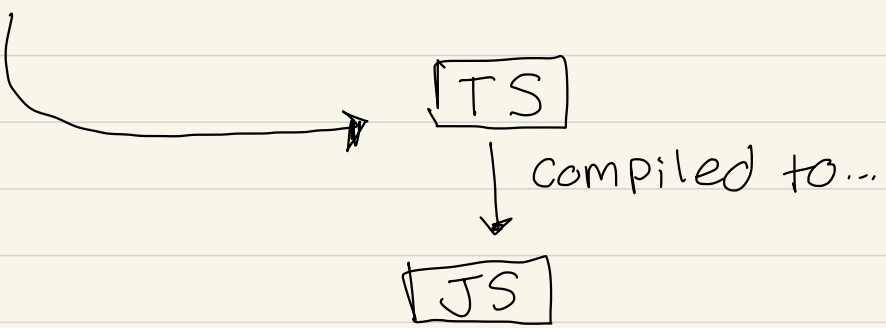
# Understanding Typescript

What is TypeScript?

* Adds new features + Advantages to JS
  - Browser can't execute it!

* Is a JS superset
  - A language building up on JS

* Features are compiled to JS "workarounds", possible errors are thrown



# Why TypeScript?

```
function add(num1, num2){
  return num1 + num2;
}

console.log(add('2', '3'));
```

unwanted behavior at runtime → 23 not 5

Mitigation strategies: Add if check to add function
Validate and sanitize user input.

Developers can still write invalid code!

Typescript is a "tool" that helps developers write better code!

HTML File

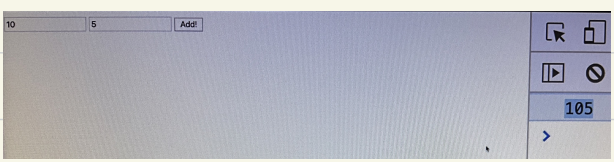
```html
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Understanding TypeScript</title>
    <script src="js-only.js" defer></script>
  </head>
  <body>
    <input type="number" id="num1" placeholder="Number 1" />
    <input type="number" id="num2" placeholder="Number 2" />
    <button>Add!</button>
  </body>
</html>
```

JS File (only JS)

```js
const button = document.querySelector("button");
const input1 = document.getElementById("num1");
const input2 = document.getElementById("num2");

function add(num1, num2) {
  return num1 + num2;
}

button.addEventListener("click", function() {
  console.log(add(input1.value, input2.value));
});
```




→ concatenation

Not expected result

In JS always the value of an input is a string!!!

```js
const button = document.querySelector("button");
const input1 = document.getElementById("num1");
const input2 = document.getElementById("num2");

function add(num1, num2) {
  if (typeof num1 === "number" && typeof num2 === "number") {
    return num1 + num2;
  } else {
    return +num1 + +num2;
  }
}

button.addEventListener("click", function() {
  console.log(add(input1.value, input2.value));
});
```

Fix using only JS (vanilla JS)

But it can be improved...

index.html > html > head > script

```html
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Understanding TypeScript</title>
    <script src="using-ts.js" defer></script>
  </head>
  <body>
    <input type="number" id="num1" placeholder="Number 1" />
    <input type="number" id="num2" placeholder="Number 2" />
    <button>Add!</button>
  </body>
</html>
```

HTML File

using-ts.js > ...

```js
var button = document.querySelector("button");
var input1 = document.getElementById("num1");
var input2 = document.getElementById("num2");
function add(num1, num2) {
    return num1 + num2;
}
button.addEventListener("click", function () {
    console.log(add(+input1.value, +input2.value));
});
```

JS File using TS

using-ts.ts > ...

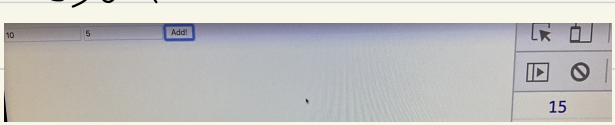
```ts
const button = document.querySelector("button");
const input1 = document.getElementById("num1")! as HTMLInputElement;
const input2 = document.getElementById("num2")! as HTMLInputElement;

function add(num1: number, num2: number) {
  return num1 + num2;
}

button.addEventListener("click", function() {
  console.log(add(+input1.value, +input2.value));
});
```

TS File

Result...

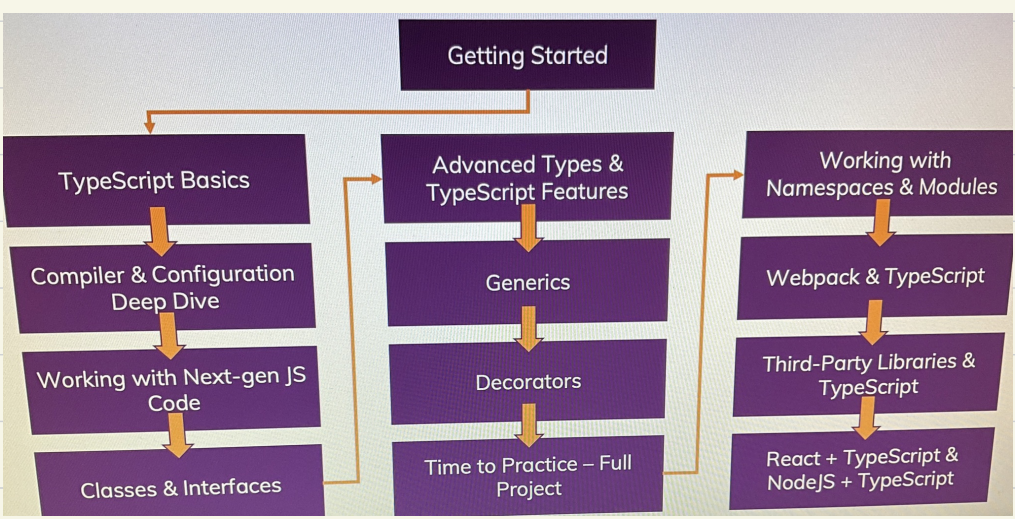



* Type annotations
* cleaner, better and less error prompted code.

Typescript adds...

* Types!
* Next-gen JS features (compiled down for older Browsers)
  - → Like Babel in JS
* Non-JS Features like interfaces or generics
* Meta-programming features like decorators
* Rich configuration options
* Modern tooling that helps even in non-TS projects

---



Compile TS file...

In a terminal run this...

"tsc app.ts"

↳ A JS file called app.js will be created in the project tree

Working with types...
Core syntax and features.

## Core types

* number → 1, 5.3, -10
* string → 'Hi', "Hi", `Hi`
* boolean → true, false

The key difference is...

JS uses "dynamic types" (resolved at runtime)

TS uses "static types" (set during development)

---

## Important: Type Casing

In TypeScript, you work with types like `string` or `number` all the times.

**Important**: It is `string` and `number` (etc.), **NOT** `String`, `Number` etc.

**The core primitive types in TypeScript are all lowercase!**

## Core Types

| | | |
|---|---|---|
| number | 1, 5.3, -10 | All numbers, no differentiation between integers or floats |
| string | 'Hi', "Hi", `Hi` | All text values |
| boolean | true, false | Just these two, no "truthy" or "falsy" values |
| object | {age: 30} | Any JavaScript object, more specific types (type of object) are possible |

Nested objects and types...

## Nested Objects & Types

Of course object types can also be created for **nested objects**.

Let's say you have this JavaScript **object**:

```
const product = {
  id: 'abc1',
  price: 12.99,
  tags: ['great-offer', 'hot-and-new'],
  details: {
    title: 'Red Carpet',
    description: 'A great carpet - almost brand-new!'
  }
}
```

This would be the **type** of such an object:

```
{
  id: string;
  price: number;
  tags: string[];
  details: {
    title: string;
    description: string;
  }
}
```

So you have an object type in an object type so to say.

## Core Types

| | | |
|---|---|---|
| number | 1, 5.3, -10 | All numbers, no differentiation between integers or floats |
| string | 'Hi', "Hi", `Hi` | All text values |
| boolean | true, false | Just these two, no "truthy" or "falsy" values |
| object | {age: 30} | Any JavaScript object, more specific types (type of object) are possible |
| Array | [1, 2, 3] | Any JavaScript array, type can be flexible or strict (regarding the element types) |

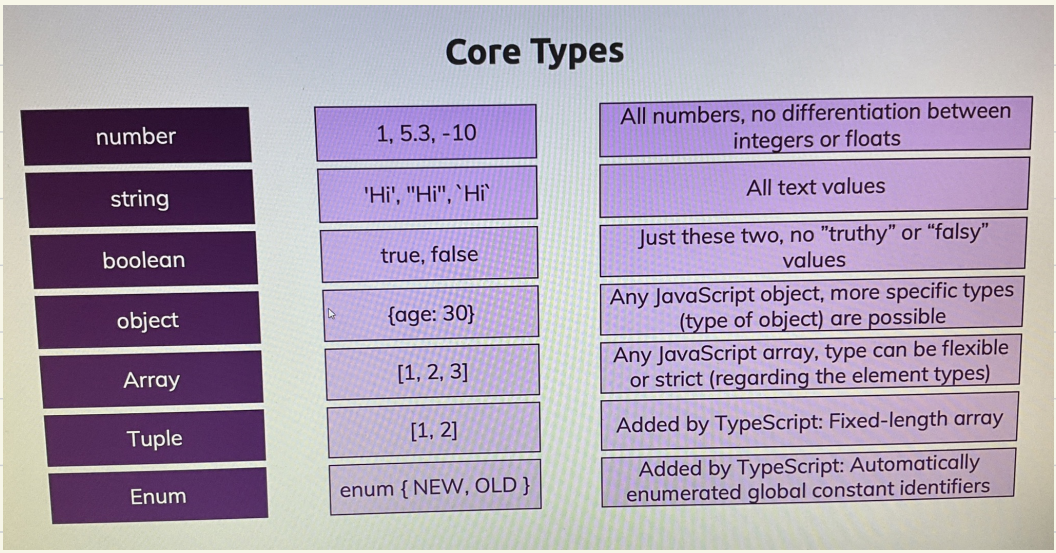
any in TS = do whatever you want

---

## Core Types

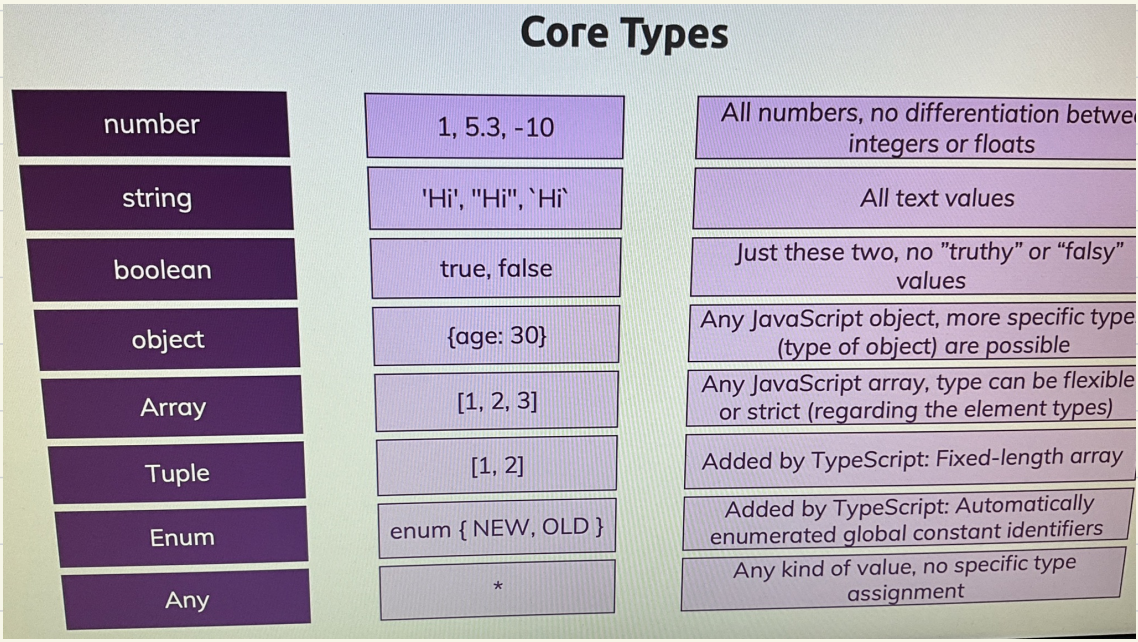
| | | |
|---|---|---|
| number | 1, 5.3, -10 | All numbers, no differentiation between integers or floats |
| string | 'Hi', "Hi", `Hi` | All text values |
| boolean | true, false | Just these two, no "truthy" or "falsy" values |
| object | {age: 30} | Any JavaScript object, more specific types (type of object) are possible |
| Array | [1, 2, 3] | Any JavaScript array, type can be flexible or strict (regarding the element types) |
| Tuple | [1, 2] | Added by TypeScript: Fixed-length array |

# Enum type...




Important: Often, you'll see enums with all-uppercase values but that's not a "must-do". You can go with any values names.

"Any" type => most flexible type that you can assign in TS.



# Type aliases and objects

## Type Aliases & Object Types

Type aliases can be used to "create" your own types. You're not limited to storing union types though - you can also provide an alias to a (possibly complex) object type.

For example:

```
type User = { name: string; age: number };
const u1: User = { name: 'Max', age: 30 }; // this works!
```

This allows you to avoid unnecessary repetition and manage types centrally.

For example, you can simplify this code:

```
function greet(user: { name: string; age: number }) {
  console.log('Hi, I am ' + user.name);
}

function isOlder(user: { name: string; age: number }, checkAge: number) {
  return checkAge > user.age;
}
```

To:

```
type User = { name: string; age: number };

function greet(user: User) {
  console.log('Hi, I am ' + user.name);
}

function isOlder(user: User, checkAge: number) {
  return checkAge > user.age;
}
```

Undefined ⇒ can be a valid type in TS.

But a function can't retur an undefined type.

# The Typescript Compiler

Configuring and using the TS compiler.

"Watch mode"

```
tsc app.ts --watch

or

tsc app.ts -w
```

```
... Starting compilation in watch mode...
```

---

To create a tsconfig.json file you should run:

```
tsc --init
```

# Including and excluding files...

In the tsconfig.json file you can place a code-block:

```
"exclude": [
    "analytics.ts"
]
```

this will exclude than while compilation process.

also...

```
*.dev.ts
**/*.dev.ts
```

also you can "include": [

]