Beyond TypeScript 101

Matthisk Heimensen - 2019 Xebia

TypeScript 101

Basic Types

```
type Primitive = number
               | boolean
               | string;
```

TypeScript 101

Variable declaration

```
let color: string = "red";
```

TypeScript 101

Functions

```
function add(x: number, y: number): number {
  return x + y;
}
```

TypeScript 101

Interfaces

```
interface Drawable {
  draw(): void
}
```

TypeScript 101

Generics

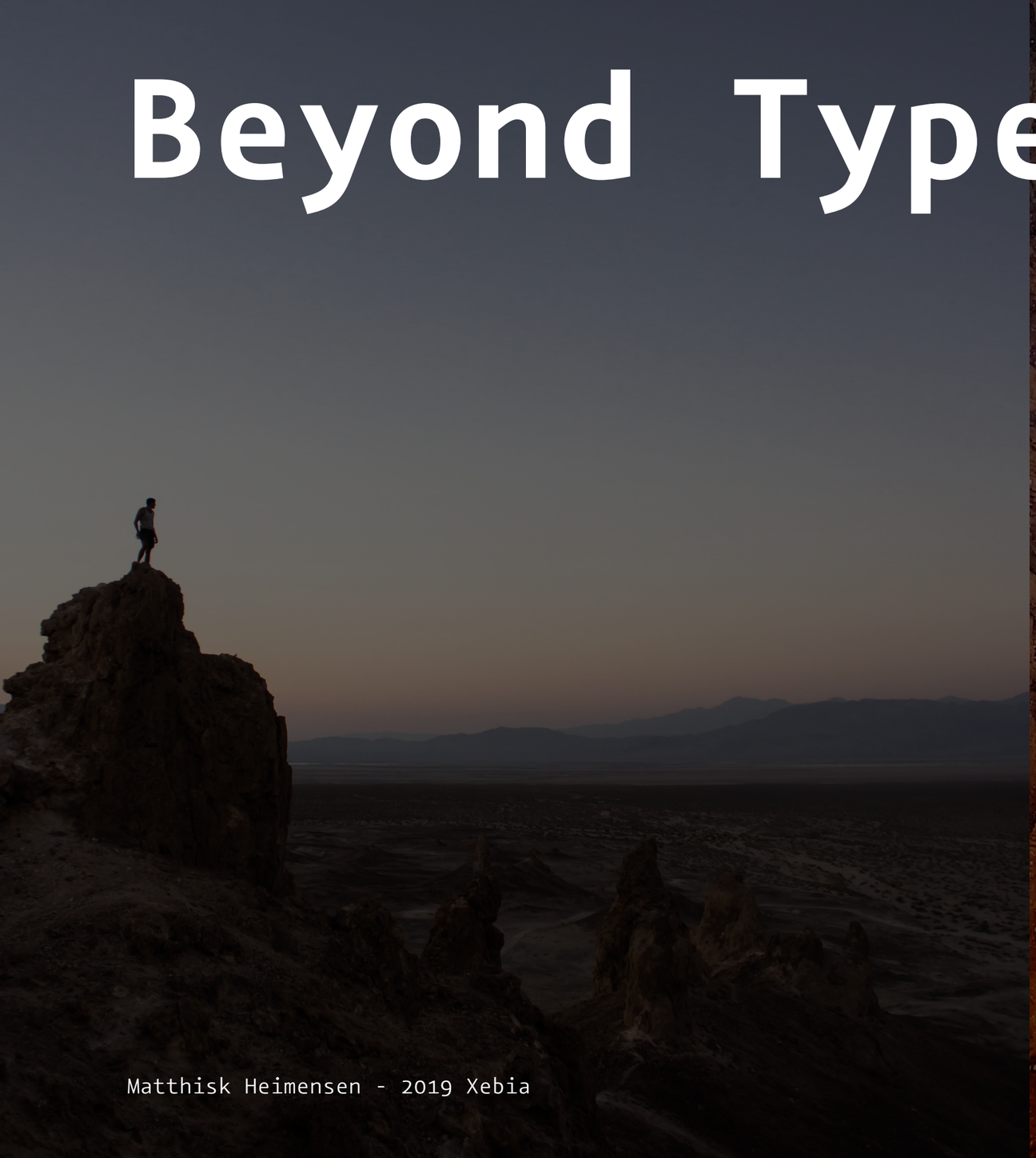
```
function tuple2<T, U>(arg0: T, arg1: U): [T, U] {
  return [arg0, arg1];
}
```

TypeScript 101

Type aliases

```
type MyNumber = number;
```

Beyond TypeScript 101



Matthisk Heimensen - 2019 Xebia

Type this!

Underscore.js

```
/**
 * Return the property name from all elements in a list
 */
_.pluck(list, propertyName)

/**
 * Return a copy of the object, filtered to only have
 * values for the whitelisted keys (or array of valid keys).
 */
_.pick(object, *keys)
```

Type this!

Object.freeze

```
/**
 * freezeObject makes an object readonly
 * and prevents new properties from being added.
 */
Object.freeze;
```

Type this!

Validating form values

```
/**
 * Validate form fields, returning a partial variant of the input object
 * with form field errors as property values.
 */
declare function validateForm(formValues);
```

Type this!

object rest property

```
function itemWithoutAddress(item) {
  const { address, ...rest } = item;
  return rest;
}
```

Type this!

Object.assign

```
/**
 * Use `Object.assign` to initialize class instance properties.
 */
class Pizza {
  slices: number
  name: string

  constructor(init) {
    Object.assign(this, init)
  }
}
```

TypeScript to the rescue

1. Index types
2. Mapped types
3. Conditional types

1) Index types

How do we express dynamic member selection in our typesystem?

```
function get(item, propertyName) {
  return item[propertyName]
}
```

1) Index types

```
type Keys<T> = keyof T;

type MemberSelection<T, K> = T[K];
```

Code examples

2) Mapped types

How do we express the transformation of an input type to a different output type?

```
/**
 * Return a copy of the object, filtered to only have
 * values for the whitelisted keys (or array of valid keys).
 */
_.pick(object, *keys)
```

2) Mapped types

How do we express the transformation of an input type to a different output type?

```
type Keys = 'option1' | 'option2';

type Flags = { [K in Keys]: boolean };
```

Code examples

3) Conditional types

How can we express non-uniform type mappings?

```
x > 0 ? true : false;
```

3) Conditional types

How can we express non-uniform type mappings?

```
T extends U ? X : Y
```

Code examples

Matthisk Heimensen

***Twitter*: @tthisk**

***Github*: github.com/matthisk**

References

- http://www.typescriptlang.org/docs/handbook/advanced-types.html
- https://github.com/piotrwitek/utility-types
- https://github.com/matthisk/beyond-ts-101