On Static Typing and First-class Functions

Please seat at tables with at least 3 people, ideally 4! **Only use Zones ABC. No DEF!**

Kris Jordan / August 26, 2024 / COMP423 / Class 03



Consider the following JavaScript function:

function add(a, b) { return a + b;

What is its return type?

Consider the following JavaScript code:

- let a = 423;
- a = a + ".9";
- console.log(a);
- a = Math.floor(a);
- console.log(a);

Are there errors? What is the output?

Dynamically Typed Languages

- Variable types are determined at runtime not at design/compile time
- No explicit type declarations* (because there's no type checking!)
 - * There are exceptions, like Python, where type annotations are now 1st class language features but they are not enforced at runtime.
- Flexible variable and parameter reassignment (e.g. a variable can be assigned any type of data)

to specify or check types at compile time... what could go wrong?

There is less upfront thought work and simpler programs when you don't need

Static Types and Software Engineering

"The competent programmer is fully aware of the strictly limited size of their own skull; therefore they approach the programming task in full humility, and among other things they avoid clever tricks like the plague." -Dijkstra

- Improve Code Quality
- Early, Automated Error Detection
- Better Developer Experience via Tooling

Type declarations make explicit the implicit and serve as documentation

• Static type checking (one form of static analysis) occurs ahead of runtime while your code is "at rest" (static!). Entire categories of errors fixed here.

• Static types make code autocomplete, automated refactoring, code navigation, and more features readily and unambiguously possible.

First-class Functions Consider the following TypeScript code:

return num > 0; } let test = isPositive; console.log(test(423));

What is the variable <u>test's type?</u> What is the output?

function isPositive(num: number): boolean {

First-class Functions Consider the following TypeScript code:

- function isPositive(num: number): boolean {
 - return num > 0;
- } let test = isPositive;
- console.log(`test.name: \${test.name}`);
- console.log(test === isPositive);
- In languages with first-class functions, functions are values that can be assigned to variables, passed to parameters, stored in data structures, and returned from function/method calls.

A Function's Type: Parameter Types + Return Type The type of a function is its "shape"

- What guarantees it is possible to substitute one function call for another, assuming the same arguments, in a piece of code?
 - Agreement between the function's parameter types and return type
- In TypeScript, there are multiple ways of specifying a function type. We will default to a *function type interface*:
 - interface Predicate {
 - (num: number): boolean;



Arrow Functions Short-hand Syntax for Defining lil Functions

function isPositive(num: number): boolean { return num > 0;

The above definition is "equivalent*" to:

let isPositive = (num: number): boolean => { return num > 0;

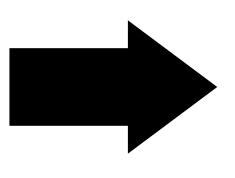
* The scenarios in which these definitions are not equivalent to one another requires a more nuanced understanding of JavaScript, including its prototypal object model and its *interesting* implementation of the **this** keyword.



}

}

Notice we can use an anonymous arrow function here!



```
interface Predicate {
    (num: number): boolean;
function filter(xs: number[], test: Predicate) {
    let result: number[] = [];
    for (let x of xs) {
        if (test(x)) {
            result.push(x);
    }
    return result;
console.log(
    filter(
        [1, -2, 3, -4, 5],
        (x: number): boolean => {
            return x < 0;
```

Structural Typing As opposed to reified, nominal typing, the shape of values determines valid uses.

- - expected.
- exploit anonymous functions.
 - As seen on the previous slide!

• Functions can conform to function type interfaces without declaring they do!

Contrast this with Java's class/implements relationship: just defining the methods of an interface is not enough, a class must also state it implements an interface in order to be used where the interface is

This is very powerful for enabling common usage scenarios in TypeScript that

 An interface specifies the shape of a value and any value that conforms to that shape is automatically considered to be an implementation of the interface.





Type Inference TypeScript can use context to infer types without explicit declarations.

- What is the type of x in the following statement: let x = "hello, world";
- What is the return type of the following function: let f = (x: number) => { return x * 2; }
- of filter, what is `x`'s type and what is the return type? filter([-1, 0, 1], (x) => { return x > 0});

• In the bolded anonymous arrow function, given the previous slide's definition

The Right-hand Example is Idiomatic Thanks to Inference

```
interface Predicate {
    (num: number): boolean;
}
function filter(xs: number[], test: Predicate) {
    let result: number[] = [];
    for (let x of xs) {
        if (test(x)) {
            result.push(x);
        }
    }
    return result;
}
console.log(
    filter(
        [1, -2, 3, -4, 5],
        (x: number): boolean => {
            return x < 0;
);
```

```
interface Predicate {
    (num: number): boolean;
function filter(xs: number[], test: Predicate) {
    let result: number[] = [];
    for (let x of xs) {
        if (test(x)) {
             result.push(x);
    return result;
console.log(
    filter(
        [1, -2, 3, -4, 5],
        (x) => \{
            return x < 0;</pre>
);
```



Software Engineering Lessons in TypeScript

- Better tools, better teams!
- Better verification, better user experiences!
 - time if using TypeScript, or equivalent*
- Well designed, layered systems can build confidence on top of simpler systems
 - TypeScript is a superset of JavaScript! (Taken further, all programming languages) ultimately transform down to machine code...)

*: https://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/gao2017javascript.pdf

 Adding static type annotations to the JavaScript language, alongside tools for checking, compiling, and niceties in IDEs like VSCode, the TypeScript language enabled software engineering teams to collaborate on large code bases more confidently and productively.

 2017 study out of University College London and Microsoft Research found 15% of JavaScript bugs that made it to production systems would have been found at compile

Before building a whole new system (or language), ask if you can build a new layer



