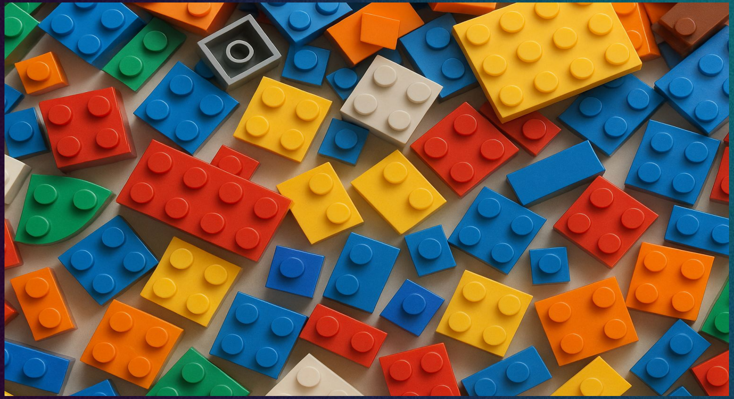
Type Level Programing



The next frontier of advanced abstractions and safety.

Diego Pacheco

What it Type-Level Programing?

- Functional Programing (FP)
- Actually advanced FP
- Types as first-class citizen
- Compile-time

What is a variable?

How many parts can we split a variable?

ID (Identifier)

Type*

Value*

What is the issue with variables?

What is the issue with bad Objects?

1. Is it safe to use?
2. Do I need to run a validator function?
3. Did I run it or not?

Functions are usually Runtime (unless your in doing comptime in Zig).

Good Objects: Constructor

Safer for the consumer.

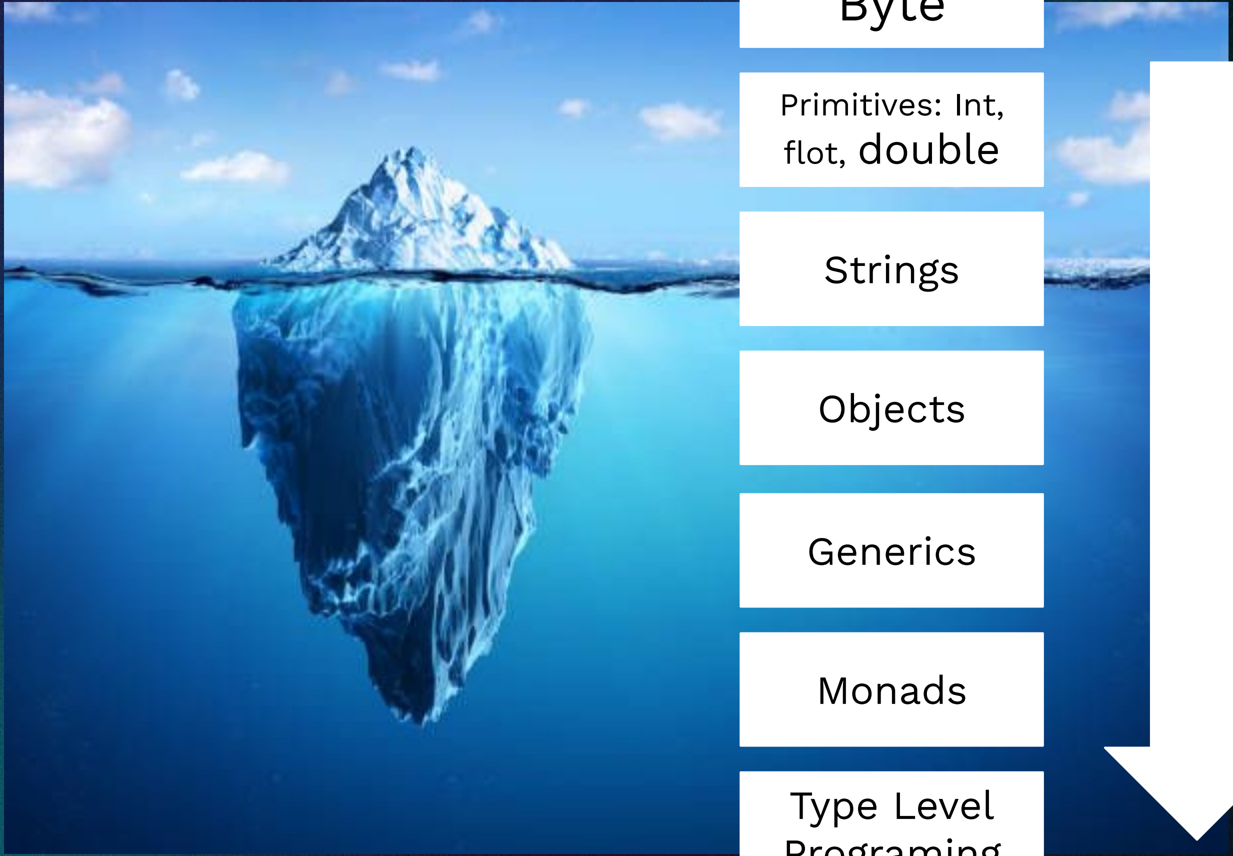
How much Safety assign a variable give to you?

What if is a String?

Strings accept Anything...

Forcing you to validate after the assignment.

Not all things
Provide the same
Levels of abstractions /
Correctness ✓



Byte

Primitives: Int, flot, double

Strings

Objects

Generics

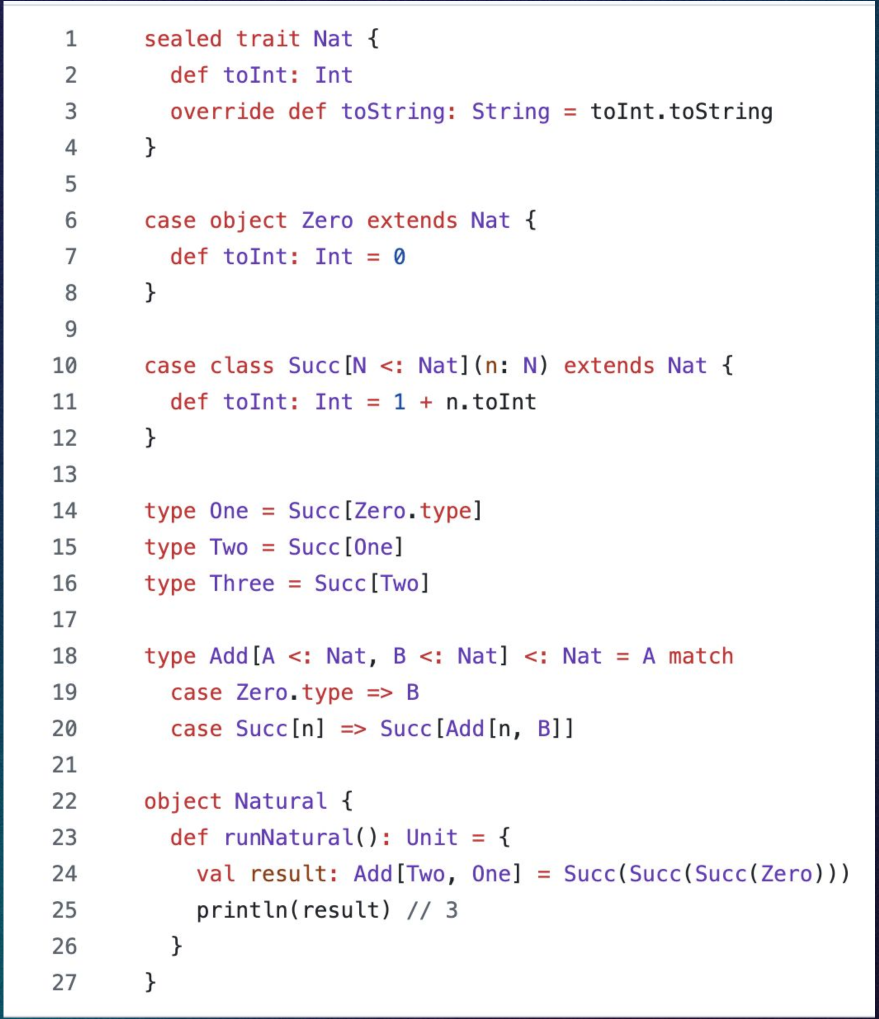
Monads

Type Level Programing

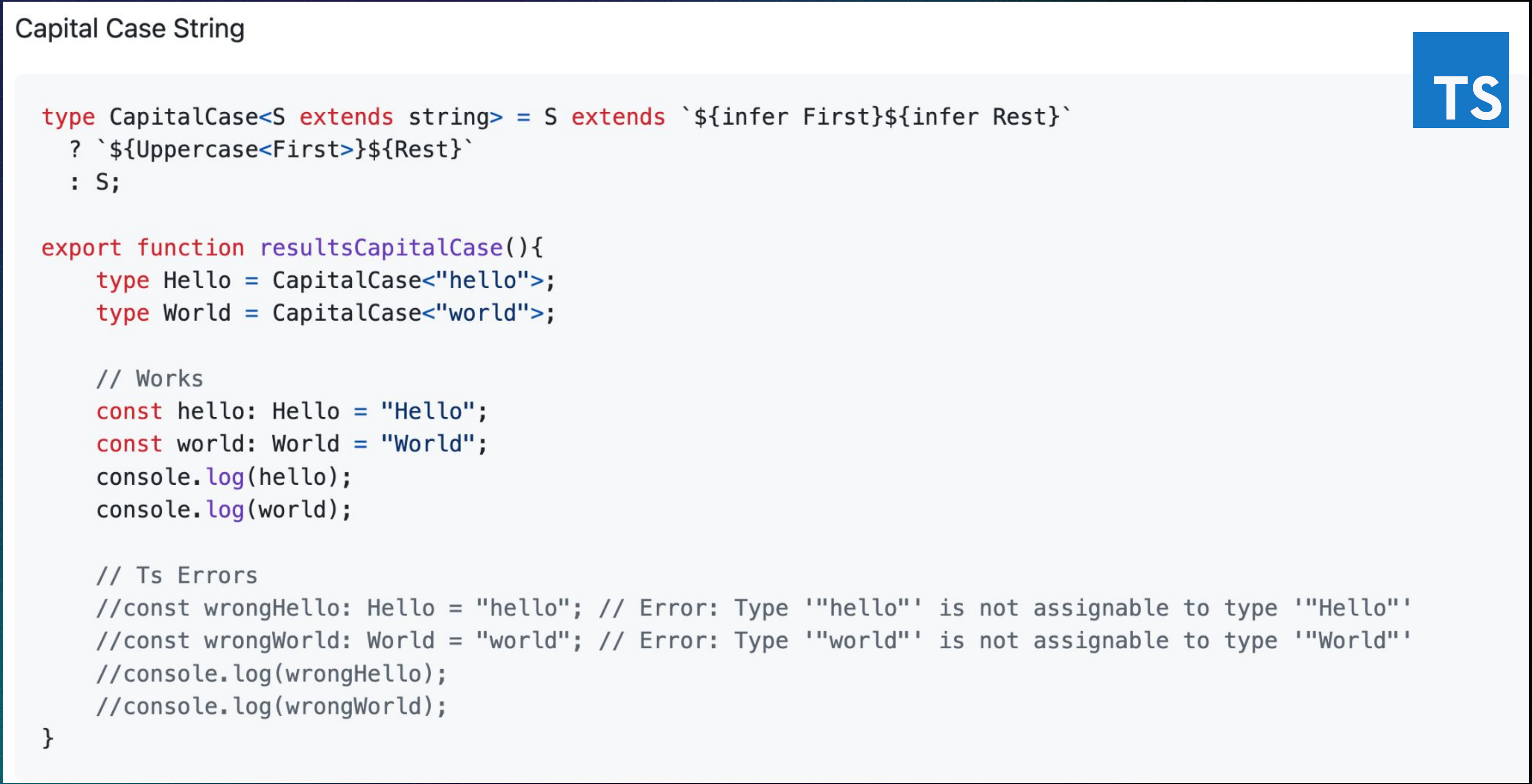
Abstraction Power

Safety / Correctness

It’s all about Shift Left.
Leverage the compiler.
Do Less tests.



```scala
sealed trait Nat {
  def toInt: Int
  override def toString: String = toInt.toString
}

case object Zero extends Nat {
  def toInt: Int = 0
}

case class Succ[N <: Nat](n: N) extends Nat {
  def toInt: Int = 1 + n.toInt
}

type One = Succ[Zero.type]
type Two = Succ[One]
type Three = Succ[Two]

type Add[A <: Nat, B <: Nat] <: Nat = A match
  case Zero.type => B
  case Succ[n] => Succ[Add[n, B]]

object Natural {
  def runNatural(): Unit = {
    val result: Add[Two, One] = Succ(Succ(Succ(Zero)))
    println(result) // 3
  }
}
```

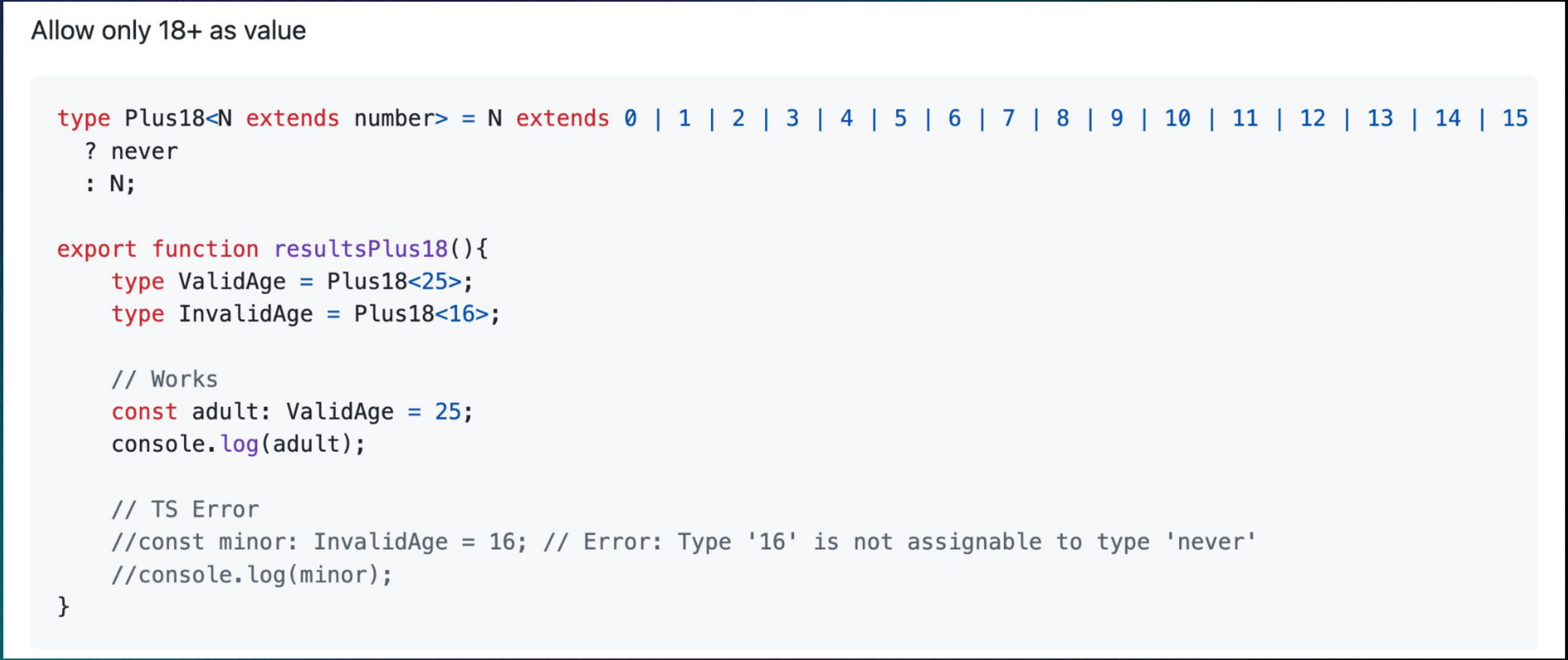


Capital Case String

TS

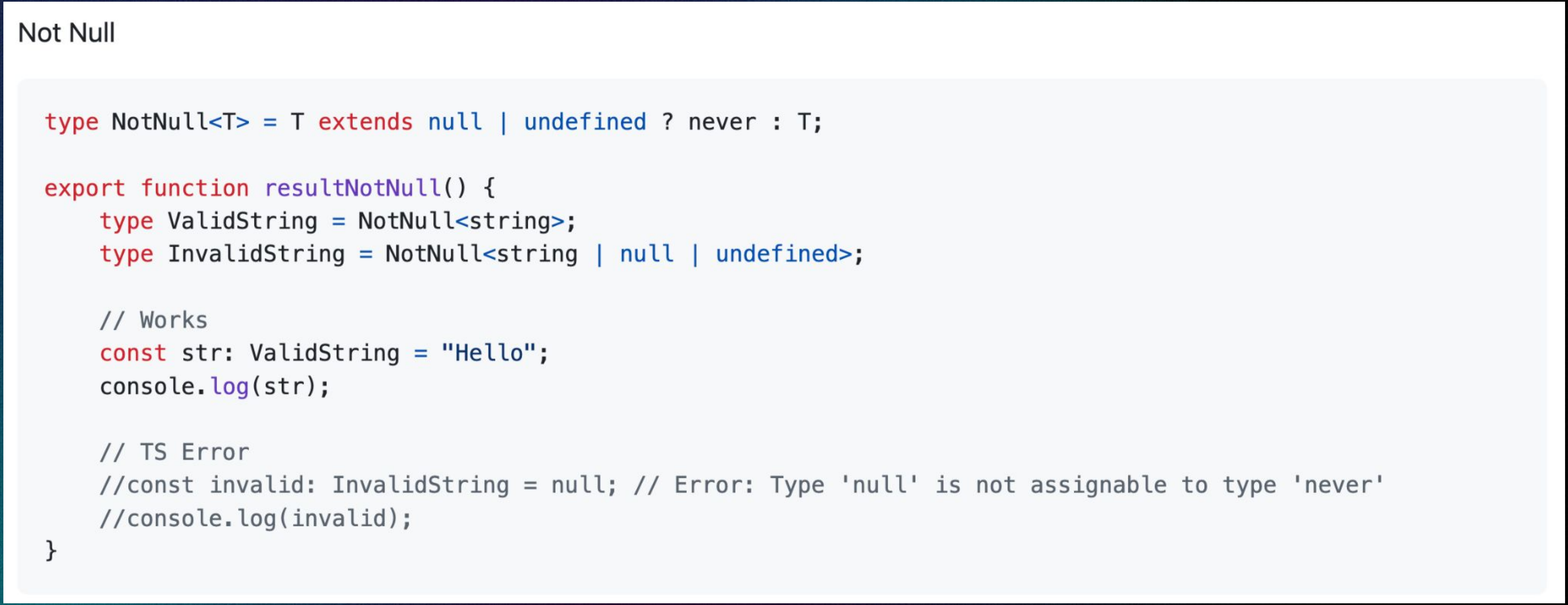
```typescript
type CapitalCase<S extends string> = S extends `${infer First}${infer Rest}`
  ? `${Uppercase<First>}${Rest}`
  : S;

export function resultsCapitalCase(){
    type Hello = CapitalCase<"hello">;
    type World = CapitalCase<"world">;

    // Works
    const hello: Hello = "Hello";
    const world: World = "World";
    console.log(hello);
    console.log(world);

    // Ts Errors
    //const wrongHello: Hello = "hello"; // Error: Type '"hello"' is not assignable to type '"Hello"'
    //const wrongWorld: World = "world"; // Error: Type '"world"' is not assignable to type '"World"'
    //console.log(wrongHello);
    //console.log(wrongWorld);
}
```

TS



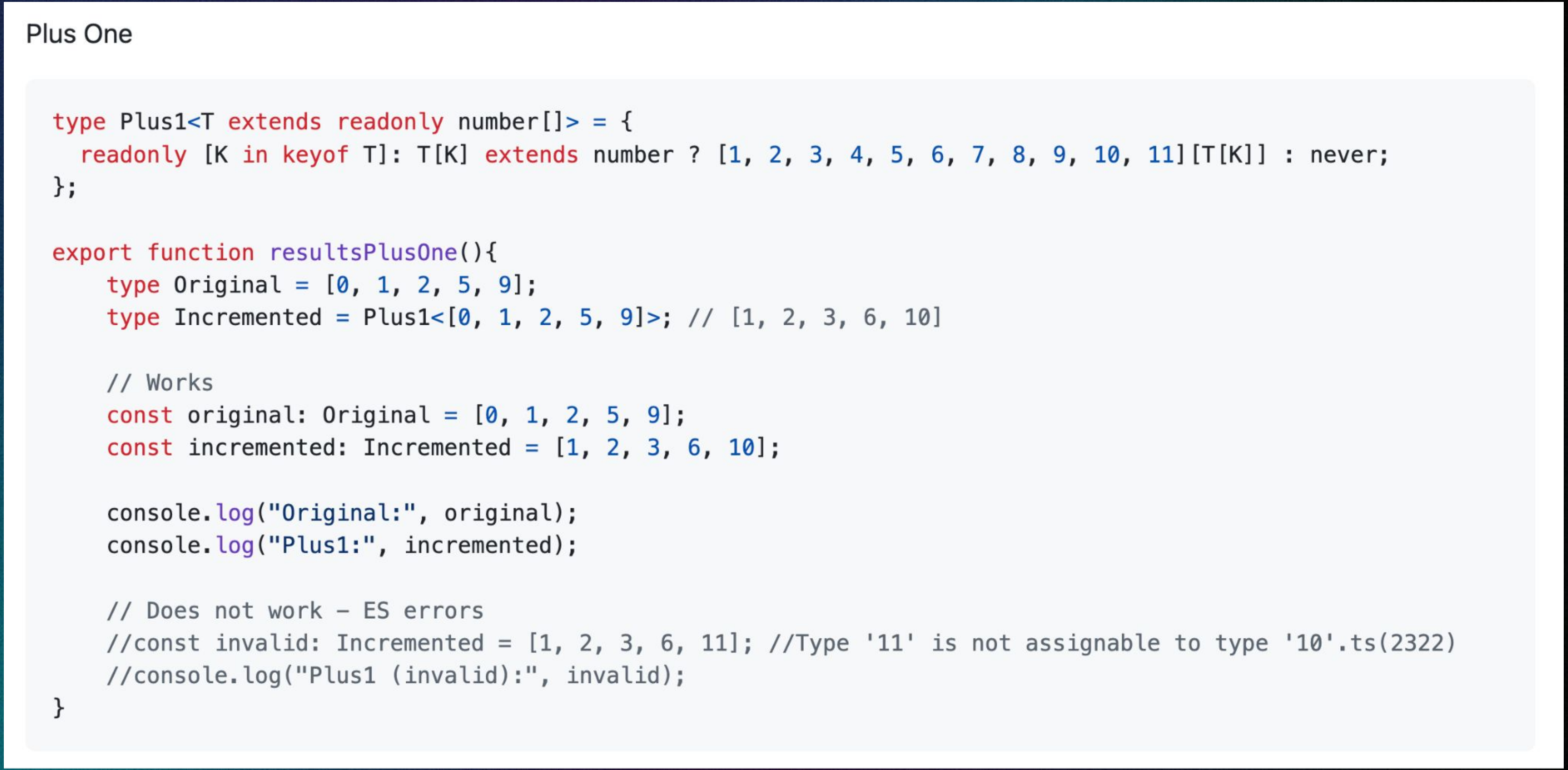
Allow only 18+ as value

```
type Plus18<N extends number> = N extends 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15
  ? never
  : N;

export function resultsPlus18(){
    type ValidAge = Plus18<25>;
    type InvalidAge = Plus18<16>;

    // Works
    const adult: ValidAge = 25;
    console.log(adult);

    // TS Error
    //const minor: InvalidAge = 16; // Error: Type '16' is not assignable to type 'never'
    //console.log(minor);
}
```



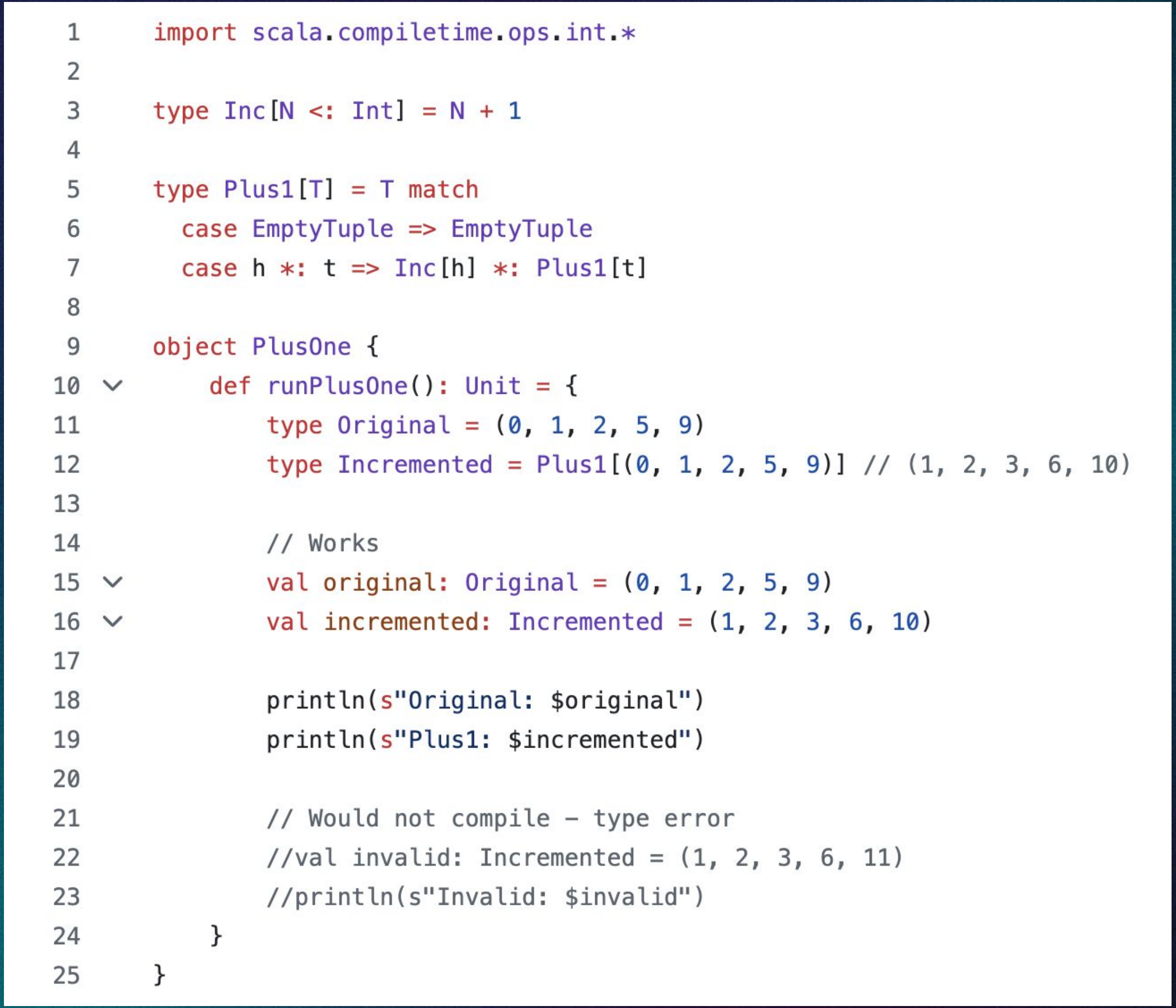
Not Null

```
type NotNull<T> = T extends null | undefined ? never : T;

export function resultNotNull() {
    type ValidString = NotNull<string>;
    type InvalidString = NotNull<string | null | undefined>;

    // Works
    const str: ValidString = "Hello";
    console.log(str);

    // TS Error
    //const invalid: InvalidString = null; // Error: Type 'null' is not assignable to type 'never'
    //console.log(invalid);
}
```

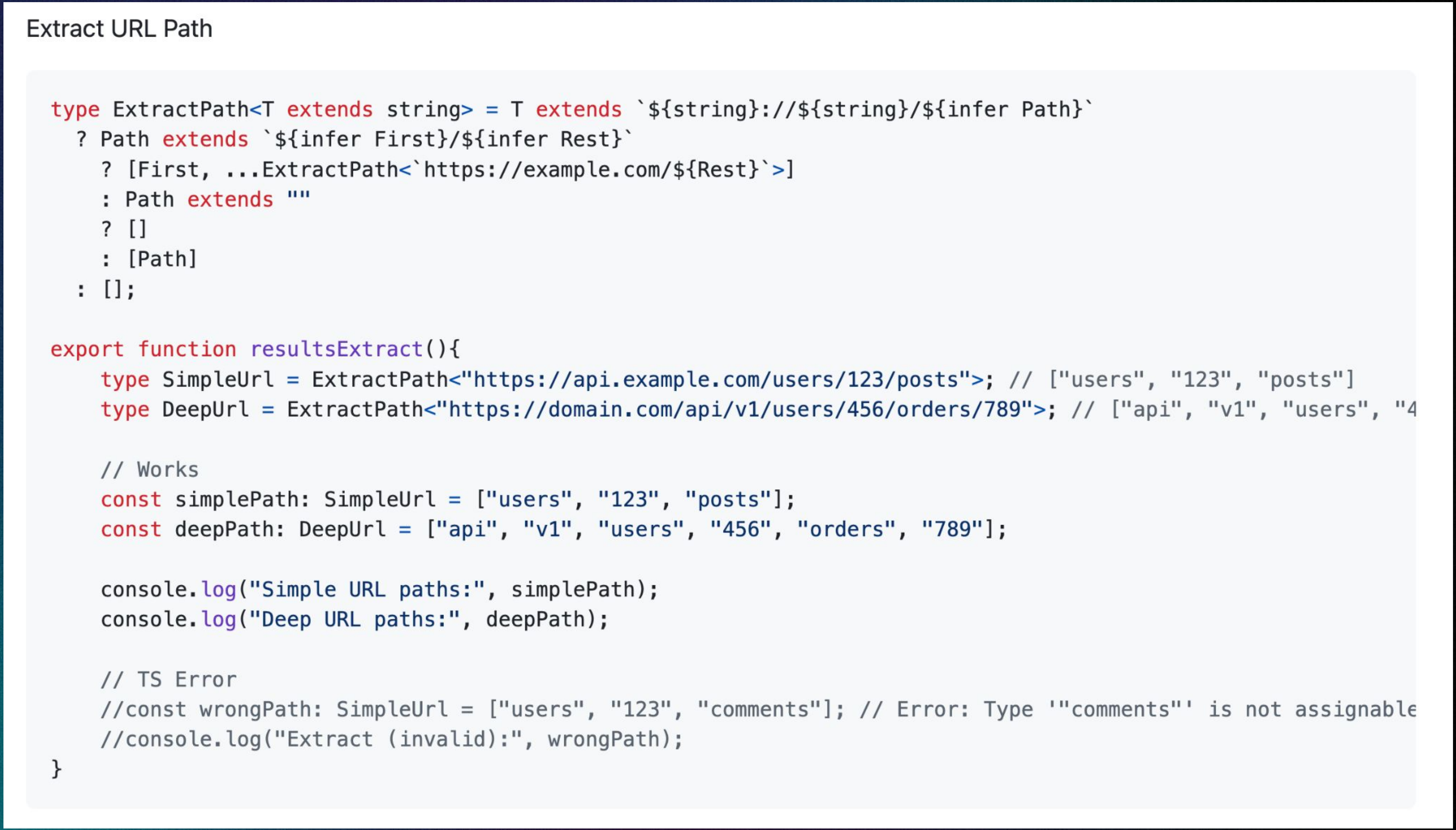


Plus One

```typescript
type Plus1<T extends readonly number[]> = {
  readonly [K in keyof T]: T[K] extends number ? [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11][T[K]] : never;
};

export function resultsPlusOne(){
    type Original = [0, 1, 2, 5, 9];
    type Incremented = Plus1<[0, 1, 2, 5, 9]>; // [1, 2, 3, 6, 10]

    // Works
    const original: Original = [0, 1, 2, 5, 9];
    const incremented: Incremented = [1, 2, 3, 6, 10];

    console.log("Original:", original);
    console.log("Plus1:", incremented);

    // Does not work - ES errors
    //const invalid: Incremented = [1, 2, 3, 6, 11]; //Type '11' is not assignable to type '10'.ts(2322)
    //console.log("Plus1 (invalid):", invalid);
}
```



```scala
import scala.compiletime.ops.int.*

type Inc[N <: Int] = N + 1

type Plus1[T] = T match
  case EmptyTuple => EmptyTuple
  case h *: t => Inc[h] *: Plus1[t]

object PlusOne {
    def runPlusOne(): Unit = {
        type Original = (0, 1, 2, 5, 9)
        type Incremented = Plus1[(0, 1, 2, 5, 9)] // (1, 2, 3, 6, 10)

        // Works
        val original: Original = (0, 1, 2, 5, 9)
        val incremented: Incremented = (1, 2, 3, 6, 10)

        println(s"Original: $original")
        println(s"Plus1: $incremented")

        // Would not compile - type error
        //val invalid: Incremented = (1, 2, 3, 6, 11)
        //println(s"Invalid: $invalid")
    }
}
```



Extract URL Path

```
type ExtractPath<T extends string> = T extends `${string}://${string}/${infer Path}`
  ? Path extends `${infer First}/${infer Rest}`
    ? [First, ...ExtractPath<`https://example.com/${Rest}`>]
    : Path extends ""
    ? []
    : [Path]
  : [];

export function resultsExtract(){
    type SimpleUrl = ExtractPath<"https://api.example.com/users/123/posts">; // ["users", "123", "posts"]
    type DeepUrl = ExtractPath<"https://domain.com/api/v1/users/456/orders/789">; // ["api", "v1", "users", "4

    // Works
    const simplePath: SimpleUrl = ["users", "123", "posts"];
    const deepPath: DeepUrl = ["api", "v1", "users", "456", "orders", "789"];

    console.log("Simple URL paths:", simplePath);
    console.log("Deep URL paths:", deepPath);

    // TS Error
    //const wrongPath: SimpleUrl = ["users", "123", "comments"]; // Error: Type '"comments"' is not assignable
    //console.log("Extract (invalid):", wrongPath);
}
```

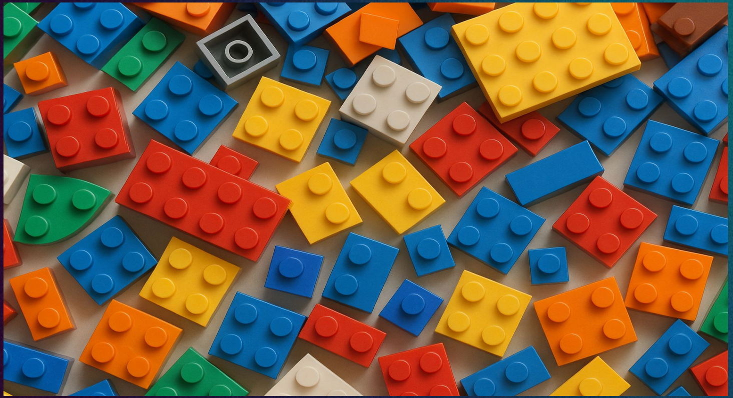
Benefits

- Correctness
- More Powerful abstractions
- Compile type-safety
- Shift Left
- Less tests
- Better maintainability

TypeScript Gist with code
https://gist.github.com/diegopacheco/f56724d3026d3fbfc4f950c01bc02685

Scala 3.x project with code
https://github.com/diegopacheco/scala-playground/tree/master/scala-3-7-3-type-level

Type Level Programing



The next frontier of advanced abstractions.

Diego Pacheco