# TYPED KNIGHT

## MASTER TYPESCRIPT IN THE SHADOWS


Discover essential TypeScript tips that will greatly improve your code.

DIEGO FLORIANO

# Disclaimer

his material is an educational parody inspired by superhero themes. It is not affiliated with or endorsed by DC Comics or Warner Bros. All character names and trademarks mentioned are property of their respective owners.

# INTRODUCTION

## Essential Tips to Become a TypeScript Knight

TypeScript is like the Batman of front-end development: silent, powerful, and always one step ahead of the bugs. In this eBook, we'll explore how this static typing hero can save your application in the dark.


# 1

# BATMAN'S MASK

# UNDERSTANDING TYPES

# BASIC TYPES

The secret identity of a variable

Just like Bruce Wayne hides behind the Batman mask, variables in TypeScript have their own "faces": types. This keeps your code clear and less prone to surprises.

batman.ts

```ts
let name: string = "Bruce Wayne"
let age: number = 35
let isHero: boolean = true
```

Types make your code predictable and safe. No Joker-style chaos.

# TYPE INFERENCE

## The Dark Knight detective

Even without being explicit, TypeScript figures out the type – just like a good detective.

batman.ts

```ts
let city = "Gotham" // Inferred as string
city = 123 // ❌ Error! city is a string
```

Trust TypeScript’s brain, but know when to be clear.

2

# GOTHAM'S ALLIES
# FUNCTIONS AND TYPES

# TYPED FUNCTIONS

Alfred always knows what to expect

Typing parameters and return values gives your code the order and safety of Alfred in the Batcave.

batman.ts

```ts
function greet(name: string): string {
  return `Good evening, ${name}.`
}
```

Well-typed functions mean fewer emergency calls.

# OPTIONAL AND DEFAULT PARAMETERS

Sometimes Robin shows up,
sometimes he doesn’t

Sometimes Robin’s off on another mission, and your function needs to handle it.

batman.ts

```ts
function patrol(city: string = "Gotham",
                partner?: string) {
  console.log(`Patrolling ${city}
                with ${partner ?? "alone"}`)
}
```

Optional parameters prevent crashes when a sidekick bails.

3

# BATMAN'S ARSENAL
# OBJECTS AND INTERFACES

# INTERFACES

The Batcave blueprint

Interfaces define what an object should look like –
just like Batman's gadgets follow a detailed design.

batman.ts

```ts
interface Villain {name: string
  dangerous: boolean
}

const joker: Villain = {name: "Joker",
  dangerous: true
}
```

Interfaces help you organize your heroes and villains.

# TYPE VS INTERFACE

Different gadgets, same goal

type and interface look alike – like a Batarang and a Batclaw – but they have subtle differences.

batman.ts

```ts
type Hero = {name: string}

interface AntiHero {name: string}
```

Use “interface” for objects, “type” for type unions and compositions.

# 4

# THE JUSTICE LEAGUE

# ADVANCED TYPES

# UNION TYPES

A team of different heroes

A variable that can hold different types – like Batman teaming up with Flash or Aquaman.

batman.ts

```ts
function showIdentity(
    id: string | number) {
        console.log(`ID: ${id}`)
}
```

Union types allow flexibility without losing control.

# TYPE GUARDS

Spotting threats in disguise

Like Batman exposing an impostor in the League, type guards help confirm the right type.

batman.ts

```ts
function checkHero(hero: string | number) {
  if (typeof hero === "string") {
    console.log("Named hero:", hero)
  } else {
    console.log("Coded hero:", hero)
  }
}
```

Always investigate the type before acting.

# 5

# SHADOWS AND SECRETS

## Generics

# GENERICS

## A plan that adapts

Batman always has a plan — no matter the enemy.
Generics let your functions do the same.

batman.ts

```ts
function protect<T>(item: T): T {
  return item
}

const secret = protect<string>("Secret Identity")
```

Use generics for reusable and safe code.

# GENERICS WITH CONSTRAINTS

Rules even for chaos

Even with freedom, there must be limits. Bruce knows this.

batman.ts

```ts
function log<T extends { name: string }>(target: T) {
  console.log(`Tracking: ${target.name}`)
}
```

Constrained generics prevent Riddler-style surprises.

# 6

# THE KNIGHT AND CLEAN CODE

# READONLY AND CONSTANTS

The Bat-rules never change

Some things should never change – like Batman’s moral code.

batman.ts

```ts
interface Gadget {
  readonly name: string
}

const batTool = { name: "Batarang" }
// batTool.name = "Batmobile" // ❌ Error
```

Use “readonly” to protect what must not change.

# NEVER AND VOID

When there’s no return

Some functions never return — like when Batman vanishes in the night.

batman.ts

```ts
function disappear(): never {
  throw new Error("Batman vanished!")
}

function patrolGotham(): void {
  console.log("Gotham is safe.")
}
```

Use “never” for functions that break the flow, “void” for those that simply do a task.

7

# THE WATCHTOWER

# Enums, Decorators and Testing

# ENUMS

Naming the enemies in the shadows

Enums let you define a fixed set of named values – like Batman keeping a classified list of Gotham's threats.

batman.ts

```ts
enum VillainType {
  Mastermind,
  Thug,
  Trickster
}

function identifyThreat(villain: VillainType) {
  if (villain === VillainType.Mastermind) {
    console.log("High-level threat detected.")
  }
}
```

Enums give your values names and context – no more magic numbers in the dark.

# DECORATORS

## Custom tech straight from Lucius Fox

Decorators are like gadgets — they extend classes and methods with extra functionality, without changing the core. (Note: decorators are still an experimental feature in TypeScript.)

batman.ts

```ts
function logAction(target: any, name: string, descriptor: PropertyDescriptor) {
  const original = descriptor.value
  descriptor.value = function (...args: any[]) {
    console.log(`Calling ${name} with`, args)
    return original.apply(this, args)
  }
}

class BatComputer {
  @logAction
  scanArea(area: string) {
    console.log(`Scanning ${area}...`)
  }
}
```

Decorators enhance your classes like Lucius upgrading the Batmobile.

# TESTING WITH TYPESCRIPT

Even the Bat tests his gear

Batman doesn’t enter the field without testing his tools. Same goes for your code.Using TypeScript with a test framework like Jest gives type-safe tests that prevent mistakes early.

batman.ts

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

// Jest test
test('add works correctly', () => {
  expect(add(2, 3)).toBe(5)
})
```

Typed tests ensure your app survives every mission.

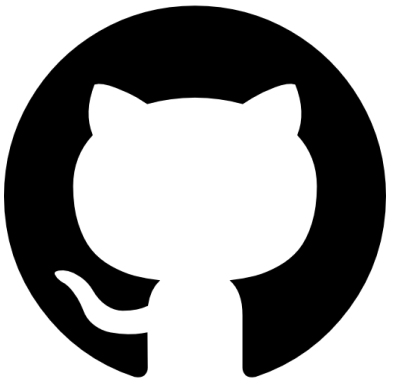
# ACKNOWLEDGMENTS

# THANK YOU FOR READING THIS FAR.

This eBook was generated with the help of AI and formatted by a human. The step-by-step process is available on my GitHub.

This material was created with the goal of sharing knowledge in a clear, accessible, and practical way for those who want to grow with TypeScript.

I hope this content has been useful to you.



https://github.com/diegofloriano/prompts-recipe-to-create-a-ebook.git

Author: Diego Floriano