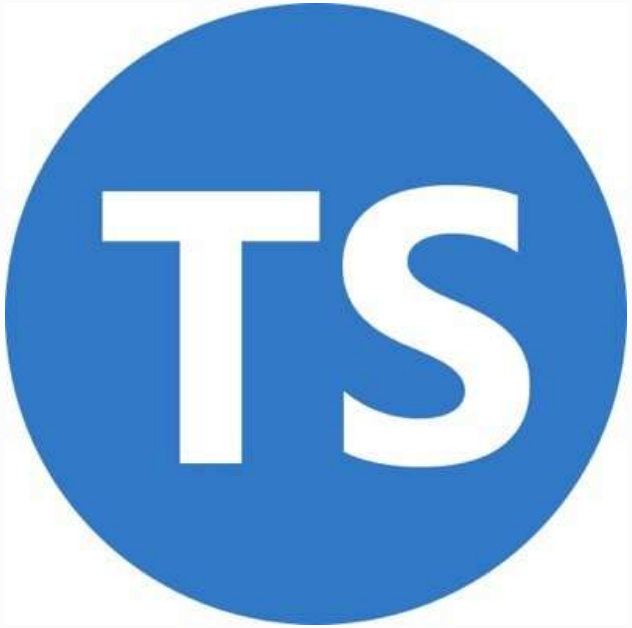
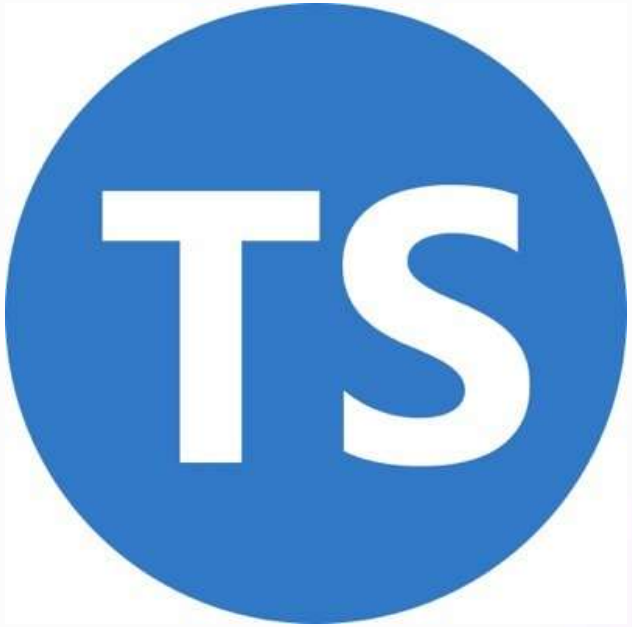
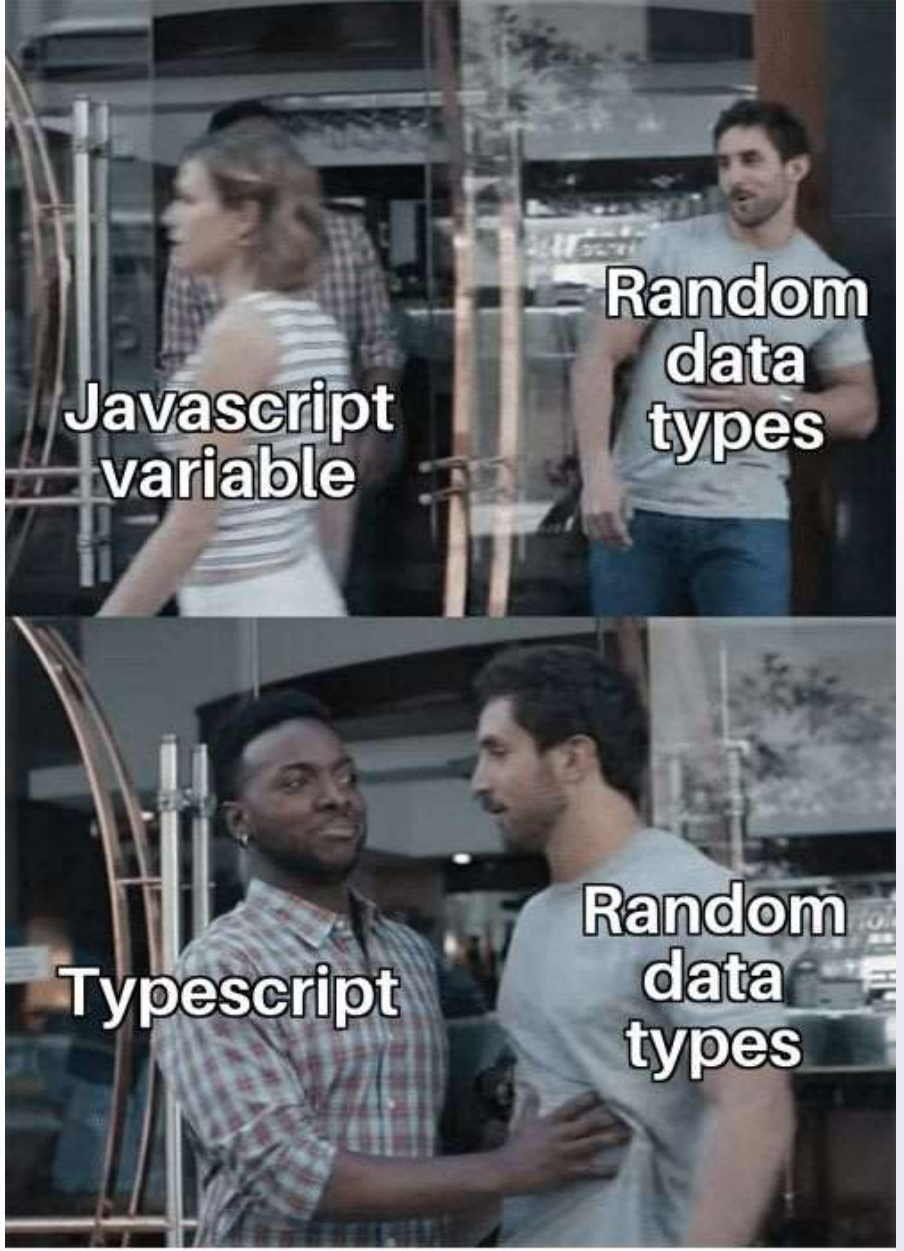
sourcemind

TypeScript

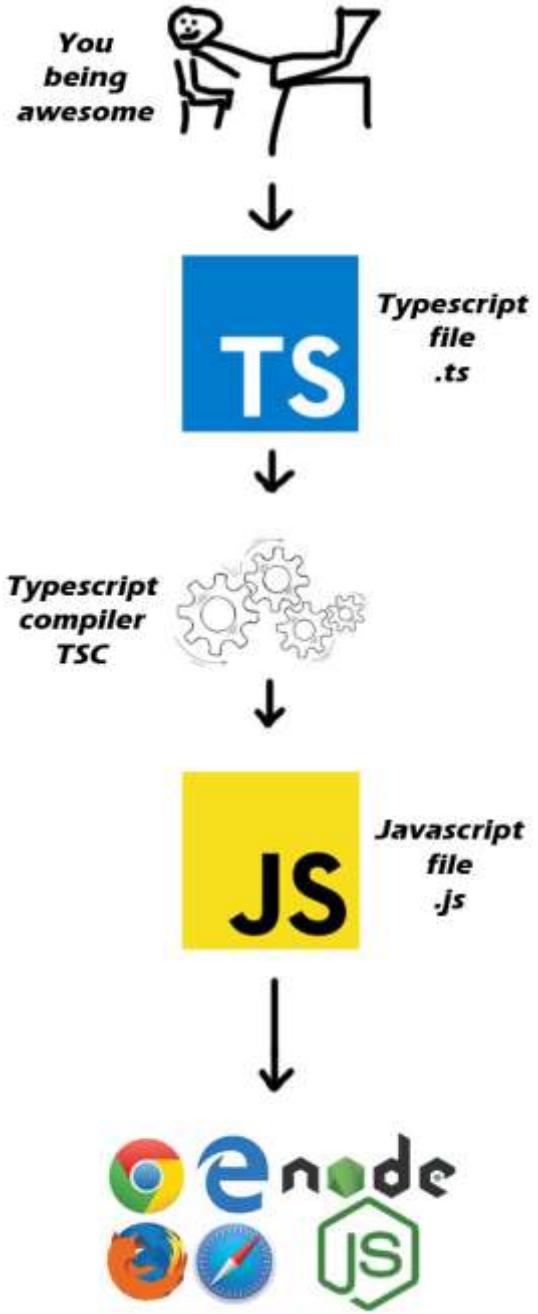




TypeScript is **JavaScript with syntax for types.**
TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.



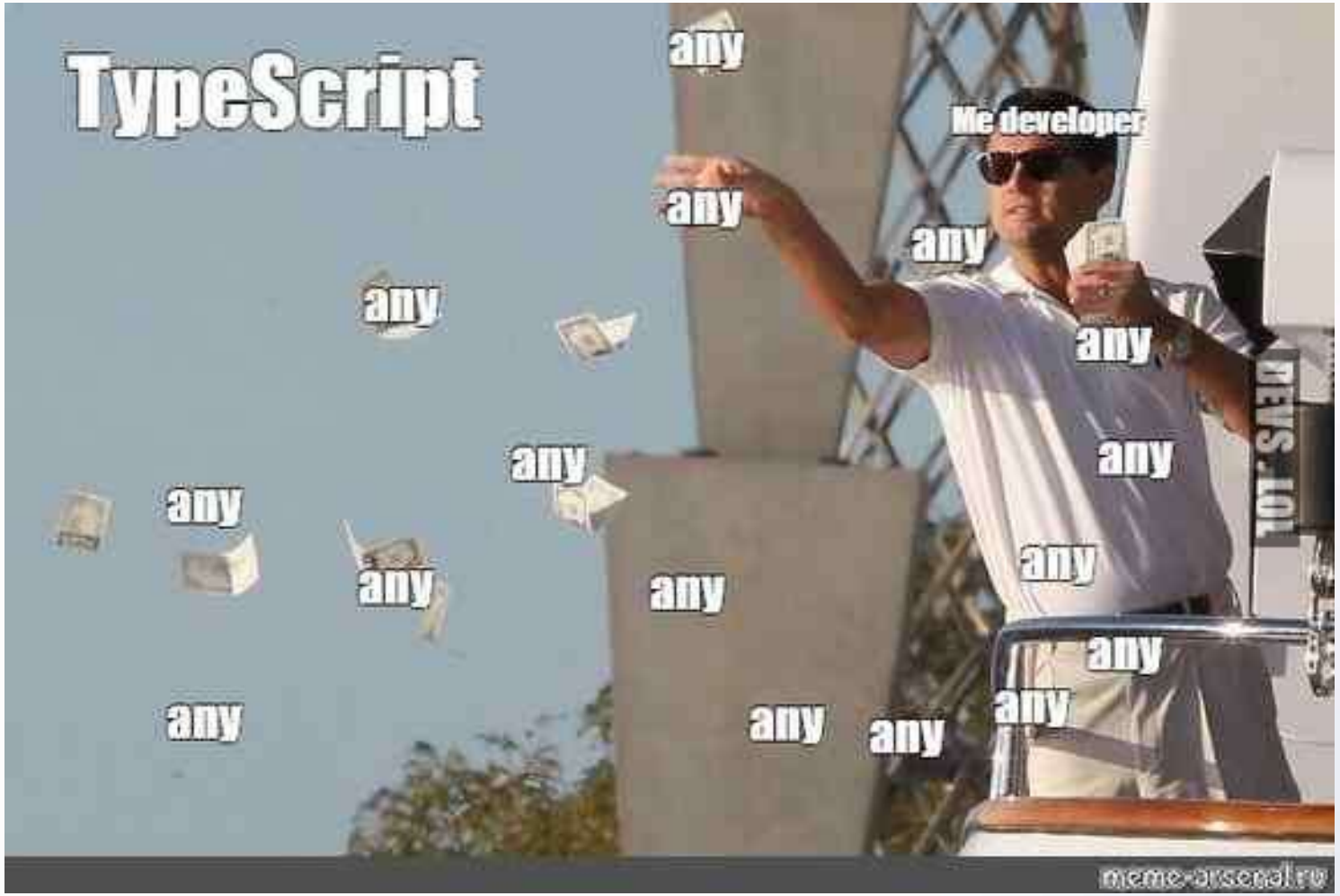
I don't think you're allowed to go here Mr. Data



- JavaScript is not originally designed for large complex applications (mostly a scripting language, with functional programming constructs), lacks structuring mechanisms like Class, Module, Interface.
- Typescript is a typed superset of JavaScript that compiles to plain JavaScript.
- Adds additional features like Static Type (optional), Class, Module etc. to JavaScript
- Microsoft technology.
- Open Source.

Type Annotation

- Any
  - Any Type is a super set of all types
    - let x : any;
    - let y;
- Primitive
  - Number
    - Does not have separate integer and float/double type.
    - let num : number = 20;
    - let num = 20;
  - String
    - Both single quote or double quotes can be used.
    - let name : string = “hello”;
    - let name =’hello’;
  - Bool
    - let isOpen =true;


TypeScript
Me developer
any
DEVS 101
meme-arsenal.ru


TypeScript code
as any

Functions

```
// Parameter type annotation
function greet(name: string) {
  console.log("Hello, " + name.toUpperCase() + "!!");
}
```

```
// Would be a runtime error if executed!
greet(42);
```

```
Argument of type 'number' is not assignable to parameter of type 'string'.
```

Return Type Annotations

```
function getFavoriteNumber(): number {
  return 26;
}
```

Object Types

```
// The parameter's type annotation is an object type
function printCoord(pt: { x: number; y: number }) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}
printCoord({ x: 3, y: 7 });
```

Optional Properties

Object types can also specify that some or all of their properties are *optional*. To do this, add a `?` after the property name:

```
function printName(obj: { first: string; last?: string }) {
  // ...
}
// Both OK
printName({ first: "Bob" });
printName({ first: "Alice", last: "Alisson" });
```

Union Types

```
function printId(id: number | string) {
  console.log("Your ID is: " + id);
}
// OK
printId(101);
// OK
printId("202");
// Error
printId({ myID: 22342 });
```

```
Argument of type '{ myID: number; }' is not assignable to parameter of type 'string | number'.
```

Try

Type Aliases

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

// Exactly the same as the earlier example
function printCoord(pt: Point) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}

printCoord({ x: 100, y: 100 });
```

Interface

An *interface declaration* is another way to name an object type:

```
interface Point {
  x: number;
  y: number;
}

function printCoord(pt: Point) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}

printCoord({ x: 100, y: 100 });
```


I'M TRYING TO
SPOT THE DIFFERENCE!
meme-arsenal.ru

Differences Between Type Aliases and Interfaces

Interface	Type
Extending an interface	Extending a type via intersections

`interface Animal {`
`  name: string`
`}`
 	
`interface Bear extends Animal {`
`  honey: boolean`
`}`
 	
`const bear = getBear()`
`bear.name`
`bear.honey`
	`type Animal = {`
	`  name: string`
	`}`
	
	`type Bear = Animal & {`
	`  honey: boolean`
	`}`
	
	`const bear = getBear();`
	`bear.name;`
	`bear.honey;`

Adding new fields to an existing interface

```
interface Window {
  title: string
}

interface Window {
  ts: TypeScriptAPI
}

const src = 'const a = "Hello World"';
window.ts.transpileModule(src, {});
```

A type cannot be changed after being created

```
type Window = {
  title: string
}

type Window = {
  ts: TypeScriptAPI
}

 // Error: Duplicate identifier 'Window'.
```

Using With React

```
npx create-react-app my-app --template typescript
```

```
class App extends React.Component<{ message: string }, { count: number }> {
  state = { count: 0 };
  render() {
    return (
      <div onClick={() => this.increment(1)}>
        {this.props.message} {this.state.count}
      </div>
    );
  }
  increment = (amt: number) => {
    // like this
    this.setState((state) => ({
      count: state.count + amt,
    }));
  };
}
```

sourcemind