Introduction to: TypeScript

Passionate People

Passionate People

Frontend Development Consultancy

Tomas Corral Casas

Head of Frontend Technologies and Academy

Consulting for: BACKBASE

Who created TypeScript?

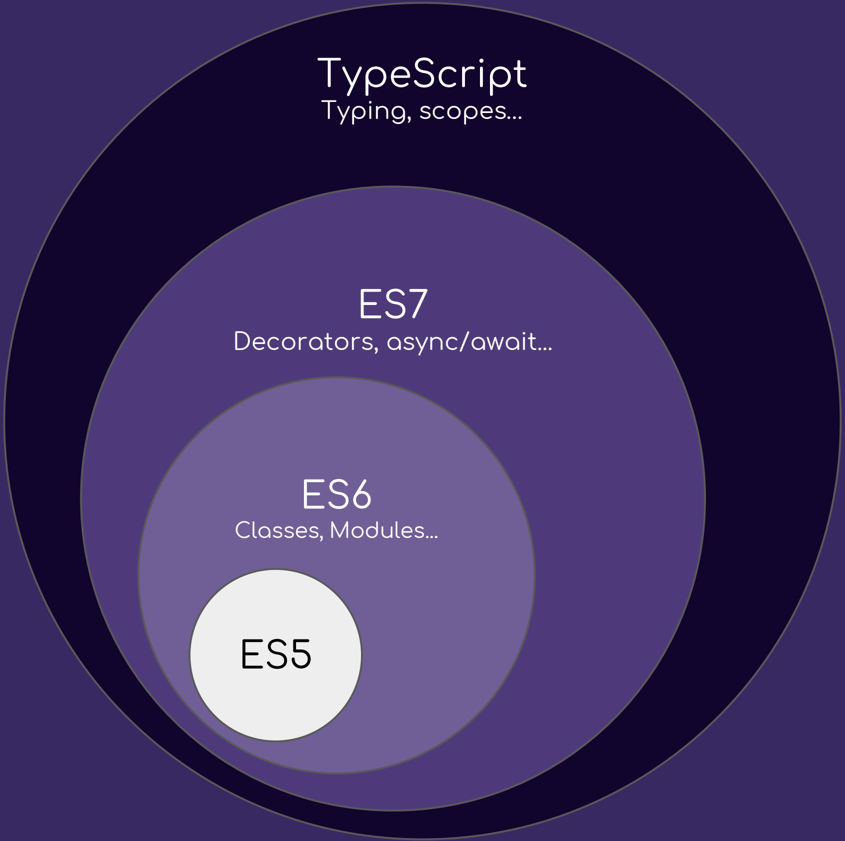
Anders
Hejlsberg

Passionate People

What is TypeScript?

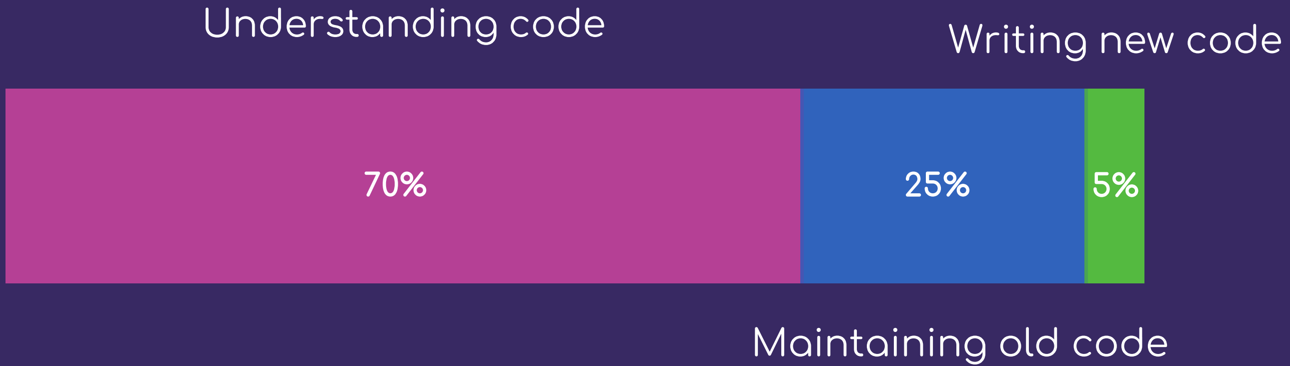
“ TypeScript is a
typed superset of JavaScript
that compiles to plain JavaScript. ”

Passionate People



Passionate People

What does a developer spend time on?



Passionate People

Before Typescript

Hungarian Notation

Hungarian notation

Prefix	Type	Example
b	Boolean	var bIsVisible = true;
e	DOM element	var eHeader = document.getElementById('header');
f	Function	var fCallback = function () { ... };
n	Number	var nSides = 4;
o	Object	var oPoint = new Point(200, 345);
a	Array	var aNames = ['Luke', 'Nick', 'David'];
s	String	var sUrl = 'http://www.google.com';
$	jQuery	var $Header = $('#header');

Passionate People

Scope

Prefix	Type	Example
_	Private	var _transformData = function (data) { ... return result; }
	Public	var render = function () { ... };
🚫	Protected	🚫

Passionate People

JSDoc comments

Passionate People

JSDoc comments

@enum

@class

@public

@protected

@param {string} color

@private

@access

@constant

@constructor

@param {LACE_TYPES} type

@package

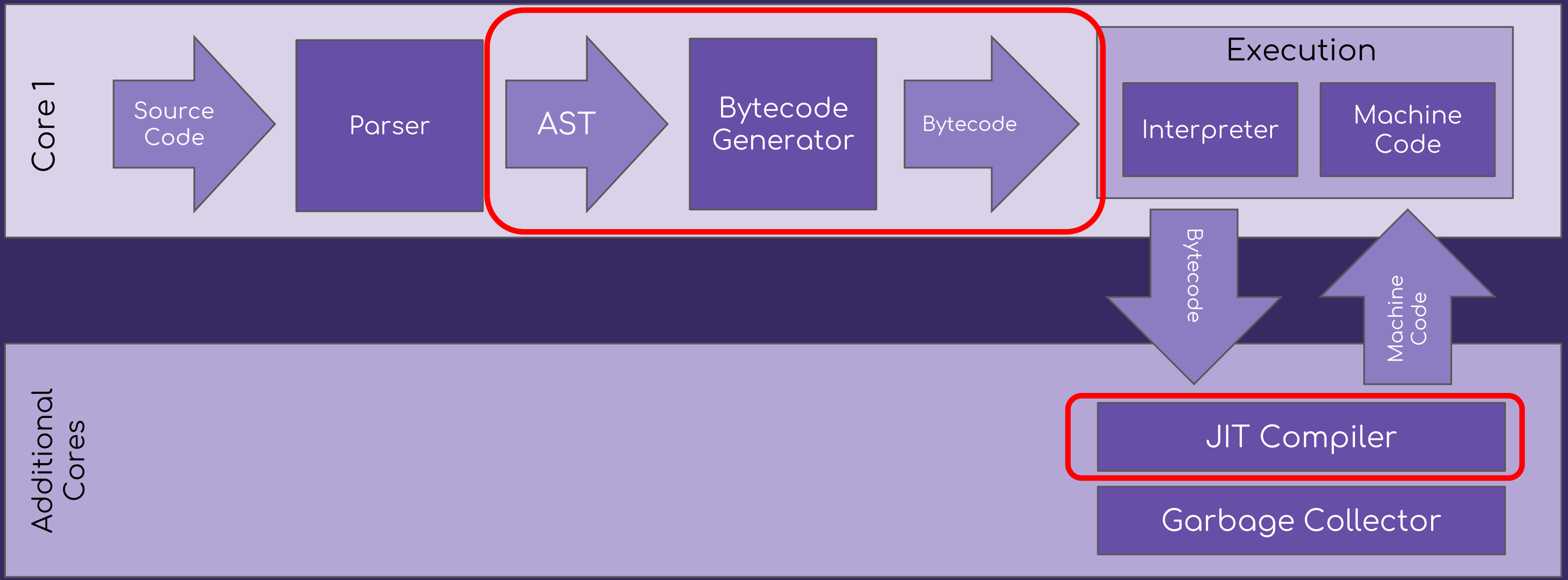
@typedef

Passionate People

Why types are so important?

Passionate People

Why types are so important?



Basic Types

boolean	never	number	any
string	Object	Array	Tuple
Union	null	void	undefined

Type assertions

Passionate People

Basic Types - Inferred

```
let month = 5;
let name = 'Mik';
let isVisible = false;
let cities = ['New York', 'Oxford', 'Valladolid'];
let person = {
    firstName: 'Gerardo',
    lastName: 'Abelardo'
};
```

Passionate People

Basic Types - Explicit

```
let month: number;

let name: string;

let isVisible: boolean;

let person: object;

let employee: any;
```

```
let cities: string[];

let customers: Array<any>;

let value: null;

let value2: undefined;

let value3: void;
```

```
let address: [string, number, string, string];
```

Passionate People

Basic Types

Type Assertions

```
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
let strLength2: number = (<string>someValue).length;
```

Passionate People

Basic Types - Union

```typescript
let area: string| undefined = property('area');
let element: HTMLElement | null = document.getElementById('passion');
let menu: Array<Link | Divider> = getMenu();
```

How TypeScript type-checking works?

Passionate People

Type-checking focuses on the *shape* that values have.

A.K.A "duck typing" or "structural subtyping".

Enums

Number Enums

String Enums

Passionate People

Number Enums

```
enum WeatherType {
    Sunny,
    Cloudy,
    Rainy,
    Storm,
    Frost,
    Nightly
}
```

```
enum WeatherType {
    Sunny = 1,
    Cloudy = 2,
    Rainy = 3,
    Storm = 4,
    Frost = 5,
    Nightly = 6
}
```

Number Enums - Usage

```
class Weather {
    getWeatherIcon(weatherType: WeatherType) {
        switch (weatherType) {
            case WeatherType.Sunny: return 'weather_sunny';
            case WeatherType.Cloudy: return 'weather_cloudy';
            case WeatherType.Rainy: return 'weather_rain';
            case WeatherType.Storm: return 'weather_storm';
            case WeatherType.Frost: return 'weather_frost';
            case WeatherType.Nightly: return 'weather_nightly';
        }
    }
}
```

String Enums

```
enum WeatherType {
   Sunny = 'weather_sunny',
   Cloudy = 'weather_cloudy',
   Rainy = 'weather_rain',
   Storm = 'weather_storm',
   Frost = 'weather_frost',
   Nightly = 'weather_nightly',
}
```

Passionate People

Functions

Function Types

Overloading

Passionate People

Function Types 1/3

```
function add(x: number, y: number): number {
    return x + y;
}
function multiply(x: number, y: number): number {
    return x * y;
}
let myAdd: (x: number, y: number) => number = add;

type twoOperandsType = (x: number, y: number) => number;

let myMultiply: twoOperandsType = multiply;
```

Passionate People

Function Types 2/3

```
function buildName(firstName: string, lastName?: string) {
    if (lastName) { return `${firstName} ${lastName}`; }
    else { return firstName; }
}


function buildAddress(street: string, postalCode: string = 'UNDEFINED') {
    if (postalCode) { return `${street} ${postalCode}`; }
    else { return street; }
}
```

Function Types 3/3

```
type buildStringOneOrMoreArguments = (x: string, y: string) => string;
const buildName1: buildStringOneOrMoreArguments = buildName;
const buildAddress1: buildStringOneOrMoreArguments = buildAddress;


function buildFamily(father: string, ...restOfFamily: string[]) {
    return `${father}  ${restOfFamily.join(" ")}`;
}
```

Passionate People

This

```
const mother = {
    name: 'Maria',
    sons: [
        'Lucho',
        'Mauro'
    ],
    daughters: [
        'Silvia',
        'Romani'
    ],
    getChildren(): string[] {
        return this.sons.concat(this.daughters);
    }
};
```

```
const father = {
    name: 'Manolo',
    sons: [
        'Lucho',
        'Mauro'
    ],
    daughters: [
        'Silvia',
        'Romani'
    ],
    getChildren(this: void) { // Avoid using this.
        return this.sons.concat(this.daughters);
    }
}
```

Arrow Function

```
let deck = {
   suits: ["hearts", "spades", "clubs", "diamonds"],
   cards: Array(52),
   createCardPicker: function() {
       return () => {
           let pickedCard = Math.floor(Math.random() * 52);
           let pickedSuit = Math.floor(pickedCard / 13);
           return {
               suit: this.suits[pickedSuit],
               card: pickedCard % 13
           };
       }
   }
};
```

Passionate People

Overloading

```
type propertiesObject = { [name: string]: any };
const properties: propertiesObject = {};


function property(propertyName: string, defaultValue: any ): any;
function property(propertyName: string): any | undefined;
function property(propertyName: string, defaultValue?: any) {
   if (defaultValue) {
       properties[propertyName] = defaultValue;
   }
   return properties[propertyName];
}
```

Passionate People

Interfaces

interface

implements

Interfaces

```
interface TreeLink {
   type: string;
   title: string;
   url: string;
   isCurrent: boolean;
   preferences: any;
   children: TreeLink[];
}
```

```
const treeLink: TreeLink = {
   type: 'external',
   title: 'Google',
   url: 'http://www.google.nl',
   isCurrent: false,
   preferences: {
       navIcon: 'accounts'
   },
   children: []
};
```

Passionate People

Implements

```
interface Flyable { fly: () => void }
interface Quackable { quack: () => void; }


class Duck implements Flyable, Quackable {
   fly() { console.log('Fly, Fly, Fly'); }
   quack() { console.log('Quack, Quack'); }
}
class Turkey implements Flyable {
   fly() { console.log('Ufff!'); }
}
```

Passionate People

Classes

class extends implements abstract

public private protected readonly

getters setters

static methods and properties

Passionate People

Classes

```
class Animal {
    name: string;
    constructor(name: string) {
        this.name = name;
    }
}
```

```
class Animal {
    constructor(
     public name: string
    ) {}
}
```

Passionate People

Abstract Classes

```
abstract class Animal {
    constructor(
     public readonly name: string
    ) {}
    abstract makeSound(): void;
    abstract move(): void;
    abstract getOffspring(): void;
}
```

```
abstract class Mammal extends Animal{
    constructor(
     public readonly name: string,
     public readonly legs: number
    ) {
        super(name);
    }
}
```

Passionate People

Inheritance 1/3

```
class Whale extends Mammal {
    constructor(){  super('Whale', 0); }
    makeSound() { console.log('waaoaoooooooooooaoaoo'); }
    move() { console.log('Swim, Swim, Swim, Swim, Swim!'); }
    getOffspring() { console.log('Get birth a calf!'); }
}
```

Passionate People

Inheritance 2/3

```
class Dog extends Mammal {
    constructor() { super('Dog', 4); }
    makeSound() { console.log('Woof, Woof, Woof!'); }
    move() { console.log('Walk, Walk, Walk!'); }
    getOffspring() { console.log('Get birth a dozen puppies!'); }
}
```

Passionate People

Inheritance 3/3

```
class Platypus extends Mammal {
   constructor() { super('Platypus', 4); }
   makeSound() { console.log('hrrrrrrrrrrrh'); }
   move() { console.log('Swim, Swim, Walk, Walk!'); }
   getOffspring() { console.log('Put eggs and wait!'); }
}
```

Generics

Generic Types

Generic Classes

Passionate People

Generics

```
function echo<T>(arg: T): T {
    console.log(arg);
    return arg;
}
```

```
echo('19th October');
echo<string>('Friday');
echo<number>(19);
```

Passionate People

Generic Types

```
function identity<T>(arg: T): T {
    return arg;
}


let myIdentity: <T>(arg: T) => T = identity;


let myIdentity2: <U>(arg: U) => U = identity;
```

Passionate People

Generic Classes

```
class Operator<T> {
   initialValue: T;
   operation: (x: T, y: T) => T;
}
const numAdder = new Operator<number>();
numAdder.initValue = 10;
numAdder.operation = function(x, y) {
  return this.initValue + x + y;
};
```

```
const strConc = new Operator<string>();
strConc.initValue = '';
strConc.operation = function (x, y) {
  return `${this.initValue} ${x} ${y}`;
};
```

Passionate People

Modules and Namespaces

export	import
Default exports	Working with other libraries
Namespacing	Multi-file namespace

Export

```
export class Cat {
   constructor(private name: string, private legs: number){}
}


export default {
   getCatInstance(name: string, legs: number) {
       return new Cat(name, legs);
   }
}
```

Import

```
import { Cat } from './Cat';
import { getCatInstance } from './utils';
```

Namespaces

```
export namespace Shapes {
   export class Triangle { /* ... */ }
   export class Square { /* ... */ }
}
import * as shapes from "./shapes";
let t = new shapes.Shapes.Triangle(); // shapes.Shapes?
```

Passionate People

Thanks.

Follow me

Passionate People