# Lecture 4
# TypeScript

Frontend Web Development

100 SECONDS OF
JS
TS
TypeScript

## Why?

```
const languages = {
  it: { baseUrl: "/it" },
  en: { baseUrl: "/en" },
};

export default function indexPage() {
  const languageKey = window.localStorage.getItem("lang");

  const path = languages[languageKey].baseUrl;

  return redirect(path);
}
```

## A sneakier example

```
<body>
  <input id="input" type="number" />
  <p id="result">Can you guess the 2 bugs here?</p>
  <script>
    const p = document.getElementById('result');
    const input = document.getElementById('result');
    function sum(a, b) { return a + b; }
    p.textContent = sum(input.value, 10);
  </script>
</body>
```

*It works =/= It's correct*!

## Typos

```
const announcement = "Hello World!";

// How quickly can you spot the typos?
announcement.toLocaleLowercase();
announcement.toLocalLowerCase();

// We probably meant to write this...
announcement.toLocaleLowerCase();
```

## Uncalled functions

```
function flipCoin() {
  // Meant to use Math.random()
  return Math.random < 0.5;
  Operator '<' cannot be applied to types '() => number' and 'number'
}
```

## Basic logic errors

```
const value = Math.random() < 0.5 ? "a" : "b";
if (value !== "a") {
  // ...
} else if (value === "b") {
  This condition will always return 'false' since the types '"a"' and
  '"b"' have no overlap
  // Oops, unreachable
}
```

DYNAMIC TYPING
STATIC TYPING
DONE!

# TypeScript

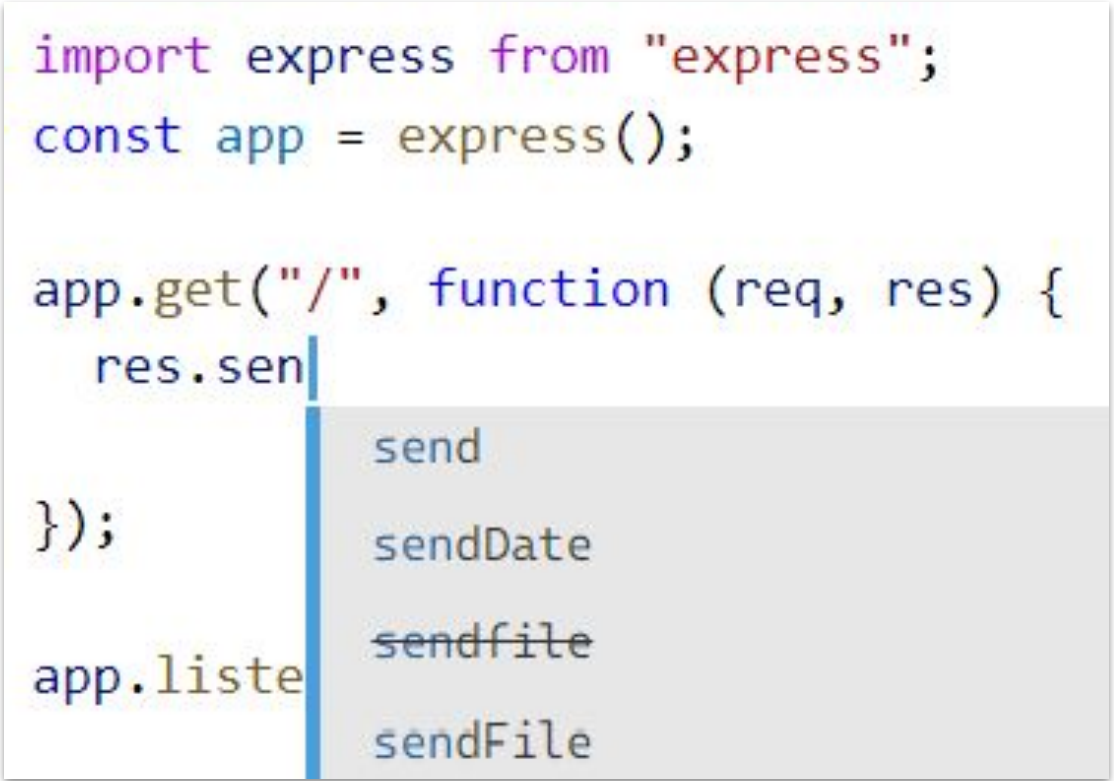
## What is TypeScript?

- A statically-typed superset of JavaScript
  - Can be progressively adopted
  - Inter-operates with existing JS code
  - Structurally-typed
- Just a compiler, no runtime
  - Transpiles to JavaScript
- Developed by Microsoft in 2012
- Open-source

## Advantages

- *Optional* static typing
- Better code readability
- IDE support (auto-completion)
- Easier refactoring
- Integration of newer ES standards
    - built-in transpiling (downleveling) to older versions
- Massive community

## IDE support

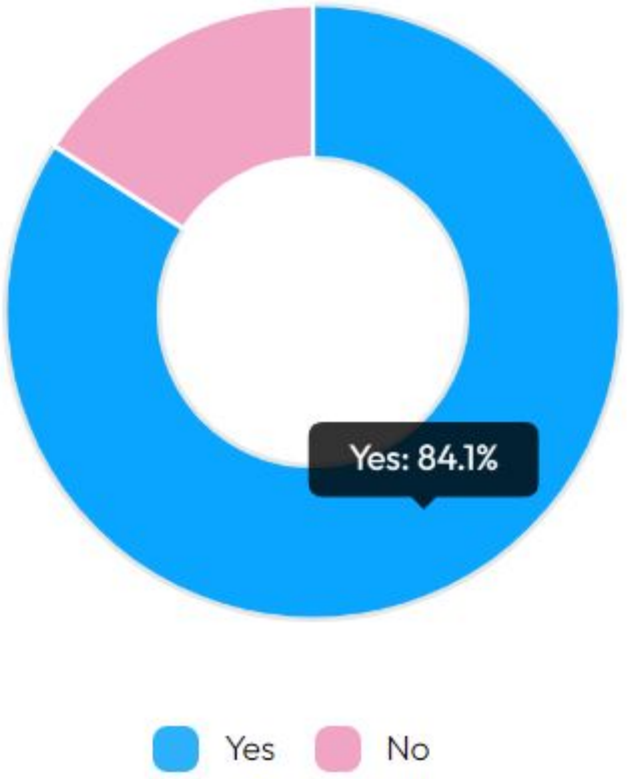


```
import express from "express";
const app = express();

app.get("/", function (req, res) {
  res.sen

});

app.liste
```

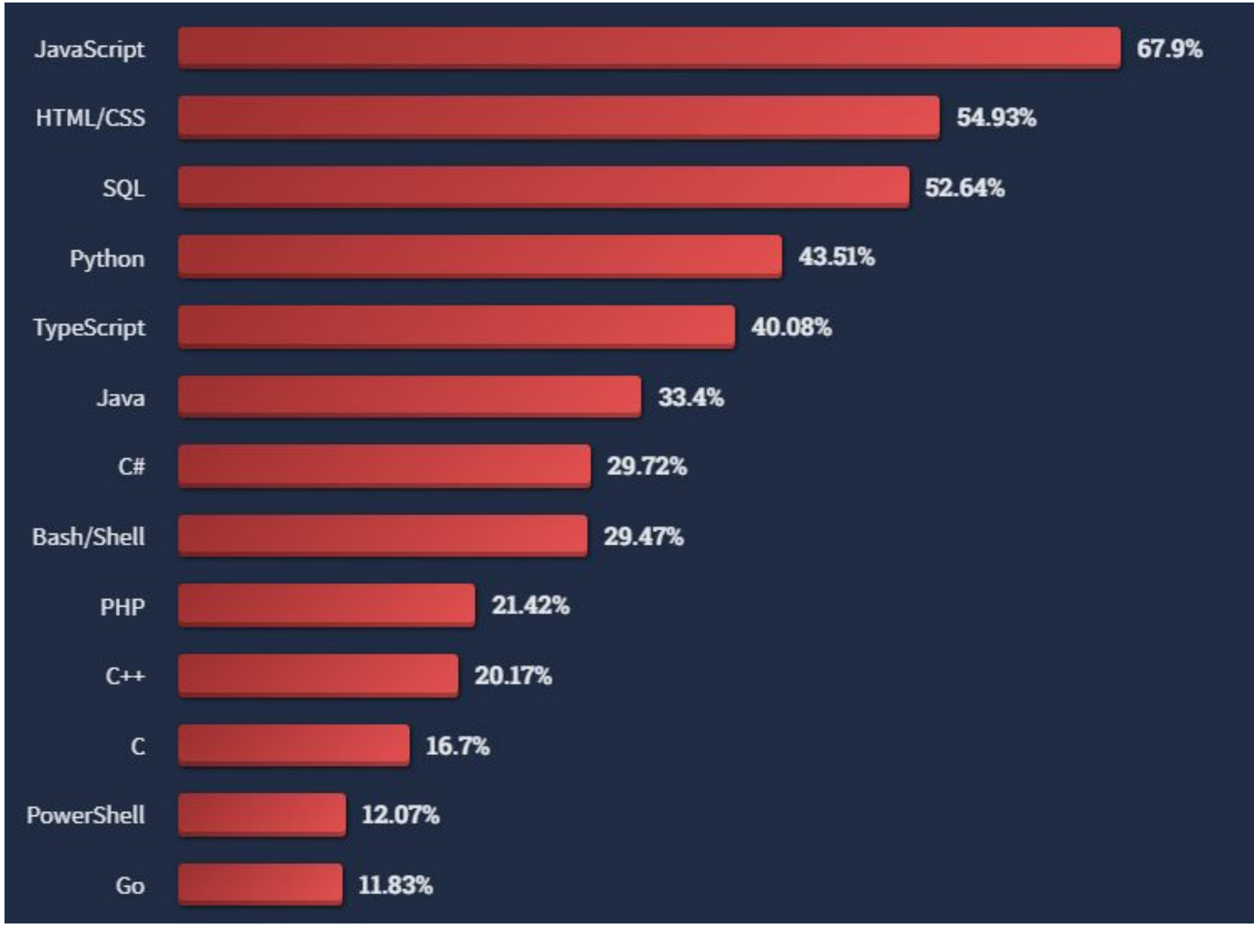
## The State of Frontend 2022

“Over the last year, have you used TypeScript?”

https://tsh.io/state-of-frontend/#typescript



## Stack Overflow Developer Survey 2022



https://survey.stackoverflow.co/2022

TypeScript
JavaScript

## Basics of typing

```
let myName: string = "Alice";

function greet(person: string, date: Date): void {
    console.log(`Hello ${person}, today is ${date.toDateString()}!`);
}

greet("Brendan");
```

```
Expected 2 arguments, but got 1
```

```
greet("Maddison", Date());
```

```
Argument of type 'string' is not assignable to parameter of type 'Date'
```

## Basics of typing: Type erasure

Compiled output:

```
"use strict";
function greet(person, date) {
    console.log("Hello " + person + ", today is " +
date.toDateString() + "!");
}
greet("Maddison", new Date());
```

## Basics of typing: Built-in types

- All primitives: `string`, `number`, `boolean, …`
  - Note: never use the capital (`String`, `Number`, `Boolean`) alternatives!
- Arrays: `number[]`, or `Array<number>`
- Tuples: `[number, number]`
  - Actually just arrays under the hood
- `any`: turns off type-checking entirely
- `unknown`: as the name implies 🙂

## any

When you don't know what type you should use in TypeScript
any

## Interfaces

```
interface PaintOptions {
   shape: string;
   xPos?: number;
   yPos?: number;
}

function paintShape(opts: PaintOptions) {
   // ...
}
```

## Extending Interfaces

```
interface Colorful {
   color: string;
}

interface Circle {
   radius: number;
}

interface ColorfulCircle extends Colorful, Circle {}

const cc: ColorfulCircle = {
   color: "red",
   radius: 42,
};
```

## Implementing Interfaces

```
interface Pingable {
   ping(): void;
}

class Sonar implements Pingable {
   ping() {
      console.log("ping!");
   }
}
```

## Type Inference

```
let msg = "hello there!";
```

```
let msg: string
```

```
let x = [0, 1, null];
```

```
let x: (number | null)[]
```

## Type Inference: Contextual typing

```
const names = ["Alice", "Bob", "Eve"];

names.forEach(function (s) {
  console.log(s.toUppercase());
  Property 'toUppercase' does not exist on type 'string'. Did you mean 'toUpperCase'?
});
```

## Generics

```
// Only numbers
function identity(arg: number): number {
   return arg;
}

// Any type for input, and not necessarily the same type for output
function identity(arg: any): any {
   return arg;
}

// Generic type
function identity<Type>(arg: Type): Type {
   return arg;
}
```

## Type assertion (casting)

```
const myCanvas = document.getElementById("main_canvas") as
HTMLCanvasElement;
```

Only allows more specific or less specific type casts (cannot be unrelated)

```
const x = "hello" as number;
```

```
Conversion of type 'string' to type 'number' may be a mistake because neither type
sufficiently overlaps with the other
```

```
const y = ("hello" as any) as number;
```

## Do not abuse *`any`*

Code smell.

Removes all the benefits of static typing (safety, IDE support, ...).

## Type aliases

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

type UserInputSanitizedString = string;

function sanitizeInput(str: string): UserInputSanitizedString {
 return sanitize(str);
}

// Create a sanitized input
let userInput = sanitizeInput(getInput());

// Can still be re-assigned with a string though
userInput = "new input";
```

## Function types

```
type Logger = (msg: string) => void;

let toFixed: (digits: number | undefined) => string;


function add(a: string, b: string): string;
function add(a: number, b: number): number;
function add(a: any,    b: any):    any {
  return a + b;
}
```

## Union types & type narrowing

```
type ID = number | string;

function printId(id: number | string) {
  if (typeof id === "string") {
    // In this branch, id is of type 'string'
    console.log(id.toUpperCase());
  } else {
    // Here, id is of type 'number'
    console.log(id);
  }
}
```

## Dependent/Literal types

Values can be used as types.

```
type Alignment = 'left' | 'right' | 'center';
function printText(s: string, alignment: Alignment) { /**/ }


printText("Hello, world", "left");
printText("Top of the mornin' to ya", "centre");
```

```
Argument of type '"centre"' is not assignable to parameter of type '"left" | "right" | "center"'
```

## Discriminated unions

```
interface Shape {
  kind: "circle" | "square";
  radius?: number;
  sideLength?: number;
}
function getArea(shape: Shape) {
  if (shape.kind === "circle") {
    return Math.PI * shape.radius ** 2;
                 Object is possibly 'undefined'.
  }
}
```

## Discriminated unions

```
interface Circle {
 kind: "circle";
 radius: number;
}

interface Square {
 kind: "square";
 sideLength: number;
}

type Shape = Circle | Square;
```

```
function getArea(shape: Shape) {

  return Math.PI * shape.radius ** 2;
```

```
Property 'radius' does not exist on type 'Shape'
 Property 'radius' does not exist on type 'Square'
```

```
}

function getArea(shape: Shape) {
  if (shape.kind === "circle") {
    return Math.PI * shape.radius ** 2;

  }
}
```

```
(parameter) shape: Circle
```

## Utility types

```
interface Todo {
  title: string;
  description: string;
  completed: boolean;
}

function updateTodo(todo: Todo, fieldsToUpdate: Partial<Todo>) {
  return { ...todo, ...fieldsToUpdate };
}

type TodoPreview = Pick<Todo, "title" | "completed">;
```

## Utility types

```
type Payload = Record<string, any>;
type A = Awaited<Promise<string>>; // type A = string
type B = ReturnType<() => number>; // type B = number
type C = Parameters<(s: string) => void>; // type C = [s: string]
```

**Intrinsic string manipulation types**:

```
type HTTPVerb = 'get' | 'post';
type Method = Uppercase<HTTPVerb>; // ‘GET’ | ‘POST’
```

[Utility Types (The TypeScript Handbook)]

## Structural typing

https://www.typescriptlang.org/play/?q=482#example/structural-typing

## DefinitelyTyped

A massive repo for 3rd-party TypeScript definitions for libraries without official TS support.

Around 16k contributors

Simply `npm install --save-dev @types/lodash`

https://github.com/DefinitelyTyped/DefinitelyTyped

## Resources

- TypeScript Playground: https://www.typescriptlang.org/play
- The TypeScript Handbook: https://www.typescriptlang.org/docs/handbook/intro.html
- TS in 5 minutes: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Remember: Entities specific to TypeScript (like interfaces, type aliases, enums, etc.) do not exist in runtime and are either only used in compilation process (e.g. interfaces) or compiled into pure plain JavaScript (e.g. enums).