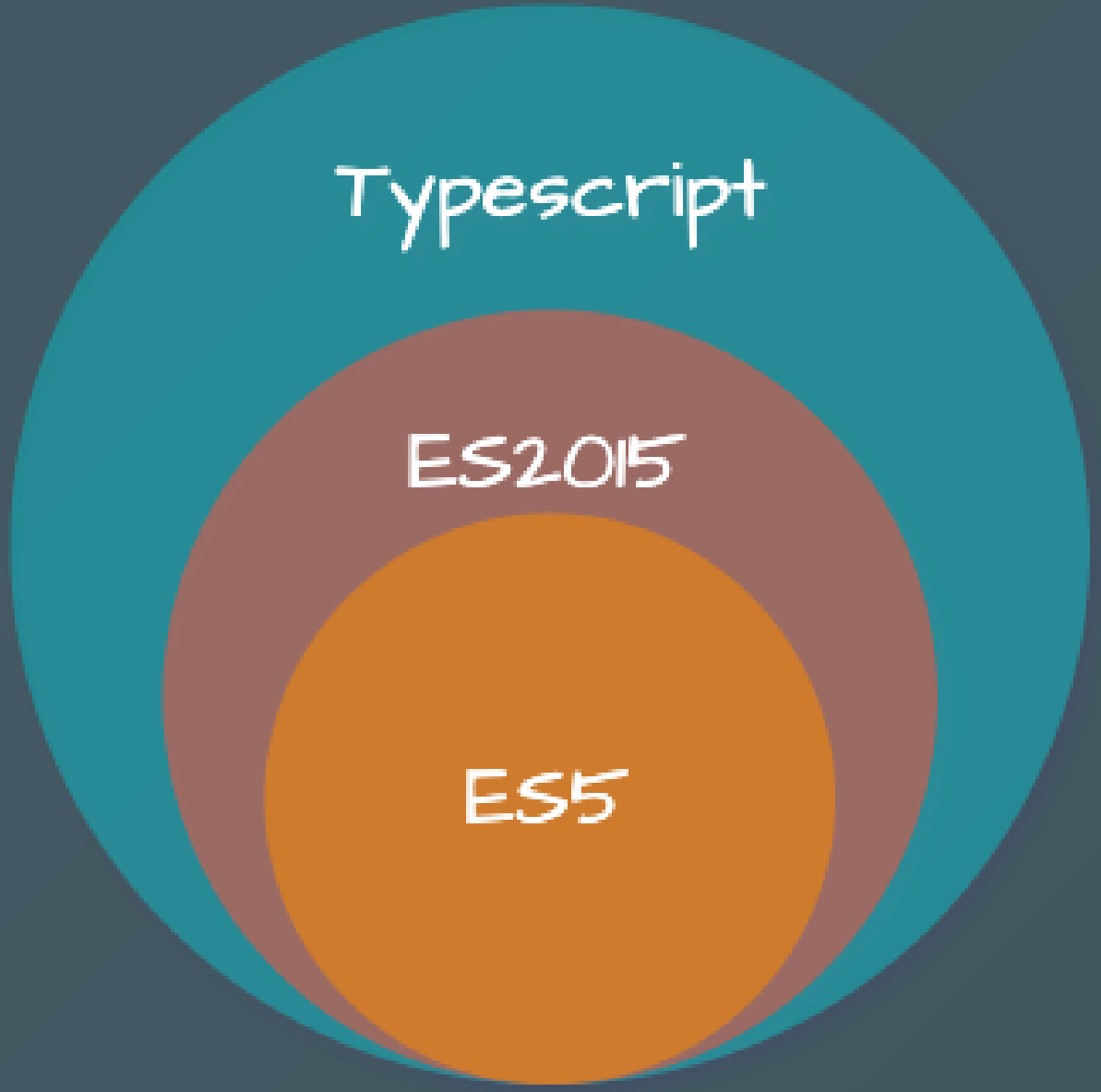
# Typescript

- What is Typescript?
- Why Typescript?
- Typescript vs. Javascript

# What is Typescript?

- Typescript is a superset of Javascript
- Typescript is a strongly typed language
- Typescript is compiled to Javascript



# Why Typescript?

 why-typescript.js why-typescript.ts 

- Type safety
- Better tooling
- Improved scalability
- Improved maintainability
- Compatibility with Javascript

# Typescript vs. Javascript

| Typescript | Javascript |
| --- | --- |
| Compiled | Interpreted |
| Static typing | Dynamic typing |
| Compile-time errors | Runtime errors |
| Transpiled to Javascript | Direct use in browser |
| More verbose | Less verbose |

# Install Typescript

https://www.typescriptlang.org/download

```
npm install -g typescript
```

# Compile Typescript

```
tsc <filename>.ts
```

# Typescript Types

types.ts  

- Boolean
- Number
- String
- Array
- Tuple
- Enum

- Object

# Type Assertion

assertion.ts

- `as` keyword

```
const message = 'Hello World';
const length = (message as string).length;
```

# Type Guard

type-guard.ts  

- `typeof` keyword
- `number`, `string`, `boolean`, `symbol`

```
const message = 'Hello World';
if (typeof message === 'string') {
  const length = message.length;
}
```

- `instanceof` keyword

# Types Union

union.ts

- Union: `type1 | type2`
  - Narrowing down the type

# Discriminated Union

discriminated-union.ts

```
interface Shape {
  kind: 'circle' | 'square';
  radius?: number;
  sideLength?: number;
}
```

# Type Aliases

alias.ts

- `type` keyword

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

# Type Interface

interface.ts

- `interface` keyword

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

- Type alias vs. Type interface

# Typescript Functions

functions.ts

- Function type expression
- Constructor type expression
- Generic function
- Optional and default parameters

# Optional and Default Parameters

```
function log(message: string, userId?: string): void {
  const time = new Date().toLocaleTimeString();
  console.log(time, message, userId || 'Not signed in');
}
```

```
function log(message = 'logging', userId?: string): void {
  const time = new Date().toLocaleTimeString();
  console.log(time, message, userId || 'Not signed in');
}
```

# Function Overloading

overloading.ts

- Function overloading is a way to provide multiple function signatures for the same function name
- Function overloading is *not* supported in Javascript

# Object Types

object-types.ts

- Optional properties
- Readonly properties
- Excess property checks
- Index signatures
- Extending object types

# Classes

classes.ts

- Class members
- Constructor
- Inheritance
- Member Visibility
- Generic classes

# Guidances for writing generics

generics.ts

1. Use multiple type parameters to describe all the types of a function's arguments
2. Use `extends` keyword to constrain the type parameters
3. Use `keyof` keyword to describe index types
4. Use mapped types to describe generic objects
5. Use type parameters to describe relationships between arguments
6. Use type parameters to enforce relationships between return types and argument types
7. Use type parameters to describe the shape of callbacks

# Utility Types

utility-types.ts

- `Partial<T>`
- `Readonly<T>`
- `Record<K, T>`
- `Pick<T, K>`
- `Omit<T, K>`
- `Exclude<T, U>`
- `Extract<T, U>`
- `ReturnType<T>`

...