
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