

How to get names of enum entries?



I would like to iterate a TypeScript an `enum` type and get each enumerated symbol name, e.g.:

215



```
enum myEnum { entry1, entry2 }  
  
for (var entry in myEnum) {  
    // use entry's name here, e.g., "entry1"  
}
```



22

enums

typescript

edited Jun 6 at 21:56



Alexander Abakumov

5,456 5 50 76

asked Aug 7 '13 at 19:01



CalvinDale

2,438 4 18 27

21 Answers



181



The code you posted will work; it will print out all the members of the enum, including the values of the enum members. For example, the following code:

```
enum myEnum { bar, foo }  
  
for (var enumMember in myEnum) {  
    console.log("enum member: ", enumMember);  
}
```

Will print the following:

```
Enum member: 0  
Enum member: 1
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

If you instead want only the member names, and not the values, you could do something like this:

```
for (var enumMember in myEnum) {  
    var isValueProperty = parseInt(enumMember, 10) >= 0  
    if (isValueProperty) {  
        console.log("enum member: ", myEnum[enumMember]);  
    }  
}
```

That will print out just the names:

Enum member: bar

Enum member: foo

Caveat: this slightly relies on an implementation detail: TypeScript compiles enums to a JS object with the enum values being members of the object. If TS decided to implement them different in the future, the above technique could break.

answered Aug 7 '13 at 19:30



Judah Gabriel Himango

43.4k 32 145 200

14 To be clear, the above answer still works as of TS 2.3. However, if you use "const enum", rather than just "enum", only then will it not work. Using const enum is basically telling TS to do a search-and-replace; every place you use MyEnum.Foo, it will be replaced with a corresponding numeric value. –

Judah Gabriel Himango May 11 '17 at 15:56



Though the answer is already provided, Almost no one pointed to the [docs](#)

209

Here's a snippet



```
enum Enum {  
    A  
}  
  
let nameOfA = Enum[Enum.A]; // "A"
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



3,177 2 14 21

22 It also says there "Keep in mind that string enum members do not get a reverse mapping generated at all." – [jbojcic](#) Oct 20 '17 at 11:30

How about displaying 0 or 1 from this enum ? `export enum Octave { ZERO = 0, ONE = 1 }` – [Stephane](#) Jun 22 at 20:31

Assuming you stick to the rules and only produce enums with numeric values, you can use this code. This correctly handles the case where you have a name that is coincidentally a valid number

51

```
enum Color {
  Red,
  Green,
  Blue,
  "10" // wat
}

var names: string[] = [];
for(var n in Color) {
  if(typeof Color[n] === 'number') names.push(n);
}
console.log(names); // ['Red', 'Green', 'Blue', '10']
```

answered Aug 7 '13 at 19:46

[Ryan Cavanaugh](#)

111k 30 180 186

Warning In modern typescript (tsc 2.5.2 atm) you are not even allowed to have a numeric string as a key to begin with. As such Himango's answer is better, since it covers all cases and has no downsides. – [srcspider](#) Nov 3 '17 at 9:04

For me an easier, practical and direct way to understand what is going on, is that the following enumeration:

40

```
enum colors { red, green, blue };
```

```
.....
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```
var colors = { red: 0, green: 1, blue: 2,
               [0]: "red", [1]: "green", [2]: "blue" }
```

Because of this, the following will be true:

```
colors.red === 0
colors[colors.red] === "red"
colors["red"] === 0
```

This creates a easy way to get the name of an enumerated as follows:

```
var color: colors = colors.red;
console.log("The color selected is " + colors[color]);
```

It also creates a nice way to convert a string to an enumerated value.

```
var colorName: string = "green";
var color: colors = colors.red;
if (colorName in colors) color = colors[colorName];
```

The two situations above are far more common situation, because usually you are far more interested in the name of a specific value and serializing values in a generic way.

answered Sep 11 '16 at 18:52



[Michael Erickson](#)

2,745 1 14 8

▲ If you only search for the names and iterate later use:

29

▼

```
Object.keys(myEnum).map(key => myEnum[key]).filter(value => typeof value === 'string')
as string[];
```

answered Mar 29 '17 at 11:02

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

9 Or with the ES2017 lib: `Object.values(myEnum).filter(value => typeof value === 'string') as string[]`; – [J.Money](#) Mar 6 '18 at 14:39

With current TypeScript Version 1.8.9 I use typed Enums:

21

```
export enum Option {  
    OPTION1 = <any>'this is option 1',  
    OPTION2 = <any>'this is option 2'  
}
```

with results in this Javascript object:

```
Option = {  
    "OPTION1": "this is option 1",  
    "OPTION2": "this is option 2",  
    "this is option 1": "OPTION1",  
    "this is option 2": "OPTION2"  
}
```

so I have to query through keys and values and only return values:

```
let optionNames: Array<any> = [];  
for (let enumValue in Option) {  
    let optionNameLength = optionNames.length;  
  
    if (optionNameLength === 0) {  
        this.optionNames.push([enumValue, Option[enumValue]]);  
    } else {  
        if (this.optionNames[optionNameLength - 1][1] !== enumValue) {  
            this.optionNames.push([enumValue, Option[enumValue]]);  
        }  
    }  
}
```

And I receive the option keys in an Array:

```
optionNames = [ "OPTION1", "OPTION2" ];
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



BenjaminJackman

799 1 6 18



Philip Miglinci

15.3k 2 23 29

▲ This solution work too.

12

▼

```
enum ScreenType {  
    Edit = 1,  
    New = 2,  
    View = 4  
}  
  
var type: ScreenType = ScreenType.Edit;  
  
console.log(ScreenType[type]); //Edit
```

answered Oct 9 '17 at 18:25



Carlinhos

324 4 16

▲ As of Typescript 2.4, enums can contain string initializers <https://www.typescriptlang.org/docs/handbook/release-notes/typescript-2-4.html>

12

▼ This allows you to write:

```
enum Order {  
    ONE = "First",  
    TWO = "Second"  
}  
  
console.log(`One is ${Order.ONE.toString()}`);
```

and get this output:

One is First

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



John Huebner

121 1 3

You can use the `enum-values` package I wrote when I had the same problem:

6

[Git: enum-values](#)

```
var names = EnumValues.getNames(myEnum);
```

answered Jul 23 '16 at 10:20



Slava Shpitalny

2,108 1 10 14

2 You aren't really answering the question, it would be better to document your answer with code/etc but I did find the package useful. – [lucuma](#) Aug 21 '17 at 16:03

Based on some answers above I came up with this type-safe function signature:

6

```
export function getStringValuesFromEnum<T>(myEnum: T): keyof T {  
    return Object.keys(myEnum).filter(k => typeof (myEnum as any)[k] === 'number') as any;  
}
```

Usage:

```
enum myEnum { entry1, entry2 };  
const stringVals = getStringValuesFromEnum(myEnum);
```

the type of `stringVals` is `'entry1' | 'entry2'`

[See it in action](#)

answered Apr 18 '18 at 8:13

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

1 The function should return `(keyof T)[]` instead of `keyof T`. Also, the `export` stops your playground from working. – Joald Aug 7 '18 at 8:12

Starting from typescript 2.4, the enum would not contain the key as a member anymore. [source from Typescript readme](#)

5

The caveat is that string-initialized enums can't be reverse-mapped to get the original enum member name. In other words, you can't write `Colors["RED"]` to get the string "Red".

My solution:

```
export const getColourKey = (value: string) => {
  let colourKey = '';
  for (const key in ColourEnum) {
    if (value === ColourEnum[key]) {
      colourKey = key;
      break;
    }
  }
  return colourKey;
};
```

edited Jan 25 '18 at 8:15



Pang

7,111 16 68 108

answered Jan 25 '18 at 7:55



kitko112

183 1 2 10

Let `ts-enum-util` ([github](#), [npm](#)) do the work for you and provide a lot of additional type-safe utilities. Works with both string and numeric enums, properly ignoring the numeric index reverse lookup entries for numeric enums:

5

String enum:

```
import {$enum} from "ts-enum-util";

enum Option {
  OPTION1 = 'this is option 1',
  OPTION2 = 'this is option 2'
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).


```
// value: ["OPTION1", "OPTION2"]
const keys= $enum(Option).getKeys();

// type: Option[]
// value: ["this is option 1", "this is option 2"]
const values = $enum(Option).getValues();
```

Numeric enum:

```
enum Option {
    OPTION1,
    OPTION2
}

// type: ("OPTION1" | "OPTION2")[]
// value: ["OPTION1", "OPTION2"]
const keys= $enum(Option).getKeys();

// type: Option[]
// value: [0, 1]
const values = $enum(Option).getValues();
```

answered Feb 24 '18 at 6:00



Jeff Lau

111 1 3

wow wow wow. I don't know how you managed to get exhaustiveness checking into your library, but I just tried it and I am so incredibly pleased. This library is so valuable I almost wish that it was in the TypeScript standard library! – [GreeneCreations](#) Jul 3 at 15:13

@GreeneCreations Thanks :). It only required a bit of generics wizardry. The source code is there in github with plenty of code comments for you to study. I'm playing around with some ideas to make type checking even more strict for a few methods, but still struggling a bit with deciding what the true purpose of the methods are and whether the additional strictness is actually good, or a possible annoyance. Think I just need to make a decision, go with it, and see if I get any complaints. – [Jeff Lau](#) Jul 5 at 14:51

maybe start an issue thread or something and people can give early feedback. You can cc me in there and aid love to give any input I can provide. My GitHub handle is dgreene1 – [GreeneCreations](#) Jul 6 at 15:08



Another interesting solution [founded here](#) is using ES6 Map:

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```
    mid,
    high
  }

  export const TypeLabel = new Map<number, string>([
    [Type.low, 'Low Season'],
    [Type.mid, 'Mid Season'],
    [Type.high, 'High Season']
  ]);
```

USE

```
console.log(TypeLabel.get(Type.low)); // Low Season
```

answered Mar 12 at 14:06



[manzapanza](#)

4,640 3 29 38

The only solution that works for me in all cases (*even if values are strings*) is the following :

```
var enumToString = function(enumType, enumValue) {
  for (var enumMember in enumType) {
    if (enumType[enumMember]===enumValue) return enumMember
  }
}
```

answered Jan 18 '18 at 15:44



[user2080105](#)

506 2 6 22

According to typescript documentation, we can do this via Enum with static functions.

Get Enum Name with static functions

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```
}  
  
namespace myEnum {  
    export function GetmyEnumName(m: myEnum) {  
        return myEnum[m];  
    }  
}
```

now we can call it like below
`myEnum.GetmyEnumName(myEnum.entry1);`
// result entry1

for reading more about Enum with static function follow the below link <https://basarat.gitbooks.io/typescript/docs/enums.html>

answered Sep 4 '18 at 13:39



[Shahid Ahmad](#)

524 5 14

The best way I think is to just declare the desired enum values. That way accessing them is clean and pretty (every time).

```
enum myEnum { entry1 = 'VALUE1', entry2 = 'VALUE2' }  
  
2 for (var entry in myEnum) {  
    console.log(entry);  
}
```

will produce:

VALUE1
VALUE2

answered Oct 9 '18 at 11:10



[stemadsen](#)

968 2 8 13

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

2

```
export class EnumUtils {  
  /**  
   * Returns the enum keys  
   * @param enumObj enum object  
   */  
  static getEnumKeys(enumObj: any, valueType: string): any[] {  
    return EnumUtils.getEnumValues(enumObj, valueType).map(value => enumObj[value]);  
  }  
  
  /**  
   * Returns the enum values  
   * @param enumObj enum object  
   */  
  static getEnumValues(enumObj: any, valueType: string): any[] {  
    return Object.keys(enumObj).filter(key => typeof enumObj[key] === valueType);  
  }  
}
```

How to use it:

```
enum TestEnum{  
  A= 0,  
  B= 1  
}  
  
EnumUtils.getEnumKeys(TestEnum, "number");  
EnumUtils.getEnumValues(TestEnum, "number");
```

Result for keys: ["A", "B"]

Result for values: [0, 1]

answered Feb 4 at 12:47




[Arnold Vakaria](#)

51 5

I find that solution more elegant:

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



```
console.log( val );  
}
```

It displays:


```
bar  
foo
```

answered Sep 7 '16 at 14:53




[Anthony Brenelière](#)

24k 12 29 44



My Enum is like this:

0



```
export enum UserSorting {  
    SortByFullName = "Sort by FullName",  
    SortByLastname = "Sort by Lastame",  
    SortByEmail = "Sort by Email",  
    SortByRoleName = "Sort by Role",  
    SortByCreatedAt = "Sort by Creation date",  
    SortByCreatedBy = "Sort by Author",  
    SortByUpdatedAt = "Sort by Edit date",  
    SortByUpdatedBy = "Sort by Editor",  
}
```

so doing this return *undefined*:

```
UserSorting[UserSorting.SortByUpdatedAt]
```

To resolve this issue, I choose another way to do it using a Pipe:

```
import { Pipe, PipeTransform } from '@angular/core';  
  
@Pipe({  
    name: 'enumKey'  
})
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

let enumValue = args[0];
var keys = Object.keys(value);
var values = Object.values(value);
for (var i = 0; i < keys.length; i++) {
    if (values[i] == enumValue) {
        return keys[i];
    }
}
return null;
}
}

```

And to use it:

```
return this.enumKeyPipe.transform(UserSorting, [UserSorting.SortByUpdatedAt]);
```

answered Dec 15 '18 at 18:21



Cedric Arnould

581 3 21



I found this question by searching "typescript iterate over enum keys". So I just want to post solution which works for me in my case. Maybe it'll help to someone too.

0



My case is the following: I want to iterate over each enum key, then filter some keys, then access some object which has keys as computed values from enum. So this is how I do it without having any TS error.

```

enum MyEnum = { ONE = 'ONE', TWO = 'TWO' }
const LABELS = {
    [MyEnum.ONE]: 'Label one',
    [MyEnum.TWO]: 'Label two'
}

```

```

// to declare type is important - otherwise TS complains on LABELS[type]
// also, if replace Object.values with Object.keys -
// - TS blames wrong types here: "string[] is not assignable to MyEnum[]"
const allKeys: Array<MyEnum> = Object.values(MyEnum)

```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
const allowedLabels = allowedKeys.map((type) => ({
  label: LABELS[type]
})))
```

answered Apr 17 at 16:21



Alendorff

1,539 6 30 53

It is not exactly answer of your question but it is a trick to tackle your problem.

-2

```
export module Gender {
  export enum Type {
    Female = 1,
    Male = 2
  };

  export const List = Object.freeze([
    { Female: Type.Female },
    { Male: Type.Male }
  ]);
}
```

You can extend your list model in a way you want.

```
export const List = Object.freeze([
  { Female: { key: 'Female', value: Type.Female } },
  { Male: { key: 'Male', value: Type.Male } }
]);
```

Now, you can use it in this way:

```
for(let gender of Gender.List){
  console.log(gender.Female.key);
  console.log(gender.Female.value);
}
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```
if(i === Gender.Type.Male){  
  console.log("I'm a man.");  
}
```

[edited May 11 at 14:26](#)

answered Jan 12 at 22:49

[Pedram Ahmadpour](#)**137** 2 3

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).