to address the root causes of the various cybersecurity problems. The lab works closely with industry to ensure that its work both addresses the most pressing problems of the time, and provides the scientific basis for solutions that can stand the test of time.

**Biorobotics Lab**

Research at the USF BioRobotics Lab covers various areas of robotics and biologically-inspired systems. The lab is involved in various domestic and international research collaborations related to behavioral and neural modeling of spatial cognition in animals. Robot research focuses on autonomous mobile platforms, including single and multi-robot wheeled robots, humanoid robots, aerial and marine systems. The biorobotics lab is also home to the USF RoboBulls autonomous robot soccer team.

**Computing Education Research & Evolutionary Algorithms Lab (CEREAL)**

This lab gathers researchers from diversified disciplines who share an interest in; 1) identifying and understanding the learning barriers encountered by students of the computing disciplines and developing and evaluating innovative, technology-supported pedagogies to address them, 2) developing Evolutionary techniques that are able to tackle challenging application domains that require very significant adaptive capabilities (e.g., interactive or time-dependent optimization problems), and 3) applying our experience with the above to develop Evolutionary-Aided Teaching and Learning approaches (e.g., autonomous design of practice problems, automated discovery of concept inventories). As a result of these interests, our work spans both the Computing Education research and the Evolutionary Computation fields.

**Computer Vision and Pattern Recognition Group**

The Computer Vision and Pattern Recognition Group Invents Technologies Resulting in Commercial Products that Enhance the Security, Health and Quality of Life. The lab leverages USF's strengths in Video and Image Analysis Technology, Biometric Technology, Affective Computing, Classification and Knowledge Discovery, and Medical Data Analysis Technology to impact domestic security, quality of life, and healthcare.

**Relational vs. Non-Relational Database**

A relational database is structured, meaning the data is organized in tables. Many times, the data within these tables have relationships with one another, or dependencies. A non relational database is document-oriented, meaning, all information gets stored in more of a laundry list order. Within a single construct, or document, you will have all of your data listed out.

**SQL Databases (Relational)**

SQL is short for Structured Query Language, basically meaning a very firm way of sorting through data in the form of tables, columns, and rows. How is data structured in an SQL database? The table itself would be made up really of one variable or object that we would be looking through. The column would represent the data point itself that needs to be stored and the row is a record of the data points per column.

**NoSQL Databases (Non-Relational)**

In contrast to a relational database, a NoSQL database is one that is less structured/confined in format, and thus, allows for more flexibility and adaptability. If you are going to be dealing with a dataset that isn’t clearly defined, meaning not organized or structured, you likely won’t have the luxury of establishing defined tables and relationships amongst the dataset.

**List of Data Science Tools**

* SAS
* Apache Hadoop
* Tableau
* TensorFlow
* BigML
* Knime
* RapidMiner
* Excel
* Apache Flink
* PowerBI
* DataRobot
* Apache Spark
* Sap Hana
* MongoDB
* Python
* Trifacta
* Minitab
* R
* Apache Kafka
* QlikView
* MicroStrategy
* Google Analytics
* Julia
* SPSS
* MATLAB

**What is Search?**

Search is a utility that enables its user to find documents, files, media, or any other type of data held inside a database. Search works on the simple principle of matching the criteria with the records and displaying it to the user. In this way, the most basic search function works.

**What is Binary Search?**

A binary search is an advanced type of search algorithm that finds and fetches data from a sorted list of items. Its core working principle involves dividing the data in the list to half until the required value is located and displayed to the user in the search result. Binary search is commonly known as a half-interval search or a logarithmic search.

**How Binary Search Works?**

The binary search works in the following manner:

* The search process initiates by locating the middle element of the sorted array of data
* After that, the key value is compared with the element
* If the key value is smaller than the middle element, then searches analyses the upper values to the middle element for comparison and matching
* In case the key value is greater than the middle element then searches analyses the lower values to the middle element for comparison and matching

## **Example Binary Search**

The above image illustrates the following:

1. You have an array of 10 digits, and the element 59 needs to be found.
2. All the elements are marked with the index from 0 – 9. Now, the middle of the array is calculated. To do so, you take the left and rightmost values of the index and divide them by 2. The result is 4.5, but we take the floor value. Hence the middle is 4.
3. The algorithm drops all the elements from the middle (4) to the lowest bound because 59 is greater than 24, and now the array is left with 5 elements only.
4. Now, 59 is greater than 45 and less than 63. The middle is 7. Hence the right index value becomes middle – 1, which equals 6, and the left index value remains the same as before, which is 5.
5. At this point, you know that 59 comes after 45. Hence, the left index, which is 5, becomes mid as well.
6. These iterations continue until the array is reduced to only one element, or the item to be found becomes the middle of the array.

**The 7 V’s of Big Data**

**Volume**

Volume is how much data we have – what used to be measured in Gigabytes is now measured in Zettabytes (ZB) or even Yottabytes (YB). The IoT (Internet of Things) is creating exponential growth in data. The volume of data is projected to change significantly in the coming years.

**Velocity**

Velocity is the speed in which data is process and becomes accessible. I remember the days of nightly batches, now if it’s not real-time it’s usually not fast enough.

**Variety**

Variety describes one of the biggest challenges of big data. It can be unstructured and it can include so many different types of data from XML to video to SMS. Organizing the data in a meaningful way is no simple task, especially when the data itself changes rapidly.

**Variability**

Variability is different from variety. A coffee shop may offer 6 different blends of coffee, but if you get the same blend every day and it tastes different every day, that is variability. The same is true of data, if the meaning is constantly changing it can have a huge impact on your data homogenization.

**Veracity**

Veracity is all about making sure the data is accurate, which requires processes to keep the bad data from accumulating in your systems. The simplest example is contacts that enter your marketing automation system with false names and inaccurate contact information. How many times have you seen Mickey Mouse in your database? It’s the classic “garbage in, garbage out” challenge.

**Visualization**

Visualization is critical in today’s world. Using charts and graphs to visualize large amounts of complex data is much more effective in conveying meaning than spreadsheets and reports chock-full of numbers and formulas.

**Value**

Value is the end game. After addressing volume, velocity, variety, variability, veracity, and visualization – which takes a lot of time, effort and resources – you want to be sure your organization is getting value from the data.

# Design Patterns: Elements of Reusable Object-Oriented Software 1st Edition

