This is <u>Devhints.io cheatsheets</u> — a collection of cheatsheets I've written.
CHEATSHEET FOR
TypeScript
f ¥
TypeScript is just like ES2015 with type-checking. All ES2015 (classes, etc) should work.
Basic types
any void
boolean number string
<pre>null undefined string[] /* or Array<string> */ [string, number] /* tuple */</string></pre>
string null undefined /* union */ never /* unreachable */
<pre>enum Color {Red, Green, Blue = 4} let c: Color = Color.Green</pre>
Declarations
<pre>let isDone: boolean let isDone: boolean = false</pre>
<pre>function add (a: number, b: number): number { return a + b</pre>
<pre>// Return type is optional function add (a: number, b: number) { }</pre>
Type assertions
Variables
<pre>let len: number = (input as string).length let len: number = (<string> input).length /* not allowed</string></pre>
<pre>functions function object(this: {a: number, b: number}, a: number, b</pre>
<pre>this.a = a; this.b = b; return this; }</pre>
<pre>// this is used only for type declaration let a = object(1,2); // a has type {a: number, b: number}</pre>
Interfaces
<pre>function printLabel (options: { label: string }) { console.log(options.label)</pre>
<pre>} // Note the semicolon function getUser (): { name: string; age?: number } {</pre>
Explicit
<pre>interface LabelOptions { label: string }</pre>
<pre>function printLabel(options: LabelOptions) { }</pre>
<pre>Optional properties interface User { name: string,</pre>
age?: number }
<pre>Read only interface User { readonly name: string</pre>
Dynamic keys
{
<pre>[key: string]: Object[] }</pre>
}
Type aliases
<pre>Type aliases type Name = string string[] Function types interface User { }</pre>
<pre>Type aliases type Name = string string[] Function types</pre>
<pre>Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback</pre>
<pre> Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number } </pre>
<pre>Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } }</pre>
<pre> Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } </pre>
<pre>Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {}</pre>
<pre>Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance</pre>
<pre>Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} class Point3D extends Point {} interface Colored {}</pre>
<pre>type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number y: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} class Point3D extends Point {} interface Colored {} class Pixel extends Point implements Colored {}</pre>
<pre>type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} class Point and extends Point {} interface Colored {} class Pixel extends Point implements Colored {} Short fields initialisation class Point { static instances = 0; }</pre>
<pre>Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} class Point {} class Pixel extends Point implements Colored {} Short fields initialisation class Point { static instances = 0; constructor(public x: number, public y: number,)(}</pre>
<pre>type Aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} class Point applied initialisation class Point { static instances = 0; constructor(public x: number, public y: number,){} }</pre>
Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} class Point3D extends Point {} interface Colored {} class Pixel extends Point implements Colored {} Short fields initialisation class Point { static instances = 0; constructor : number, public y: number, } }} Fields which do not require initialisation class Point { public y: number; } }
Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point {
Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }}) Classes class Point { x: number y: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} class Pointable extends Point {} interface Colored {} Short fields initialisation class Pixel extends Point implements Colored {} Short fields initialisation class Point { static instances = 0; constructor(public x: number, public y: number, } }} Fields which do not require initialisation Class Point { public someUselessValue!: number; Generics
<pre>type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number, y: number this.x = x this.y = y } } Inheritance class Point {} class Pointal 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, public y: number, } } Fields which do not require initialisation class Point { public x: number, public x: number, public y: number, } } Fields which do not require initialisation class Point { public x: number, public x: number, public y: number; } Generics class Greeter<<>> { greeting: T constructor(message: T) { this.greeting = message } } </pre>
Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) *> any) { callback getUser(function (user: User) { }) Classes class Point { x: number y: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} class Point {} class Point a static instances = 0; constructor(public x: number, public y: number, public y: number, public someUselessValuel: number; Generics class Greeter <f> { greeting: T constructor(message: T) { this.greeting = message } }</f>
type Name = string string[] Function types interface User { } function getUser(callback: (user: User) -> any) { callback getUser(function (user: User) { }) Classes class Point {
Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) => any) { callback getUser(function (user: User) { }}) Classes class Point {
Type aliases type Name = string string[] Function types interface User { } function getUser(cattback: (user: User) -> any) { cattback getUser(function (user: User) { })) Classes class Point {
Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) -> any) { callback getUser(function (user: User) { }}) Class Point { x: number y: number static instances = 0 constructor(x: number, y: number) { this.x = x this.y = y } } Inheritance class Point {} fistatic instances = 0; constructor(pebtic x: number, pebtic y: number,); } Fields which do not require initialisation class Point { public y: number, }; } fill Generics class GreeterCo { greeting: romeUselessValuel: number; } Generics class GreeterCo { greeting: romeUselessValuel: number; } Modules export interface User { } Type extraction interface fullding { cont: string { } cont: string { } Type extraction
Type aliases type Name = string string[] Function types interface User (} function getUser(allback: (user: User) => any) { callback getUser(function (user: User) + }) Classes class Point {
Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) -> any) { callback getUser(function (user: User) { })} Classes Class Point {
Type aliases type Name = string string[] Function types interface User { } function getUser(callback: (user: User) -> any) { callback getUser(function (user: User) { })} Classes Class Point {
Type aliases type Name - string string[] Function types interface User { } function getUser(callbacks (user: User) => any) { callback getUser(function (user: User) { }}) Classes class Point { xx number y; number y; number y; number y; number y; number this.x = x this.y = y } } Inheritance class Point { static instances = 0; constructor(string) sublic x; number, yol) } Fields which do not require initialisation class Point { public x; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisation class Point { public y; number, yol) } Fields which do not require initialisati