

Typescript // Cheatsheet

Installing typescript

You can install **typescript** in two different ways.

- Via Npm
- or through Visual studio plugin

Installing Typescript through NPM and running a typescript file

1. Download and install node.js
2. open cmd/terminal
3. enter the following

```
> npm install -g typescript
```

4. create a new file with '.ts' extension
5. open the terminal/cmd and type

```
> tsc <nameofthefile>.ts
```

Typescript

What is typescript?

Typescript is a syntatically typed javascript.

Typescript basics

Data Types

There are 3 data types

- number
- string
- boolean

```
var num: number = 25;
var aName: string = "renz";
var isAlive: boolean = true;
console.log(num + ' ' + name + ' ' + isAlive); // 25 renz
true
```

You can declare a variable with **any** type of data in it.

```
var someData1: any = 25;
var someData2: any = true;
var someData3: any = "bazingga";

console.log(someData1 + ' ' + someData1 + ' ' + someData1
); // 25 true bazingga
```

Constant variables

You can declare a constant variable in typescript. constant variables are unchangeable values.

```
const num1 = 25;  
console.log(num1) // 25  
num1 = 30; // false
```

Functions

You can declare a function that returns nothing (void)

```
function theresNothingHere(): void {  
    console.log("Hodor");  
}  
console.log(theresNothingHere()); // Hodor
```

You can also declare functions that returns a certain type of data

```
function returnsNumbers(num1: number, num2: number): number {  
    return num1 + num2;  
}
```

```
function returnFullName(fname: string, lname: string): string {
    return fname + ' ' + lname;
}

console.log(returnsNumbers(25 , 25)); // 50
console.log(returnFullName('Renz', 'Pulvira')); // Renz Pulvira
```

In function arguments. you can put **Optional types** in the function arguments. it means that you can put a value on the arguments or *not*.

```
function getAgeOrHeight(age?: number, height?: number) : number {
    if(height == NaN){
        return age;
    } else if(age == NaN) {
        return height;
    } else {
        return 404;
    }
}

console.log("Age: " + getAgeOrHeight()); // age: 404
```

Interface

Interface are a special type that you can edit/create yourself for other values. for example you can create a type that has the same properties of a car.

```
/*  
 *  
 *   When you wan't to create a complex data type  
 *   Using Interface is a good way to create a variable  
 *   with a property inside of it.  
 *  
 */  
  
// This variable has certain properties  
interface dog {  
    sound: string;  
    theName: string;  
    age: number;  
}  
  
// Defining the values of properties with the 'dog' inter  
face  
let dogProp:dog = {  
    sound: "Bark!",  
    theName: "Marco",
```

```
    age: 4
};

// outputting the declared variables
console.log(dogProp.sound);
console.log(dogProp.theName);
console.log(dogProp.age);
```

You can also use **optional types** when declaring the interface properties.

```
interface carProp {
    unit: string,
    color: string,
    wheels?: number,
    volume():? void;
}

let myCar: carProp = {
    unit: 'BMW',
    color: 'red'
    volume: () => {
        console.log("VRUUUM");
    }
}

console.log(myCar.unit); // BMW
```

Classes

Creating a class in typescript is a little similar to javascript when declaring classes. classes behaves like a blueprint.

```
class Dog {  
  name: string;  
  constructor(theName: string){  
    this.name = theName;  
  }  
  
  sayName(){  
    return this.name;  
  }  
}  
  
let Animal = new Dog("Marco");  
console.log("This dog's name is " + Animal.sayName()); //  
  This dog's name is Marco
```

Arrow functions

Arrow functions anonymous functions that is more concise in syntax.

they are sometimes called 'fat arrow' because of the '=>' symbol in arrow functions.

arrow functions are more concise in writing function expressions.

```
// With arguments
let Monster = (itHas: string, itSounds: string) => {
    console.log("It has " + itHas);
    console.log("and it sounds " + itSounds);
};

// Without arguments
let Monster = () => { console.log('Monster unknown') };
```

Inheritance

Inheritance are an important part in OOP. you can **extend** some certain class to another class and use it's variables/functions.

```
class Dog {
    constructor(action){

    }
}

class Cat extends Dog{
    constructor(action, name){
        super(action);
    }
}
```



```
}  
}
```

Super class

Super class are a way of using the original variables on the extended class.

```
class Dog {  
    constructor(name, action){  
        this.name = name;  
        this.action = action;  
    }  
}  
  
class Cat extends Dog {  
    constructor(name, action){  
        this.action = super(action);  
    }  
}
```

Getters and Setters

Getters are very useful and important to OO Programming. the **get** method

usually used for returning data, and **set** method sets a value for

constant or
special variables.

```
class Animal {  
    private _weight: number;  
  
    get weight():number {  
        return weight;  
    }  
  
    set weight(weight) {  
        this._weight = weight;  
    }  
}
```

Generic functions
