15 Essential C# Interview Questions *

Toptal sourced essential questions that the best C# developers can answer. Driven from our community, we encourage experts to submit questions and offer feedback.

1. What is the output of the short program below? Explain your answer.

```
class Program {
  static String location;
  static DateTime time;

  static void Main() {
    Console.WriteLine(location == null ? "location is null" : location);
    Console.WriteLine(time == null ? "time is null" : time.ToString());
  }
}
```

Hide answer

The output will be:

Although both variables are uninitialized, `String` is a reference type and `DateTime` is a value type. As a value type, an unitialized `DateTime` variable is set to a default value of midnight of 1/1/1 (yup, that's the year 1 A.D.), *not* `null`.

2. Given an array of ints, write a C# method to total all the values that are even numbers.

Hide answer

There are of course many ways to do this, but two of the most straightforward would be either:

```
static long TotalAllEvenNumbers(int[] intArray) {
  return intArray.Where(i => i % 2 == 0).Sum(i => (long)i);
}
```

or:

```
static long TotalAllEvenNumbers(int[] intArray) {
  return (from i in intArray where i % 2 == 0 select (long)i).Sum();
}
```

Here are the key things to look for in the answer:

1. Does the candidate take advantage of the C# language constructs which make a one-liner solution possible (i.e., rather than employing a more lengthy solution which contains a loop, conditional statement, and accumulator)?

2. Does the candidate consider the possibility of overflow. For example, an implementation such as `return intArray.Where(i => i % 2 == 0).Sum()` (regardless of the return type of the function) might be an "obvious" one-line solution, but the probability of overflow here is high. While the approach used in the answers above of converting to `long` doesn't eliminate the possibility, it makes it a highly unlikely that an overflow exception will occur.

```
static DateTime time;
/* ... */
if (time == null)
{
        /* do something */
}
```

Hide answer

One might think that, since a `DateTime` variable can never be null (it is automatically initialized to Jan 1, 0001), the compiler would complain when a `DateTime` variable is compared to `null`. However, due to type coercion, the compiler does allow it, which can potentially lead to headfakes and pull-out-your-hair bugs.

Specifically, the `==` operator will cast its operands to different allowable types in order to get a common type on both sides, which it can then compare. That is why something like this will give you the result you expect (as opposed to failing or behaving unexpectedly because the operands are of different types):

```
double x = 5.0;
int y = 5;
Console.WriteLine(x == y);  // outputs true
```

4. Given an instance `circle` of the following class:

```
public sealed class Circle {
  private double radius;

  public double Calculate(Func<double, double> op) {
    return op(radius);
  }
}
```

Hide answer

The preferred answer would be of the form:

```
circle.Calculate(r => 2 * Math.PI * r);
```

Since we don't have access to the *private* `radius` field of the object, we tell the object itself to calculate the circumference, by passing it the calculation function inline.

A lot of C# programmers shy away from (or don't understand) function-valued parameters. While in this case the example is a little contrived, the purpose is to see if the applicant understands how to formulate a call to `Calculate` which matches the method's definition.

Alternatively, a valid (though less elegant) solution would be to retrieve the radius value itself from the object and then perform the calculation with the result:

```
var radius = circle.Calculate(r => r);
var circumference = 2 * Math.PI * radius;
```

5. What is the output of the program below? Explain your answer.

```
class Program {
  private static string result;

  static void Main() {
    SaySomething();
    Console.WriteLine(result);
  }

  static async Task<string> SaySomething() {
    await Task.Delay(5);
    result = "Hello world!";
    return “Something”;
  }
}
```

Hide answer

The answer to the first part of the question (i.e., the version of the code with `await Task.Delay(5);`) is that the program will just output a blank line (*not* "Hello world!"). This is because `result` will still be uninitialized when `Console.WriteLine` is called.

Most procedural and object-oriented programmers expect a function to execute from beginning to end, or to a `return` statement, before returning to the calling function. This is not the case with C# `async` functions. They only execute up until the first `await` statement, then return to the caller. The function called by `await` (in this case `Task.Delay`) is executed asynchronously, and the line after the `await` statement isn't signaled to execute until `Task.Delay` completes (in 5 milliseconds). However, within that time, control has already returned to the caller, which executes the `Console.WriteLine` statement on a string that hasn't yet been initialized.

Calling `await Task.Delay(5)` lets the current thread continue what it is doing, and if it's done (pending any awaits), returns it to the thread pool. This is the primary benefit of the async/await mechanism. It allows the CLR to service more requests with less threads in the thread pool.

Asynchronous programming has become a lot more common, with the prevalence of devices which perform over-the-network service requests or database requests for many activities. C# has some excellent programming constructs which greatly ease the task of programming asynchronous methods, and a programmer who is aware of them will produce better programs.

6. What is the output of the program below? Explain your answer.

```
delegate void Printer();

static void Main()
{
        List<Printer> printers = new List<Printer>();
        int i=0;
        for(; i < 10; i++)
        {
            printers.Add(delegate { Console.WriteLine(i); });
        }

        foreach (var printer in printers)
```

```
}
```

Hide answer

This program will output the number 10 ten times.

Here's why: The delegate is added in the for loop and "reference" (or perhaps "pointer" would be a better choice of words) to `i` is stored, rather than the value itself. Therefore, after we exit the loop, the variable `i` has been set to 10, so by the time each delegate is invoked, the value

7. It is possible to store mixed datatypes such as int, string, float, char all in one array?

Hide answer

Yes! It is possible to do so because the array can be of type `object` that can not only store any datatype but also the object of the class as shown below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication8
{
    class Program
    {
        class Customer
        {
            public int ID { get; set; }
            public string Name { get; set; }
            public override string ToString()
            {
                return this.Name;
            }
        }
        static void Main(string[] args)
        {
            object[] array = new object[3];
```

```
            c.ID = 55;
            c.Name = "Manish";
            array[2] = c;
            foreach (object obj in array)
            {
                Console.WriteLine(obj);
            }
            Console.ReadLine();
        }
    }
}
```

8. Compare structs and classes in C#. What do they have in common? How do they differ?

Hide answer

Classes and Structs in C# do have a few things in common, namely:

- Are compound data types
- Can contain methods and events
- Can support interfaces

But there are a number of differences. Here's a comparison:

Classes:

- Support inheritance
- Are reference (pointer) types
- The reference can be null
- Have memory overhead per new instance

Structs:

- Do not support inheritance

®

- Cannot have a null reference (unless Nullable is used)
- Do not have memory overhead per new instance (unless “boxed”)

9. What is the output of the program below?

```
public class TestStatic
    {
        public static int TestValue;

        public TestStatic()
        {
            if (TestValue == 0)
            {
                TestValue = 5;
            }
        }
        static TestStatic()
        {
            if (TestValue == 0)
            {
                TestValue = 10;
            }

        }

        public void Print()
        {
            if (TestValue == 5)
            {
                TestValue = 6;
            }
            Console.WriteLine("TestValue : " + TestValue);

        }
    }

 public void Main(string[] args)
        {

            TestStatic t = new TestStatic();
```

Hide answer

```
TestValue : 10
```

The static constructor of a class is called before any instance of the class is created. The static constructor called here initializes the `TestValue` variable first.

10.

```
class ClassA
{
  public ClassA() { }

  public ClassA(int pValue) {  }
}
```

At the client side:

```
class Program
{
  static void Main(string[] args)
  {
    ClassA refA = new ClassA();
  }
}
```

Question:

Is there a way to modify `ClassA` so that you can you call the constructor with parameters, when the Main method is called, without creating any other new instances of the `ClassA`?

Hide answer

The `this` keyword is used to call other constructors, to initialize the class object. The following shows the implementation:

```
  public ClassA() : this(10)
  { }

  public ClassA(int pValue)
  {  }
}
```

11. What does the following code output?

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace main1
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("Hello");
            }
            catch (ArgumentNullException)
            {
                Console.WriteLine("A");
            }
            catch (Exception)
            {
                Console.WriteLine("B");
            }
            finally
            {
                Console.WriteLine("C");
            }
            Console.ReadKey();
        }
    }
}
```

```
Hello
C
```

12. Describe dependency injection.

Hide answer

Dependency injection is a way to de-couple tightly linked classes, thereby reducing the direct dependency of classes upon each other. There are different ways by which dependency injection can be achived:

1. Constructor dependency
2. Property dependency
3. Method dependency

13. Write a C# program that accepts a distance in kilometers, converts it into meters, and then displays the result.

Hide answer

```
using system;
class abc
{
    public static Void Main()
    {
        int ndistance, nresult;
        Console.WriteLine("Enter the distance in kilometers");
        ndistance = convert.ToInt32(Console.ReadLine());
        nresult = ndistance * 1000;
        Console.WriteLine("Distance in meters: " + nresult);
        Console.ReadLine();
```

®

14. Describe boxing and unboxing. Provide an example.

Hide answer

Boxing is an implicit conversion of a value type to the type `object` or to any interface type implemented by the value type. Boxing a value type creates an object instance containing the value and stores it on the heap.

Example:

```
int x = 101;
object o = x;  // boxing value of x into object o

o = 999;
x = (int)o;    // unboxing value of o into integer x
```

15. You're given a word string containing at least one `$` symbol, e.g.:

`"foo bar foo $ bar $ foo bar $ "`

Question: How do you remove all but the first occurrence of `$` from a given string?

Hide answer

This problem has two parts: Preserving the first occurrence and replacing all the others.

We can solve it using a regular expression and `String.Replace()`:

```
using System;
using System.Text.RegularExpressions;

class MainClass {
  public static void Main (string[] args) {
```

```
    GroupCollection halves = Regex.Match(s, @"([^$]*\$)(.*)").Groups;
    string answer = halves[1].Value + halves[2].Value.Replace("$", "");

    Console.WriteLine("after: {0}", answer);
    // like for example $  you don't have   network   access
  }
}
```

Explanation:

- `([^$]*\$)` —Group 1 captures any number of non-`$` characters, plus a single `$` character (escaped with a `\`)
- `(.*)` —Group 2 (greedily) captures everything else

With the first occurrence of `$` preserved in `halves[1].Value`, we can simply use `String.Replace()` on `halves[2].Value` to eliminate all `$` characters found in the remainder

* These sample questions are intended as a starting point for your interview process. If you need additional help, explore our hiring resources—or let Toptal find the best developers, designers, marketing experts, product managers, project managers, and management consultants for you.

Submit an interview question

Submitted questions and answers are subject to review and editing, and may or may not be selected for posting, at the sole discretion of Toptal, LLC.

Name *

®

Enter Your Question Here ... *

Enter Your Answer Here ... *

* I agree with the Terms and Conditions of Toptal, LLC's Privacy Policy

* All fields are required

Submit a Question

®

Talent All Over The World.

Join the Toptal community.

Learn more

Skills in High Demand by Clients

AI Engineer Jobs

React.js Developer Jobs

Python Developer Jobs

JavaScript Developer Jobs

WordPress Developer Jobs

Node.js Developer Jobs

About

Top 3%

Clients

Toptal Talent Assessment

Ultimate Freelancing Guide

Blog

Community

About Us

Contact

Contact Us

Press Center

Careers

FAQ

Toptal®

The World's Top Talent, On Demand ™