

# Training TypeScript

## Module: Type Queries



Peter Kassenaar

[info@kassenaar.com](mailto:info@kassenaar.com)

# We know the typeof operator

Types in plain JavaScript :

```
console.log(typeof 123); // 'number'  
console.log(typeof 'this is a string'); // 'string'  
console.log(typeof [10, 20, 30]); // 'object'  
console.log(typeof {}); // 'object'
```

In TypeScript we can use typeof in a slightly different manner

```
// Lets say we have a type/object literal as such  
const employee = {  
  name: 'Boris',  
  age: 27  
};
```

```
// we can now QUERY the employee object and assign it's properties  
// to the Person type.
```

```
type Person = typeof employee;
```

```
let person: Person = {  
  name: 'Brandon',  
  age: 19  
};
```

```
20 // we can no type Person = { assign it's properties  
21 // to the Pe name: string;  
22 type Person age: number;  
23 }  
24 let person: Person = {  
25   name: 'Brandon',  
26   age: 19  
27 };
```

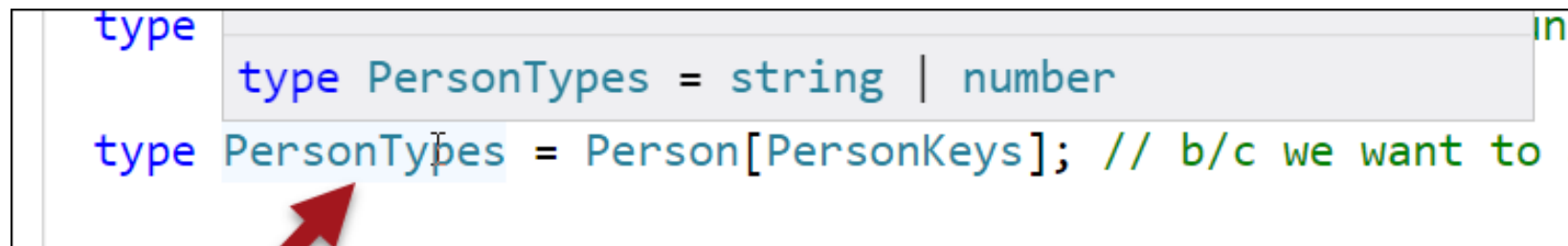
In JavaScript, `typeof` would have given us `'object'`.

In TypeScript, it gives us the properties of the inferred type!

# Keyof index type queries

```
// PersonKeys is now a union type "name" | "age"  
type PersonKeys = keyof Person;
```

```
// b/c we want to access the exact types  
// of the 'name' and 'age' properties  
type PersonTypes = Person[PersonKeys];
```



```
type PersonTypes = string | number  
type PersonTypes = Person[PersonKeys]; // b/c we want to
```

# Creating a custom Lookup function

Let's say we want to look up the value of a given property on an object – BUT DO IT TYPE SAFE


This would be the type-unsafe notation:

```
function getPropertyValue(obj : object, key: string) {  
    return obj[key]; // type unsafe way  
}
```

Solution: create a generic type, and use extends keyof:

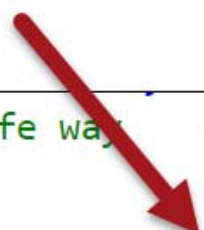
```
function getPropertyValue<T, K extends keyof T>(obj: T, key: K) {  
    return obj[key]; // type safe way  
}  
  
const personName = getProperty(person, 'name');
```

*“K is a subtype of T, and has the keys that are looked up”*



```
const personName: string
const personName = getProperty(person, 'name');
console.log(personName);
```

First of all – we get the correct typing on the variable the value is assigned to.



```
return obj[key]; // type safe way
}
const personName = getProperty(person, 'test');
console.log(personName);
```

[ts] Argument of type '"test"' is not assignable to parameter of type '"name" | "age"'.  
~~~~~

Second – we get an error if we try to look up a non-existing property

# Workshop

- Study the example `../21-type-queries.ts`
- Create an example of your own, using one of your own data types:
  - 1. Use `typeof` as a type query
  - 2. Use `keyof` and see what type is returned
  - 3. Create a lookup function that returns the value of one of the properties of your own custom object

