Intro to Typescript

Javascript The Bad Parts

> Wat
> https://www.destroyallsoftware.com/talks/wat

```
infinity/-0
[0, -1, -2].sort();
```

```
const array = [{"id":1,"name":"Leanne Graham","username":"Bret","email":"Sincere@april.biz","address":{"street":"Kulas Light","suite
const doh = array.map(index => {
  index.company = undefined;
  return index;
});
```

- Lack of support for object composition
- `this.foo === "anyones best guess"`

Fixing the Bad Parts

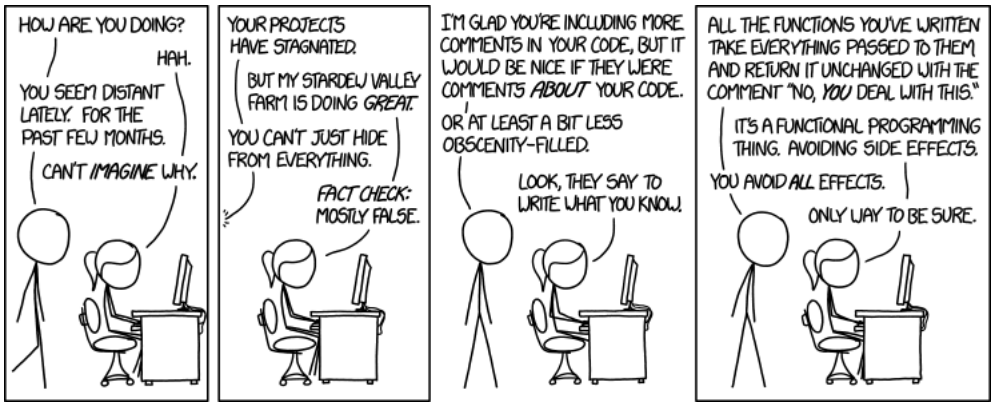
Functional Programming

- Javascript is the first major programming language to embrace the Functional Programming paradigm (Sorry Scala)
- Functional Programming is an approach of structuring software based on the mathematical concept of a function
- Functional Programming emphasizes
  - Writing software based on functions/c instead of classes.
    - Closures

```
function greatGrandparent() {
  let message = "I am a string";
  const grandparent = ()=> {
    function parent(){
      let child = (consoleMessage) => {
        console.log(consoleMessage)
      }
      child(message);
    }
    parent()
  }
```

```
  return {message, grandparent}
}
```

- Avoidance of side effects
  - A Side Effect is any operation that affects the outside scope of a function.



- Immutability of data
  - Javascript is a pass by reference language.
- Use of pure functions
  - Returns the same output for the same input.
  - Does not cause side effects outside the function

Typescript

- Typescript is a strongly typed superset of JavaScript that transpiles into plain JavaScript
- Typescript uses the same syntax JavaScript developers know
- Typescript incorporates standard OOP paradigms and cutting edge javascript features

JavaScript

```
var Todo = (function () {
        function Todo(todoId, todoText) {
                this.todoId = todoId;
                this.todoText = todoText;
        }
        Todo.prototype.getTodoId = function () {
                return (this.todoId);
        };
        Todo.prototype.setTodoId = function (newTod
                this.todoId = newTodoId;
        };
        Todo.prototype.getTodoText = function () {
                return (this.todoText);
        };
        Todo.prototype.setTodoText = function (newT
                this.todoText = newTodoText;
        };
        return Todo;
}());
```

TypeScript

```
class Todo {
        constructor(
                private todoId: number,
                private todoText: string
        ) {}

        getTodoId() : number {
                return(this.todoId);
        }

        setTodoId(newTodoId: number) {
                this.todoId = newTodoId;
        }

        getTodoText() : string {
                return(this.todoText);
        }

        setTodoText(newTodoText: string) {
                this.todoText = newTodoText;
        }
}
```

- Features include
  - strict type checking
  - Mixins/traits and interface support
  - nulll coalescing operator
  - optional chaining
- For a five minute intro to Typescript checkout Typescript handbook