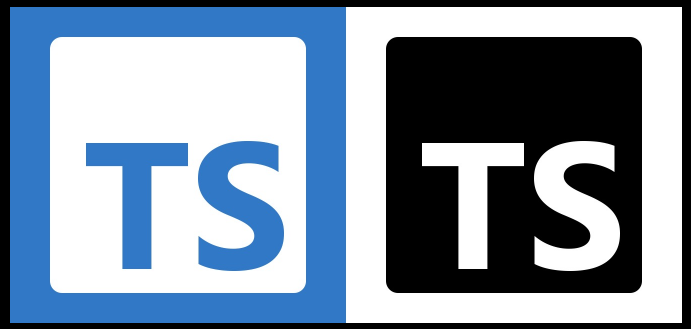


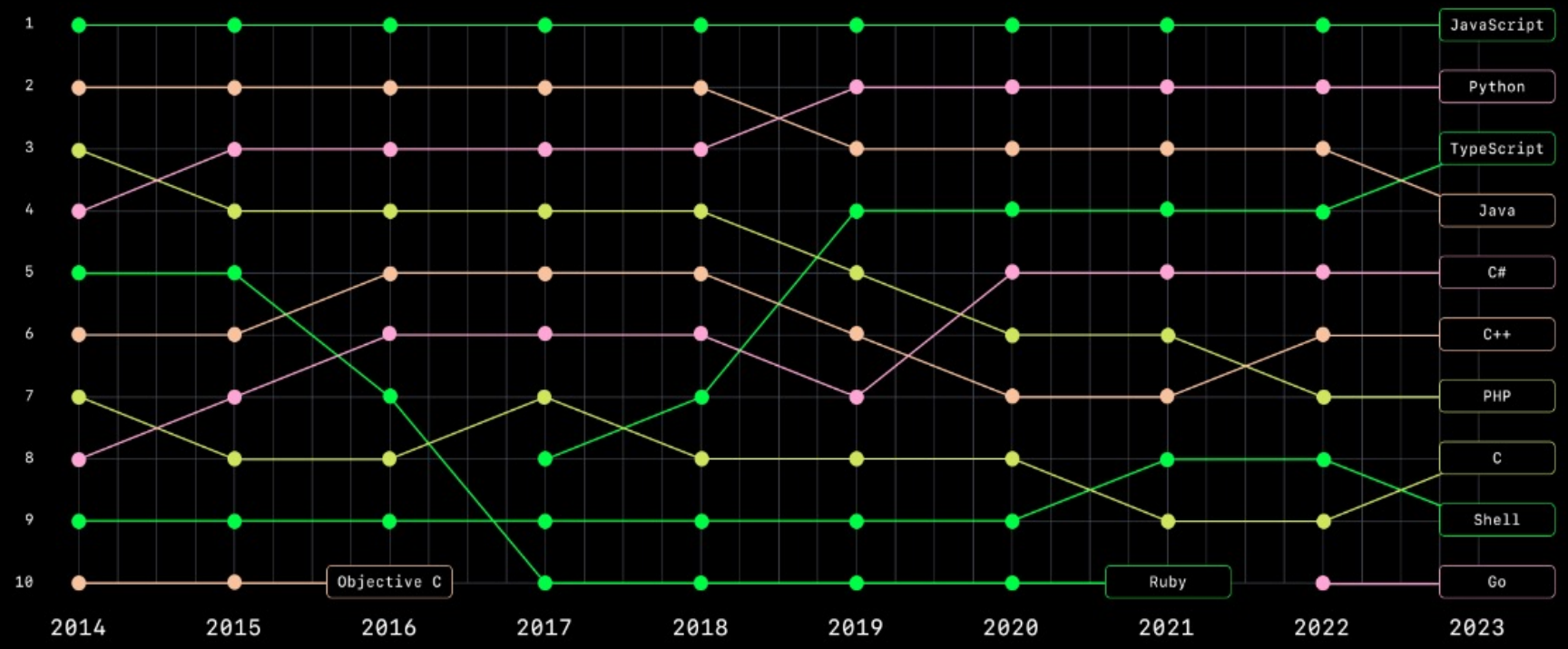
TypeScript

Source code - https://github.com/diarmuidoconnor/typescript-demos

Background

- Open source language, developed by Microsoft (2010–12).
- Anders Hejlsberg – the creator of C# and Turbo Pascal
- Based on ECMAScript 4 and 6.
- A superset of Javascript.
- We still write JS, but augmented by the class-based OOP of ES6, and the structural type system of ES4.
- TS is compiled to regular JS and runs in any browser, any host, and OS.
- “… and one thing TS got right: local type inference” Bernard Eich
- “What impressed me the most is what TS doesn't do; it does not output type checking into your JS code” Nicholas C Zakas .
- TS is a a language for application-scale JavaScript development.

Top 10 programming languages on GitHub



File Extensions.

- .ts – source code file extension.
- .d.ts – declaration files.

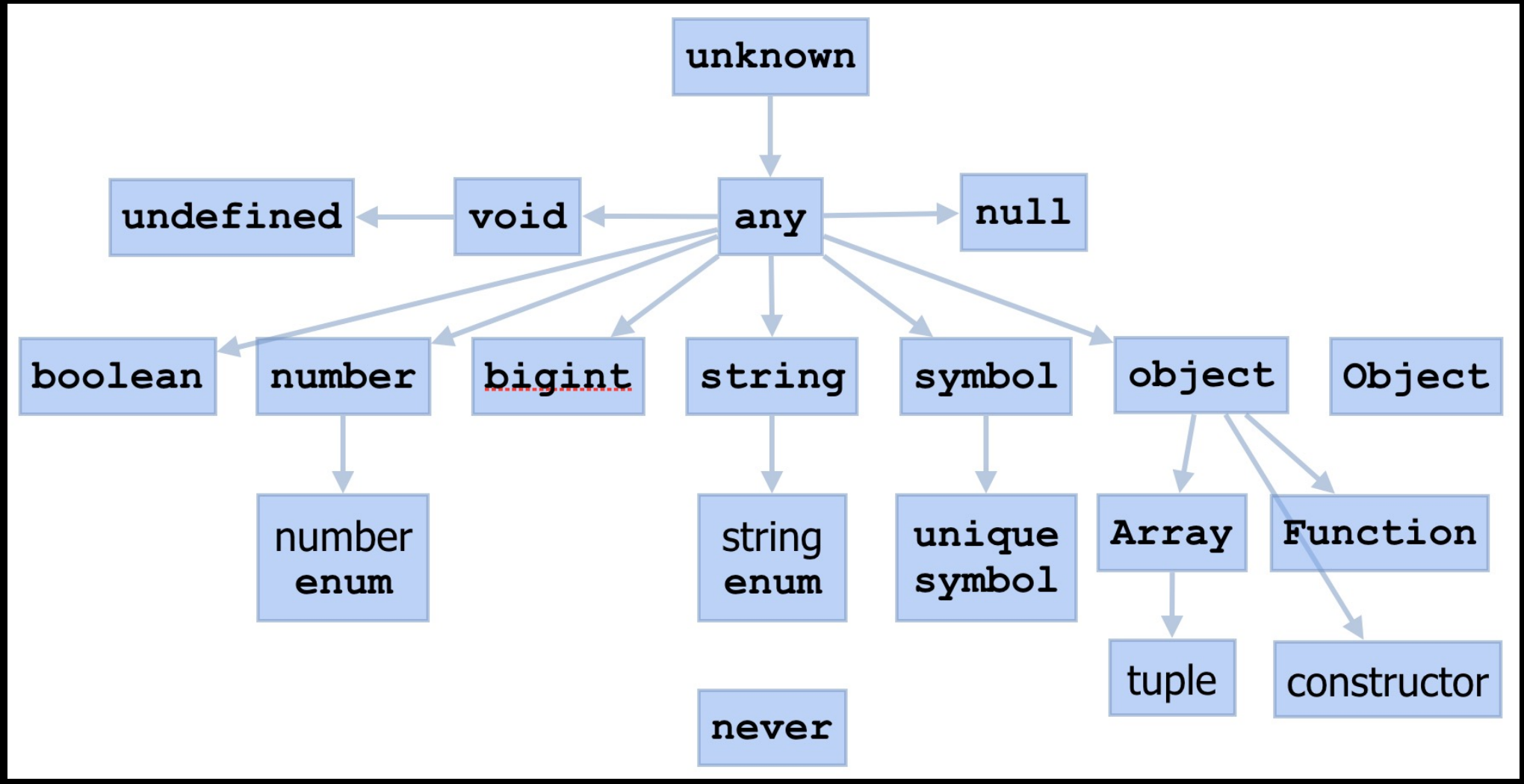
- Declaration source files:
  - Provide type definitions, separate from the source code.
  - Analogous to header files in C/C++.
  - Also used to describe the exported virtual types of a third-party JavaScript library, allowing TS developers to consume it.
  - Tooling – Gives type safety, intellisense and compile error detection during development.

Types

- Primitive Types:
  - number – represents integers. Floats, doubles.
  - boolesn
  - string – single or double quote.
  - null.
  - undefined.

- Object Types:
  - Class, module, interface and literal types.
  - Supports typed arrays.
- The Any type:
  - All types are subtypes of a single top type called the Any type.
  - Represents any JavaScript value with no constraints.

TypeScript Type Hierarchy



Type Annotations.

- (Optional) static typing.
- Lightweight way to record the intended contract of a variable or function.
- Applied using a post-fix syntax.
  e.g. let me : string = “Diarmuid O; Connor”

- Typed Array:

```
let myNums: number[ ] = [1, 2, 3, 5]
```

- Can also apply annotations to function signature:

```
function add(a: number, b: number) {
    return a + b;
}
```

Classes

- Support for ECMAScript 6 alike classes.
- public or private member accessibility.
- Parameter property declarations via constructor.
- Supports single-parent inheritance.
- Derived classes make use of super calls to parent methods..

```
class Animal {
    constructor(public name) { }
    move(meters) {
        alert(this.name + " moved " + meters + "m.");
    }
}

class Snake extends Animal {
  move() {
    alert("Slithering...");
    super.move(5);
  }
}

class Horse extends Animal {
  move() {
    alert("Galloping...");
    super.move(45);
  }
}
```

Interfaces

- Designed for development tooling support only.
- No output when compiled to JavaScript.
- Open for extension (may declare across multiple files).
- Supports implementing multiple interfaces.

```
interface Drivable {
    start(): void;
    drive(distance: number): void;
    getPosition(): number;
}

class Car implements Drivable {
  private isRunning: bool = false;
  private distanceFromStart: number;

  public start(): void {
    this.isRunning = true;
  }
  public drive(distance: number): void {
    if (this.isRunning) {
      this.distanceFromStart += distance;
    }
  }
  public getPosition(): number {
    return this.distanceFromStart;
  }
}
```

Interface Data Types.

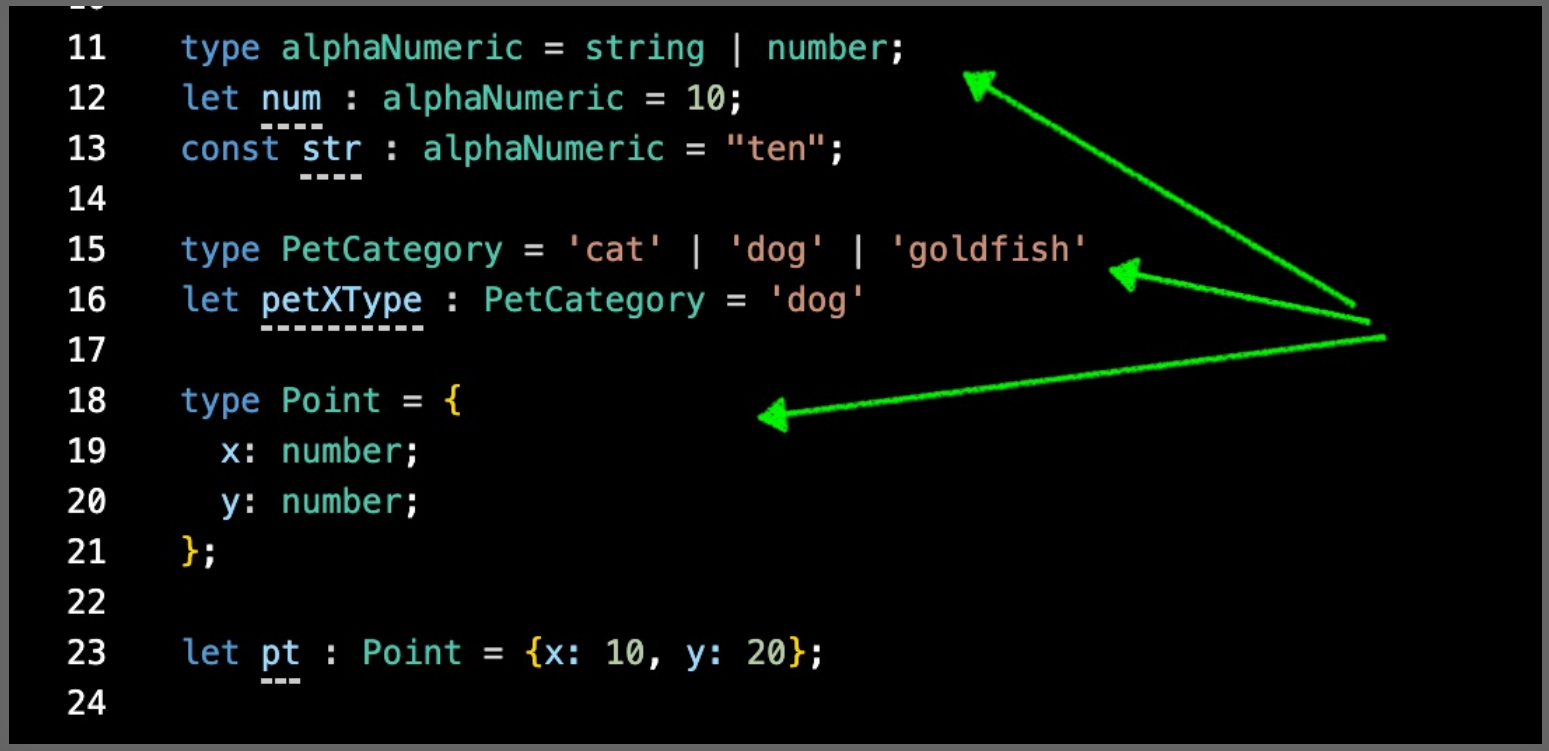
- An interface data type tells the TypeScript compiler about the property names an (data) object must have and their corresponding value types. Therefore, interface is a type and is an abstract type since it is composed of primitive types.

```
interface Person {
   first: string;
   last: string;
}

const me: Person = {
   first: "diarmuid",
   last: "o connor",
};
```

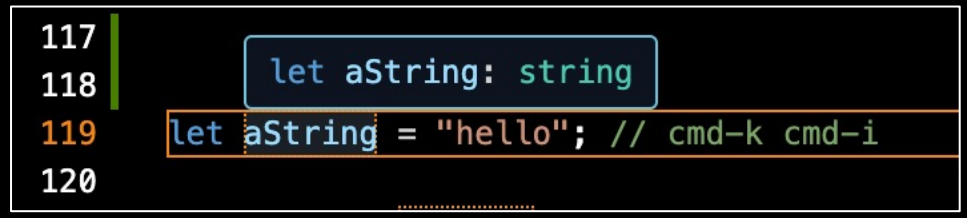
Type Aliases.

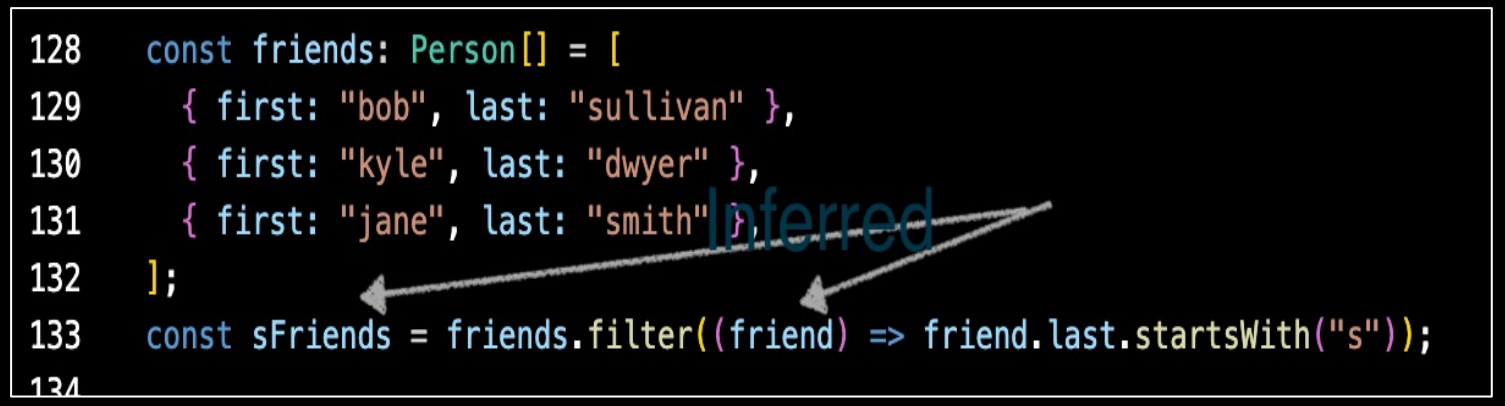
- Type aliases create a new name for a type. Type aliases are sometimes similar to interface data types, but can name primitives, unions, tuples, and any other types.



Type Inference.

- TS compiler can infer the types of variables based on their values.





- Inferencing increases developer productivity.

Functions

- Declaring the types in a function's signature.

```
4   function addNumbers(a: number, b: number): number {
5     return a + b;
6   }
```

- Compiler can often infer the return type.

```
8
9   TS i   function addtoNumberArray(nums: number[], inc: number): number[]
10  export function addtoNumberArray(nums: number[], inc: number) {      You, 8 m
11    const newNums = nums.map((num) => num + inc);
12    return newNums;
13
```

Higher Order Functions.

- Declaring the callback's type in a custom HOF.
  callback : (param1: type, param2: type, …) => return_type

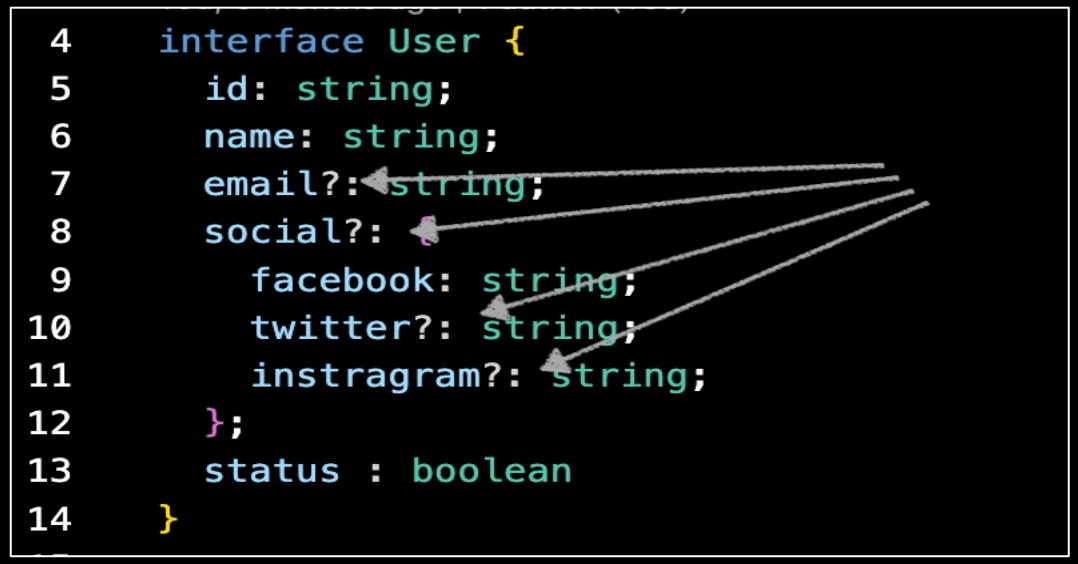
```
4   export function printToConsole(
5     text: string,
6     callback: (s: string) => string
7   ): void {
8     const response = callback(text);
9     console.log(response);
10  }
```

```
12  export function arrayMutate(
13    numbers: number[],
14    mutate: (num: number) => number
15  ): number[] {
16    return numbers.map(mutate);
17  }
```

- Can use type aliases to improve readability of calllback's signature.

Optionals

- Optional object properties are properties that can hold a value or be undefined.



- May also be used with function parameters.
  - An optional parameter cannot precede a required one.
  - Must accommodate undefined case in body – otherwise compiler errors may arise.

Union types & Type Literals

- Union types are used when a value can be more than a single type, e.g.

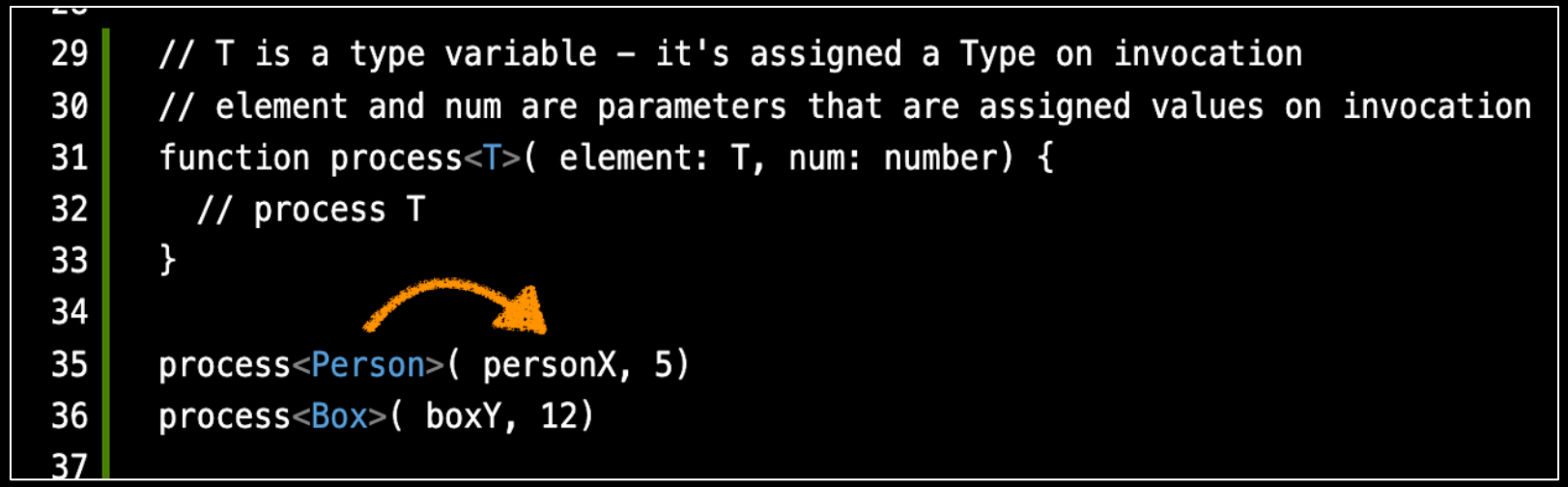
```
type Size = string | number. // Union type
let glassSz : Size = ‘medium’
let bottleSz: Size = 2. // litre
type Role = Student | Lecturer | Manager    // Union type
```

- Literal types – three sets of literal types available in TS: strings, numbers, and booleans; by using literal types you can allow an exact value which a string, number, or boolean must have.

```
e.g.  type DegreeNomination = ‘BSc’   | ‘BEng’  | ‘BA’  | ‘BBs’
```

Generics

- A major part of software engineering is building components that not only have well-defined and consistent APIs, but are also reusable, i.e. can be used for multiple data types.
- Generics allow creating 'type variables' to create classes, functions & type aliases that don't need to explicitly define the data types that they use.



Utility types

- TypeScript provides several utility types to facilitate common type transformations.
- These utilities are available globally.