Typescript for Javascript Programmers

Typescript

CHEATSHEET FOR

TypeScript

Typescript for Javascript programmers

https://devhints.io/typescript

Essential Features

Basic types

```
any
void

boolean
number
string

null
undefined

string[]          /* or Array<string> */
[string, number]  /* tuple */

string | null | undefined   /* union */

never  /* unreachable */
```

```
enum Color {Red, Green, Blue = 4}
let c: Color = Color.Green
```

Declarations

```
let isDone: boolean
let isDone: boolean = false
```

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

// Return type is optional
function add (a: number, b: number) { ... }
```

Classes

```
class Point {
  x: number
  y: number
  static instances = 0
  constructor(x: number, y: number) {
    this.x = x
    this.y = y
  }
}
```

Interfaces

Explicit

```
interface LabelOptions {
  label: string
}

function printLabel(options: LabelOptions) { ... }
```

Inline

```
function printLabel (options: { label: string }) {
  console.log(options.label)
}

// Note the semicolon
function getUser (): { name: string; age?: number } {
}
```

Modules

```
export interface User { ... }
```

Optional Features

Type assertions

Variables

```
let len: number = (input as string).length
let len: number = (<string> input).length  /* not allowed in JSX */
```

Functions

```
function object(this: {a: number, b: number}, a: number, b: number) {
  this.a = a;
  this.b = b;
  return this;
}

// this is used only for type declaration
let a = object(1,2);
// a has type {a: number, b: number}
```

Interfaces

Optional properties

```
interface User {
  name: string,
  age?: number
}
```

Read only

```
interface User {
  readonly name: string
}
```

Dynamic keys

```
{
  [key: string]: Object[]
}
```

Type aliases

```
type Name = string | string[]
```

Function types

```
interface User { ... }

function getUser(callback: (user: User) => any) { callback({...}) }

getUser(function (user: User) { ... })
```

Classes

Fields which do not require initialisation

```
class Point {
  public someUselessValue!: number;
  ...
}
```

Inheritance

```
class Point {...}

class Point3D extends Point {...}

interface Colored {...}

class Pixel extends Point implements Colored {...}
```

Short fields initialisation

```
class Point {
  static instances = 0;
  constructor(
    public x: number,
    public y: number,
  ){}
}
```

Generics

```
class Greeter<T> {
  greeting: T
  constructor(message: T) {
    this.greeting = message
  }
}

let greeter = new Greeter<string>('Hello, world')
```

Type extraction

```
interface Building {
  room: {
    door: string,
    walls: string[],
  };
}

type Walls = Building['room']['walls']; // string[]
```