TECHNOLOGY

Coding Bootcamp

simplilearn

Angular

TECHNOLOGY

Getting Started with Angular

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Work with TypeScript to develop code in Angular
- Develop a comprehensive understanding of fundamentals of Angular
- Work with Angular CLI commands to execute various Angular operations

TECHNOLOGY

TypeScript

TypeScript

TypeScript is a strongly typed programming language that builds on JavaScript, giving a better tooling at any scale.

● It is a specific, object-oriented, and compiled language.

● It is both a programming and a scripting language and is a superset of JavaScript.

Features of TypeScript

It provides suggestion and code correction during development like a programming language.

Its code is transpiled into its JavaScript equivalent when executed. It reuses the JavaScript systems, tools, and libraries.

Any valid file with a .js extension can be changed to a .ts extension.

It is used across programs, gadgets, and working frameworks.

It does not need a dedicated runtime environment to compile unlike other programming languages such as Java.

Advantages of TypeScript

Strict typing: Everything is defined the way it is intended to be used.

Structural typing: It has a structural flow and a defined way of coding making it easier to understand.

Fast refactoring: A developer can easily change the code efficiently using features like rename symbol, extract methods, and many more.

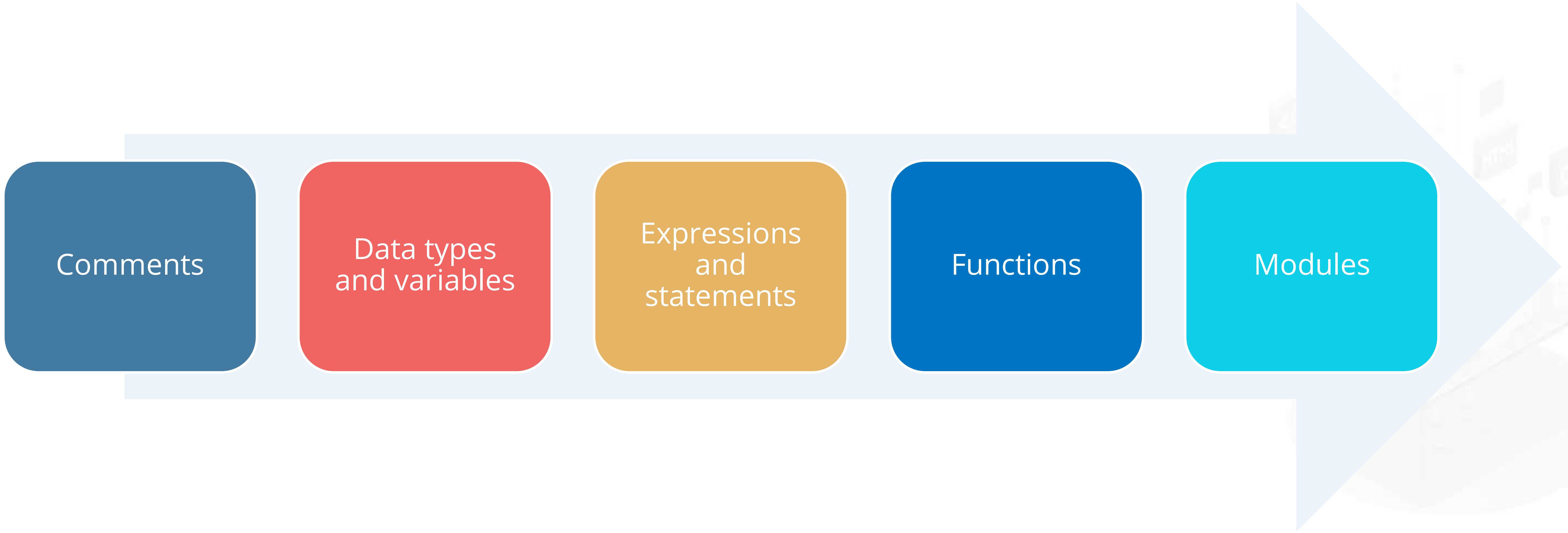
Compile-time check: Many bugs in TypeScript's code can be spotted at compile time preventing it from making any blunders during run-time.

TECHNOLOGY

TypeScript Program

TypeScript Program

A TypeScript program consists of:



Compiling and Executing a TypeScript Program

The steps to compile and execute a TypeScript program are:

Start by saving the file with a TypeScript extension

Open the saved file with the .ts extension in the command prompt

Use this command to compile the program:
tsc file_name.ts

Use this command to run the program:
Node file_name.js

Use this command to compile multiple files at once: **tsc file_1.ts file_2.ts**

Comments

Comments label specific parts of the code to be readable and easily understood by other developers.

```
                         Single line comment:
// This is a single line comment.

                         Multi-line comment:
/*
This is a multi-line comment.
There is no limit to the number of line that can be put inside a comment
block.
                              */
```

Data Types and Variable

Data types and variables work together. A variable is a name space in system memory that stores values, and data types describe the type of value stored in a variable.

- A variable's value can change. Thus, a variable acts as a container to store a value.
- A developer can define a variable in four ways as shown in the example:

1) **var a=10:** It will take the data type of the value automatically.

2) **let a=10:** It will take the data type of the value automatically.

3) **var a:number=10:** It will take the data type as a number.

4) **cons a=10:** It will take the data type of the value automatically. However, the value of the const variable cannot be changed once defined.

```
Option1:
      var a=10;
Option2:
      let a=10;
Option3:
  var a:number=10;
Option4:
     const a=10;
```

Data Types

The list of data types in TypeScript is:

Data Types	Uses	Keyword
Any	It can store any type of data in any kind of variable.	any
Number	It is used for integers, fractions, or any number.	number
String	It is used for sequences of Unicode or char.	string
Boolean	It is used for logical values such as true and false.	boolean
Void	It is used for function return types to represent non-returning functions.	void
Null	It is used to indicate the absence of value of an object.	Null
Undefined	It is used for all uninitialized variables.	undefined

var vs. let

var and **let** both are used to declare a variable but have several differences as shown below:

	var	let
1	Syntax: var name = value	Syntax: let name = value
2	It has function scope.	It has a block scope.
3	It can be declared again.	Once declared, it cannot be declared again in the same scope or block.
4	Hoisting is possible in **var.**	Hoisting is not allowed in **let.**
5	It is an ECMAScript1 feature.	It is an ES6 feature.
6	It is supported in all the versions of any browser.	It is supported in chrome49, Edge 12, firefox44, safari11, and opera36.

Naming a Variable

When naming a variable, several best practices should be considered. Ideally, JavaScript naming conventions are also followed in TypeScript.

Numeric digits and alphabets are allowed in the variable name.

White spaces and special characters, except an underscore _ , are not allowed.

Variable name must not start with a digit.

If a variable has multiple words in the name, camel casing should be followed. For example: **branchOwnerName**

Statements and Expressions

Statements and expressions are used in TypeScript to tell the program what to do and how to do it.

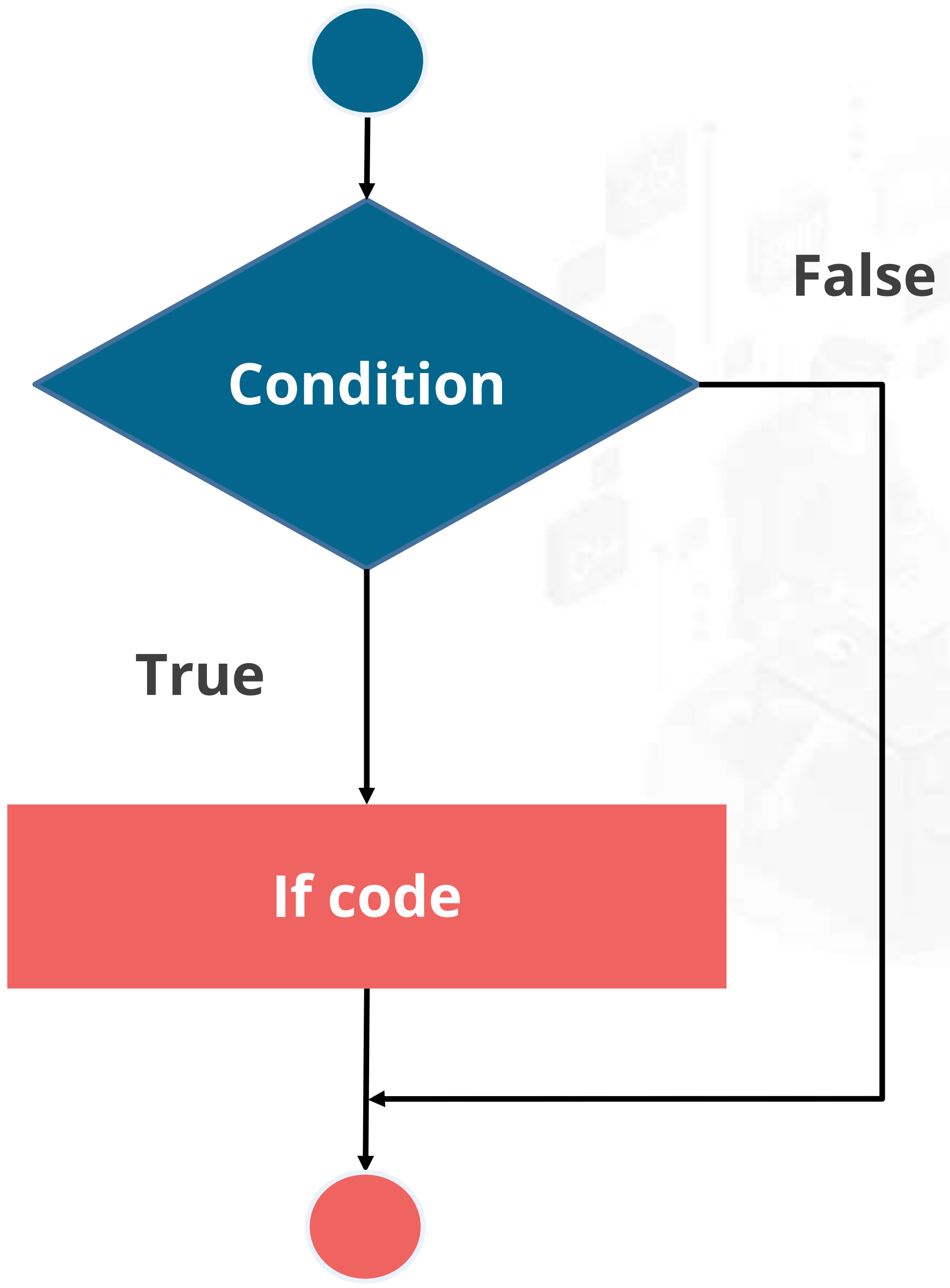
- Statements can be declared, and they explain what is the operation that is expected to be performed in the following block.
- Expressions provide a single-line code that can be part of a statement and used as unique code.

Conditional Statements

Conditional statements are used to execute a block of code based on the result of a Boolean condition. It decides whether the block code will be executed based on the condition specified.

Syntax

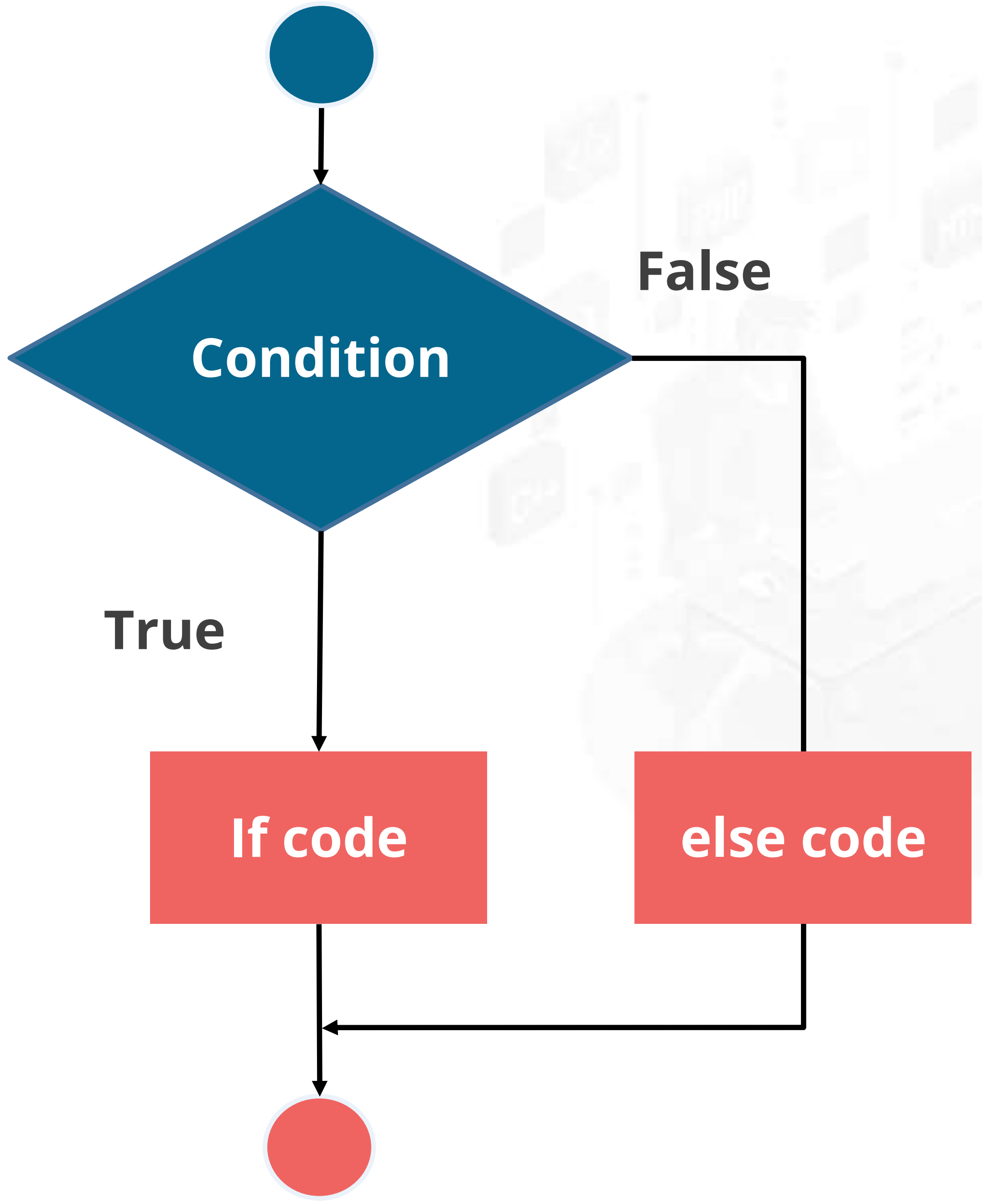
```
if(condition) {

/*
These expressions or code block will be
executed if the result of the condition
is true.
*/

}
```



The If-Else Statement

An **if-else** statement returns or executes a block of code. When the condition in the if block is false, the **else** block is executed automatically.

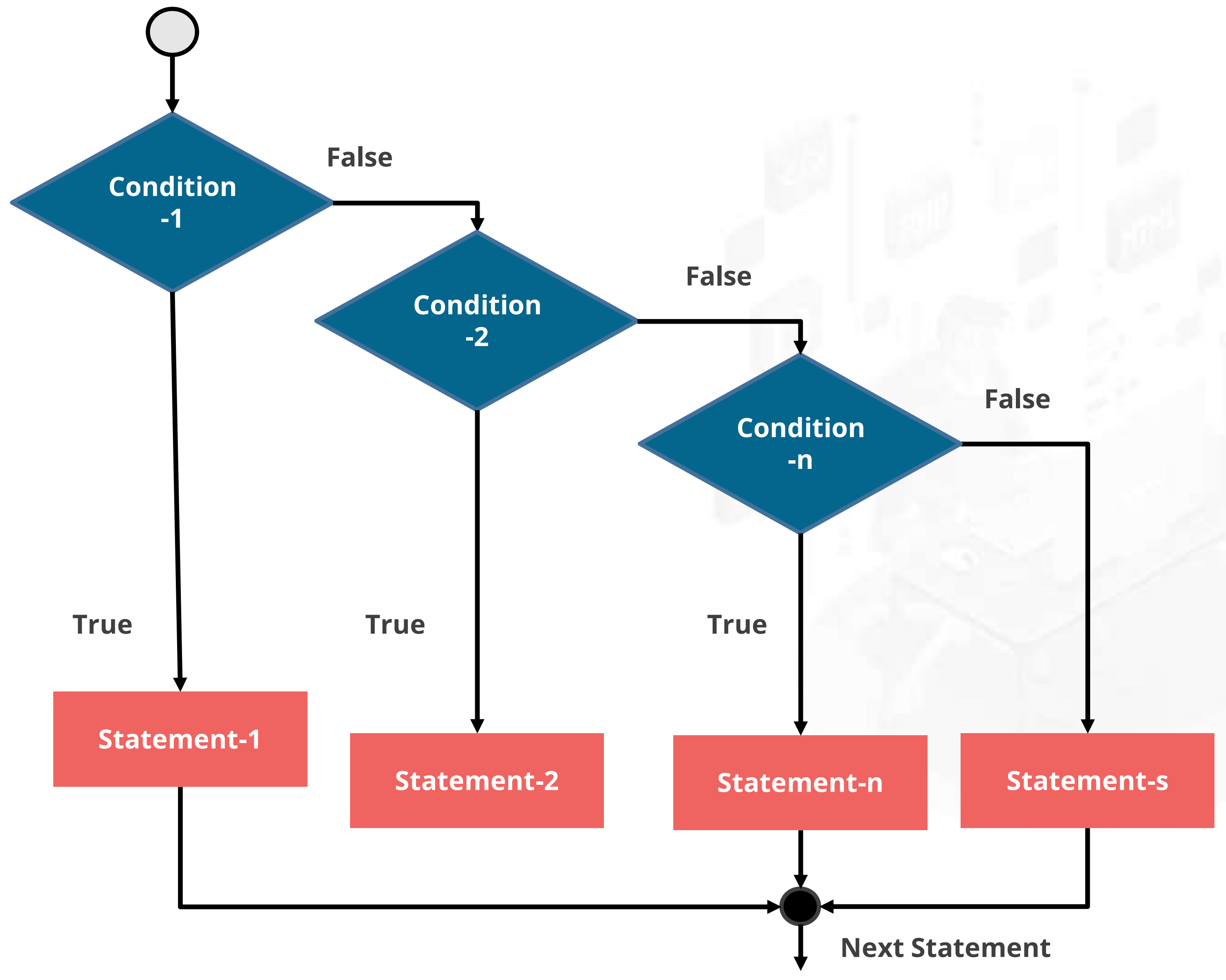
```
if(condition) {
  // code to be executed
} else {
   // code to be executed
}
```



The If-Else Ladder Statement

Users can choose from multiple options, and it runs on a top-down approach.

```
if(condition_1){
//statement to be executed if condition_1 is
true
}else if(condition_2){
//statement to be executed if condition_2 is
true
}
  else if(condition_3){
//statement to be executed if condition_3 is
true
}
else{
//statement to be executed if all the
conditions are false
}
```



Nested If Statement

In this statement, the **if** statement focuses on another if statement inside the parent **if** block.

```
if(condition1) {
    //Nested if else inside the body of
"if"
    if(condition2) {
       //Code inside the body of nested
"if"
    }
    else {
       //Code inside the body of nested
"else"
    }
}
else {
    //Code inside the body of "else."
}
```

Looping Statements

Loops in TypeScript provide constructs to handle repetitive tasks by executing code blocks multiple times.

Key points to remember:

- **Initialization:** Establishing a loop control variable (for example, let i = 0 in a for loop)
- **Condition:** A boolean expression evaluated before each iteration. The loop continues if the condition is true and stops if it is false.
- **Increment/Decrement:** Modifying the loop control variable (for example, i++ in a for loop)
- **Block of code:** The set of statements executed with each iteration of the loop

While Loop

It is used to emphasize a piece of the program over and over until the predetermined Boolean condition is valid.

Syntax:

```
Initializer;
while(condition) {
 increment/decrement;
}
```

Example:

```
let i = 0;
while (i < 5) {
  console.log(i);  // Outputs: 0, 1, 2, 3, 4
  i++;
}
```

Do-While Loop

The do-while looks at the condition toward the finish of the loop body.

Syntax:

```
initializer;
do {
 // statements to be executed
 increment/decrement;
} while(condition);
```

Example:

```
let i = 0;
do {
  console.log(i);  // Outputs: 0, 1, 2, 3, 4
  i++;
} while (i < 5);
```

For Loop

The four main parts of the **for loop** are:

- Initialization
- Condition
- Increment/Decrement
- Body of **for loop**

For Loop

Syntax:

```
for(initializer; condition;
increment/decrement) {
  // statements
}
```

Example:

```
for (let i = 0; i < 5; i++) {
  console.log(i);// Outputs: 0, 1, 2, 3, 4
}
```

Functions

Functions are the essential structure of any application in TypeScript.

Are used to assemble layers of abstraction

Make the code coherent, viable, and reusable

Are used to impersonate data hiding and modules

Functions in TypeScript

Functions are used to execute a set of code together. Here is the syntax of defining and calling a function in TypeScript:

```
function function_name( para1,para2..)
{
// Code to be executed
}

function_name();
```

Anonymous Function

A function is called an anonymous function when it is defined and called simultaneously. Below is an example of the same:

```
let obj = function (x,y) {
//the statements to be executed
};
```

It is used in scenarios where a small computation is to be done, which might not be used later in the application.

Mandatory and Optional Parameter

A user can define which parameters are mandatory and which are optional. The optional ones are not required to be passed when calling the same function.

```
//Mandatory
function test (name : string, age : number){
//statements to be executed
}

//Optional
function test (name : string, age?: number){
//Here the age parameter is optional
}
```

Modules

A module is used to create a group of related variables, classes, functions, interfaces, and more. It is used when a user wants to export a complete feature and import it into other feature files to access it anywhere.

Modules execute not in a global scope but in a local scope.

The module is created using the export keyword.

Advantages of Using Modules

Encapsulation: Modules help encapsulate code, reducing the risk of name conflicts.

Reusability: Code in modules can be easily reused across different parts of an application or in other projects.

Maintainability: Modules make it easier to manage and maintain large codebases by breaking down the application into smaller, manageable pieces.

Collaboration: Modules facilitate collaboration among multiple developers, as they can work on different modules without interfering with each other's code.

Working with Modules

Below is an example of exporting and importing a module in TypeScript:

```
// math.ts
export function multiply(x: number, y: number): number {
  return x * y;
}

export function divide(x: number, y: number): number {
  return x / y;
}

// app.ts
import { multiply, divide } from './math';

console.log(multiply(6, 3)); // 18
console.log(divide(6, 3)); // 2
```

TECHNOLOGY

Introduction to Angular

What Is Angular?

Angular is a platform and framework designed for creating single-page client applications with HTML and TypeScript.

It includes various packages and modules for developing scalable web apps and a set of developer tools for writing, building, testing, and updating code.

Why Angular?

These are the reasons why users should use the Angular framework:

- **Scalability**: It is suitable for building large-scale, complex applications due to its modular architecture and robust tooling.
- **Performance**: Its change detection and rendering mechanisms are optimized for high performance.
- **Community and ecosystem**: It has a large community and ecosystem with extensive documentation, tutorials, and third-party libraries.
- **Cross-platform development**: It supports building applications for different platforms, including web, mobile (using Angular with Ionic), and desktop (using Angular with Electron).

Features of Angular

These are some features that make Angular easy to use:

- Cross-platform
- Angular APIs
- MVC architecture

Features of Angular

These are some features that make Angular easy to use:

- Less code framework
- Directives
- Virtual scrolling

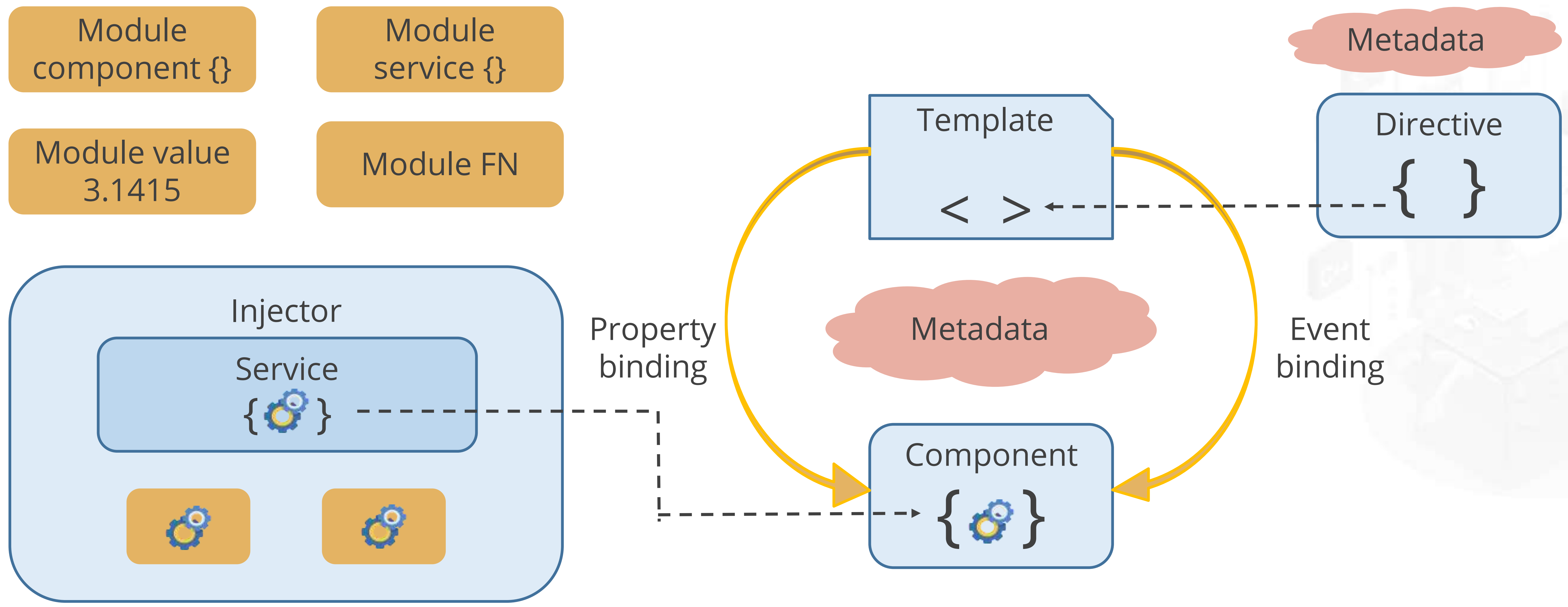
Features of Angular

These are some features that make Angular easy to use:

- Angular UI library
- Angular CLI
- Two-way data binding

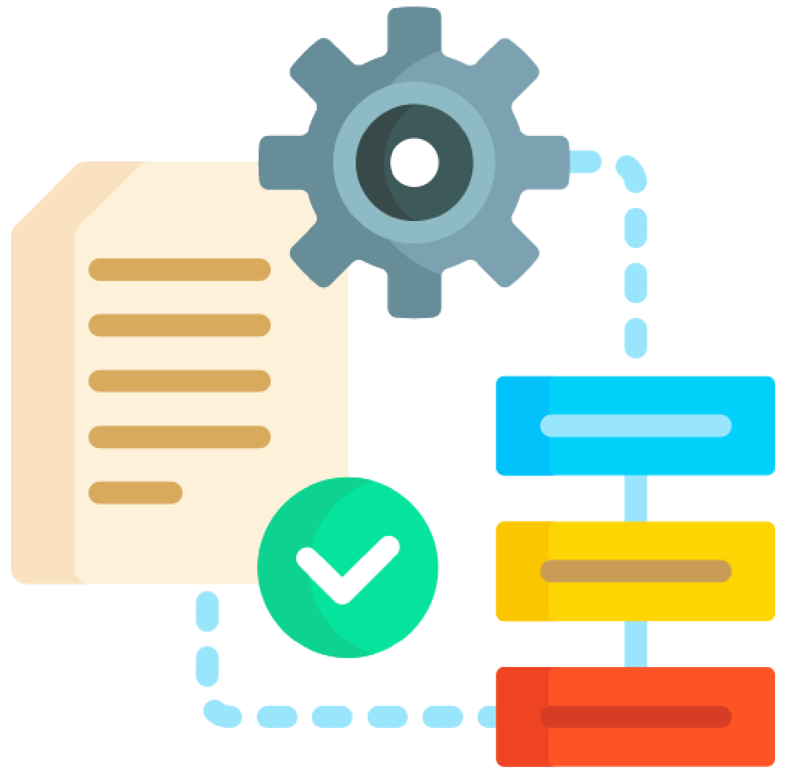
Architecture of Angular Apps

The architecture diagram of an angular application:



Development Environment Setup

It is important to have NodeJS already installed in the system before setting up Angular. The following are the steps to set up an Angular development environment:



- Install the Angular CLI
- Verify the version of Angular
- Create a new Angular app
- Launch Visual Studio Code
- Run the Angular app

Development Environment Setup

The following are the commands to set up Angular and execute a basic Angular application:

- Install the Angular CLI
- Verify the version of Angular
- Create a new Angular app
- Launch Visual Studio Code
- Run the Angular app

The Angular CLI is a command-line interface tool that users can use from a command shell to initialize, create, scaffold, and maintain Angular apps.

```
sudo npm install -g @angular/cli
```

Development Environment Setup

The following are the commands to set up Angular:

- Install the Angular CLI
- **Verify the version of Angular**
- Create a new Angular app
- Launch Visual Studio Code
- Run the Angular app

```
ng --version
```

Development Environment Setup

The following are the commands to set up Angular:

- Install the Angular CLI
- Verify the version of Angular
- Create a new Angular app
- Launch Visual Studio Code
- Run the Angular app

```
ng new app_name
```

Development Environment Setup

The following are the commands to set up Angular:

- Install the Angular CLI
- Verify the version of Angular
- Create a new Angular app
- Launch Visual Studio Code
- Run the Angular app

Open the Angular app on Visual Studio Code.

Development Environment Setup

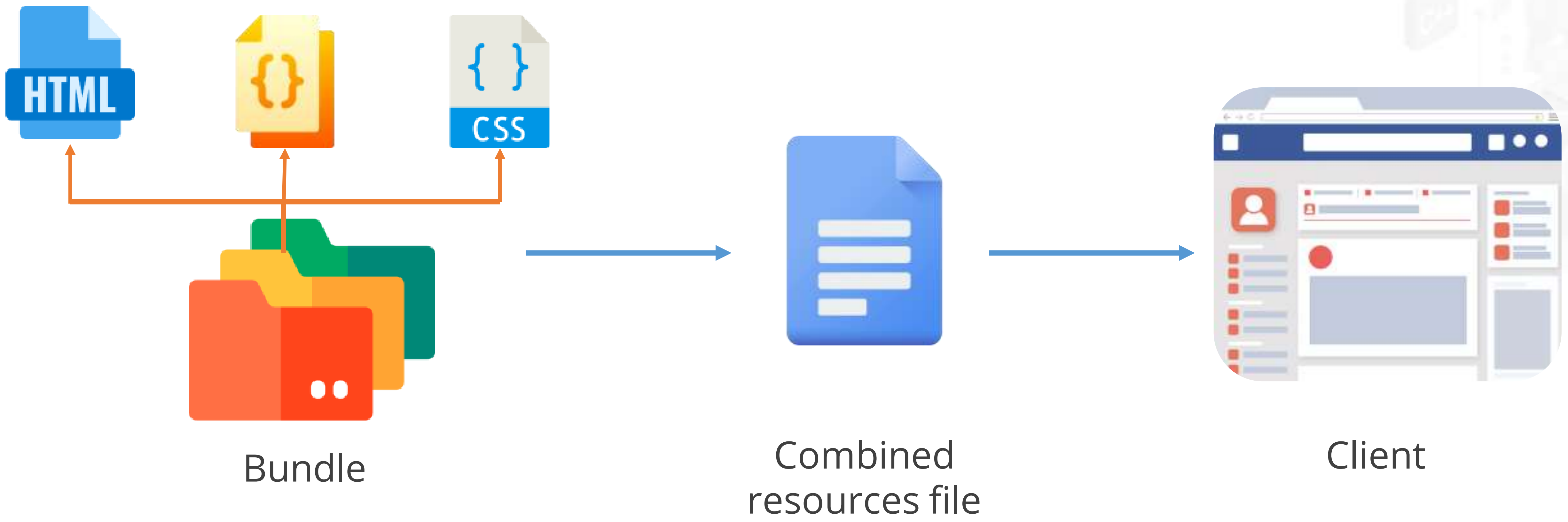
The following are the commands to set up Angular:

- Install the Angular CLI
- Verify the version of Angular
- Create a new Angular app
- Launch Visual Studio Code
- Run the Angular app

```
ng serve --open
```

What Is Webpack?

- Webpack is an effective module bundler.
- In Angular projects, Webpack operates behind the scenes through the Angular CLI (Command Line Interface). When you create and build an Angular project with the Angular CLI, Webpack automatically handles bundling and optimization tasks.
- HTML, CSS, JavaScript, and virtually any other file type can be included in a bundle. A Webpack is created when the application is compiled.



Building Blocks of Angular

The main building blocks of Angular are as follows:

- Modules
- Data binding
- Components
- Directives
- Templates
- Services
- Pipes
- Dependency injection

Creating the First Web App

Duration: 15 min.

Problem Statement:

You have been assigned a task to create an Angular application.

Outcome:

By following these steps, you will be able to successfully create your first Angular application along with the comprehensive knowledge on the project structure.

Note: Refer to the demo document for the detailed steps:
01_Create_the_First_Web_App

Assisted Practice: Guidelines

Steps to be followed:

1. Creation of an Angular project.

TECHNOLOGY

Working with Angular CLI

simplilearn

Angular CLI Syntax

Syntax:

```
ng [optional-arg] [options]
```

Example:

```
ng build my-app -c production
```

- There are aliases for most commands and some options.
- The syntax statement for each command displays aliases.
- The option names are prefixed with double dash (--) characters.
- The option aliases are prefixed with a single dash (-) character. Arguments are not prefixed.

Angular CLI Commands

Command	Alias	Description
add	-	It adds support for an external library to the users' projects.
analytics	-	It configures the gathering of Angular CLI usage metrics.
build	b	It compiles an Angular application or library into an output directory named **dist/** at the given output path.
cache	-	It creates a persistent disk cache and retrieves cache statistics.
completion	-	It sets up Angular CLI auto-completion for the user's terminal.
config	-	It retrieves Angular configuration values in the **angular.json** file for the workspace.

Angular CLI Commands

Command	Alias	Description
deploy	-	It invokes the deploy builder for a specified project.
doc	d	It opens the official Angular documentation (angular.io) in a browser and searches for a given keyword.
e2e	e	It builds and serves an Angular application, then runs end-to-end tests.
extract-i18n	-	It extracts **i18n** messages from the source code.
generate	g	It generates and/or modifies files based on a configuration.

Angular CLI Commands

Command	Alias	Description
lint	-	It runs linting tools on Angular application code in each project folder.
new	n	It creates a new Angular workspace.
run	-	It runs an Architect target with an optional custom builder configuration.
serve	s	It builds and serves users' applications, rebuilding on file changes.
test	t	It runs unit tests in a project.
update	-	It updates the users' workspace and dependencies.

Understanding Files in Angular

Following are the workspace configuration files:

Configuration files	Purpose
.editorconfig	It is a configuration file for code editors.
.gitignore	It specifies intentionally untracked files that Git should ignore.
README.md	It is an introductory document for the root application.
angular.json	It provides workspace-wide and project-specific configuration defaults.
package.json	It configures npm package dependencies that are available to all projects in the workspace.

Understanding Files in Angular

Following are the workspace configuration files:

Configuration files	Purpose
package-lock.json	It provides version information for all the packages installed into **node_module** by the npm client.
src/	It is a source file for the root-level application.
node_modules/	It provides npm packages to the entire workspace.
tsconfig.json	It specifies root files and the compiler options required to compile the project.

Understanding Files in Angular

Following are the application source files:

Application source files	Purpose
app/	It contains the component files in which users' application logic and date are defined.
assets/	It contains images and other asset files to be copied when users build their applications.
favicon.ico	It is an icon used for the application in the bookmark bar.
index.html	It is the main HTML page.
main.ts	It is the main entry point of the application.
styles.css	It is a list of CSS files that supply styles for a project

Understanding Files in Angular

Following are the application configuration files:

Application configuration files	Purpose
tsconfig.app.json	It is an application-specific TypeScript configuration, including TypeScript and Angular template compiler options.
tsconfig.spec.json	It is a TypeScript configuration for the application tests.

Key Takeaways

- Angular is a client-side TypeScript-based full stack web framework and a front-end development system.
- Angular CLI is used to initialize, develop, and maintain Angular applications directly from a command shell.
- The MVC Architecture is used to develop web applications and consists of the model, view, and controller.
- TypeScript is a superset of JavaScript; it is the recommended language for building Angular applications.

TECHNOLOGY

Thank You