Using _ (underscore) variable with arrow functions in ES6/Typescript

Asked 2 years, 7 months ago Active 1 year, 9 months ago Viewed 29k times

91

19

I came across this construct in an Angular example and I wonder why this is chosen:

```
_ => console.log('Not using any parameters');
```

I understand that the variable _ means don't care/not used but since it is the only variable is there any reason to prefer the use of _ over:

```
() => console.log('Not using any parameters');
```

Surely this can't be about one character less to type. The () syntax conveys the intent better in my opinion and is also more type specific because otherwise I think the first example should have looked like this:

```
(_: any) => console.log('Not using any parameters');
```

In case it matters, this was the context where it was used:

```
submit(query: string): void {
    this.router.navigate(['search'], { queryParams: { query: query } })
      .then(_ => this.search());
}
```

javascript typescript ecmascript-6 arrow-functions

edited Dec 11 '16 at 13:08
vaxquis
8,106 5 40 58

asked Dec 11 '16 at 10:28
Halt
882 1 11 20

5 stackoverflow.com/questions/18300654/… – yurzui Dec 11 '16 at 12:01

3 Personally, the _=> pattern reduces the number of brackets making it easier to read: doStuff().then(()=>action()) vs doStuff().then(_=>action()). – Damien Golding Jul 11 '18 at 2:24

4 Answers

72

The reason why this style can be used (and possibly why it was used here) is that `_` is one character shorter than `()`.

Optional parentheses fall into the same style issue as optional curly brackets. This is a matter of taste and code style for the most part, but verbosity is favoured here because of consistency.

While arrow functions allow a single parameter without parentheses, it is inconsistent with zero, single destructured, single rest and multiple parameters:

```
let zeroParamFn = () => { ... };
let oneParamFn = param1 => { ... };
let oneParamDestructuredArrFn = ([param1]) => { ... };
let oneParamDestructuredObjFn = ({ param1 }) => { ... };
let twoParamsFn = (param1, param2) => { ... };
let restParamsFn = (...params) => { ... };
```

Although `is declared but never used` error was fixed in TypeScript 2.0 for underscored parameters, `_` can also trigger `unused variable/parameter` warning from a linter or IDE. This is a considerable argument against doing this.

`_` can be conventionally used for ignored parameters (as the other answer already explained). While this may be considered acceptable, this habit may result in a conflict with `_` Underscore/Lodash namespace, also looks confusing when there are multiple ignored parameters. For this reason it is beneficial to have properly named underscored parameters (supported in TS 2.0), also saves time on figuring out function signature and why the parameters are marked as ignored (this defies the purpose of `_` parameter as a shortcut):

```
let fn = (param1, _unusedParam2, param3) => { ... };
```

For the reasons listed above, I would personally consider `_ => { ... }` code style a bad tone that should be avoided.

86.8k 25 140 261

1 It's one character shorter but it is the same amount of keypresses for most IDEs, since pressing the `(` usually comes with a `)`. I personally prefer using `p` for parameter, I also wonder if it has any performance issue – Mojimi Feb 11 at 13:17

No performance differences that could be taken into account. – Estus Flask Feb 11 at 13:27

54

> The `()` syntax conveys the intent better imho and is also more type specific

Not exactly. `()` says that the function does not expect any arguments, it doesn't declare any parameters. The function's `.length` is 0.

If you use `_`, it explicitly states that the function will be passed one argument, but that you don't care about it. The function's `.length` will be 1, which might matter in some frameworks.

So from a type perspective, it might be more accurate thing to do (especially when you don't type it with `any` but, say, `_: Event`). And as you said, it's one character less to type which is also easier to reach on some keyboards.

answered Dec 11 '16 at 12:00
Bergi
398k 66 628 950

6 My first thought was that _ makes it only obvious by convention that there are no arguments to consider when trying to understand the function. Using () makes it explicit, no need for scanning the code for a possible use of _ (which would violate the convention). But you opened my eyes to also consider the value of documenting that there is a value passed to the function, which would otherwise not always be obvious. – Halt Dec 11 '16 at 15:51

I just realized my code is full of unused `_` s arrow function variables, I wonder if there's any performance difference compared to using `()` – Mojimi Feb 11 at 13:15

21

I guess `_ =>` is just used over `() =>` because `_` is common in other languages where it is not allowed to just omit parameters like in JS.

`_` is popular in Go and it's also used in Dart to indicate a parameter is ignored and probably others I don't know about.

3 Python also follows this convention, I think. – Jaime RGP Dec 11 '16 at 23:22

7 This usage of _ was presumably borrowed from functional languages such as ML and Haskell, where it long predates the invention of Python (let alone Go, Dart, or TypeScript). – ruakh Dec 12 '16 at 6:47

1 also Ruby does that (po-ru.com/diary/rubys-magic-underscore) and F# too (and other languages influenced by ML family) – Mariusz Pawelski Dec 16 '16 at 10:44

10

It is possisble to distinguish between the two usages, and some frameworks use this to represent different types of callbacks. For example I think nodes express framework uses this to distinguish between types of middleware, for example error handlers use three arguments, while routing uses two.

Such differentiation can look like the example below:

```
const f1 = () => { } // A function taking no arguments
const f2 = _ => { }  // A function with one argument that doesn't use it

function h(ff) {
  if(ff.length==0) {
    console.log("No argument function - calling directly");
    ff()
  } else if (ff.length==1) {
    console.log("Single argument function - calling with 1");
    ff(1)
  }
}

h(f1)
h(f2)
```

edited Dec 12 '16 at 6:48
Günter Zöchbauer
358k 82 1135 1035

answered Dec 12 '16 at 6:35
Michael Anderson
48.1k 7 100 150

This is based off Bergi's answer, but I thought adding an example was a little more editing than I was happy to do to someone elses post. – Michael Anderson Dec 12 '16 at 6:37