CONCEPTS TypeScript

Basic npm commands for TS project

```
npm init -y
npm i typescript
npx tsc --init
npx tsc --init --rootdir src --outdir lib
npx tsc --watch
```

New JS type Bigint

```
// new type
let bigint: bigint = 24n;

// Tuple type
let tuple: [number, number] = [0, 0];
```

Tuple type

```
// Tuple type
let tuple: [number, number] = [0, 0];
```

Duck type annotation

```
type Point2d = { x: number; y: number };
type Point3d = { x: number; y: number; z: number };

let point2d: Point2d = { x: 0, y: 0 };
let point3d: Point3d = { x: 0, y: 0, z: 0 };

/* EXTRA info ok */

point2d = Point3d;
function takesPoint2d(point2d: Point2d) {}
takesPoint2d(point3d);

// Error !!!
point3d = Point2d; // Error
function takesPoint3d(point3d: Point3d) {}
takesPoint3d(point2d); // Error
```

Unknown type

1. If you don't know what type you are looking

```
let exampleUnknown:unknown; // I don't now what is the type

if (typeOf exampleUnknown === "string") {
    exampleUnknown.trim()
}

if (typeOf exampleUnknown === "boolean") {
    let boolean: boolean = exampleUnknown
}
```

Assertions type

1. We are telling the type script compiler "I now what is a type"
2. don't use angel brackets <string>
3. when you use this type and changed type to another TS don't show error

```
let hello = load();

const timed = (hello as string).trim();
hello = 0; // now can't see !ErrorTS
```

+(Casting) type

1. We are telling the type script compiler "I now what is a type"

```
let letters = "hello";
const number = +letters;
console.log(number === 123123);
```

Declaration type

1. Declaration new global variable `env.d.ts`
    1. created file `env.d.ts`
    2. `export declared const process: any;`
    3. access to this variable `process.env.USER`

```
process.env.User; // can access
```

Preparation for work with TS

1. Working with node.js files
    1. Node.s cant Working with TS files!
        1. Need convert to JS use command `npx tsc`
        2. `node index.js`
    2. Skip this process:
        1. Need install package `npm install ts-node`
        2. use command `npm ts-node index.ts`
        3. OR set `package.json`

```
"scripts": {start: "ts-node index.ts"}
```

2. Working with node.js variables for example: process.env.User
    1. Need use nmp command `npm install @types/node`
    2. You'll get access to `env` variable form node.js

```
env; // can access
import fs from "fs"; // can access
fs.writeFileSync("hello.txt", "hello");
```

3. Working with TS ExpressJS

```
npm install express
npm install @types/express
```

INTERMEDIATE level

Readonly modifier

1. When need only read in object keys and values

```
type Point = {
  readonly x: number;
  y: number;
};

point = { x: 1, y: 1 };
//Can't property assignment
point.x = 1; //!Not working
```

Union type

1. => | pipe operator
2. use extra pipe operator

```
type Union = number | string;
```

Literal type

1. If need use uniq type string

```
let direction: "North";
direction: "South"; //!ERROR: Invalid type
```

2. use union type string

```
let direction: "North" | "East" | "West" | "South";
```

Narrowing type (Compare to type "instanceof" operator)

1. Compare primitive type "string" or "number" we can use typeOf

```
let name: "string" | "number";

if(typeOf name == "string"){...}
```

2. Compare some not primitive type "object" or "array"

```
class Cat {
  meow() {
    console.log("Meow");
  }
}
class Dog {
  bark() {
    console.log("Woof");
  }
}

type Animal = Cat | Dog;

function speak(animal: Animal) {
```

```
  if (anima instanceof Cat) {
    animal.meow();
  }
}
```

3. Compare keys in objects

```
type Square = {
  sizes: number;
};

type Rectangle = {
  width: number;
  height: number;
};

type Shape = Square | Rectangle;

function area(shape: Shape) {
  if ("sizes" in shape) {
    return shape.sizes * shape.size;
  }
}
```

Class parameters property

Omit unnecessary type declared properties

```
class Person {
  //Omit! public name: string;
  //Omit! public age: number;

  constructor(public name: string, public age: number) {
    //Omit! this.name = name;
    //Omit! this.age = age;
  }
}

const adam = new Person("Adam", 12000);
adam.name, adam.age;
```

.filter() value is possible undefined

Fixed this TS error

```
type User = {
  name: string;
```

```
  age: number;
};

const users: User[] = [
  {
    name: "Adam",
    age: 12000,
  },
  {
    name: "Via",
    age: 1222,
  },
];

function getUserAge(name: string): number {
  const user = users.find((user) => user.name === name);

  if (user == null) {
    throw new Error(`User not found ${name}`);
  }
  return user.age; //! Error Object is possible undefined
}
```

using == equal operator for

1. Use double equal (==) for check null or undefined values

```
console.log(null == null); // true;
console.log(undefined == undefined); // true;

console.log(undefined == null); // true! Surprised!!!;
```

2. Null is not equal to other values

```
console.log(false == null); // false
console.log(0 == null); // false
console.log("" == null); // false
```

3. Check result only null or undefined

```
const result: number | undefined | null;

if (result == null) {
  console.log(result); // result only have null | undefined
}

function result(value: number | undefined | null) {
```

```
  if (value == null) {
    return value; // undefined| null
  }

  return value + 2; // number

  console.log(result(1)); //  3
  console.log(result(null)); //null
  console.log(result(undefined)); //undefined
}
```

4. if result only boolean

```
if (result != null) {
  console.log(result); // result only have true | false
}
```

Intersection type

1. Extend one type to other type

```
type Point2D = {
  x: number;
  y: number;
};

type Point3D = Point3D & {
  z: number;
};
```

2. EXAMPLE:

```
type Person = {
  name: string;
};

type Email = {
  email: string;
};
// OR
type ContactDetails = Person & Email;

function contact(details: Person & Email) {
  console.log(
    `Dear ${details.name}. I hope you will received ${details.email}`
  );
}
```

```
contact({
  name: "John",
  //!ERROR email: 'john@example.com
});
```

Option modifier

```
type Person = {
  name: string;
  email?: string;
};

const alfred: Person = {
  name: "John",
  email: undefined, // or "john@example.com
};

class Point {
  x?: number | null;
  y?: number;
}

const point = new Point();
console.log(point.x); // undefined

// we cant assign number or undefined
point.x = undefined;
point.x = 0;

point.x = null; // added union type null
```

Non-null Assertion Operator => point!.x

1. Whe I now this variable wont be null

```
type Point = {
  x: number;
  y: number;
};

let point: Point;
function initialize() {
  point = { x: 0, y: 0 };
}
initialize();

console.log("After Initialized", point!.x, point!.y);
```

```ts
type Person = {
  name: string;
  email?: string | null | undefined;
};

function sendEmail(email: string) {
  console.log("Sending email", email);
}

function ensureContactable(person: Person) {
  if (person.email == null)
    throw new Error(`You must provide ${person.name}`);
}

function contact(person: Person) {
  ensureContactable(person);
  //ERROR argument  of type "string | null | undefined
  sendEmail(person.email!); // use =>> !
}
```

Difference between Intersection type and interface

1. If need extend interfaces || use Intersection type

```ts
type PointIntersection = {
    x: number,
    y: number,
}

type Point2DIntersection = PointIntersection & {
    z: number
}

interface PointInterface {
    x: number,
    y: number,
}

type Point2DInterface extend PointInterface {
    z: number
}

export const point: Point2DInterface = {
    x: 0,
    y: 0,
    // z: 0, !Error
}
```

interface Declaration merging

1. It is possible to combine two interfaces.

```
export interface Request {
  body: any;
}

export interface Request {
  json: any;
}

function handleRequest(req: Request) {
  req.body = req.body;
  // you will get access to jason property of instance Request interface
  req.json = req.json;
}
```

Benefits Type then Interfaces

1. Type:
    1. Unions
    2. Intersections
    3. Primitives
    4. Shorthand functions
    5. Advanced type functions

```
type InputOnChange = (newValue: InputValue) => void;
type InputValue = string;
type InputType = "text" | "email";

export type InputProps = {
  type: InputType;
  value: InputValue;
  onChange: InputOnChange;
};
```

2. Interfaces:
    1. Declaration Merging
    2. Familiarity (extends interfaces)

```
export interface InputProps = {
    type: "text"| "email",
    value: string,
```

```
            onChange:  (new: InputValue) => void,
        }
```

Newer type

1. If need handle ERROR if we not have all successfully handle cases
2. If on the future we decide to support new types will be ERROR

```
type Square = {
  kind: "Square";
  size: number;
};

type Rectangle = {
  kind: "Rectangle";
  width: number;
  height: number;
};

type Circle = {
  kind: "Circle";
  radius: number;
};

type Shape = Square | Rectangle | Circle;

function area(s: Shape) {
  if (s.kind === "Square") {
    return s.size * s.size;
  } else if (s.kind === "Rectangle") {
    return s.width * s.height;
  }
  // NEED added statement

  const _ensureAllCaseAreHandle: never = s; // Circle ERROR

  return _ensureAllCaseAreHandle; // ensure type will be return number
}
```