1

Basic Technologies

- Hypertext Markup Language
- Cascading Style Sheets

2

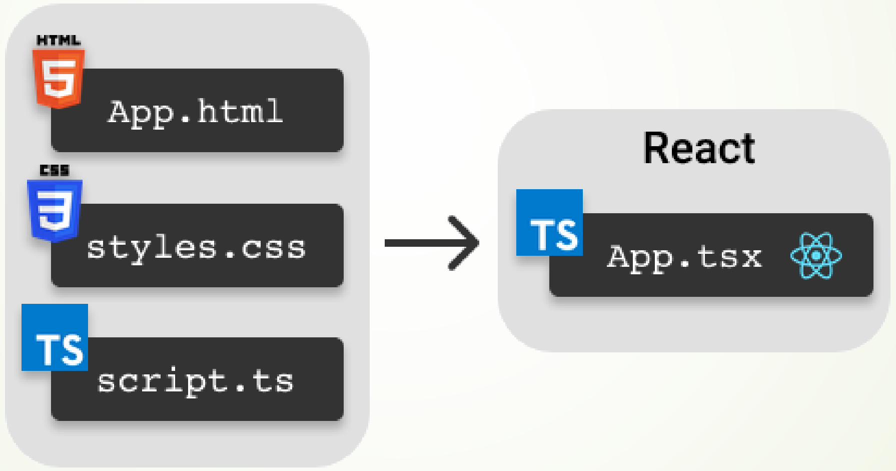
Interactive Web

- JavaScript
- Critical Rendering Path
- Json & Ajax
- jQuery
- JavaScript ES6
- Websockets
- TypeScript
- React

React

What is React

- open-source front-end JavaScript library for building user interfaces



- widely used as a base in building single-page websites and mobile applications
  - a single-page application is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages

II.2

React

Introduction

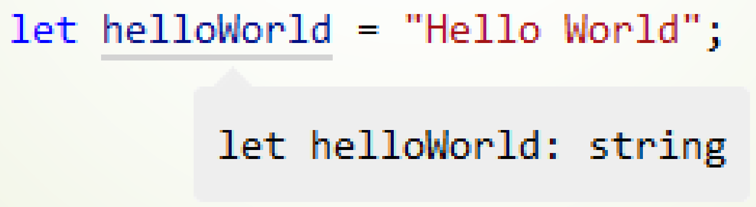
- a JavaScript library that originated at Facebook for building interactive user interfaces or UIs
- lets developers create sizeable web apps or complex UIs by integrating a small, isolated code snippet
- often called a framework because of its behavior and capabilities to build full-fledged applications. However, it is technically a library; it requires more libraries to form complex solutions.
- React + TypeScript
  intended to avoid typical problems that arise with the dynamic type system of JavaScript. These include improved maintainability of code in larger and long-lived applications.

TypeScript

What is TypeScript

- a syntactic superset of JavaScript
- offers all of JavaScript’s features,
- and an additional layer on top of these: TypeScript’s type system

it offers a type-system without needing to add extra characters to make types explicit in the code



```
let helloWorld = "Hello World";
```

```
let helloWorld: string
```

II.2

TypeScript

Why TypeScript?

- Each and every value in JavaScript has a set of behaviors that can be observed from running different operations
  - is `message` really callable?
  - does it have a property called `toLowerCase` on it?
  - if it does, is `toLowerCase` even callable?
  - if both of these values are callable, what do they return?
- use a static type system to make predictions about what the code is expected to do before it runs

```
// Accessing property
message.toUpperCase();

// calling something
message();




message = "Hello world";
```

II.2

TypeScript

Playground

- https://www.typescriptlang.org/play

TypeScript in VS Code

- required TypeScript compiler tsc
  - via npm - Node Package Manager
    a library and registry for JavaScript software packages
  - requires Node.js
    an asynchronous event-driven JavaScript runtime
    1. https://nodejs.org/en/download/
    2. npm install -g typescript
    3. tsc --version

II.2

TypeScript

TypeScript in VS Code

- compile *.ts-scipt via integrated terminal `tsc <script>`
- execute resulting script `node <script>`
- debug script
  - add config file tsconfig.json
  - start debugging using Node.js-debugger

```
{
  "compilerOptions": {
    "target": "ES2015",
    "module": "CommonJS",
    "outDir": "out",
    "sourceMap": true
  }
}
```

TypeScript

Defining Types

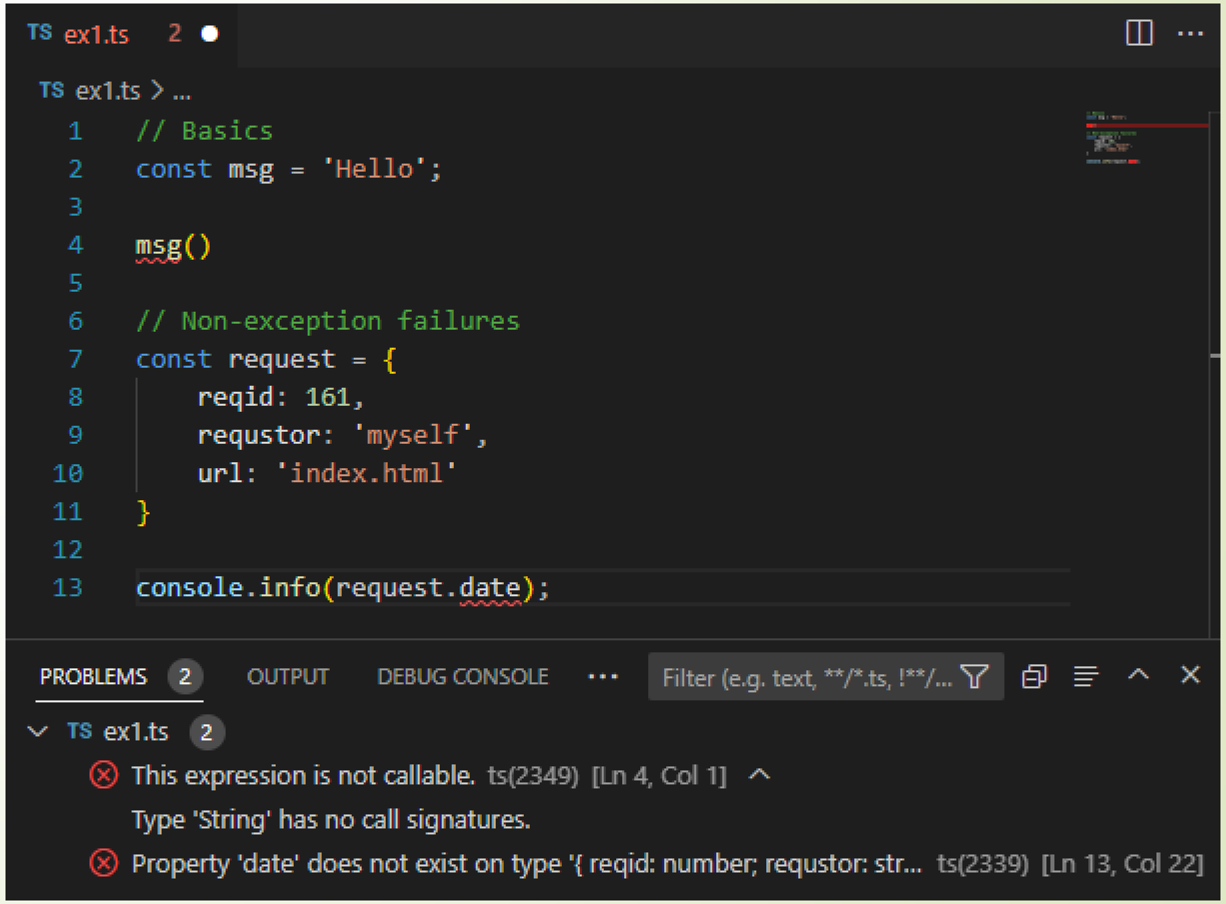
- TypeScript supports an extension of the JavaScript, which offers places to tell TypeScript what the types should be
- static types systems describe the shapes and behaviors of what values will be when being used uses that information and tells when things might be going wrong
- example
  - object creation
  - shape/interface of an object
  - interface declaration with class

```
interface User {
  name: string;
  id: number;
}

class UserAccount {
  name: string;
  id: number;

  constructor(name: string, id: number) {
    this.name = name;
    this.id = id;
  }
}

const user: User =
    new UserAccount("user_abc", 12);
```

TypeScript

Examples

- Basics: static type-checking



```
// Basics
const msg = 'Hello';

msg()

// Non-exception failures
const request = {
    reqid: 161,
    requstor: 'myself',
    url: 'index.html'
}

console.info(request.date);
```

```
This expression is not callable. ts(2349) [Ln 4, Col 1]
Type 'String' has no call signatures.
Property 'date' does not exist on type '{ reqid: number; requstor: str... ts(2339) [Ln 13, Col 22]
```

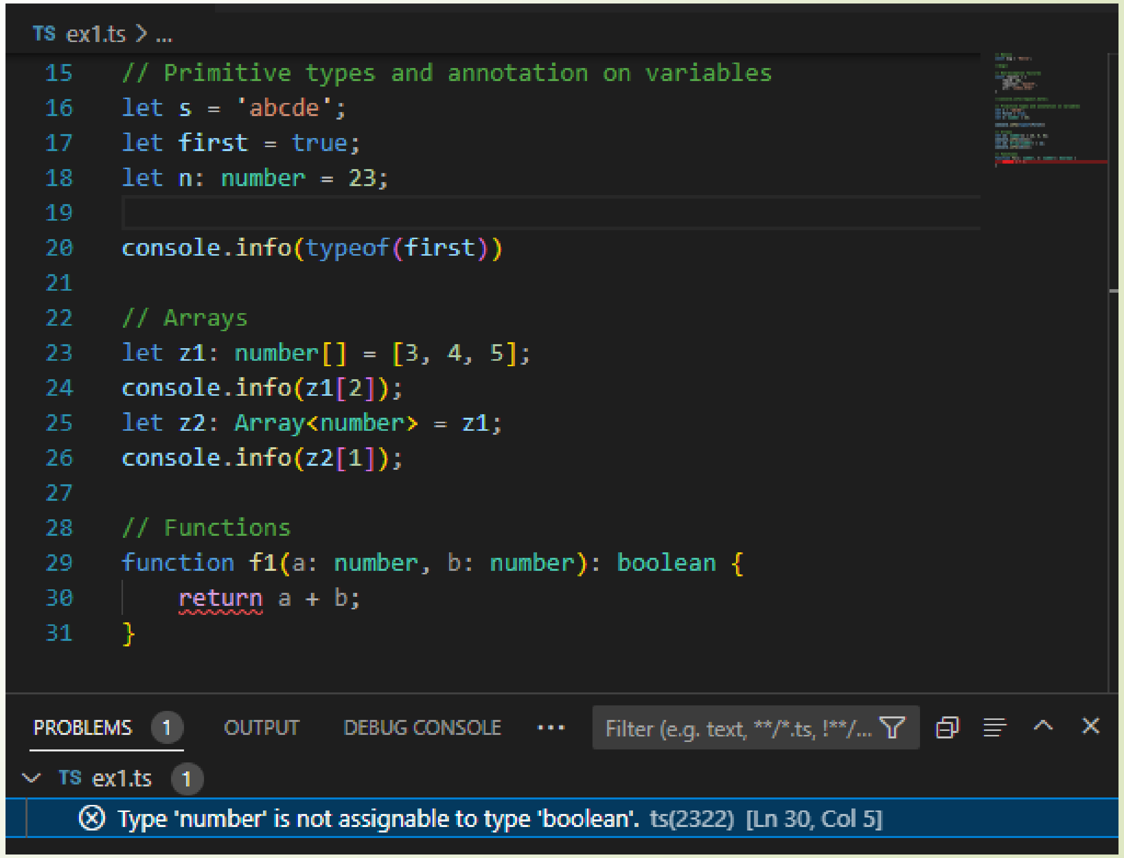
TypeScript

Everyday Types

- primitives: string, number, boolean
- arrays
- any

Type Annotations

- declaration of variables
- functions
  - formal parameters
  - return type



TS ex1.ts > ...

```typescript
// Primitive types and annotation on variables
let s = 'abcde';
let first = true;
let n: number = 23;

console.info(typeof(first))

// Arrays
let z1: number[] = [3, 4, 5];
console.info(z1[2]);
let z2: Array<number> = z1;
console.info(z2[1]);

// Functions
function f1(a: number, b: number): boolean {
    return a + b;
}
```

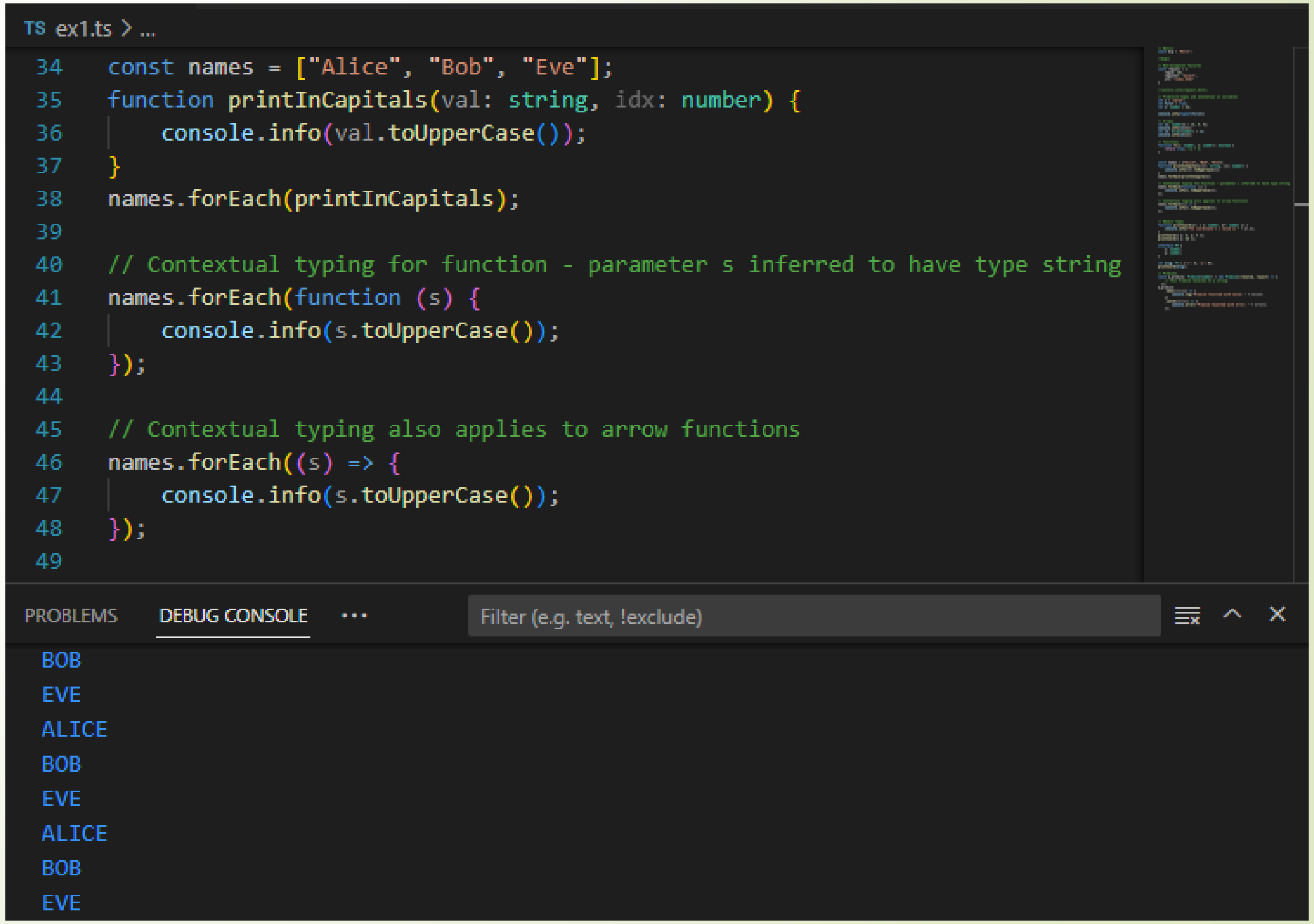
```
Type 'number' is not assignable to type 'boolean'. ts(2322) [Ln 30, Col 5]
```

II.2

TypeScript

Anonymous functions

- contextual typing: automatically determine type



TS ex1.ts > ...

```
const names = ["Alice", "Bob", "Eve"];
function printInCapitals(val: string, idx: number) {
    console.info(val.toUpperCase());
}
names.forEach(printInCapitals);

// Contextual typing for function - parameter s inferred to have type string
names.forEach(function (s) {
    console.info(s.toUpperCase());
});

// Contextual typing also applies to arrow functions
names.forEach((s) => {
    console.info(s.toUpperCase());
});

```

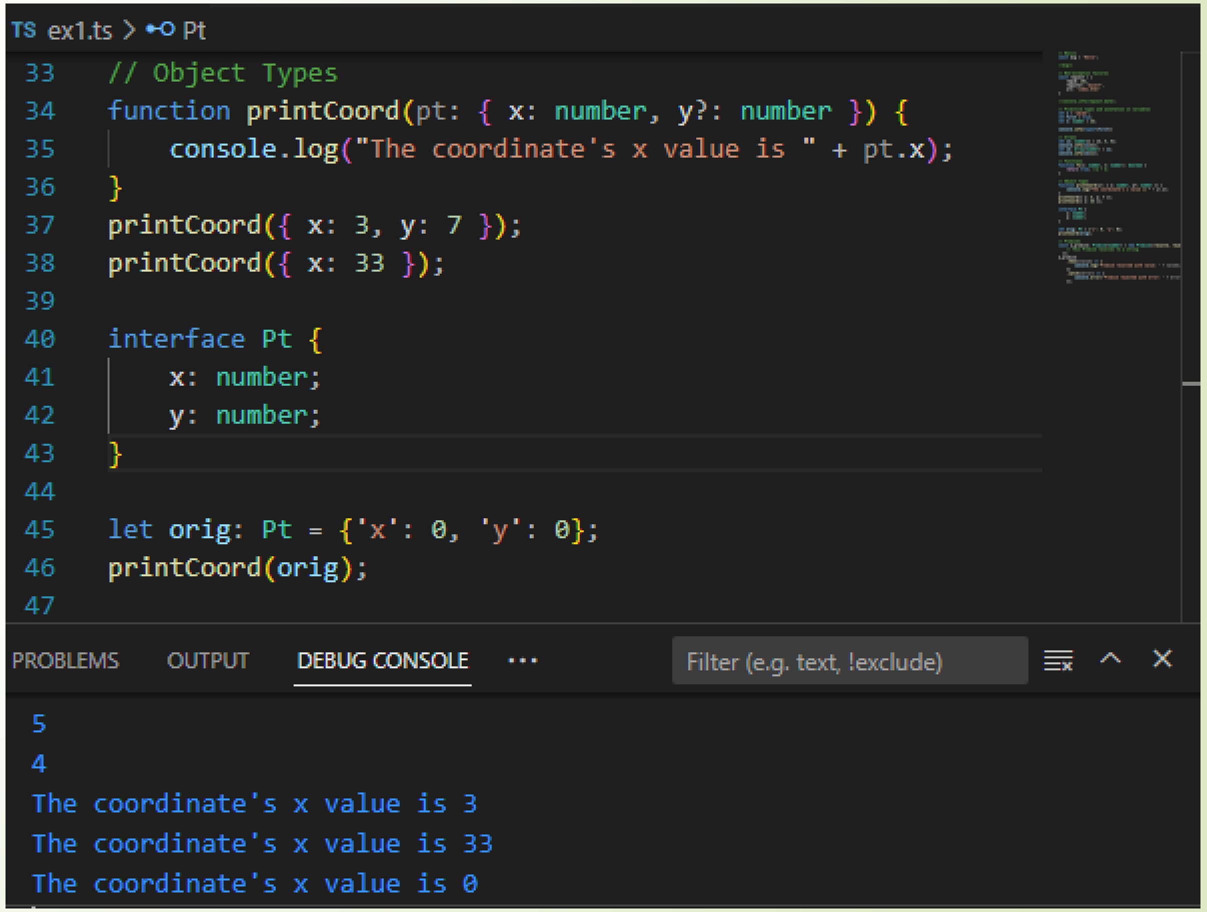
```
BOB
EVE
ALICE
BOB
EVE
ALICE
BOB
EVE
```

II.2

TypeScript

Object Types

- by type alias
- by listing
  - properties
  - and their types
- properties can be optional
- by interface declaration



```
// Object Types
function printCoord(pt: { x: number, y?: number }) {
    console.log("The coordinate's x value is " + pt.x);
}
printCoord({ x: 3, y: 7 });
printCoord({ x: 33 });

interface Pt {
    x: number;
    y: number;
}

let orig: Pt = {'x': 0, 'y': 0};
printCoord(orig);
```

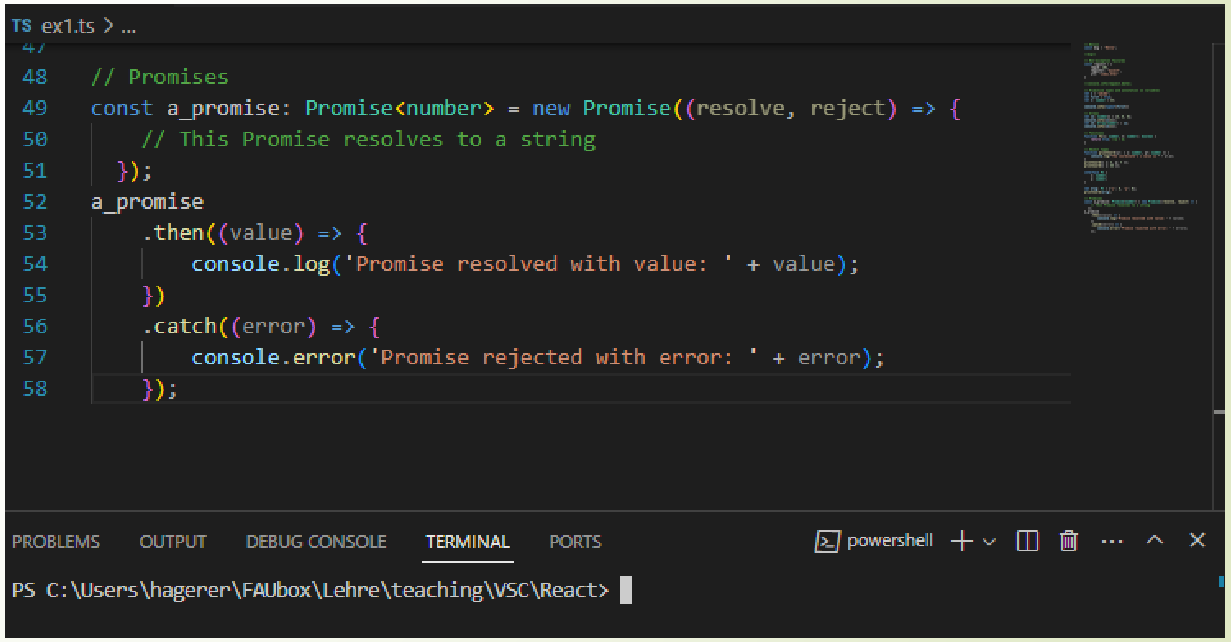
```
5
4
The coordinate's x value is 3
The coordinate's x value is 33
The coordinate's x value is 0
```

II.2

TypeScript

Promise

- as in Javascript
  handle asynchronous operations, providing better control over the flow of code
- created via constructor accepting a function which should take two parameters:
  - a function to resolve the promise
  - a function to reject the promise



```
// Promises
const a_promise: Promise<number> = new Promise((resolve, reject) => {
    // This Promise resolves to a string
  });
a_promise
    .then((value) => {
        console.log('Promise resolved with value: ' + value);
    })
    .catch((error) => {
        console.error('Promise rejected with error: ' + error);
    });
```

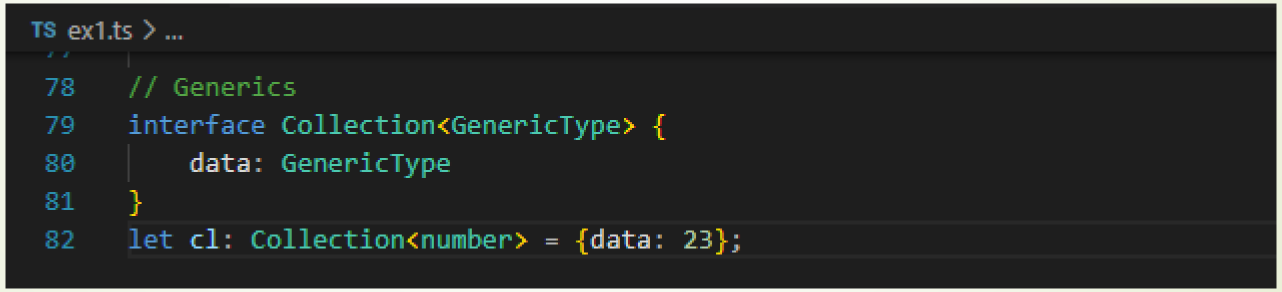
```
PS C:\Users\hagerer\FAUbox\Lehre\teaching\VSC\React>
```

II.2

TypeScript

Generics

- are used to assign multiple types to a function or variable without the value losing that specific type information upon return
- defined with < > brackets surrounding names of the generic types, like Array<T> or Map<Key, Value>



TS ex1.ts > ...

```
// Generics
interface Collection<GenericType> {
    data: GenericType
}
let cl: Collection<number> = {data: 23};
```