# Get int value from enum in C#

Asked 10 years, 3 months ago Active 6 days ago Viewed 1.4m times

1639

178

I have a class called `Questions` (plural). In this class there is an enum called `Question` (singular) which looks like this.

```
public enum Question
{
    Role = 2,
    ProjectFunding = 3,
    TotalEmployee = 4,
    NumberOfServers = 5,
    TopBusinessConcern = 6
}
```

In the `Questions` class, I have a `get(int foo)` function that returns a `Questions` object for that `foo` . Is there an easy way to get the integer value from the enum so I can do something like `Questions.Get(Question.Role)` ?

c# enums casting int

edited Feb 27 '18 at 15:05
Sae1962
735 9 24

asked Jun 3 '09 at 6:46
jim
10.1k 13 45 64

26 For the other way around: cast-int-to-enum-in-c-sharp. – nawfal Jun 9 '13 at 11:54

8 I know I'm late to the party, but instead of defining your method as `get(int foo)` you can define it as `get(Question foo)` then do your casting inside the method, the you can call your method as `Questions.Get(Question.Role)` – Joe Feb 7 '17 at 15:10

## 27 Answers

Just cast the enum, e.g.

The above will work for the vast majority of enums you see in the wild, as the default underlying type for an enum is `int`.

However, as cecilphillip points out, enums can have different underlying types. If an enum is declared as a `uint`, `long`, or `ulong`, it should be cast to the type of the enum; e.g. for

```
enum StarsInMilkyWay:long {Sun = 1, V645Centauri = 2 .. Wolf424B = 2147483649};
```

you should use

```
long something = (long)StarsInMilkyWay.Wolf424B;
```

edited Feb 27 '18 at 15:20

Sae1962
735 9 24

answered Jun 3 '09 at 6:49

Tetraneutron
24.6k 3 21 20

23 @Harry it isn't true. You can create Enumeration without casting, it is not required. and I only assign number in special cases, most of the time, I leave it as default value. but you can do `enum Test { Item = 1 }` and see that `1 == (int)Test.Item` is equal. – Jaider Jun 28 '12 at 20:47

33 @Jaider `(int)Test.Item` That is a cast! () is the explicit cast operator. – Sinthia V Jul 26 '12 at 19:02

46 @Sinthia V he said you can *create* it without casting, which is correct – Paul Ridgway Aug 17 '12 at 18:30

6 If the underlying type for `enum Question` was not `int` but `long` this cast will truncate `Role` s integral value! – quaylar Oct 29 '13 at 16:14

1 When you accept an `Enum` as a parameter, you know is only a fixed number of possible integral values you can get. On the other hand, if you take simply an `int`, then you have to validate if that `int` is within the accepted values., thus complicating the code. You can always override your signatures like ``` public void MyMethod(int x) { // do something with x } public void MyMethod(Enum x) { this.MyMethod((int) x); } ```` – percebus Aug 18 '15 at 16:52

283

Since Enums can be any integral type (`byte`, `int`, `short`, etc.), a more robust way to get the underlying integral value of the enum would be to make use of the `GetTypeCode` method in conjunction with the `Convert` class:

```
enum Sides {
    Left, Right, Top, Bottom
}
Sides side = Sides.Bottom;
```

This should work regardless of the underlying integral type.

edited Feb 27 '18 at 15:05

Sae1962

**735** 9 24

answered Jul 9 '10 at 14:54

cecilphillip

**9,250** 3 31 38

31 This technique proved its worth to me when dealing with a generic type where T:enum (actually T:struct, IConvertible but that's a different story). – aboy021 Jul 5 '11 at 23:20

1 How would you modify this to print out the hexadecimal value of side? This example shows the decimal value. The problem is that `var` is of type `object`, so you need to unbox it and it gets messier than I would like. – Mark Lakata Nov 9 '12 at 2:15

2 I think you should change the example to `object val = Convert...etc` the `var` in your example will always be `object`. – Mesh Oct 23 '13 at 8:20

2 @TimAbell All I can really say is that we found that dynamically compiled aspx pages (where you have to deploy the .cs files to the live server) were assigning the integers differently to each value. That meant that serialised objects one one machine, were deserialising with different values on a different machine and effectively getting corrupted (causing *hours* of confusion). We raised it with MS and I seem to recall they said that the autogenerated integers were not guaranteed to be the same when built across different framework versions. – NickG Mar 24 '15 at 9:42

4 @TimAbell On a separate occasion, a developer deleted an obsolete/unused Enum value causing all other values in the sequence to be out by one. As such, our coding standards now require that IDs are always specified explicitly, otherwise adding/deleting or even auto-formatting the code (e.g. sorting alphabetically) will change all the values causing data corruption. I would strongly advise anyone to specify all Enum integers explicitly. This is ultra-important if they correlate to externally (database) stored values. – NickG Mar 24 '15 at 9:46

183

Declare it as a static class having public constants:

```
public static class Question
{
    public const int Role = 2;
    public const int ProjectFunding = 3;
    public const int TotalEmployee = 4;
    public const int NumberOfServers = 5;
    public const int TopBusinessConcern = 6;
}
```

And then you can reference it as `Question.Role`, and it always evaluates to an `int` or whatever you define it as.


**735** 9 24


**1,969** 1 9 2

8 I'm surprised this hasn't got more votes - it's so obvious if you really want to use the int type natively. – CAD bloke May 15 '13 at 19:40

29 I'd use `static readonly int` because constants are compiled into their hard values. See stackoverflow.com/a/755693/492 – CAD bloke May 15 '13 at 23:16

88 This solution actually doesn't provide the real benefit of strongly typed enums. If I only wanted to pass a GameState-enum-parameter to a specific method for example, the compiler shouldn't allow me to pass any int-variable as a parameter. – thgc Apr 12 '14 at 18:24

9 @CADBloke which is precisely why you would use `const` and not `static readonly` because every time you compare `static readonly` you're making a method call to get the value of the variable whereas with a `const` you're comparing two value types directly. – blockloop Aug 14 '14 at 17:11

3 @brettof86 Yes, a const would be faster, if the compilation limitation will never be problem then it's all good. – CAD bloke Aug 15 '14 at 10:57

81

```
Question question = Question.Role;
int value = (int) question;
```

Will result in `value == 2` .

answered Jun 3 '09 at 6:48


jerryjvl
**15.8k** 6 34 55

33 The temporary variable question is unnecessary. – Gishu Jun 3 '09 at 6:51

1 So something like this Questions.Get(Convert.ToInt16(Question.Applications)) – jim Jun 3 '09 at 6:51

3 no need to convert it - just cast. – Michael Petrotta Jun 3 '09 at 6:52

5 You can simply cast in either direction; the only thing to watch is that enums don't enforce anything (the enum value could be 288, even though no Question exists with that number) – Marc Gravell ♦ Jun 3 '09 at 6:54

1 @Gishu You could say it's ... questionable. ;) – Felix D. Dec 16 '17 at 16:10

On a related note, if you want to get the int value from [illegible] than given [illegible] here:

72

You can use:

```
int i = Convert.ToInt32(e);
int i = (int)(object)e;
int i = (int)Enum.Parse(e.GetType(), e.ToString());
int i = (int)Enum.ToObject(e.GetType(), e);
```

The last two are plain ugly. I prefer the first one.

edited Aug 13 '15 at 0:12

ErikE
**35.1k** 14 120 166

answered Dec 1 '13 at 8:47

nawfal
**46.4k** 38 266 315

The second one is the fastest though. – Johan B Jun 16 at 11:47

39

It's easier than you think - an enum is already an int. It just needs to be reminded:

```
int y = (int)Question.Role;
Console.WriteLine(y); // prints 2
```

edited Mar 7 '18 at 13:52

jim
**10.1k** 13 45 64

answered Jun 3 '09 at 6:51

Michael Petrotta
**53.3k** 14 131 173

15 Nitpick: *this* enum is already an int. Other enums might be different types -- try "enum SmallEnum : byte { A, B, C }" – mqp Jun 3 '09 at 6:56

9 Absolutely true. C# reference: "Every enumeration type has an underlying type, which can be any integral type except char." – Michael Petrotta Jun 3 '09 at 6:59

Example:

```
    Raj = 1,
    Rahul,
    Priyanka
}
```

And in the code behind to get enum value:

```
int setempNo = (int)EmpNo.Raj; //This will give setempNo = 1
```

or

```
int setempNo = (int)EmpNo.Rahul; //This will give setempNo = 2
```

Enums will increment by 1, and you can set the start value. If you don't set the start value it will be assigned as 0 initially.

edited Dec 3 '18 at 12:56 — Callagher — 13 6

answered Sep 25 '09 at 10:43 — sooraj

6 Does this actually compile? – Peter Mortensen Jan 7 '16 at 20:04

Can something that is a Raj be also be a Rahul or a Priyanka? Your values conflict and should double to be unique e.g. 0, 1, 2, 4, 8, etc. This is my core concern with enums. – Timothy Gonzalez Nov 14 '16 at 21:58

4 I'm quite sure a coma is missing after `Raj = 1` , and `Public Enum` should be `public enum` – Rafalon Feb 26 '18 at 14:03

25

I have recently converted away from using enums in my code in favour of instead using classes with protected constructors and predefined static instances (thanks to Roelof - C# Ensure Valid Enum Values - Futureproof Method).

In light of that, below's how I'd now approach this issue (including implicit conversion to/from `int` ).

```
public class Question
{
    // Attributes
    protected int index;
```

```
();
    protected static readonly IDictionary<int,Question> values = new
Dictionary<int,Question>();

    // Define the "enum" values
    public static readonly Question Role = new Question(2,"Role");
    public static readonly Question ProjectFunding = new Question(3, "Project Funding");
    public static readonly Question TotalEmployee = new Question(4, "Total Employee");
    public static readonly Question NumberOfServers = new Question(5, "Number of
Servers");
    public static readonly Question TopBusinessConcern = new Question(6, "Top Business
Concern");

    // Constructors
    protected Question(int index, string name)
    {
        this.index = index;
        this.name = name;
        values.Add(index, this);
    }

    // Easy int conversion
    public static implicit operator int(Question question) =>
        question.index; //nb: if question is null this will return a null pointer
exception

    public static implicit operator Question(int index) =>
        values.TryGetValue(index, out var question) ? question : null;

    // Easy string conversion (also update ToString for the same effect)
    public override string ToString() =>
        this.name;

    public static implicit operator string(Question question) =>
        question?.ToString();

    public static implicit operator Question(string name) =>
        name == null ? null : values.Values.FirstOrDefault(item =>
name.Equals(item.name, StringComparison.CurrentCultureIgnoreCase));


    // If you specifically want a Get(int x) function (though not required given the
implicit converstion)
    public Question Get(int foo) =>
        foo; //(implicit conversion will take care of the conversion for you)
}
```

The advantage of this approach is you get everything you would have from the enum, but your code's now much more flexible, so should you need to perform different actions based on the value of `Question`, you can put logic into `Question` itself (i.e. in the preferred OO fashion) as opposed to putting lots of case statements throughout your code to tackle each scenario.

NB: Answer updated 2018-04-27 to make use of C# 6 features; i.e. declaration expressions and lambda expression body definitions. See revision history for original code. This has the benefit of making the definition a little less verbose; which had been one of the main complaints about this answer's approach.

edited Apr 27 '18 at 8:55

answered Mar 30 '13 at 1:08

JohnLBevan
**15.2k** 1 52 120

1 I guess it's the trade off between explicit cast and the code you have to write to circumvent it. Still love the implementation just wish it wasn't so lengthy. +1 – Lankymart Aug 2 '13 at 10:40

2 I've used several different type of classes structured similar to this. I find they work wonders when trying to follow the "don't let me be an idiot later" methodology. – James Haug Sep 8 '16 at 16:13

19

If you want to get an integer for the enum value that is stored in a variable, wich the type would be `Question`, to use for example in a method, you can simply do this I wrote in this example:

```
enum Talen
{
    Engels = 1, Italiaans = 2, Portugees = 3, Nederlands = 4, Duits = 5, Dens = 6
}

Talen Geselecteerd;

public void Form1()
{
    InitializeComponent()
    Geselecteerd = Talen.Nederlands;
}

// You can use the Enum type as a parameter, so any enumeration from any enumerator can
be used as parameter
void VeranderenTitel(Enum e)
```

This will change the window title to 4 because the variable `Geselecteerd` is `Talen.Nederlands`. If I change it to `Talen.Portugees` and call the method again, the text will change to 3.

I had a hard time finding this simple solution on the internet and I couldn't find it, so I was testing something and found this out. Hope this helps. ;)

edited Feb 27 '18 at 16:02
Timo
3,711 2 27 37

answered Jun 10 '12 at 22:58
Mathijs Van Der Slagt
201 2 2

2 coding in dutch. oh dear. – WiseStrawberry Jan 9 at 14:57

Unfortunately, this approach gives poor performance the more you use it. I tried it in some code of mine, and as time went on, my application got slower and slower, with less and less CPU usage. This implied that the threads were waiting on something - I'm assuming some kind of garbage collection, possibly due to boxing the enum parameter to ToInt32(). By switching to a simple int.Parse(), I was able to eliminate this poor performance completely, and the performance stayed the same no matter how long the code ran. – Greg Feb 4 at 18:38

15

To ensure an enum value exists and then parse it, you can also do the following.

```
// Fake Day of Week
string strDOWFake = "SuperDay";
// Real Day of Week
string strDOWReal = "Friday";
// Will hold which ever is the real DOW.
DayOfWeek enmDOW;

// See if fake DOW is defined in the DayOfWeek enumeration.
if (Enum.IsDefined(typeof(DayOfWeek), strDOWFake))
{
// This will never be reached since "SuperDay"
// doesn't exist in the DayOfWeek enumeration.
    enmDOW = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), strDOWFake);
}
// See if real DOW is defined in the DayOfWeek enumeration.
else if (Enum.IsDefined(typeof(DayOfWeek), strDOWReal))
{
    // This will parse the string into it's corresponding DOW enum object.
```

```
// Can now use the DOW enum object.
Console.Write("Today is " + enmDOW.ToString() + ".");
```

I hope this helps.

edited Jul 23 '11 at 20:03

Peter K.
**6,551** 4 41 67

answered Jul 22 '11 at 20:56

Nathon
**153** 1 6

---

14

Maybe I missed it but has anyone tried a simple generic extension method. This works great for me. You can avoid the type cast in your API this way but ultimately it results in a change type operation. This is a good case for programming Roselyn to have the compiler make a GetValue method for you.

```
    public static void Main()
    {
        int test = MyCSharpWrapperMethod(TestEnum.Test1);

        Debug.Assert(test == 1);
    }

    public static int MyCSharpWrapperMethod(TestEnum customFlag)
    {
        return MyCPlusPlusMethod(customFlag.GetValue<int>());
    }

    public static int MyCPlusPlusMethod(int customFlag)
    {
        //Pretend you made a PInvoke or COM+ call to C++ method that require an integer
        return customFlag;
    }

    public enum TestEnum
    {
        Test1 = 1,
        Test2 = 2,
        Test3 = 3
    }
}

public static class EnumExtensions
```

```
        T result = default(T);

        try
        {
            result = (T)Convert.ChangeType(enumeration, typeof(T));
        }
        catch (Exception ex)
        {
            Debug.Assert(false);
            Debug.WriteLine(ex);
        }

        return result;
    }
}
```

answered Sep 27 '14 at 1:30

Doug
**182** 1 6

Possibly because doing (int)customFlag is less typing all around and does more or less the same thing? – Tim Keating Nov 11 '14 at 17:32

13

One more way to do it:

```
Console.WriteLine("Name: {0}, Value: {0:D}", Question.Role);
```

Will result in:

```
Name: Role, Value: 2
```

answered Sep 19 '14 at 5:42

plavozont
**546** 5 11

12

```
    Role = 2,
    ProjectFunding = 3,
    TotalEmployee = 4,
    NumberOfServers = 5,
    TopBusinessConcern = 6
}
```

...is a fine declaration.

You do have to cast the result to int like so:

```
int Question = (int)QuestionType.Role
```

Otherwise, the type is still `QuestionType`.

This level of strictness is the C# way.

One alternative is to use a class declaration instead:

```
public class QuestionType
{
    public static int Role = 2,
    public static int ProjectFunding = 3,
    public static int TotalEmployee = 4,
    public static int NumberOfServers = 5,
    public static int TopBusinessConcern = 6
}
```

It's less elegant to declare, but you don't need to cast it in code:

```
int Question = QuestionType.Role
```

Alternatively, you may feel more comfortable with Visual Basic, which caters for this type of expectation in many areas.

edited Feb 27 '18 at 17:07

answered Mar 31 '14 at 16:35

Knickerless-Noggins

**5,754** 3 47 62

10

You can do this by implementing an **Extension Method** to your defined enum type:

```
public static class MyExtensions
{
    public static int getNumberValue(this Question questionThis)
    {
        return (int)questionThis;
    }
}
```

This simplify getting int value of current enum value:

```
Question question = Question.Role;
int value = question.getNumberValue();
```

or

```
int value = Question.Role.getNumberValue();
```

edited Dec 9 '12 at 22:51

Blachshma
**15.2k** 4 41 61

answered Dec 9 '12 at 22:09

Bronek
**7,168** 2 36 43

4 Bronek, what you did is make up uninformative syntax through a (non generic btw) extension method that actually takes longer to write. I fail to see how it is better than the original solution by Tetraneutron. Let us not make this into a chat, help is always welcome in stackoverflow and everyone here is here to help. Please take my comment as constructive criticism. – Benjamin Gruenbaum Dec 10 '12 at 0:28

2 Benjamin,first of all,why did you delete my comment?I don't understand your decisions-maybe somebody else through the community would agree with my comment.Secondly,my solution wraps Tetraneutron's one and accurately it is easier and less writing because an extension method is suggested by IntelliSense.So I think your decision is not impartial and representative.I see many similar answering on Stack and it is OK.Anyway I use my solution and maybe there are some people would choose my solution in the future,but these negative points make it harder to find.Most of all it is correct and not copy. – Bronek Dec 10 '12 at 3:20

2 @Bronek If you don't ping me I get no indication that you replied. I did *not* delete your comment I do not have the ability or want to do so. Likely a mod came by and deleted it - you're welcome to flag it for moderator attention and ask why or better yet - ask on Meta Stack Overflow. I have an opinion about your solution from a programming stand point which is perfectly in my right - this is what comments are for to begin with, no need to take it personal. – Benjamin Gruenbaum Aug 7 '13 at 14:45

```
int number = Question.Role.GetHashCode();
```

10

`number` should have the value `2`.

edited Oct 27 '15 at 12:26

Pepijn van Leeuwen
**509** 6 17

answered Sep 18 '14 at 22:53

JaimeArmenta
**109** 1 4

GetHashCode is one way to get value from Enum common mask – ThanhLD Nov 9 '18 at 4:08

---

8

How about a extension method instead:

```
public static class ExtensionMethods
{
    public static int IntValue(this Enum argEnum)
    {
        return Convert.ToInt32(argEnum);
    }
}
```

And the usage is slightly prettier:

```
var intValue = Question.Role.IntValue();
```

answered Apr 21 '14 at 2:51

SixOThree
**463** 1 8 19

---

6

```
public enum Suit : int
{
    Spades = 0,
    Hearts = 1,
    Clubs = 2
```

```
Console.WriteLine((int)(Suit)Enum.Parse(typeof(Suit), "Clubs"));

//from int
Console.WriteLine((Suit)1);

//From number you can also
Console.WriteLine((Suit)Enum.ToObject(typeof(Suit), 1));

if (typeof(Suit).IsEnumDefined("Spades"))
{
    var res = (int)(Suit)Enum.Parse(typeof(Suit), "Spades");
    Console.Out.WriteLine("{0}", res);
}
```

answered Jul 27 '18 at 6:29

Gauravsa

**3,849** 1 11 20

---

4

My fav hack with int or smaller enums:

```
GetHashCode();
```

For a enum

```
public enum Test
{
    Min = Int32.MinValue,
    One = 1,
    Max = Int32.MaxValue,
}
```

this

```
var values = Enum.GetValues(typeof(Test));

foreach (var val in values)
{
```

```
    Console.WriteLine(val);
}
```

outputs

```
one
1
1
max
2147483647
2147483647
min
-2147483648
-2147483648
```

**Disclaimer:** Doesn't work for enums based on long

edited Jul 17 '15 at 13:47

answered Jun 26 '15 at 16:51

Erik Karlsson
**174** 1 11

---

4

Following is the extension method

```
public static string ToEnumString<TEnum>(this int enumValue)
{
    var enumString = enumValue.ToString();
    if (Enum.IsDefined(typeof(TEnum), enumValue))
    {
        enumString = ((TEnum) Enum.ToObject(typeof (TEnum), enumValue)).ToString();
    }
    return enumString;
}
```

edited Nov 16 '18 at 6:25

Ashish Kamble
**904** 1 7 20

answered Dec 16 '16 at 6:58

Kamran Shahid
**1,759** 2 26 49

4

Since enums can be declared with multiple primitive types, a generic extension method to cast any enum type can be useful.

```
enum Box
{
    HEIGHT,
    WIDTH,
    DEPTH
}

public static void UseEnum()
{
    int height = Box.HEIGHT.GetEnumValue<int>();
    int width = Box.WIDTH.GetEnumValue<int>();
    int depth = Box.DEPTH.GetEnumValue<int>();
}

public static T GetEnumValue<T>(this object e) => (T)e;
```

edited Nov 16 '18 at 6:49

Ashish Kamble
**904** 1 7 20

answered Dec 6 '17 at 5:14

Jeffrey Ferreiras
**195** 6

---

3

The example I would like to suggest 'to get 'int' value from enum is,'

```
public enum Sample
{Book =1, Pen=2, Pencil =3}

int answer = (int)Sample.Book;
```

now the answer will be 1.

I hope this might help someone.

answered Aug 3 '15 at 7:12

Vivek
**72** 2 11

2

```
[modifiers] Questions Get(Question q)
{
    return Get((int)q);
}
```

where `[modifiers]` can generally be same as for `Get(int)` method. If You can't edit the `Questions` class or for some reason don't want to, You can overload the method by writing an extension:

```
public static class Extensions
{
    public static Questions Get(this Questions qs, Question q)
    {
        return qs.Get((int)q);
    }
}
```

edited Jan 28 '13 at 14:09

answered Jan 28 '13 at 13:52

Grx70

**7,774** 1 25 45

---

2

Try this one instead of convert enum to int:

```
public static class ReturnType
{
    public static readonly int Success = 1;
    public static readonly int Duplicate = 2;
    public static readonly int Error = -1;
}
```

answered Jan 14 '14 at 20:28

Nalan Madheswaran

**5,630** 1 35 29

---

In Vb, It should be

```
Public Enum Question
    Role = 2
    ProjectFunding = 3
    TotalEmployee = 4
    NumberOfServers = 5
    TopBusinessConcern = 6
End Enum

Private value As Integer = CInt(Question.Role)
```

answered Apr 12 '16 at 7:09

VPP

320 1 4 24

The question is for C#. – Ctrl S Nov 19 '18 at 13:37

2

I a bit late to the game but I came up with this extension method that includes current language features. By using dynamic, I don't need to make this a generic method and specify the type which keeps the invocation simpler and consistent. Please let me know if I've done something wrong:

```
public static class EnumEx
{
    public static dynamic Value(this Enum e)
    {
        switch (e.GetTypeCode())
        {
        case TypeCode.Byte:
            {
                return (byte) (IConvertible) e;
            }
        case TypeCode.Int16:
            {
                return (short) (IConvertible) e;
            }
        case TypeCode.Int32:
            {
                return (int) (IConvertible) e;
            }
```

```
            }
        case TypeCode.UInt16:
            {
                return (ushort) (IConvertible) e;
            }
        case TypeCode.UInt32:
            {
                return (uint) (IConvertible) e;
            }
        case TypeCode.UInt64:
            {
                return (ulong) (IConvertible) e;
            }
        case TypeCode.SByte:
            {
                return (sbyte) (IConvertible) e;
            }
        }

        return 0;
    }
```

answered Mar 2 at 17:55

Jeff
**174** 1 11

---

1

```
Question question = Question.Role;
int value = question.GetHashCode();
```

Will result in `value == 2` .

> This is only true if the enum fits inside an `int`

edited Aug 14 at 19:52

answered Aug 12 at 3:27

GinCanhViet
**44** 10

---

-14

Try this :

```
int value = YourEnum.ToString("D");
```

edited Oct 23 '13 at 7:47

bluish
**15.1k** 19 95 154

answered Jan 10 '13 at 7:37



Caryn
**11**

**protected** by Robert Levy Jul 17 '15 at 13:59

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?