**Module2: Typescript Basics & Basic Types:**

1. Understanding TypeScript OneDrive[**Folder**](https://pearsoneducationinc-my.sharepoint.com/:f:/g/personal/tony_lockhart_pearson_com/Eu1VEx97bqVOnx3iwTHpDQwBGg9KOyEw9itNFR0O6xnqzw)**:**
2. **Setup:**
   1. Open your IDE and clone, or pull the changes to the udemy-understanding -typescript GitHub repo:
      1. **Option1: Clone the repo:**
         1. cd into the udemy-understanding-typescript directory
         2. git clone [git@github.com:tlockhart/udemy-understanding-typescript.git](mailto:git@github.com:tlockhart/udemy-understanding-typescript.git)
      2. **Option2: Pull down updates**
         1. cd into the udemy-understanding-typescript directory
         2. Type git pull [Enter]
      3. **Setup the application:**
         1. Open a terminal
         2. cd into the project you wish to work in:
         3. **Install Node**
            1. [[Preferred] use nvm to install node](https://www.taniarascia.com/setting-up-a-brand-new-mac-for-development/#nodejs)
            2. [node.js.org](https://nodejs.org/en/)
         4. **Installing Typescript Globally:**
            1. npm install -g typescript
         5. **Install lite-server:**
            1. npm install --save-dev lite-server
            2. Add lite-server to start script in package.json:

"start": "lite-server"

* + - 1. **Install the app to build the node\_modules folder**
         1. Type npm install [Enter]
      2. **Compile the app.tsc file in watch mode:**
         1. tsc app.tsc [Enter]
      3. After compilation manually add .js files to index.html (Add all file dependencies first):
         1. <script src="app.js" defer></script>
      4. Open a different terminal
      5. **Start the lite-server**
         1. Type npm start [Enter]

1. **Core Syntax and Features:**
   1. **Core Types: Covered in Videos 10-19**

A picture containing timeline

Description automatically generated

1. **Type Basics & Basic Types:**
   1. **Visual Studio Code has Built-in TypeScript support**:
      1. Identifies type errors before compile time.
   2. **Explicit Type Declaration:**
      1. **Type annotations** specify the type of value a variable can contain.
      2. **Tip:** Always add a type annotation whenever declaring variables or function parameters.
      3. **Type Annotations Syntax:** Type annotations are declared on the right side of the variable. The variable and its type are separated by a colon. Note: Primitive type annotations (number, boolean, and string) are all lowercase.
         1. let myName**:** string = “Alice”;
   3. [Types of Language](https://www.educative.io/answers/statically-v-dynamically-v-strongly-v-weakly-typed-languages):
      1. **Strongly Typed** (Typescript) – Languages that do not allow conversion between unrelated types
      2. **Weakly (Loosely) Typed** (JavaScript) – Languages that do allow conversions between unrelated types.
   4. **The true benefit of TypeScript:**
      1. The IDE’s ability to identify type errors before compile time.
   5. **Type Inference**: Explicit type annotations are not necessary when a default value has been assigned.
      1. Note: Explicitly type declarations are only necessary for unassigned variables.
2. **Quiz 1: 5mins.**
3. **Video 16: Objects**
   1. Object Types vs Objects:
      1. **Object Types:** Describe the data in an object. They can be declared as a type and used in an object type declaration. They provide the structure of the object to the IDE.
         1. type Person = {

name: string;

age: number;

}

* + 1. **Objects**: Contain the actual data:
       1. const User1: Person = {

name: “Tony Lockhart”,

age: 100

}

1. **Video 18: Array Types:**
   1. Declare an array:
      1. **Syntax:** 
         1. keyword variableName: variableType[];
      2. **Example:**
         1. const hobbies: string[];
2. **Video 19: Tuples:**
   1. **Tuple**: A special array that indicates the number of elements and the associated types of those elements. Used to restrict the type of data to be stored.
   2. Declare Tuples:
      1. **Syntax:**
         1. keyword variableName: [elementType, elementType, …];
      2. **Example:**
         1. const role: [number, string];
3. **Video 20: Enum:**
   1. **Enum:** Global variable, used when you need to provide a value with a human readable label.
      1. Syntax:
         1. enum ObjectName = { PROP = value, PROP2 = value, ... }
      2. Example:
         1. enum Role { ADMIN = 'ADMIN', READ\_ONLY = 100}
4. **Video 21**: The “**any**” Type:
   1. **Any**: Disables type checking, by allowing you to store any type of value in a variable.
   2. NOTE: You should avoid using the “any” data type, if possible.
5. **Video 22**: **Union Types**:
   1. **Union**: Declares all the possible accepted types for a variable type.
      1. **Syntax:** 
         1. variableName: number | string = value;
      2. **Example:**
         1. team: number | string = “Creative Technology”;
6. **Video 23: Literal Types**
   1. **Literal Types**: Declare the exact value that can be assigned to a variable.
      1. **Syntax:**
         1. variableName: value1 | value2;
      2. **Example:** 
         1. resultConversion: ‘as-number’ | ‘as-text’;
7. **Video 24: Type Alias:**
   1. **Type Alias**: Allows you to combine types and create an alias for it.
   2. **Example:**
      1. type Combinable = number | string;
      2. type ConversionDescriptor = 'as-number' | 'as-text';
      3. input1: **Combinable**,
      4. input2: **Combinable**,
      5. resultConversion: **ConversionDescriptor**) {}
8. **Video 26: Types & Voids**:
   1. **Type Alias**: Allows you to combine types and create an alias for it.
   2. **Example**:
      1. Typescript can infer the return types of functions.
      2. Void data type indicates a function will not return a value.
      3. Undefined data type, means a function is returning an undefined value.
9. **Video 27: Function as Types**:
   1. **Function Type**: Types that describe the parameters and the return type of a function.
      1. **Note**: The typescript parameter names, in the function definition, does not have to match the actual parameter names in the function.
      2. **Function Example:** 
         1. function add(n1: number, n2: number): number {
         2. return n1 +n2;
         3. }
      3. **Syntax:**
         1. let combineValues = (param1: type, param2: type) => returnType;
         2. combineValues = add;
10. **Video 28: Function Types and Callbacks:**
    1. **Callback Function:** A function passed as an argument to another function.
    2. **Note:** If a function’s returned type is declared as void, any value that is returned from it will be ignored.
    3. **Callback Example:**

function addAndHandle(n1: number, n2: number, cb:(num: number) => void) {

    const result = n1 + n2;

    cb(result);

}

addAndHandle(10, 20, (result) => {

    console.log(result);

return result;

})

1. **Video 29: The “unknown” Type:**
   1. **Unknown variables** can store any value without getting errors.
   2. **Example:**

let userInput: unknown;

let userName: string;

userInput = 5;

userInput = 'Max'; //Variables with defined types cannot accept values of type unknown, unless the variable is defined as any

userName = userInput;

* 1. **Note:** If you would like to assign an unknown variable (userInput) to a string variable (userName), you would need to perform type checking before assigning userInput to userName.
  2. **Example:**

if (typeof userInput === 'string') {

    userName = userInput;

}

1. **Video 30: “never” Type:**
   1. **“never” type**: Should be used to a declare a function’s return type, if it never returns any value.  If an error is thrown, or an infinite loop is generated, the return type should be never.
      1. **Example:**

function generateError(message: string, code: number): never{

    throw {message: message, errorCode: code};

}

const result = generateError("An error occurred!", 500);

console.log(result);