
TS

Superset of JavaScript

Types add safety

Types enable faster development

Compiles to plain JavaScript

- ES3
- ES5
- ES6/ES2015

Cross-platform

Open source

Library Manager

Books

Authors

Librarians

Magazines

Declaring variables and constants

- var
- let
- const

Specifying types

Basic data structures

- enums
- arrays
- tuples

var	let and const
Globally available in the function in which it is declared	Only available in the block in which it is declared
"Hoisted" to the top of the function	Not "hoisted" to the top of the block
Variable name may be declared a second time in the same function	Variable name may only be declared once per block

var Versus let

```
function ScopeTest() {
    if(true) {
        var foo = 'use anywhere';
        let bar = 'use in this block';
        // do some more stuff
    }
    console.log(foo); // works!!
    console.log(bar); // error!!
}
```

Boolean

Number

String

Array

Enum

Any

Void

Type Inference

```
let myString = 'this is a string';
myString = 42; // error!!

function ReturnNumber() {
    return 42;
}
let anotherString = 'this is also a string';
anotherString = ReturnNumber(); // error!!
```

Adding Type Annotations

```
let myString: string = 'this is a string';
myString = 42; // error!!

function ReturnNumber(): number {
    return 42;
}
let anotherString: string = 'this is also a string';
anotherString = ReturnNumber(); // error!!
```

Enums

```
enum Category { Biography, Poetry, Fiction }; // 0, 1, 2
enum Category { Biography = 1, Poetry, Fiction }; // 1, 2, 3
```

Enums

```
enum Category { Biography, Poetry, Fiction }; // 0, 1, 2
enum Category { Biography = 1, Poetry, Fiction }; // 1, 2, 3
enum Category { Biography = 5, Poetry = 8, Fiction = 9 }; // 5, 8, 9

let favoriteCategory: Category = Category.Biography;
```

```
let strArray1: string[] = ['here', 'are', 'strings'];

let strArray2: Array<string> = ['more', 'strings', 'here'];

let anyArray: any[] = [42, true, 'banana'];
```

Arrays

Can be declared two different ways

Accessed and used much like JavaScript arrays

Declare as an array of “any” to store any type in the same array

Functions in TypeScript versus JavaScript

Parameter types and return types

Arrow functions

Function types

Parameters

- Optional parameters
- Default parameters
- Rest parameters

Overloaded functions

Functions in TypeScript Versus JavaScript

TypeScript	JavaScript
Types (of course!)	No types
Arrow functions	Arrow functions (ES2015)
Function types	No function types
Required and optional parameters	All parameters are optional
Default parameters	Default parameters (ES2015)
Rest parameters	Rest parameters (ES2015)
Overloaded functions	No overloaded functions

Parameter Types and Return Types

```
function CreateCustomerID(name: string, id: number): string {
    return name + id;
}
```

```
let arr = allBooks.filter(function(book) {
    return book.author === 'Herman Melville';
});
```

```
let arr = allBooks.filter(book => book.author === 'Herman Melville');
```

Arrow Functions

Concise syntax for anonymous functions

“this” is captured at function creation – not invocation

```typescript
function PublicationMessage(year: number): string {
    return 'Date published: ' + year;
}
let publishFunc: (someYear: number) => string;
publishFunc = PublicationMessage;
let message: string = publishFunc(2016);
```

Function Types

Combination of parameter types and return type

Variables may be declared with function types

Function assigned must have the same signature as the variable type

```
function CreateCustomer(name: string, age?: number) { }
```

Optional and Default Parameters

Optional parameters denoted with "?" after parameter name

```
function CreateCustomer(name: string, age?: number) { }

function GetBookByTitle(title: string = 'The C Programming Language') { }
```

Optional and Default Parameters

Optional parameters denoted with "?" after parameter name

Must appear after all required parameters

Default parameters may be set to a literal value or an expression

```
function GetBooksReadForCust(name: string, ...bookIDs: number[]) { }

let books = GetBooksReadForCust('Leigh', 2, 5);
```

Rest Parameters

Collects a group of parameters into a single array

Denoted with an ellipsis prefix on last parameter

Implementing Function Overloads

```
function GetTitles(author: string): string[];
function GetTitles(available: boolean): string[];
function GetTitles(bookProperty: any): string[] {
    if(typeof bookProperty == 'string') {
        // get books by author, add to foundTitles
    }
    else if(typeof bookProperty == 'boolean') {
        // get books by availability, add to foundTitles
    }
    return foundTitles;
}
```

```
interface Book {
    id: number;
    title: string;
    author: string;
    pages?: number;

}
```

Defining an Interface

“interface” keyword

List properties with their types

Interfaces for Function Types

```
function CreateCustomerID(name: string, id: number): string {
    return name + id;
}
interface StringGenerator {
    (chars: string, nums: number): string;
}
let IdGenerator: StringGenerator;
IdGenerator = CreateCustomerID;
```

Extending Interfaces

```
interface LibraryResource {
    catalogNumber: number;
}
interface Book {
    title: string;
}
interface Encyclopedia extends LibraryResource, Book {
    volume: number;
}
```

```
let refBook: Encyclopedia = {
    catalogNumber: 1234,
    title: 'The Book of Everything
    volume: 1
}
```

Class Types

```
interface Librarian {
    doWork: () => void;
}
class ElementarySchoolLibrarian implements Librarian {
    doWork() {
        console.log('Reading to and teaching children...');
    }
}
let kidsLibrarian: Librarian = new ElementarySchoolLibrarian();
kidsLibrarian.doWork();
```

What is a class?

Similarity to classes in other languages

Class members

- Constructors
- Properties
- Methods

Inheritance

Abstract classes

Class expressions

Template for creating objects

Provides state storage and behavior

Encapsulates reusable functionality

Define Types

Properties and Methods

Constructors

Access Modifiers

Inheritance

Abstract Classes

```
class ReferenceItem {
    constructor(title: string, publisher?: string) {
        // perform initialization here
    }
}
let encyclopedia = new ReferenceItem('WorldPedia', 'WorldPub');
```

Constructors

Method named "constructor" – maximum of one per class

Use optional parameters to call different ways

Executed by using the "new" keyword

Properties and Methods

```
class ReferenceItem {
    numberOfPages: number;
    get editor(): string {
        // custom getter logic goes here, should return a value
    }
    set editor(newEditor: string) {
        // custom setter logic goes here
    }
    printChapterTitle(chapterNum: number): void {
        // print title here
    }
}
```

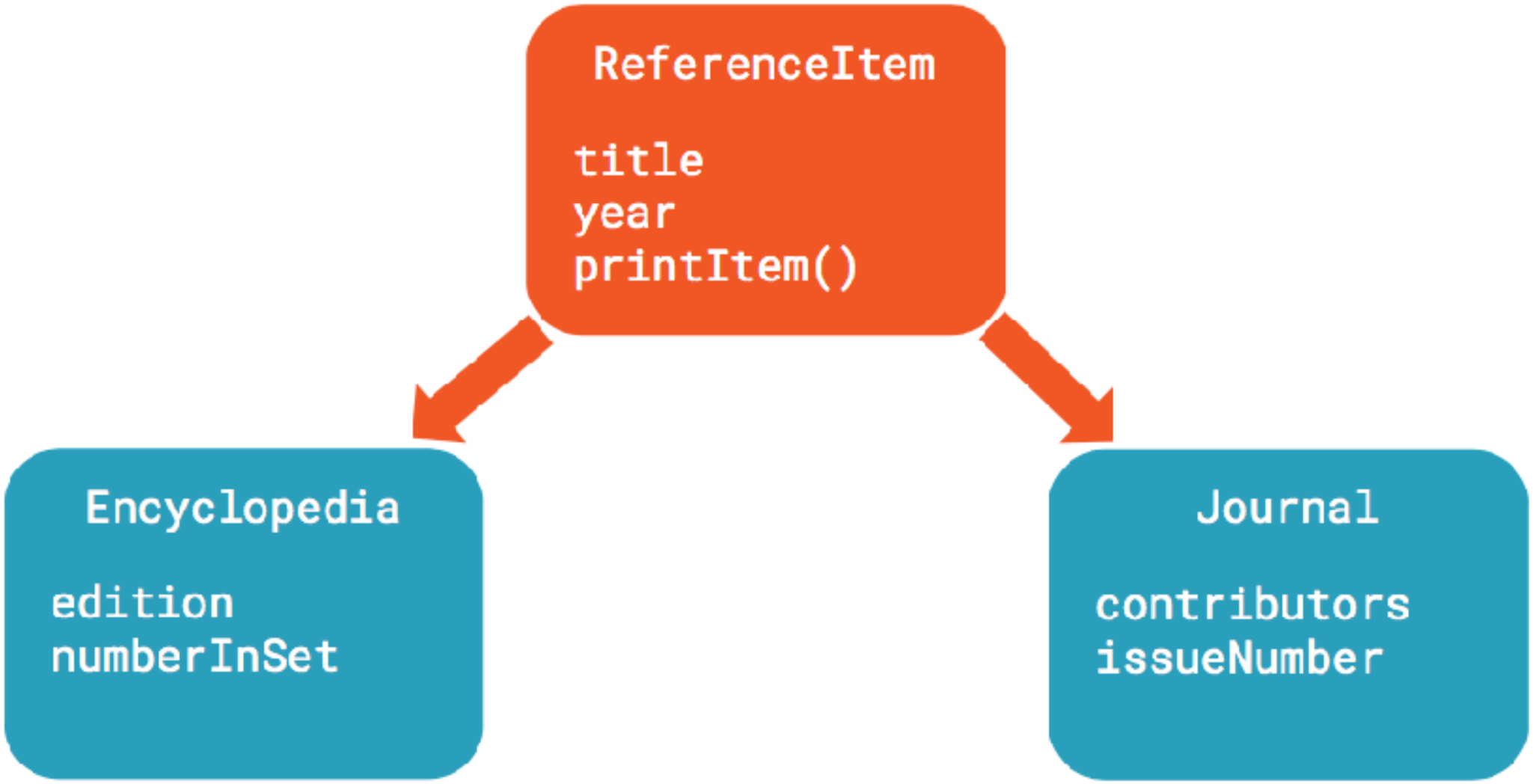
Parameter Properties

```
class Author {
    name: string;
    constructor(authorName: string) {
        name = authorName;
    }
}
class Author {
    constructor(public name: string) { }
}
```

Static Properties

```
class Library {
    constructor(public name: string) { }
    static description: string = 'A source of knowledge.';
}
let lib = new Library('New York Public Library');
let name = lib.name; // avaiable on instances of the class
let desc = Library.description; // available on the class
```

Inheritance



Extending Classes with Inheritance

```
class ReferenceItem {
    title: string;
    printItem(): void { // print something here }
}

class Journal extends ReferenceItem {
    constructor() {
    super();
    }
    contributors: string[];
}
```

They're modular!!!

Maintainable

Reusable

Native to Node and ES2015

Organized simply in files and folders

CommonJS

Asynchronous Module Definition (AMD)

Universal Module Definition (UMD)

System

ES2015

Require.js

http://requirejs.org

SystemJS

https://github.com/systemjs/systemjs

Exporting from a Module

```
// periodicals.ts
export interface Periodical {
    issueNumber: number;
}
export class Magazine implements Periodical {
    issueNumber: number;
}
export function GetMagazineByIssueNumber(issue: number): Magazine {
    // retrieve and return a magazine
}
```

Exporting from a Module

```
// periodicals.ts
interface Periodical {
    issueNumber: number;
}
class Magazine implements Periodical {
    issueNumber: number;
}
function GetMagazineByTitle(title: string): Magazine {
    // retrieve and return a magazine
}
export { Periodical, Magazine, GetMagazineByTitle as GetMag}
```

Importing from a Module

```
// news.ts
import { Magazine, GetMag as GetMagazine } from './periodicals';
let newsMag: Magazine = GetMagazine('Weekly News');


// kids.ts
import * as mag from './periodicals';
```