Contents

[Type Basics 2](#_Toc471890963)

[Task 01. Basic Types 2](#_Toc471890964)

[Task 02. Enum 3](#_Toc471890965)

[Functions 4](#_Toc471890966)

[Task 03. Arrow Functions 4](#_Toc471890967)

[Task 04. Function Type 5](#_Toc471890968)

[Task 05. Optional, Default and Rest Parameters 6](#_Toc471890969)

[Task 06. Function Overloading 7](#_Toc471890970)

[Interfaces 8](#_Toc471890971)

[Task 07. Defining an Interface 8](#_Toc471890972)

[Task 08. Defining an Interface for Function Types 9](#_Toc471890973)

[Task 09. Extending Interface 10](#_Toc471890974)

[Task 10. Interfaces for Class Types 11](#_Toc471890975)

[Classes 12](#_Toc471890976)

[Task 11. Creating and Using Classes 12](#_Toc471890977)

[Task 12. Extending Classes 13](#_Toc471890978)

[Task 13. Creating Abstract Classes 14](#_Toc471890979)

[Modules and Namespaces 15](#_Toc471890980)

[Task 14. Using Namespaces 15](#_Toc471890981)

[Task 15. Export and Import 16](#_Toc471890982)

[Task 16. Default Export 18](#_Toc471890983)

[Generics 19](#_Toc471890984)

[Task 17. Generic Functions 19](#_Toc471890985)

[Task 18. Generic Interfaces and Classes 20](#_Toc471890986)

[Task 19. Generic Constraints 21](#_Toc471890987)

[Decorators 22](#_Toc471890988)

[Task 20.1. Class Decorators 22](#_Toc471890989)

[Task 20.2. Class Decorators that replace constructor functions 23](#_Toc471890990)

[Task 21. Method Decorator 24](#_Toc471890991)

[Asynchronous Patterns 25](#_Toc471890992)

[Task 22. Callback Functions 25](#_Toc471890993)

[Task 23. Promises 27](#_Toc471890994)

[Task 24. Async/await 28](#_Toc471890995)

# Types Basics

## Task 01. Basic Types

function getAllBooks() {

let books = [

{ title: 'Refactoring JavaScript', author: 'Evan Burchard', available: true },

{ title: 'JavaScript Testing', author: 'Liang Yuxian Eugene', available: false },

{ title: 'CSS Secrets', author: 'Lea Verou', available: true },

{ title: 'Mastering JavaScript Object-Oriented Programming', author: 'Andrea Chiarelli', available: true }

];

return books;

}

function logFirstAvailable(books): void {

let numberOfBooks: number = books.length;

let firstAvailable: string = '';

for(let currentBook of books) {

if(currentBook.available) {

firstAvailable = currentBook.title;

break;

}

}

console.log(`Total Books: ${numberOfBooks}`);

console.log(`First Available: ${firstAvailable}`);

}

// ---------------------------------------------

console.log(getAllBooks());

const allBooks = getAllBooks();

logFirstAvailable(allBooks);

## Task 02. Enum

enum Category { JavaScript, CSS, HTML, TypeScript, Angular2 }

function getAllBooks() {

let books =[

{ title: 'Refactoring JavaScript', author: 'Evan Burchard', available: true, category: Category.JavaScript },

{ title: 'JavaScript Testing', author: 'Liang Yuxian Eugene', available: false, category: Category.JavaScript },

{ title: 'CSS Secrets', author: 'Lea Verou', available: true, category: Category.CSS },

{ title: 'Mastering JavaScript Object-Oriented Programming', author: 'Andrea Chiarelli', available: true, category: Category.JavaScript }

];

return books;

}

function logFirstAvailable(books): void {

…

}

function getBookTitlesByCategory(categoryFilter: Category): Array<string> {

console.log(`Getting books in category: ${Category[categoryFilter]}`);

const allBooks = getAllBooks();

const filteredTitles: string[] = [];

for(let currentBook of allBooks) {

if(currentBook.category === categoryFilter) {

filteredTitles.push(currentBook.title);

}

}

return filteredTitles;

}

function logBookTitles(titles: string[]): void {

for(let title of titles) {

console.log(title);

}

}

// ---------------------------------------------

console.log(getAllBooks());

const allBooks = getAllBooks();

logFirstAvailable(allBooks);

const javaScriptBooks = getBookTitlesByCategory(Category.JavaScript);

logBookTitles(javaScriptBooks);

# Functions

## Task 03. Arrow Functions

function getAllBooks() {

let books =[

{ id: 1, title: 'Refactoring JavaScript', author: 'Evan Burchard', available: true, category: Category.JavaScript },

{ id: 2, title: 'JavaScript Testing', author: 'Liang Yuxian Eugene', available: false, category: Category.JavaScript },

{ id: 3, title: 'CSS Secrets', author: 'Lea Verou', available: true, category: Category.CSS },

{ id: 4, title: 'Mastering JavaScript Object-Oriented Programming', author: 'Andrea Chiarelli', available: true, category: Category.JavaScript }

];

return books;

}

function getBookByID(id: number) {

const allBooks = getAllBooks();

return allBooks.find(book => book.id === id);

}

// -----------

//logBookTitles(javaScriptBooks);

javaScriptBooks.forEach((val, idx, arr) => console.log(++idx + ' - ' + val));

## Task 04. Function Type

function createCustomerID(name: string, id: number): string {

return `${name}${id}`;

}

let myID = createCustomerID('Ann', 10);

console.log(myID);

// the names of parameters are not important

let IdGenerator: (chars: string, num: number) => string;

IdGenerator = (name: string, id: number) => `${name}${id}`;

IdGenerator = createCustomerID;

myID = IdGenerator('Ann', 20);

console.log(myID);

## Task 05. Optional, Default and Rest Parameters

function logFirstAvailable(books = getAllBooks()): void {

…

}

function getBookTitlesByCategory(categoryFilter: Category = Category.JavaScript): Array<string> {

…

}

function createCustomer(name: string, age?: number, city?: string): void {

console.log(`Creating customer ${name}`);

if(age) {

console.log(`Age: ${age}`);

}

if(city) {

console.log(`City: ${city}`);

}

}

function сheckoutBooks(customer: string, ...bookIDs: number[]): string[] {

console.log(`Checking out books for ${customer}`);

let booksCheckedOut: string[] = [];

for(let id of bookIDs) {

let book = getBookByID(id);

if (book && book.available) {

booksCheckedOut.push(book.title);

}

}

return booksCheckedOut;

}

// ----

createCustomer('Ann');

createCustomer('Boris', 6);

createCustomer('Clara', 12, 'Atlanta');

let myBooks: string[] = сheckoutBooks('Ann', 1, 3, 4);

myBooks.forEach(title => console.log(title));

## Task 06. Function Overloading

function getTitles(author: string): string[];

function getTitles(available: boolean): string[];

function getTitles(bookProperty: any): string[] {

const allBooks = getAllBooks();

const foundTitles: string[] = [];

if(typeof bookProperty == 'string') {

// get all books by a particular author

for(let book of allBooks) {

if(book.author === bookProperty) {

foundTitles.push(book.title);

}

}

}

else if(typeof bookProperty == 'boolean') {

// get all books based on specified availability

for(let book of allBooks) {

if(book.available === bookProperty) {

foundTitles.push(book.title);

}

}

}

return foundTitles;

}

// ---------------------------------------------

let checkedOutBooks = getTitles(false);

checkedOutBooks.forEach(title => console.log(title));

# Interfaces

## Task 07. Defining an Interface

interface Book {

id: number;

title: string;

author: string;

available: boolean;

category: Category;

pages?: number;

markDamaged?: (reason: string) => void;

}

function getAllBooks(): Book[] {

…

}

function getBookByID(id: number): Book | undefined {

…

}

function PrintBook(book: Book): void {

console.log(`${book.title} by ${book.author}`);

}

// ------

**let myBook: Book = {**

**id: 5,**

**title: 'Colors, Backgrounds, and Gradients',**

**author: 'Eric A. Meyer',**

**available: true,**

**category: Category.CSS,**

**pages: 200,**

**markDamaged: (reason: string) => console.log(`Damaged: ${reason}`)**

**};**

**PrintBook(myBook);**

**myBook.markDamaged('missing back cover');**

## Task 08. Defining an Interface for Function Types

interface Book {

id: number;

title: string;

author: string;

available: boolean;

category: Category;

pages?: number;

markDamaged?: DamageLogger;

}

interface DamageLogger {

(reason: string): void;

}

// ---------

let logDamage: DamageLogger;

logDamage = (damage: string) => console.log('Damage reported: ' + damage);

logDamage('coffee stains');

## Task 09. Extending Interface

interface Person {

name: string;

email: string;

}

interface Author extends Person {

numBooksPublished: number;

}

interface Librarian extends Person {

department: string;

assistCustomer: (custName: string) => void;

}

// -----

let favoriteAuthor: Author = {

email: 'Anna@gmail.com',

name: 'Anna',

numBooksPublished: 3

};

let favoriteLibrarian: Librarian = {

name: 'Boris',

email: 'Boris@gmail.com',

department: 'Classical Literature',

assistCustomer: (name: string) => console.log(`Assist ${name}`)

};

## Task 10. Interfaces for Class Types

class UniversityLibrarian implements Librarian {

name: string;

email: string;

department: string;

assistCustomer(custName: string): void {

console.log(`${this.name} is assisting ${custName}`);

}

}

// ----

// let favoriteLibrarian: Librarian = {

// name: 'Boris',

// email: 'Boris@gmail.com',

// department: 'Classical Literature',

// assistCustomer: (name: string) => console.log(`Assist ${name}`)

// };

let favoriteLibrarian: Librarian = new UniversityLibrarian();

favoriteLibrarian.name = 'Anna';

favoriteLibrarian.assistCustomer('Boris');

# Classes

## Task 11. Creating and Using Classes

class ReferenceItem {

// title: string;

// year: number;

private \_publisher: string;

static department: string = 'Research';

// constructor(newTitle: string, newYear: number) {

// console.log('Creating a new ReferenceItem...');

// this.title = newTitle;

// this.year = newYear;

// }

constructor(public title: string, private year: number) {

console.log('Creating a new ReferenceItem...');

}

printItem(): void {

console.log(`${this.title} was published in ${this.year}.`);

console.log(`Department: ${ReferenceItem.department}`);

}

get publisher(): string {

return this.\_publisher.toUpperCase();

}

set publisher(newPublisher: string) {

this.\_publisher = newPublisher;

}

}

// ----

let ref: ReferenceItem = new ReferenceItem('Updated Facts and Figures', 2016);

ref.printItem();

ref.publisher = 'Random Data Publishing';

console.log(ref.publisher);

## Task 12. Extending Classes

class Encyclopedia extends ReferenceItem {

constructor(newTitle: string, newYear: number, public edition: number) {

super(newTitle, newYear);

}

printItem(): void {

super.printItem();

console.log(`Edition: ${this.edition} (${this.year})`);

}

}

class ReferenceItem {

…

constructor(public title: string, protected year: number) {

console.log('Creating a new ReferenceItem...');

}

}

let refBook: ReferenceItem = new Encyclopedia('WorldPedia', 1900, 10);

refBook.printItem();

## Task 13. Creating Abstract Classes

abstract class ReferenceItem {

…

abstract printCitation(): void;

}

class Encyclopedia extends ReferenceItem {

…

printCitation(): void {

console.log(`${this.title} - ${this.year}`);

}

}

// let ref: ReferenceItem = new ReferenceItem('Updated Facts and Figures', 2016);

// ref.printItem();

// ref.publisher = 'Random Data Publishing';

// console.log(ref.publisher);

let refBook: ReferenceItem = new Encyclopedia('WorldPedia', 1900, 10);

refBook.printItem();

refBook.printCitation();

# Modules and Namespaces

## Task 14. Using Namespaces

**// utility-functions.ts**

namespace Utility {

export namespace Fees {

export function CalculateLateFee(daysLate: number): number {

return daysLate \* .25;

}

}

export function MaxBooksAllowed(age: number): number {

return age < 12 ? 3 : 10;

}

function privateFunc(): void {

console.log('This is private...');

}

}

// index.html

<html>

<head></head>

<body>

<script src="utility-functions.js"></script>

<script src="app.js"></script>

</body>

</html>

// app.ts

/// <reference path="utility-functions.ts" />

import util = Utility.Fees;

let fee = util.CalculateLateFee(10);

console.log(`Fee: ${fee}`);

## Task 15. Export and Import

// enums.ts

enum Category { JavaScript, CSS, HTML, TypeScript, Angular2 };

export { Category };

// interfaces.ts

import { Category } from './enums';

interface Book {

id: number;

title: string;

author: string;

available: boolean;

category: Category;

pages?: number;

markDamaged?: DamageLogger;

}

interface DamageLogger {

(reason: string): void;

}

interface Person {

name: string;

email: string;

}

interface Author extends Person {

numBooksPublished: number;

}

interface Librarian extends Person {

department: string;

assistCustomer: (custName: string) => void;

}

export { Book, DamageLogger as Logger, Author, Librarian };

// classes.ts

import \* as Interfaces from './interfaces';

class UniversityLibrarian implements Interfaces.Librarian {

name: string;

email: string;

department: string;

assistCustomer(custName: string) {

console.log(this.name + ' is assisting ' + custName);

}

}

abstract class ReferenceItem {

private \_publisher: string;

static department: string = 'Research';

constructor(public title: string, protected year: number) {

console.log('Creating a new ReferenceItem...');

}

printItem(): void {

console.log(`${this.title} was published in ${this.year}.`);

console.log(`Department: ${ReferenceItem.department}`);

}

get publisher(): string {

return this.\_publisher.toUpperCase();

}

set publisher(newPublisher: string) {

this.\_publisher = newPublisher;

}

abstract printCitation(): void;

}

export { UniversityLibrarian, ReferenceItem };

// app.ts

import { Category } from './enums';

import { Book, Logger, Author, Librarian } from './interfaces';

import { UniversityLibrarian, ReferenceItem } from './classes';

// ----

let logDamage: Logger;

logDamage = (damage: string) => console.log('Damage reported: ' + damage);

logDamage('coffee stains');

## Task 16. Default Export

// encyclopedia.ts

import { ReferenceItem } from './classes';

export default class Encyclopedia extends ReferenceItem {

constructor(newTitle: string, newYear: number, public edition: number) {

super(newTitle, newYear);

}

printItem(): void {

super.printItem();

console.log(`Edition: ${this.edition} (${this.year})`);

}

printCitation(): void {

console.log(`${this.title} - ${this.year}`);

}

}

// app.ts

import RefBook from './encyclopedia';

// ---

let refBook: ReferenceItem = new RefBook('WorldPedia', 1900, 10);

refBook.printItem();

# Generics

## Task 17. Generic Functions

// lib/utility-functions.ts

export function purge<T>(inventory: Array<T>): Array<T> {

return inventory.splice(2, inventory.length);

}

// enums.ts

enum Category { JavaScript, CSS, HTML, TypeScript, Angular2, Software };

// app.ts

import { purge } from './lib/utility-functions';

// ---

let inventory: Array<Book> = [

{ id: 10, title: 'The C Programming Language', author: 'K & R', available: true, category: Category.Software },

{ id: 11, title: 'Code Complete', author: 'Steve McConnell', available: true, category: Category.Software },

{ id: 12, title: '8-Bit Graphics with Cobol', author: 'A. B.', available: true, category: Category.Software },

{ id: 13, title: 'Cool autoexec.bat Scripts!', author: 'C. D.', available: true, category: Category.Software }

];

let purgedBooks: Array<Book> = purge<Book>(inventory);

purgedBooks.forEach(book => console.log(book.title));

let purgedNums: Array<number> = purge<number>([1, 2, 3, 4]);

console.log(purgedNums);

## Task 18. Generic Interfaces and Classes

// interfaces.ts

interface Magazine {

title: string;

publisher: string;

}

export { Book, DamageLogger as Logger, Author, Librarian, Magazine };

// shelf.ts

export default class Shelf<T> {

private \_items: Array<T> = new Array<T>();

add(item: T): void {

this.\_items.push(item);

}

getFirst(): T {

return this.\_items[0];

}

}

// app.ts

import { Book, Logger, Author, Librarian, Magazine } from './interfaces';

import Shelf from './shelf';

// ----

// let purgedBooks: Array<Book> = purge<Book>(inventory);

// purgedBooks.forEach(book => console.log(book.title));

// let purgedNums: Array<number> = purge<number>([1, 2, 3, 4]);

// console.log(purgedNums);

let bookShelf: Shelf<Book> = new Shelf<Book>();

inventory.forEach(book => bookShelf.add(book));

let firstBook: Book = bookShelf.getFirst();

console.log(firstBook.title);

let magazines: Array<Magazine> = [

{ title: 'Programming Language Monthly', publisher: 'Code Mags' },

{ title: 'Literary Fiction Quarterly', publisher: 'College Press' },

{ title: 'Five Points', publisher: 'GSU' }

];

let magazineShelf: Shelf<Magazine> = new Shelf<Magazine>();

magazines.forEach(mag => magazineShelf.add(mag));

let firstMagazine: Magazine = magazineShelf.getFirst();

console.log(firstMagazine.title);

## Task 19. Generic Constraints

// shelf.ts

interface ShelfItem {

title: string;

}

export default class Shelf<T extends ShelfItem> {

…

find(title: string): T {

return this.\_items.filter(item => item.title === title)[0];

}

printTitles(): void {

this.\_items.forEach(item => console.log(item.title));

}

}

//app.ts

// ---

magazineShelf.printTitles();

let softwareBook = bookShelf.find('Code Complete');

console.log(`${softwareBook.title} (${softwareBook.author})`);

# Decorators

## Task 20.1. Class Decorators

// decorators.ts

export function sealed(name: string) {

return function(target: Function): void {

console.log(`Sealing the constructor: ${name}`);

Object.seal(target);

Object.seal(target.prototype);

}

}

// classes.ts

import { sealed } from './decorators';

@sealed('UniversityLibrarian')

class UniversityLibrarian implements Interfaces.Librarian {

…

}

## Task 20.2. Class Decorators that replace constructor functions

// decorator.ts

export function logger<TFunction extends Function>(target: TFunction): TFunction {

let newConstructor: Function = function() {

console.log(`Creating new instance.`);

console.log(target);

}

newConstructor.prototype = Object.create(target.prototype);

newConstructor.prototype.constructor = target;

return <TFunction>newConstructor;

}

// classes.ts

import { sealed, logger} from './decorators';

@logger

@sealed('UniversityLibrarian')

class UniversityLibrarian implements Interfaces.Librarian {

…

}

## Task 21. Method Decorator

// decorators.ts

export function writable(isWritable: boolean) {

return function(target: Object,

propertyKey: string,

descriptor: PropertyDescriptor) {

console.log(`Setting ${propertyKey}.`);

descriptor.writable = isWritable;

}

}

// classes.ts

import { sealed, logger, writable } from './decorators';

@logger

@sealed('UniversityLibrarian')

class UniversityLibrarian implements Interfaces.Librarian {

…

@writable(true)

assistFaculty() {

console.log('Assisting faculty.');

}

@writable(false)

teachCommunity() {

console.log('Teaching community.');

}

}

// app.ts

let lib1 = new UniversityLibrarian();

try {

lib1.assistFaculty = () => console.log('assistFaculty replacement method');

lib1.teachCommunity = () => console.log('teachCommunity replacement method');

} catch (error) {

console.log(error.message);

}

lib1.assistFaculty();

lib1.teachCommunity();

# Asynchronous Patterns

## Task 22. Callback Functions

// lib/utility-functions.ts

import { Category } from './../enums';

import { Book } from './../interfaces';

export function purge<T>(inventory: Array<T>): Array<T> {

…

}

export function getAllBooks(): Book[] {

…

}

export function getBookTitlesByCategory(categoryFilter: Category = Category.JavaScript): Array<string> {…

}

export function logFirstAvailable(books = getAllBooks()): void {

…

}

export function logBookTitles(titles: string[]): void {

…

}

export function getBookByID(id: number): Book | undefined {

…

}

export function createCustomerID(name: string, id: number): string {

…

}

export function createCustomer(name: string, age?: number, city?: string): void {

…

}

export function сheckoutBooks(customer: string, ...bookIDs: number[]): string[] {

…

}

export function getTitles(author: string): string[];

export function getTitles(available: boolean): string[];

export function getTitles(bookProperty: any): string[] {

…

}

export function PrintBook(book: Book): void {

…

}

interface LibMgrCallback {

(err: Error, titles: string[]): void;

}

export function getBooksByCategory(category: Category, callback: LibMgrCallback): void {

setTimeout(() => {

try {

let foundBooks: string[] = getBookTitlesByCategory(category);

if(foundBooks.length > 0) {

callback(null, foundBooks);

}

else {

throw new Error('No books found.');

}

} catch (error) {

callback(error, null);

}

}, 2000);

}

export function logCategorySearch(err: Error, titles: string[]): void {

if(err) {

console.log(`Error message: ${err.message}`);

}

else {

console.log(`Found the following titles:`);

console.log(titles);

}

}

// app.ts

import { purge, getAllBooks, getBookTitlesByCategory, logFirstAvailable,

logBookTitles, getBookByID, createCustomerID, createCustomer, сheckoutBooks,

getTitles, PrintBook, getBooksByCategory, logCategorySearch } from './lib/utility-functions';

// Callback functions

console.log('Beginning search...');

getBooksByCategory(Category.JavaScript, logCategorySearch);

getBooksByCategory(Category.Software, logCategorySearch);

console.log('Search submitted...');

## Task 23. Promises

// lib/utility-functions.ts

export function getBooksByCategoryPromise(cat: Category): Promise<string[]> {

let p: Promise<string[]> = new Promise((resolve, reject) => {

setTimeout(() => {

let foundBooks: string[] = getBookTitlesByCategory(cat);

if(foundBooks.length > 0) {

resolve(foundBooks);

}

else {

reject('No books found for that category.');

}

}, 2000);

});

return p;

}

// app.ts

import { purge, getAllBooks, getBookTitlesByCategory, logFirstAvailable,

logBookTitles, getBookByID, createCustomerID, createCustomer, сheckoutBooks,

getTitles, PrintBook, getBooksByCategory, logCategorySearch,

getBooksByCategoryPromise } from './lib/utility-functions';

// Promises

console.log('Beginning search...');

getBooksByCategoryPromise(Category.Angular2)

.then(titles => {

console.log(`Found titles: ${titles}`);

throw 'something bad happened';

// return titles.length;

}, reason => { return 0; })

.then(numOfBooks => console.log(`Number of books found: ${numOfBooks}`))

.catch(reason => console.log(`Error: ${reason}`));

console.log('Search submitted...');

## Task 24. Async/await

// app.ts

import "babel-polyfill";

// lib/utility-functions.ts

export async function logSearchResults(category: Category) {

let foundBooks = await getBooksByCategoryPromise(category);

console.log(foundBooks);

}

// app.ts

// async/await

console.log('Beginning search...');

logSearchResults(Category.JavaScript)

.catch(reason => console.log(reason));

console.log('Search submitted...');