// If you get stucked, here is the docs: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

// **Exercise 1**,

// Create the Product interface based on the following example products.

// - type can be only "Program" or "Course"

// - currency can be only "USD", "HUF", "EUR"

//

// If you get stuck with the createdAt, check what is the return type of

// Date.parse (you can hover your mouse over it).

export interface Product {

/\* TODO: fill the type declarations \*/

}

const products: Product[] = [

{

id: 4,

title: "How to Hack NASA with HTML",

price: "5000.00",

createdAt: Date.parse("2022-05-18T14:48:00"),

currency: "HUF",

type: "Course",

relatedCourses: [],

},

{

id: 6,

title: "Cat Grooming Masterclass",

price: "10.00",

createdAt: Date.parse("2022-05-19T16:00:00"),

currency: "USD",

type: "Program",

relatedCourses: [

{

id: 11,

title: "Lying Yourself, that you are the Master",

price: "0.00",

createdAt: Date.parse("2022-05-18T16:00:00"),

currency: "USD",

type: "Course",

relatedCourses: [],

},

{

id: 16,

title: "Taming your cat, a life long learning",

price: "0.00",

createdAt: Date.parse("2022-05-17T16:00:00"),

currency: "USD",

type: "Course",

relatedCourses: [],

},

],

},

]

// **Exercise 2**,

// Add type annotations to the arguements and return types

// of these two functions.

function filterCourses(products /\* add type annotation here \*/) /\* add type annotation here \*/ {

return products.filter(product => product.type === 'Course')

}

function getTitles(products /\* add type annotation here \*/) /\* add type annotation here \*/ {

return products.map(product => product.title)

}

// **Exercise 3**,

// When Typescript infers correctly the types and when it is necessary

// to define them explicitly? Try to remove type annotations from the

// filterCourses and getTitles functions

// above. Hover the mouse to the variables to check the inferred types.

// When do you see "any", and when something else?

// This two functions just here to check the proper return type in the tests.

const courses = filterCourses(products)

const titles = getTitles(products)

// **Exercise 4**,

// Can I pass a Product object to the format Price function without

// typescript error? Why?

// Spot that the inline type annotation here is different than the

// Product's type definition.

function formatPrice(product: {price: string, currency: string}) {

return `${product.price} ${product.currency}`

}

// passing a product to the function, for tests only.

const price = formatPrice(products[0])

**Everyday Types**

// If you get stucked, here is the docs: https://www.typescriptlang.org/docs/handbook/2/everyday-types.html

// **Exercise 1) Primitives and arrays**

// TODO: remove the "any" type, and add a concerete type for these basic primitives

// How they are working, if you remove all type definitions? How inference is working here?

let price: any /\* add the correcy type annotation here instead of any \*/

price = 100.5

let title: any /\* add type annotation here \*/

title = "How to Hack NASA with HTML?"

let option: any /\* add type annotation here \*/

option = true

let prices: any /\* add type annotation here \*/

prices = [3, 5, 100, 3.5]

let titles: any /\* add type annotation here \*/

titles = ["How to Hack NASA with HTML?", "Cat Taming Masterclass"]

let options: any /\* add type annotation here \*/

options = [true, true, false]

// **Exercise 2) Any**

// Here we have a product, which type is an explicit any.

// Unforunately we have here a cat instead. It is clearly seen,

// that everything is accepted, typescript basically switched off.

// We will got a lot of runtime errors and unexpected undefineds

// here.

// TODO: Create a proper type definition based on the usage of the product,

// correct the input data and the function usage below based on that.

const anyProduct: any = {name: 'Mr. Fluff', kind: 'British Shorthair', age: 4}

const productTitle = anyProduct.title

const priceWithTaxes = anyProduct.price \* (1.25)

const upperCaseTitle = anyProduct.price.toUpperCase()

// **Exercise 3) Anonymus Functions**

// In JS we are putting anonymus functions to a lot of place,

// typically in the higher order functions like map. Typescript

// can figure out the anonymus functions types based on the usage.

// TODO: correct the parameter's type of createKeysFromTitles. Spot out

// how it is changing the map function's types.

const titelsToConvert = ["How to Hack NASA with HTML?", "Cat Taming Masterclass"]

const createKeysFromTitles = (titles /\* \*/) => {

return titles.map(title => title.toLowerCase().replace(" ", "\_").replace("?", ""))

}

const keys = createKeysFromTitles(titelsToConvert)

// **Exercise 4) Union types**

// We have a common Course type in our codebase, unfortunately

// it is not correctly typed, because some API endpoints return

// the price in a string other endpoints in number format.

// TODO: Change the Course interface to conform all possible formats.

// (Check the type errors in the usages below.)

interface Course {

title: string,

price: number,

}

const checkoutCourse: Course = {

title: "What You can Learn from Your Cat?",

price: 100.0,

}

const shoppingCartCourse: Course = {

title: "What You can Learn from your Cat?",

price: "100.0"

}

// TODO: Ooops, after the Course interface is changed,

// something is gone wrong here. Correct the funtion body for now

// creatively, in the Narrowing chapter we will see a lot of

// patterns to handle these cases.

const getTax = (course: Course) => {

return course.price \* 0.25

}

**// Exercise 5) Types Aliases**

//

// We can use type aliases with

// type keyword for any annotations.

// TODO: fill the Type Alias for the account object

// based on the example object below. Spot out

// the differences betweeb the interface declarations.

// Note type alias can be used for any type, not just

// objects. Check the examples in the handbook.

type Account = {

}

const account: Account = {

id: 5,

name: "Awesome Account",

currency: "USD",

}

const getAccountName = (account: Account) => account.name

// TODO: Interesting, here we are not using the Account Type Alias,

// however the function is correctly typed, and accepts accounts.

// Why?

const getCurrency = (account: {name: string, currency: string}) => account.currency

const accountName = getAccountName(account)

const accountCurrency = getCurrency(account)

// **Exercise 6) Type Assertions**

//

// It is possible to tell Typescript how to

// handle some data. Typically this data is

// comes from the API.

// TODO: The fetch account method just fetch a general object,

// In our application we trust in the API. Assert it to an

// Account type (declared above) to be able to use it as an Account

// in the other parts of the application.

const fetchAccount = (id: number): object => ({id: id, name: "Some Account", currency: "USD"})

const currentAccount = fetchAccount(4) /\* add Type Assertion here \*/

const currentAccountName = currentAccount.name

// **Exercies 6) Literal types**

//

// This is an important exercise. If a type is a

// concerete value like "USD" or 7, it is handled as

// a type "constant". We have already used it in the

// first chapter in the Product.type property.

// Check here the variable types and the error messages.

//

// https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-aliases

type USD = 'USD'

type EUR = 'EUR'

// TODO: Correct the Currency type, to accept

// both EUR and USD. How can you define two possible

// types for one type? (we have seen before

// with numbers and strings).

type Currency = string

const firstCurrency: Currency = 'USD';

const secondCurrency: Currency = 'EUR'

const usd: USD = firstCurrency;

const eur: EUR = secondCurrency;

// TODO: When corrected the Currency type, another issue come up

// later in the code.

// Check the inferred type of the someAccount variable.

// It is inferred to string, but in the gerSomeCurrency

// function we using our Currency type. How add some Type

// assertion to the someAccount object to correct the later

// usage of the someAccount variable.

//

// https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-aliases

const someAccount = {

name: "My Awesome Account",

currency: "USD"

}

const getSomeCurrency = (account: {currency: Currency}) => account.currency

const someCurrency = getSomeCurrency(someAccount)

// **Exercise 7) null and undefined**

//

// Null and undefined are interchangeable

// in Javascript. In typescript it depends on

// the strictNullChecks compiler options.

// In this editor, and in our production code

// it is switched on. Check how does it works.

//

// https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#null-and-undefined

// TODO correct AccountWithOrWithoutCurrency or

// the removeCurrency function body to get rid off

// the type errors.

type AccountWithOrWithoutCurrency = {

name: string,

currency: 'USD' | 'EUR' | undefined

}

const removeCurrency = (account: AccountWithOrWithoutCurrency): AccountWithOrWithoutCurrency => {

return {

...account,

currency: null

}

}