# Generics and the Type Level

# Type level programming

```
type AType = SomeOtherType

type _check = AssertAssignable<any, number>;

const foo = 1;
type FooType = typeof foo; // FooType is number
```

# Subscripting Types

```
type O = {
    foo: number,
    bar: string
}

type _1 = AssertAssignable<number, O['foo']>
type _2 = AssertAssignable<string, O['bar']>

type _3 = AssertAssignable<number | string, O['foo' | 'bar']>
```

# `keyof` and subscripting unions

```
type O = {
  foo: number,
  bar: string
}

type _1 = AssertAssignable<'foo' | 'bar', keyof O>

type _2 = AssertAssignable<number | string, O['foo' | 'bar']>

type _3 = AssertAssignable<number | string, O[keyof O]>
```

# Generics

```
type LList<T> = Cons<T> | null;
interface Cons<T> {
  value: T;
  rest: LList<T>;
}
const numberList: LList<number> = {
  value: 1,
  rest: {
    value: 2,
    rest: null
  }
};
```

# Generic functions

```
function first<T>(list: LList<T>): T {
  return list.value;
}

// The type variable `T` is derived from the type of `llist`
const val = first(numberList);
// val is statically known to be a number
```

# Generic type constraints

```ts
export type AssertAssignable<T1, T2 extends T1> = never;
```

```ts
function get<T extends object, K extends keyof T>(obj: T, key: K): T[K] {
  return obj[key];
}

const x = get({foo: 1}, 'foo')

type _1 = AssertAssignable<number, typeof x>
```

# Exercise 7

- Start with some basic type-level programming and work toward increasing sophistication.
- We'll finish with generics in the last exercise.
- Comment out each test after making it pass. The prior tests will break when you make subsequent tests pass.