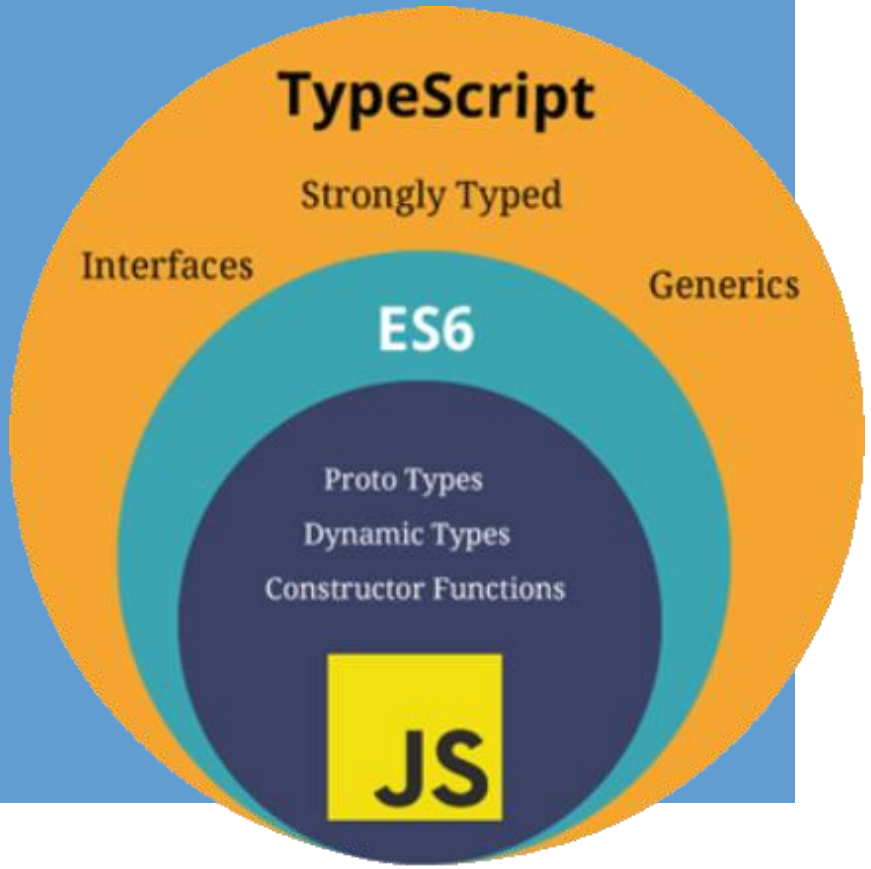
TYPESCRIPT FUNDAMENTALS

Prof. Andrew Sheehan

Boston University
Metropolitan College
Computer Science Department

WHAT IS TYPESCRIPT

TypeScript is a strongly typed programming language that builds on JavaScript



CREATED BY

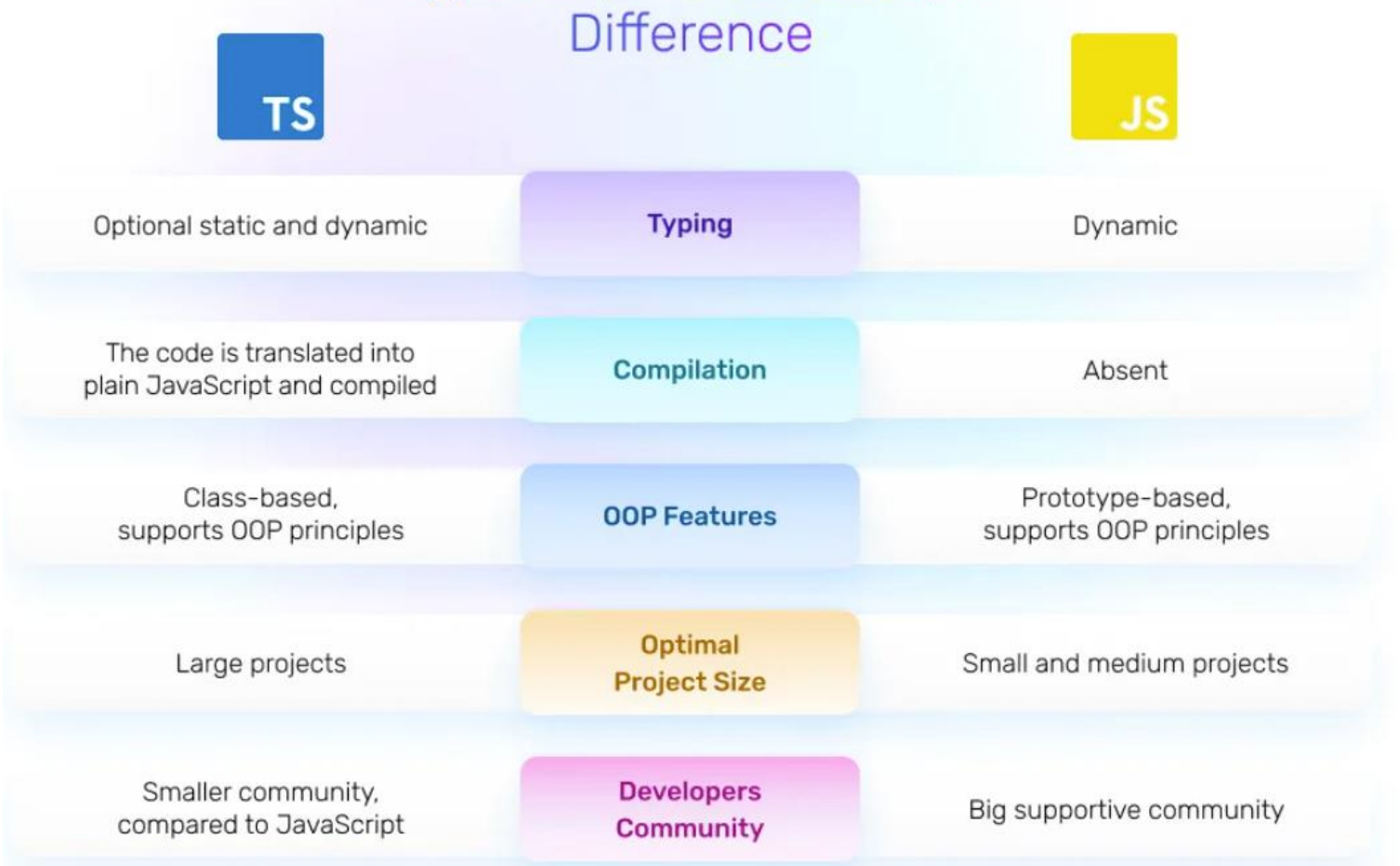
Microsoft

TypeScript

Paradigm	Multi-paradigm: functional, generic, imperative, object-oriented
Designed by	Microsoft
Developer	Microsoft
First appeared	1 October 2012; 10 years ago[1]
Stable release	5.0.4[2] / 7 April 2023; 41 days ago
Typing discipline	Duck, gradual, structural[3]
License	Apache License 2.0
Filename extensions	.ts, .tsx, .mts, .cts
Website	www.typescriptlang.org
Influenced by	
C#, Java, JavaScript, ActionScript[4]	
Influenced	
AtScript, AssemblyScript	

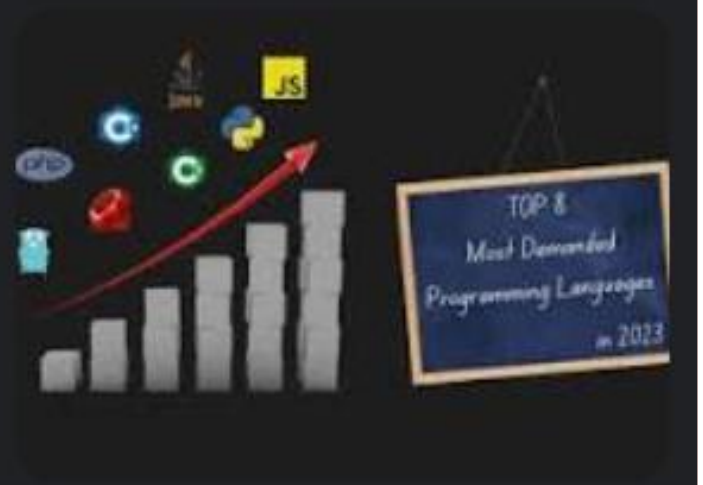
https://en.wikipedia.org/wiki/TypeScript

Typescript vs Javascript: Difference



Top 8 Most Demanded Programming Languages in 2023

1. 1 - JavaScript / TypeScript. Since its creation to make the firsts websites dynamic, JavaScript hasn't stopped gaining popularity over the years. ...
2. 2 - Python. ...
3. 3 - Java. ...
4. 4 - C# ...
5. 5 - PHP. ...
6. 6 - C/C++ ...
7. 7 - Ruby. ...
8. 8 - GO.



HOW TO COMPILE?

```
# Run a compile based on a backwards look through the fs for a tsconfig.json
tsc

# Emit JS for just the index.ts with the compiler defaults
tsc index.ts

# Emit JS for any .ts files in the folder src, with the default settings
tsc src/*.ts

# Emit files referenced in with the compiler settings from tsconfig.production.json
tsc --project tsconfig.production.json

# Emit d.ts files for a js file with showing compiler options which are booleans
tsc index.js --declaration --emitDeclarationOnly

# Emit a single .js file from two files via compiler options which take string arguments
tsc app.ts util.ts --target esnext --outfile index.js
```

```
andre@Andrew_ThinkPad /cygdrive/c/Projects/typescript/hello
$ npx -p typescript tsc
```

EXAMPLES(UPDATED)

VAR, LET AND CONST

You should know these differences.

var declarations are global while let and const are block scoped.

var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.

All are hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.

Primitives

string, number and boolean

Do Not Use: String, Number and Boolean (uppercased)

> The type names `String`, `Number`, and `Boolean` (starting with capital letters) are legal, but refer to some special built-in types that will very rarely appear in your code. *Always* use `string`, `number`, or `boolean` for types.

DATA TYPES

Any

Allows any valid javascript data type in your expression

Unknown

TypeScript 3.0 introduced a new `unknown` type which is the type-safe counterpart of the `any` type.

The main difference between `unknown` and `any` is that `unknown` is much less permissive than `any` : we have to do some form of checking before performing most operations on values of type `unknown` , whereas we don't have to do any checks before performing operations on values of type `any` .

DATA
TYPES

Never

Means "something will never occur."

For example, a function that never returns anything.

```
function throwError(errorMsg: string): never {
            throw new Error(errorMsg);
}
```

DATA TYPES

Array

Examples: string[], number[], boolean[]

DATA TYPES

Annotate the parameter and return data types

```
function greeting(name: string) :string
```

CONTEXTUAL TYPING

Typescript can figure out the type in your expression when it is not explicated indicated.

```
1  // No type annotations here, but TypeScript can spot the bug
2  const names = ["Alice", "Bob", "Eve"];
3
4  // Contextual typing for function
5  names.forEach(function (s) {
6    console.log(s.toUppercase());
7  });
8
9  // Contextual typing also applies to arrow functions
10 names.forEach((s) => {
11   console.log(s.toUppercase());
12 });
```

Object

Any JavaScript value that has properties.

```
const human = {
    name: string,
    age: number,
    dob: Date
}
```

Optional

properties

```
const human = {
    name: string,
    age?: number,
    dob: Date
}
```

Not assigned a value?

You will receive undefined

Union

```
function printId(id: number | string) {
  console.log(`SYS ${date()} ID: ${id}`);
}
```

The arguments must be of one these 2 types

It might be confusing that a *union* of types appears to have the *intersection* of those types' properties. This is not an accident - the name *union* comes from type theory. The *union* `number | string` is composed by taking the union *of the values* from each type. Notice that given two sets with corresponding facts about each set, only the *intersection* of those facts applies to the *union* of the sets themselves. For example, if we had a room of tall people wearing hats, and another room of Spanish speakers wearing hats, after combining those rooms, the only thing we know about *every* person is that they must be wearing a hat.

DATA TYPES

Type Alias

```
type ThreeDPoint = {
    x: number,
    y: number,
    z: number
}

type address_line1 = string;  // type alias
```

Note: Types cannot be implemented or extended, like you can with interface definitions

DATA TYPES

Interface

```
interface ThreeDPoint = {
    x: number,
    y: number,
    z: number
}
```

Good read: https://blog.logrocket.com/types-vs-interfaces-typescript/#differences-between-typescript-interfaces

Type v Interface

Type aliases and interfaces are very similar

Almost all the features of an interface are available in type

A type cannot add a new property after you define it. An interface is always extendable

THE CLASS

```
class Human {
    gender: string,
    date: Date,
    height: number,
    weight: number
}
```