Types

Syntax

Add type annotations with a colon `:`, right after the declaration

```
let name: string = 'nicojs';

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

let list: number[] = [2, 4, 5]; // Synonym: var list:Array<number>

name = true;
// => Error: Type 'boolean' is not assignable to type 'string'

list = add;
// => Error: Type '(x: number, y: number) => number' is not assignable to
type 'number[]'.
```

When initializing a variable while declaring, the type annotation can be left out.

Syntax explained

Types...

- Are optional
- Only at compile time
  - They are simply removed when transpiled to JavaScript
- Closely resembles JavaScript types
  - `string`, `boolean`, `number`, `array`, `undefined`, `null`, `object`
- Inferred when possible
- Comparable to static code analysis

Basic types

The primitive types match one-to-one to the JS types

```
const n = 3;
const pi = 3.14;
const bigInt = 9007199254740992n
const str = 'a string using "single quotes"';
const symbol = Symbol('foo');
```

```
const bool = true;
const nu77 = null;
const und = undefined;
```

Strings

```
const str = "a string using 'double quotes'";
const str2 = 'a string using "single quotes"';
const str3 = `a string using back thicks
can be multi line
pi is: ${pi}, or (${Math.floor(pi)})`;
```

String interpolation is supported with "backtick" (`)

Array types

Arrays work like JS arrays.

```
const list = [1, 2, 3];
let numbers: Array<number>;
let numbers2: number[];
numbers = numbers2 = list
```

... but have a *generic type*. More on generics later.

Tuples

Like an array, but with different kind of known types

```
let person: [string, number];
person = ['Henk', 20]; // => OK
person = [10, 'Henk']; // => Syntax error
// Type `[number, string]` is not assignable to type `[string, number]`.
```

Very useful in combination with type inference

```
console.log(person[0].substr(1)); // OK
console.log(person[1].substr(1)); // Error, `number` does not have
`substr`
```

Question: When would you use this?

Note: For example tuples is the result of `Promise.all const result = await Promise.all([p1, p2]);`

Fixed Length Tuples

```
let person: [string, number];

person = [42, 'foo', 'bar'];
// => error: Type '[number, string, string]' is not assignable to type
'[number, string]'
```

TS < 2.7 is less strict

```
person = [42, 'foo', 'bar'];

person[0]; // => type number
person[1]; // => type string
person[2]; // => type string | number
```

Enums

An `enum` is a map of numeric values and string counter parts

```
// TypeScript
enum MessageKind {Start, Run, Stop};
const message = MessageKind.Start;
```

String enums

We can also use string enums

```
// TypeScript
enum Colors {
    Red = "RED",
    Green = "GREEN",
    Blue = "BLUE",
}
```

The `any` type

If you need the dynamic nature of JavaScript, declare a variable as `any`.

```
let myObject: any = JSON.parse('{something: 3}');
```

Reassign to anything

```
myObject = '';
myObject = 3;
```

Be careful: TypeScripts type checking will be completely disabled

```
let a: any = 4;
a.substr(3);        // OK for TypeScript, crash at runtime
let b: boolean = a; // OK for TypeScript, crash at runtime
```

Implicit `any`

An uninitialized variable without type annotation is implicitly `any`

```
let a;
a = 'Hello';
a = 3;
a.substr(); // OK for TypeScript
```

No implicit `any`

Disable implicit `any` support with: `--noImplicitAny`

```
function log(message) {
}
// => error: Parameter 'message' implicitly has an 'any' type.
```

TypeScript tries to infer types when possible.

```
let a;

a = '123';
a.length; // => OK
```

```
a = 3;
a.length;
// => Error: Property 'length' does not exist on type 'number'.
```

Type: `undefined` and `null`

- The `undefined` type has a singleton value: `undefined`.
- The `null` type has a singleton value: `null`.

```
let a: null = null;
a = 42;
// => error: Type '42' is not assignable to type 'null'.
```

- `undefined` and `null` are sub-types of every other type.

```
let a = 3;
a = null;      // => OK
a = undefined; // => OK
```

Strict null checks

Disable the sub-typing using `--strictNullChecks` compiler option

```
let a: number = 3;
a = null;
// => error: Type 'null' is not assignable to type 'number'
a = undefined;
// => error: Type 'undefined' is not assignable to type 'number'
```

Type: `void`

`void` is a type that has 2 values: `undefined` and `null`

```
function info(message: string): void {
    console.log(message);
}

let a = info('');
a = null;      // => OK
a = undefined; // => OK
```