TYPESCRIPT

Typescript is an Object-Oriented programming language that is built on top of JavaScript. It developed and maintained by Microsoft.

The browser and node.js does not directly execute Typescript. Its needs compiler (Typescript Compiler) to compile it into JavaScript.

So after writing your Typescript code, everything will be compiled into JavaScript by the typescript compiler.

Note = All typescript files are supposed to be name with the extension ”.ts”

Note = The Main purpose of TypeScript is to allow you to use all Object-oriented programming features in JavaScript.

======================================================================================

Install Typescript

First download NODE.JS and use below npm command on your CMD to install the typescript-Complier globally so that you can run it anywhere in your systems:

npm install -g typescript

Then to check if everything is successfully working use this command:

tsc -v // tsc means typescript compiler

======================================================================================

TypeScript First Program

Create a folder in your desktop and create index.ts file in it and put the below code into it:

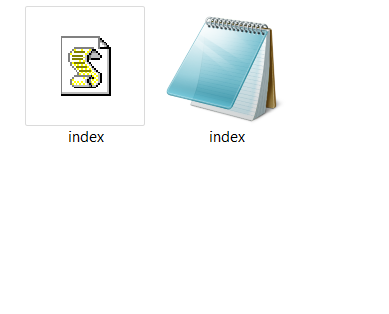
console.log("Hello World");

Now, open the cmd and navigate to the project directory and compile the above index.ts file with below command:

tsc index.ts

or you can use: tsc index.ts -w to watch for changes in your code instead of re-compiling all the time whenever you make some little changes. (the -w means watch)

It will compile the above file into JavaScript with “.js” extension.



Now what you need to do is to run the .js compiled file using Node.js with this command:

node index.js

// result: Helloo world

The key is that, all your Typescript files will be compiled into JavaScript. And the browser will run the compiled JavaScript file not the Typescript file.

======================================================================================

Compile Time Error with TypeScript

Typescript always shows a compile time error whenever there is a bug in your code and you try to compile it with tsc filename.ts. it will show you compile time error and if the error is not fixed, it will never compile your code into Javascript.

======================================================================================

TypeScript Datatypes

It’s not compulsory to explicitly use data-types in typescript as you would in Java. For example, below typescript code runs perfectly. The variable has no data-type. But remember, it good practice to always explicitly use data-types for your variables.

// variable without any data-Type

let myName ="justice";

console.log(myName);

// results: justice

These are the following datatypes in TypeScript:

* Number 🡺 This is used to represent all numbers, either integer or float there is no difference.
* String 🡺 This represents all text characters.
* Boolean 🡺 This represents true or false values.
* any 🡺 This is a dynamically type, use it when you don’t know the datatype of some value you want. It like the var in javascript
* Object 🡺 This represents a variable that has a datatype of Object. (you can use curry braces “{}” instead of Object) it simple a map or javaScript object.

Eg.

var person: Object = {

firstName: "Justice",

lastName: "Ankomah"

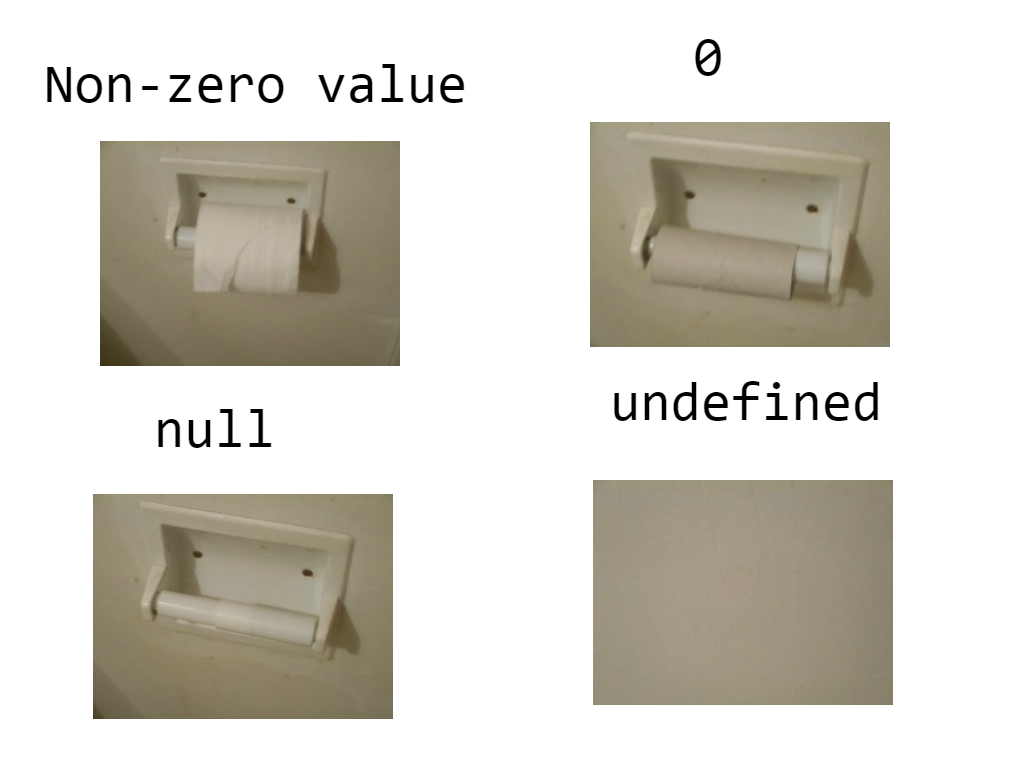
}

console.log(person["firstName"]);

// result: Justice

* Array 🡺 Array represents list of data Eg. var colors: String[] = ["red", "yellow"];
* Undefined 🡺 This represent a variable declared with no value assigned to it. Eg.  firstName: String; (it means the variable value is not there)
* Null 🡺 This is a datatype assigned to variable to indicate that it has no value. Eg. firstName: String = null; (it means the variable value is there but its empty)

The Difference between null and Undefined can be explained with this below image.



**Example:**

   export class AppComponent {

      //All the "#" in-front of the varibles means "private".

      //it the same as: private firstName:Sting | null =null;

      // This is called null variable

      // The variable is declared but it value is empty. it has value but its empty

    #firstName:String | null =null;

    // This is called "undefined" variable

    // as you can see, the variable is declared but it has no value or (it value is undefined)

    #lastName!:String;

  console.log(this.#firstName); // result: null

  console.log(this.#lastName); // result: undefined

    }

* Unioun 🡺 Union helps you to choose two or more data-types for a particular variable. Let say, if you want your variable to be allowed to be assign only a value of specific data-Types.

EG:

//  String|Number means the <myName> variable can be of type String or Number

let myName: String|Number = "Justice";

myName = 56;

console.log(myName);

// result: 56

// When creating an array of unioun-type, surround the dataType with parentheses

let colors: (String|Boolean)[] = ["red", true, "pink"];

console.log(colors);

// result: [ "red", true, "pink" ]

* Function 🡺 A Function type is a variable that will have a value of a function. REMEMBER, it start with a capital “F”

EG:

// declare a variable that has a data-Type of a function

let age:Function;

// assign a function/value to the variable

age =():Number=>{

    return 55;

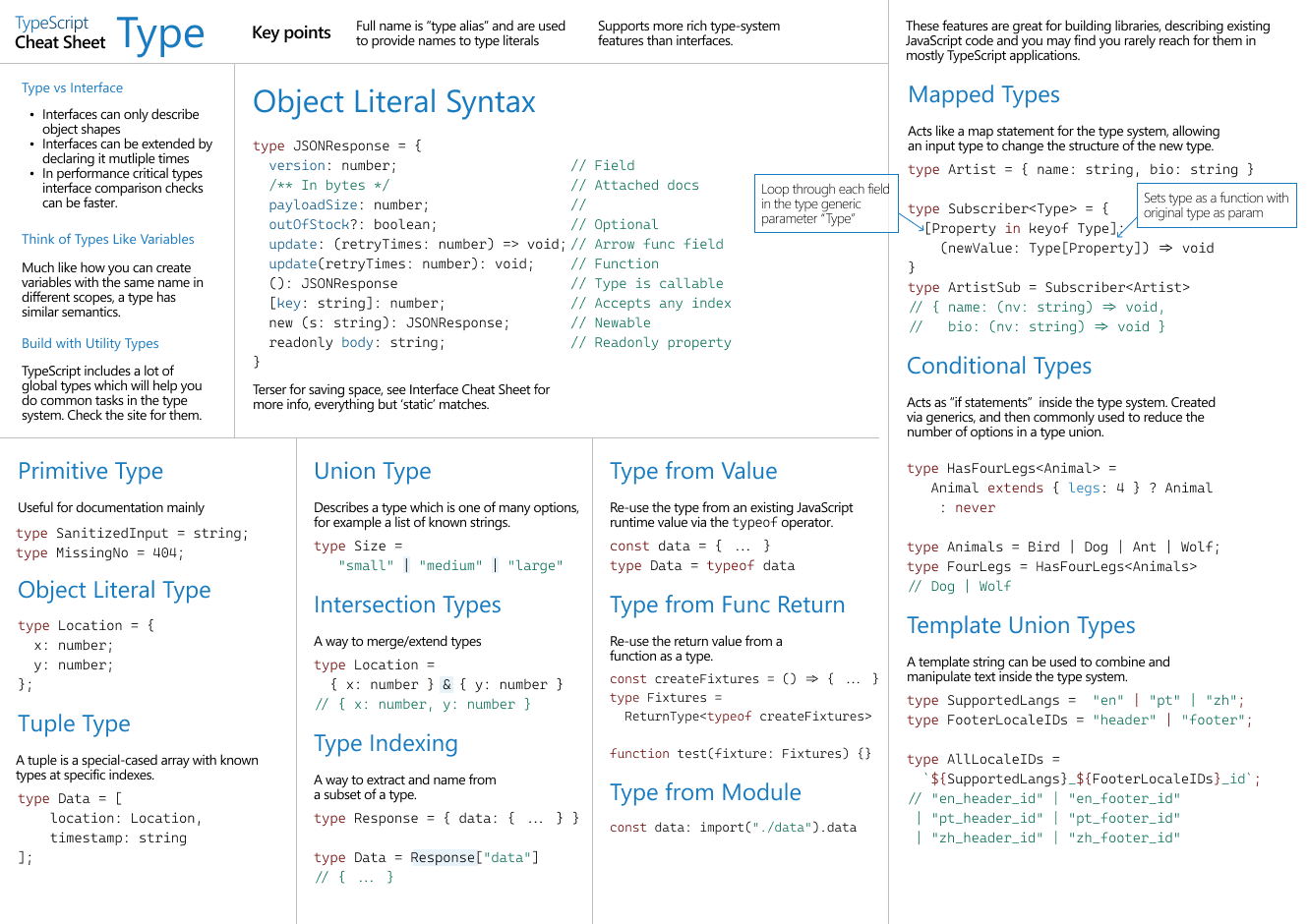
}

console.log(age());

// result: 55

You can use the typeOf() function to check the datatype of a particular variable.

Eg. console.log( typeof(“justy”) ); // result: String



======================================================================================

Optional “?” Types or Variables

You can add the question mark “?” symbol to a variable in a class or interface to make it optional. Meaning it value can be provided by the calling object or not.

**Example:**

export class AppComponent {

  ngOnInit(): void {

    // create object of the <Book> interface below but don't provide value for optional "bookName" property

    let book1: Book = {

      bookTitle: "Aki Ola"

    }

    // if no value is given to book1.bookName

    if (book1.bookName == undefined) {

      console.log("book1.bookName is undefined");

    }

    // if book1.bookName has a value

    else {

      console.log("book1.bookName is defined");

    }

  }

}

interface Book {

  // Optional "?" meaning it value can be provided by the calling object or not

  // it will have a default value "undefined" if no value is provided to it

  bookName?: String;

  bookTitle: String;

}

// Result: book1.bookName is undefined

======================================================================================

How To Use The Double Question Mark “??” Operator called “null coalescence”

It an operator used to check & provide a fallback for a particular variable.

The hack is that if the variable has some value the ?? operator will print it. But if the value of the variable is null or undefined, then it will print the fall back. Remember you have been using this in flutter already, check the example below.

**Example:**

  export class AppComponent {

    ngOnInit(): void {

    this.printName("justice");    // result: firstName is: justice

    this.printName(null);        //result: firstName is: no name

    this.printName(undefined);  //result: firstName is: no name

    }

    printName(firstName: String | null | undefined){

      // ?? = means, check if firstName has some value, then print it, else print "no name"

    console.log(`firstName is: ${firstName ?? "no name"}`);

    }

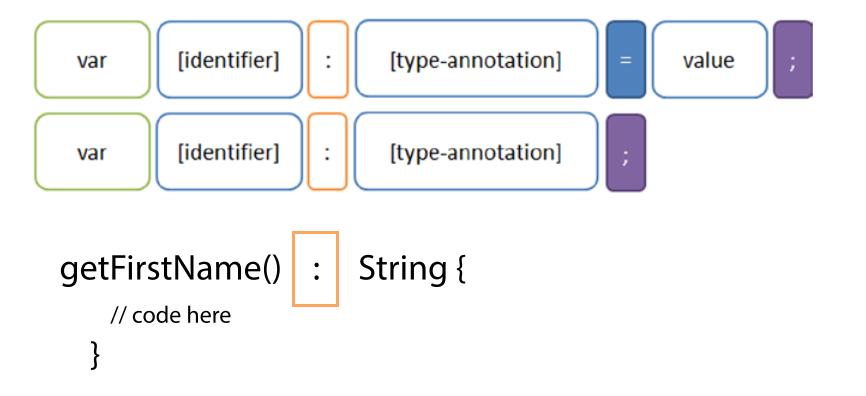
  }

======================================================================================

TypeScript Variables and Method Declaration Format

In Typescript We use colon “:” for declaring variables and methods datatype.

* The Datatype needs to come after the colon in variable Name.
* But the datatype needs to come after the bracket “()” in method declaration.



Whenever you create a variable inside a class, you don’t have to use the “var” keyword as shown in the image above. You only have to use it when you create a variable that’s not inside a class.

======================================================================================

Functions

Functions helps you to make use of codes over and over again.

Eg.

// =====  VARIABLE THAT HAS A TYPE OF A FUNCTION

// declare a variable that has a data-Type of a function

let age:Function;

// assign a function/value to the variable

age =():Number=>{

    return 55;

}

console.log(age());

// result: 55

// ===== OPTIONAL PARAMETER FUNCTION

/\*

Optional parameters are not required to be passed values in when calling the functions.

 Declare your optional parameters with question-mark symbol ?.

 Note = your optional parameters needs to come after the required parameter

\*/

 let colors = (yellow:String, red?:String)=>{

     console.log(yellow, red);

 }

 colors("yellow color");

 // result: yellow color undefined

// (if you don't passs any argument to the optional parameter, it will be undefined)

 colors("yellow color", "red color");

 // result: yellow color red color

 // ===== FUNCTION RETURN TYPES

// functions returns type can be of void, string, Number, object, any or custom object

 let books =():String=>{

 return "Aki Ola books"

 }

 console.log(books());

 // result: Aki Ola books

======================================================================================

Type Aliases

Type Aliase helps you to make use of a particular data-type instead of explicitly typing it over and over again.

For example, if you’re going to have 10 variables that are all going to have a data-Type of String. Instead of explicitly specifying a string data-type for them all, you can simply create an aliase and use it for all the 10 variables.

Eg.

// create an Alias/variable that has a value of dataType by using the "type" keyword

type stringOrNum = String | Number;

// assign the above variable of dataType String or Number  as a Type

let bookName:stringOrNum = "Aki Ola";

let bookAge:stringOrNum = 55;

console.log(bookName);

// result: Aki Ola

// create an object type aliase

type myObjectAliase = {

    firstName: String,

    age: Number

}

// use the above object type Aliase

let person:myObjectAliase = {

    firstName: "justice",

    age: 55

    // if you add any key here like "lastName", you will get an error becuase it not in the above "myObjectAliase" aliase

}

console.log(person["firstName"]);

// result: justice

======================================================================================

Class

A class is a blue print of an object.

class Books{

    // Propeties

    bookName:String;

    bookNo:Number;

    // Always Initialize your class properties in a Constructor if possible

    constructor(bookName:String, bookNo:Number){

       this.bookName = bookName;

       this.bookNo=bookNo;

    }

    // method

    getBookName():String{

        return this.bookName;

    }

}

// first object of the above class

let book1 = new Books("Flamingo", 29);

console.log(book1.getBookName());

// result: Flamingo

// second object of the above class

let book2 = new Books("Aki Oa", 50);

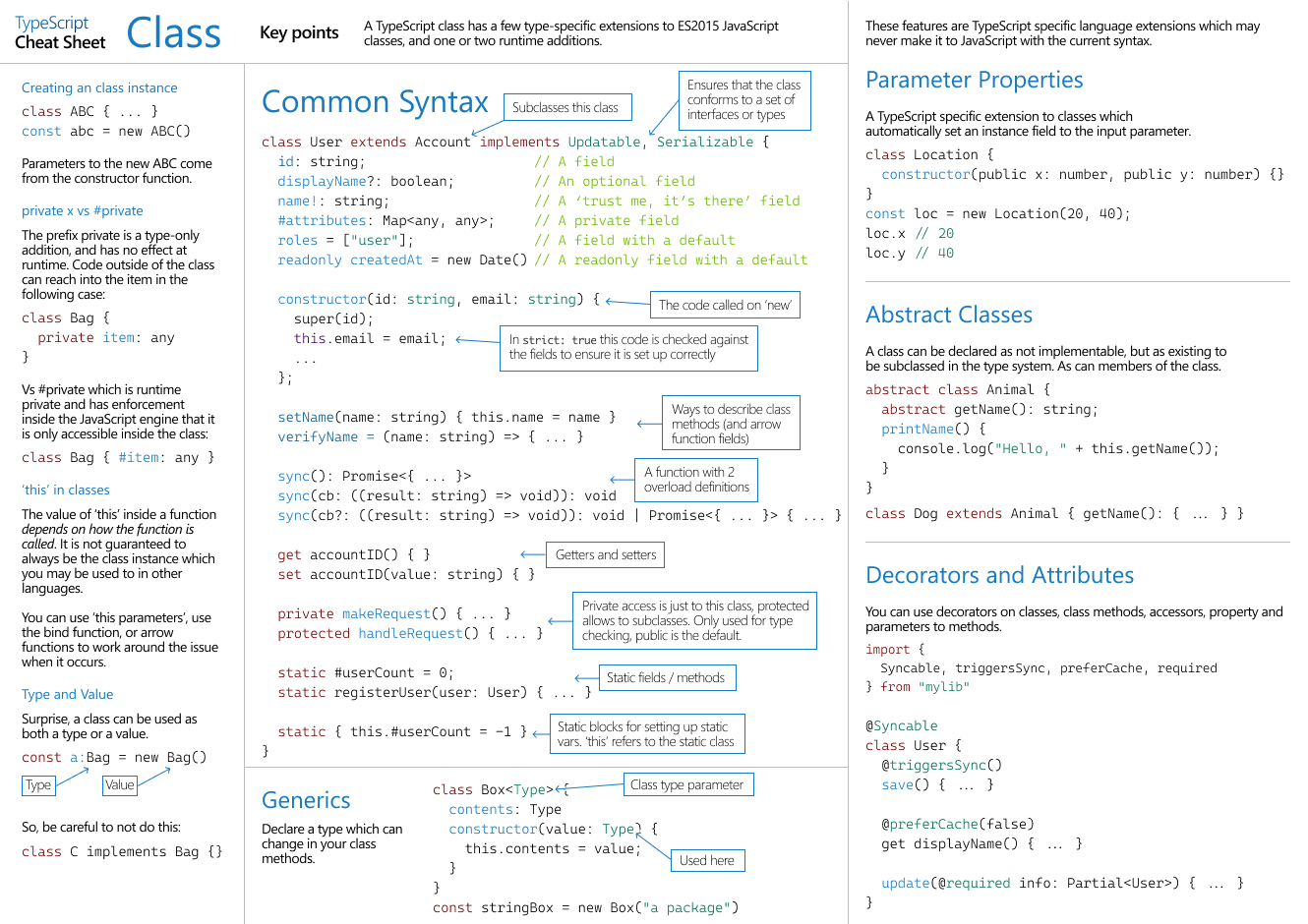
// Create an array that has a datatype of the above class

let book3:Books[]=[];

book3.push(book1);

book3.push(book2);

console.log(book3);



======================================================================================

Access Modifiers

TypeScript has 3 main access modifiers.

Public 🡺 globally scope, it accessible everywhere

Private 🡺 class scope, it accessible only with the same class. It can be created using ash “#” or “private” keyword

Readonly 🡺 read scope, a variable that you can only read it value after its initialization in a constructor. You can’t write or assign value to it.

class Books{

    // Propeties

    public bookName:String; // accessible everywhere

    private bookNo:Number; // accessible only in this class. It can be created also as: #bookNo:Number;

    readonly bookType:String; // a property that you can only read it value, you can't assign value to it after it first initialization

    // Initialize the above properties in a Constructor

    constructor(bookName:String, bookNo:Number, bookType:String){

       this.bookName = bookName;

       this.bookNo=bookNo;

       this.bookType=bookType;

    }

    // method

    getBookName():String{

        this.bookType="french"; // error, you can't assign value to a "readonly" Variable

        this.bookName = this.bookType; // correct, you are accessing it value

        return this.bookName;

    }

}

//object of the above class

let book1 = new Books("Aki Oa", 50, "math");

book1.bookType = "Science";  // Error,  you can't assign value to a "readonly" Variable

======================================================================================

Interface

In typescript, we use interface to define custom data-types for our variables. So instead of using only the built-in typescript data-types, you can define some object of particular types in an interface and use it.

Eg.

// Define an interface that you will use it as a data-type for other variables

interface PersonInterface{

    firstName: String;

    lastName: String;

    age: Number;

    // a method that accept 1 String parameter and has a void return type

    setChurch(churchName: String):void;

}

// Now create a variable that will have a data-Type of the above "PersonInterface"

// you need to provide all values for all the properties in the above without living a single one.

// And if you add an extra property to below "person1" that is not in the above "PersonInterface" there will be an error

let person1:PersonInterface = {

    firstName: "justice",

    lastName: "Ankomah",

    age: 25,

    setChurch: (churchName: String)=>{

      console.log(`my church name is ${churchName}`);

    }

}

console.log(person1.setChurch("pentecost"));

// result: my church name is pentecost

// == A METHOD THAT ACCEPTS A PARAMETER DATA-TYPE OF AN INTERFACE

 function person2(person: PersonInterface){

  console.log(`my first Name is: ${person.firstName}`);

}

person2(person1);

// result: my first Name is: justice

How to implements an interface using a class

* A class extends a class & a class implements an interface
* An interface extends an interface

Eg:

// create an interface

export interface Person{

    // create a method that returns a string

    // Note: all interface methods are abstract and they can't have a body

    getfirstName():String;

  }

  // create a class that implements the above interface

class PersonClass implements Person{

    // override the above interface method and provide implementation for its body

    getfirstName(): String {

        return "Justice"

    }

}

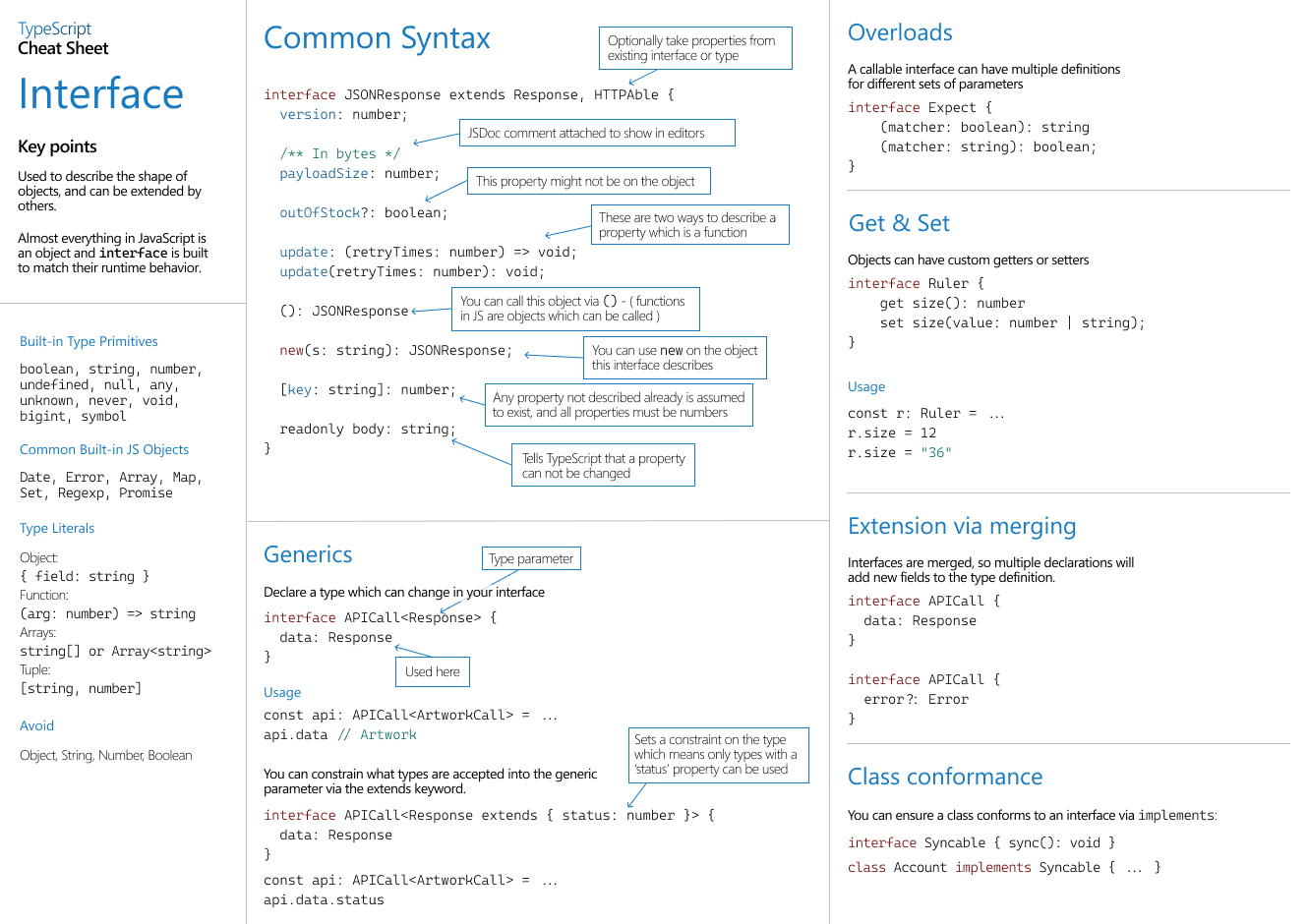
// create object of the above class

let Person1 = new PersonClass();

// access one of the class methods

console.log(Person1.getfirstName());

// result: Justice



======================================================================================

Tuples

Tuples are just arrays but they help to provide a specific datatype for each element in the array.

For example, you can use it to provide a different datatype for each element in an array.

**How to define a tuple**

To define a tuple, use the square bracket [ ] to define the order of Datatypes of your values.

**Example:**

export class AppComponent {

      ngOnInit(): void {

        // define a tuple

      let myTuple: [String, boolean, number];

      // initialize your tuple

//  myTuple=[34, true, "justice"]; this will throw an error because the order needs to be string, Boolean, number

      myTuple=["justice", true, 21]; Correct

          /\*

      You can also write the above tuple on a single line as

      const myTuple: [String, boolean, number] =["justice", true, 21];

      \*/

      console.log(myTuple);  // result: [ "justice", true, 21 ]

      // add item to the tuple, it can be an item of whaterver datatype you want

      myTuple.push("Ankomah");

      console.log(myTuple);  // result: [ "justice", true, 21, "Ankomah" ]

      }

How to pass a method or function to a tuple

Use the “Function” type when defining a tuple that will accept a function of method.

**Example:**

  export class AppComponent {

    ngOnInit(): void {

      // Define a tuple that will accept a function using the "Function" keyword

      let myTuple: [String, Function];

      // initialize the tuple and assign the below method to it

      myTuple = ["justice", this.printAge]

      // call the second item in the tuple which is the function

      // remember to add parentheses () to it because it a method

      myTuple[1]();

    }

    // Create a function

    printAge() {

      console.log("your age is 21");

    }

  }

          // result: your age is 21

How to assign a tuple to an array or object

Since tuples are just arrays, you can assign it to other arrays.

**Example:**

  export class AppComponent {

    ngOnInit(): void {

      // Define a tuple

      let myTuple: [String, number];

      // initialize the tuple values

      myTuple = ["justice", 21]

      // Define an array and assign your tuple to it

      let myArray:any[]=myTuple;

      console.log(myArray);

    }

  }

      // result: [ "justice", 21 ]

How To Create a tuple that will accept different a list of Arrays

You can create a tuple that will accept a list of arrays by adding square bracket “[]” to the tuple.

**Example:**

  export class AppComponent {

    ngOnInit(): void {

      // Define a tuple that will take an array

      let myTuple: [String, number][];

      // pass different values of array to the tuple

      myTuple = [["justice", 21],["Shadrack", 39]];

      // Loop through the tuple

       myTuple.map((eachTuple)=>{

         // Grab each first element in each tuple

         console.log(eachTuple[0]);

       });

    }

  }

      /\*

       result:

       Justice

       Shadarck

      \*/

Read Only Tuple

The best way to define a tuple is to prevent it from being modified. And you can use that by adding the “readonly” keyword to your tuple.

**Example:**

 export class AppComponent {

      ngOnInit(): void {

      // Create a tuple of type readonly. And provide types for each of it value

      // Meaning it values must be in order of types: string, boolearn, number

     let personDetials: readonly [String, boolean, number];

     // initialize the tuple values

     personDetials =["justice", true, 55];

    /\*

    You can also write the above tuple on a single line as

     const personDetials: readonly [String, boolean, number] = personDetials =["justice", true, 55];

    \*/

     console.log(personDetials); // Results: [ "justice", true, 55 ]

     // Error: Property 'push' does not exist on type 'readonly [String, boolean, number]'

     // why becuase, you can't modify the value of readonly var, you can only fetch/read it value

    personDetials.push("justice");

    }

  }

Name Tuple

You can give name for your tuple fields when defining it. This helps you to know what goes into the tuple.

So always define your tuples by sgiving name to it fields.

Note: you don’t have to access the tuple fields with their names, use their index

**Example:**

    export class AppComponent {

      ngOnInit(): void {

        // Define a tuple with a field name and it data-type

        let person: [firstName:String, age:number];

        // initialize the

        person= ["justice", 21];

        console.log(person[0]);

      }

    }

        /\*

        result:

        Justice

        \*/

**Destructing tuples**

Destructing is just a way of assigning all the values or specific values of an array/tuples to different variables. This is mostly used in react.js useState() if you can remember.

**Example:**

   export class AppComponent {

      ngOnInit(): void {

    // ==== Destructuring Concept Old Way

    // it all about assigning values in array to other variables

    // That's either  all the values in the array or specific ones

    const vehicless = ['mustang', 'f-150', 'expedition'];

      // assign each value of the above vehicles to a different variable

      const carr = vehicless[0];

      const truckk = vehicless[1];

      const suvv = vehicless[2];

      // ==== Destructuring New Way

      const vehicles = ['mustang', 'f-150', 'expedition'];

      // assign each value of the above vehicles to a different variable

      // When destructuring arrays, the order that variables are declared is important.

      const [car, truck, suv] = vehicles;

    /\*

    So now the above:

    car is assigned "mustang"

    truck is assigned "f-150"

    suv is assigned "expedition"

    \*/

    // ============= How TO DESTRUCTURE/ASSIGN ONLY SPECIFIC VALUES IN THE ARRAY/TUPLES TO VARIBLES

    // If we only want the car and suv we can simply leave out the truck but keep the comma:

    // So  the comma "," will represent the value we don't want to assign to a variable

    const newvehicles = ['mustang', 'f-150', 'expedition'];

    // you can see that in between the two "," represent the second value “f-150” in the above "newvehicles" that we don't want

    const [car, ,suv] = newvehicles;

      }

    }

How To Create A function with tuple parameters

    export class AppComponent {

      ngOnInit(): void {

        // Create a tuple

      const somePerson:[String, number] = ["justice", 21];

      // call the below method and pass in the above tuple as argument

      // But make sure your tuple has the same order & number of types as your function argument

      this.printPerson(somePerson);

      }

      // Create a function that takes a tuple parameter

      printPerson(person: [firstName: String, age:number]):void{

        console.log(person[0]);

      }

    }

    // result: justice

======================================================================================

Export & Import a File

You can create a particular interface, a function or a class in another file and export and import it into other files. This will help you to use a particular function or an interface in other files.

EG:

// Inside person.ts

// create an interface and export it with the "export" keyword

export interface Person{

    firstName:String;

    lastName:String;

  }

  // inside index.ts

// Import the "Person" interface in the src/person.ts file

// the "../" is a relative path

import {Person} from "../src/person";

// use the above imported interface

let person1:Person={

    firstName:"Justice",

    lastName: "Ankomah"

}

console.log(person1.firstName);

// result: Justice

======================================================================================

Generics

In general, generics helps you to do specific thing with different data-types. Let say you want a particular function to accept both an argument of type String & Numbers, you can easily use generics.

EG:

// ===== GENERIC FUNCTIONS ====

// create a function that accept a generic parameter

// the "T" datatype of "data" is the Generic type of the function

let myFunc = <T>(data: T):void=>{

console.log(data);

}

// call the function and pass a particular datatype for it generic

// am pasing string

myFunc<String>("my name is Justice Ankomah");

// result: my name is Justice Ankomah

// call the function again and pass a particular datatype for it generic

// am pasing Number

myFunc<Number>(77);

// result: 77

// ===== GENERIC INTERFACE ======

// create a generic interface

// THE <T> "T" can be character

interface Person<T>{

    firstName: String;

    lastName: String;

    // the "T" type below is what is coming from the above Person<T> interface

    biography: T

}

// create a variable that has a type of the above interface and provide implementation to it properties without living a single one

// remember, the "Person<T>" interface is a generic. So pass in any data-Type you want to use

// am passing in a String here

let person1:Person<String> ={

    firstName: "justice",

    lastName: "Ankomah",

    biography: "a programmer in Ghana"

}

console.log(person1.firstName);

// result: justice

// create another variable that uses the above generic interface

// here am passing "object" data-type to the generic

let person2:Person<object> ={

    firstName: "Kenneth",

    lastName: "Ankomah",

    biography: {

        job:"a programmer in Ghana",

        age: 25

    }

}

console.log(person2.biography);

// result: {job: 'a programmer in Ghana', age: 25}

//==== GENERIC extends =======

// the extends means, the generic type needs to be of a particular dataType

// below, this function will only accept Numbers

let mynewFunc = <T extends Number>(data: T):void=>{

    console.log(data);

    }

    // call the function and pass Number to it

    mynewFunc<Number>(66);

    // result: 66

    // this will be an error since the function only extends Number

    mynewFunc<String>("justice");

    // result: Error

======================================================================================

Enums

Enums are list of constant values that never changes.

**Hack** = The Enum property name is the same as it value

All the values in an enum start with an index of 0

EG:

// create enum list of constant values

// the index of an enum start with 0

enum colors{

    RED,

    YELLOW,

    WHITE,

    PINK

}

console.log(colors[3]);

// result: PINK

let books:object = {

    bookName: "Aki Ola",

    bookColor: colors[1]

}

console.log(books);

// result: { bookName: 'Aki Ola', bookColor: 'YELLOW' }

======================================================================================

How To Create And use Model

// Inside: productmodel.ts

// (1) Create an interface that will represent your actual Model

export interface ProductModel {

    // Interface Properties

    // you can make any of below properties as optional by adding '?' to it

    id?: number;

    productName: string;

    productCategory: string;

    productDate: Date;

    productFreshness: string;

    productPrice: string;

    productComment: string;

}

// (2) Create a class that perfomrs the encoding/decoding of above ProductModel interface

export class Convert\_PM {

    // Create a static method that convert a json-data from server to

    // the ProductModel above interface (Decoding/desirialization)

    public static toProductModel(json: string): ProductModel {

        // JSON.parse() will convert the json-data to javascript object/ProductModel

        return JSON.parse(json);

    }

    // Create a static method that convert above ProductModel to json

    // above interface (Encoding/Sirialization)

    public static productModelToJson(value: ProductModel): string {

        // JSON.stringify() will convert a JS object to json/String

        return JSON.stringify(value);

    }

}

// To parse above data and perform encoding/decoding of above <ProductModel> in Other component

//

// Import the "class" and the ProductModel interface from the above file

//   import { Convert\_PM, ProductModel } from "./productmodel.ts";

//

//  PERFOM DECODING...

//   let productModel = Convert\_PM.toProductModel(json\_Data\_from\_server);

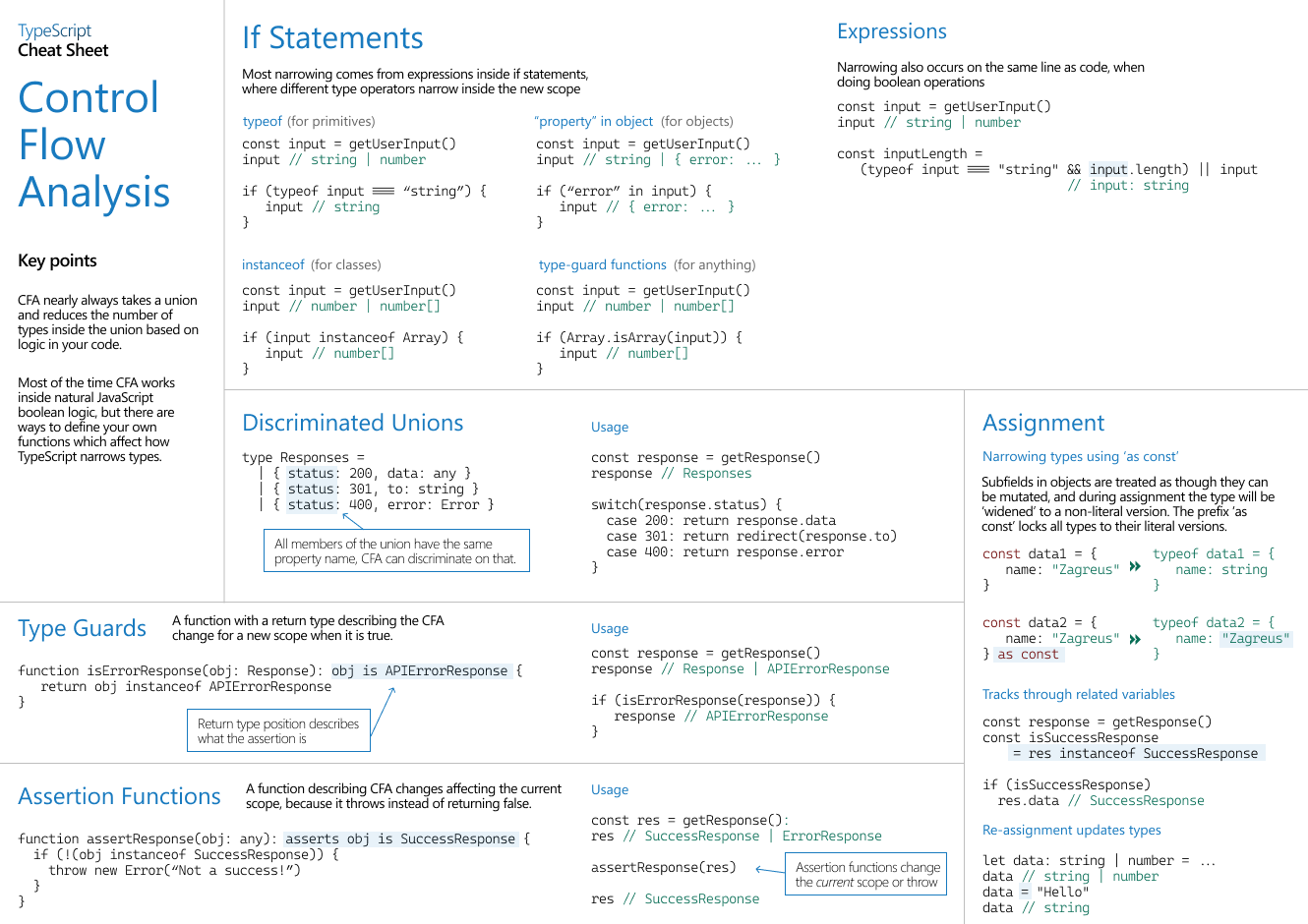
//

// PERFOM ENCODING...

// Convert.PM.productModelToJson(Object\_of\_ProductModel);

======================================================================================

**If else Conditions**



======================================================================================