

Software Engineering Stack Exchange is a question and answer site for professionals, academics, and students working within the systems development life cycle. It only takes a minute to sign up.

[Sign up to join this community](#)

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top



## Should I return from a function early or use an if statement? [closed]

Asked 8 years, 9 months ago   Active 2 years, 1 month ago   Viewed 136k times

305 I've often written this sort of function in both formats, and I was wondering if one format is preferred over another, and why.

votes



108

```
public void SomeFunction(bool someCondition)
{
    if (someCondition)
    {
        // Do Something
    }
}
```

or

```
public void SomeFunction(bool someCondition)
{
```

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

```
// Do Something  
}
```

I usually code with the first one since that is the way my brain works while coding, although I think I prefer the 2nd one since it takes care of any error handling right away and I find it easier to read

[coding-style](#)[conditions](#)[control-structures](#)

edited Jul 7 '17 at 16:43

community wiki

10 revs, 8 users 46%

[Rachel](#)

**closed** as primarily opinion-based by user40980, [GlenH7 ♦](#), user22815, [Bart van Ingen Schenau](#), [Ixrec](#) Nov 1 '15 at 1:31


Many good questions generate some degree of opinion based on expert experience, but answers to this question will tend to be almost entirely based on opinions, rather than facts, references, or specific expertise.

If this question can be reworded to fit the rules in the [help center](#), please [edit the question](#).

**locked** by [Thomas Owens ♦](#) May 18 at 10:24

This question exists because it has historical significance, but **it is not considered a good, on-topic question for this site** so please do not use it as evidence that you can ask similar questions here. This question and its answers are frozen and cannot be changed. See the [help center](#) for guidance on writing a good question.

Read more about locked posts [here](#).

- 
- 9 I'm a bit late for this discussion so I won't put this in an answer; I also thought about this two years ago: [lecterror.com/articles/view/code-formatting-and-readability](#) I find the second easier to read, modify, maintain and debug. But maybe that's just me :) – [dr Hannibal Lecter](#) Nov 12 '10 at 12:39
- 
- 8 Related: [Where did the notion of "one return only" come from?](#) – [yannis](#) Jan 23 '15 at 19:49 
- 
- 4 now this question is a fine example of an opinion-based question – [Rudolf Olah](#) Oct 27 '15 at 15:44
- 
- 2 so what if there is no absolute proof in one or another direction? When enough argumentation is provided in one and another direction and there is a voting if the answer is correct - it makes it very useful. I find closing questions like this to be harmful to the value of this site. – [gsf](#) Oct 13 '16 at 4:18
- 

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

## 19 Answers

**402** votes I prefer the second style. Get invalid cases out of the way first, either simply exiting or raising exceptions as appropriate, put a blank line in there, then add the "real" body of the method. I find it easier to read.



answered Nov 11 '10 at 19:27

community wiki  
Mason Wheeler

- 
- 7 Smalltalk calls these "guard clauses". At least that's what Kent Beck calls them in Smalltalk Best Practice Patterns; I don't know if it's common parlance. – [Frank Shearar](#) Nov 11 '10 at 19:37
- 
- 154 Exiting early allows you to pop stuff off your limited mental stack. :) – [Joren](#) Nov 12 '10 at 7:59
- 
- 50 I've previously heard this referred to as "the Bouncer Pattern" - get rid of bad cases before they get in the door. – [RevBingo](#) Nov 12 '10 at 21:02
- 
- 14 I'd even go as far and say this definitely is the ONLY right thing to do. – [Oliver Weiler](#) Dec 6 '10 at 0:34
- 
- 38 Plus you don't keep adding indents if you get rid of the border cases at the beginning. – [doppelgreener](#) Feb 18 '11 at 13:31
- 

**170** votes Definitely the latter. The former doesn't look bad right now, but when you get more complex code, I can't imagine anyone would think this:

```
public int SomeFunction(bool cond1, string name, int value, AuthInfo perms)
{
    int retval = SUCCESS;
    if (someCondition)
    {
        if (name != null && name != "")
        {
            if (value != 0)
            {
                if (perms.allow(name)
                {
                    ...
                }
            }
        }
    }
}
```

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

```
        {
            reval = PERM_DENY;
        }
    }
    else
    {
        retval = BAD_VALUE;
    }
}
else
{
    retval = BAD_NAME;
}
}
else
{
    retval = BAD_COND;
}
return retval;
}
```

is more readable than

```
public int SomeFunction(bool cond1, string name, int value, AuthInfo perms)
{
    if (!someCondition)
        return BAD_COND;

    if (name == null || name == "")
        return BAD_NAME;


    if (value == 0)
        return BAD_VALUE;

    if (!perms.allow(name))
        return PERM_DENY;

    // Do something
    return SUCCESS;
}
```

I fully admit I never understood the advantage of single exit points.

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

- 20 The advantage of single a exit point is that... there's a single exit point! With your example, there's several points which could return. With a more complex function, that could turn into a hunt-the-exit-point when the format of the return value changes. Of course, there's times when forcing a single exit point doesn't make sense. – [JohnL](#) Nov 11 '10 at 21:17
- 71 @JohnL big functions are the problem, not multiple exit points. Unless you're working in a context where an additional function call will slow down your code immensely, of course... – [Dan Rosenstark](#) Nov 11 '10 at 21:23
- 4 @Yar: True, but the point stands. I'm not going to try and convert anyone, and I was just pointing out the advantage of going for as few exit points as possible (and Jason's example was kind of straw man anyway). Just next time you're pulling your hair out trying to figure out why SomeFunction sometimes returns odd values, maybe you'll wish you could add a logging call *just before* the return. Much easier to debug if there's only one of the buggers! :) – [JohnL](#) Nov 11 '10 at 23:11
- 76 @Jason Viers As if a single `return value;` helps!?! Then one has to hunt half a dozen `value = ...`, with the disadvantage that you are never sure that value will not be changed between this assignment and the final return. At least an immediate return is clear that nothing will change the result anymore. – [Sjoerd](#) Feb 18 '11 at 17:26
- 5 @Sjoed: I second Sjoerd here. If you want to log the result, you can log at the caller site. If you want to know the reason, you have to log at each exit point/assignment so both are identical in this aspect. – [Matthieu M.](#) Feb 18 '11 at 18:46 

**32** **It depends** - In general I am not going to go out of my way to try and move a bunch of code around to break out of the function early - the compiler will generally take care of that for me. That said though, if there are some basic parameters at the top that I need and can't continue otherwise, I will breakout early. Likewise, if a condition generates a giant `if` block in function I will have it breakout early as a result of that as well.

votes

That said though, if a function requires some data when it is called, I'm usually going to be throwing an exception (see example) as opposed to just returning.

```
public int myFunction(string parameterOne, string parameterTwo) {
    // Can't work without a value
    if (string.IsNullOrEmpty(parameterOne)) {
        throw new ArgumentNullException("parameterOne");
    }
    if (string.IsNullOrEmpty(parameterTwo)) {
        throw new ArgumentNullException("parameterTwo");
    }


    // ...
    // Do some work
    ..
}
```

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

```
return value;
}
```

edited Jul 7 '17 at 16:44

community wiki  
2 revs, 2 users 93%  
rjzii

- 
- 9 And if the end result is maintainable, who cares which style was selected? – [Jeff Siver](#) Nov 11 '10 at 19:32
- 
- 3 @Jeff Siver - Thus why this tends to be a "holy-war" style question, at the end of the day it comes down to personal preference and whatever an internal style guide says. – [rjzii](#) Nov 11 '10 at 19:33
- 
- 1 The key take-away here is that he's throwing an exception instead of returning early. The return value should not be re-purposed for validity checks. What if you had various conditions and wanted to let the code using the method know why it failed? Suddenly you might return actual business data, nothing (empty result) or many different strings, codes, numers, ... from one single method. Just to describe why it failed. No thanks. – [DanMan](#) May 24 '14 at 10:51 
- 
- what about cyclomatic complexity as my compiler suggests? isnt it better to not nest code if one can help it? – [|--"-----"-----"](#) Apr 1 '16 at 3:19
- 

## 24 I prefer the early return.

votes If you have one entry point and one exit point then you always have to track the entire code in your head all the way down to the exit point (you never know if some other piece of code bellow does something else to the result, so you have to track it up until the exist). You do that no mater which branch determines the final result. This is hard to follow.

With one entry and multiple exists, you return when you have your result and don't bother tracking it all the way down to see that nobody does anything else to it (because there won't be anything else since you returned). It's like having the method body split into more steps, which each step with the possibility to return the result or let the next step try its luck.

edited Nov 11 '10 at 20:50

community wiki  
2 revs  
user7197

- 
- 13 In C programming where you have to manually clean-up there is a lot to be said for one-point return. Even if there is no need right now to clean something up, someone might edit your function, allocate something and need to clean it up before return. If that happens it will be

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

In C++ programming you have destructors and even now scope-exit guards. All these need to be here to ensure the code is exception-safe in the first place, so code is well guarded against early exit and therefore doing so has no logical downside and is purely a style issue.

I am not knowledgeable enough about Java, whether "finally" block code will get called and whether finalizers can handle the situation of needing to ensure something happens.

C# I certainly can't answer on.


D-language gives you proper built-in scope-exit guards and therefore is well-prepared for early exit and therefore should not present an issue other than style.

Functions should of course not be so long in the first place, and if you have a huge switch statement your code is probably also badly factored.

answered Feb 18 '11 at 13:21

community wiki  
[CashCow](#)

---

1 C++ allows something called return value optimization which permits the compiler to essentially omit the copy operation that would normally occur when you return a value. However, under various conditions that's hard to do, and multiple return values is one of those cases. In other words, using multiple return values in C++ can actually make your code slower. This certainly holds on MSC, and given that it would be quite tricky (if not impossible) to implement RVO with multiple possible return values, it's likely a problem in all compilers. – [Eamon Nerbonne](#) May 20 '14 at 14:27 

---

1 In C, just use `goto` and, possibly, a two-point return. Example (code formatting not possible in comments): `foo() { init(); if (bad) goto err; bar(); if (bad) goto err; baz(); return 0; err: cleanup(); return 1; }` – [mirabilos](#) May 24 '14 at 12:20

---

1 Instead of a `goto` I prefer "Extract Method". When you think you need to implement a return value variable or a `goto`, in an effort to ensure your cleanup code is always called, that's a Smell that you need to break it up into multiple functions. This allows you to simplify your complex conditional code using Guard Clauses and other early-return techniques, while still ensuring your cleanup code always runs. – [BrandonLWhite](#) Sep 17 '15 at 14:48

---

"someone might edit your function" - this is such a ridiculous explanation for decision making. Anyone can do anything in the future. It doesn't mean that you should do something specific today to prevent someone from breaking things in the future. – [Victor Yarema](#) Oct 23 '18 at 18:42

---

Called making your code maintainable. In the real world code is written for business purposes and sometimes a developer later on needs to change it. At the time I wrote that answer though I had patched CURL to introduce caching and recall the spaghetti mess that it was that really needed a proper rewrite in modern C++ – [CashCow](#) Oct 30 '18 at 14:55

---

## 9 I use both.


votes If `DoSomething` is 3-5 lines of code then the code just look beautiful using the first formatting method.

But if it has many more lines than that, then I prefer the second format. I don't like when the opening and closing brackets are not on the same screen.

answered Nov 12 '10 at 18:10

community wiki  
azheglov

2 Beautiful no way! Too much indentation! – JimmyKane Apr 24 '16 at 19:40

@JimmyKane There's only so much indentation which can happen in 3-5 lines, especially as you need 2 (?) lines per level the rest gets indented: One for the control-structure and start of block, one for end of block. – Deduplicator Oct 2 '18 at 2:47 

## 8 A classic reason for single-entry-single-exit is that otherwise the formal semantics become unspeakably ugly otherwise (same reason GOTO was considered harmful).

votes

Put another way, it's easier to reason about when your software will exit the routine if you have only 1 return. Which is also an argument against exceptions.

Typically I minimize the early-return approach.

answered Nov 11 '10 at 19:40

community wiki  
Paul Nathan

but for a formal analysis tool you can synthesize an outer function with the semantics the tool needs, and keep the code human readable. – Tim Williscroft Nov 11 '10 at 21:48

OT: This is a good question, but it's not a good question to ask on a forum. It's a good question to ask a mentor or a colleague.

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



My attitude to analysis is shaped by optimizations from the Self project. 99% of dynamic method calls can be statically resolved and eliminated. so then you can analyze some pretty straight line code. As a working programmer, I can reliably assume most code I'll work on was written by average programmers. So they won't write very nice code. – [Tim Williscroft](#) Nov 11 '10 at 22:31

@Tim: Fair enough. Although I do feel compelled to point out that static languages still have a lot of play. – [Paul Nathan](#) Nov 11 '10 at 22:36

A reason that doesn't hold in the face of exceptions. Which is why single-exit is great in C, but doesn't buy you anything in C++. – [peterchen](#) Nov 11 '10 at 23:15

7 votes Personally, I prefer to do pass/fail condition checks at the beginning. That allows me to group most of the most common failures at the top of the function with the rest of the logic to follow.

answered Nov 11 '10 at 19:26

community wiki  
[nathan](#)

6 votes It depends.

Early return if there is some obvious dead end condition to check for right away that would make running the rest of the function pointless.\*

Set Retval + single return if the function is more complex and could have multiple exit points otherwise (readability issue).

*\*This can often indicate a design problem. If you find that a lot of your methods need to check some external/paramater state or such before running the rest of the code, that's probably something that should be handled by the caller.*

answered Nov 11 '10 at 20:22

community wiki  
[Bobby Tables](#)

6 When I write code that may be shared, my mantra is "Assume Nothing. Trust No One." You should always validate your inputs and any external state you depend on. Better to throw an exception than possibly corrupt something because somebody gave you bad data. – [TMN](#) Nov 11 '10 at 20:59

@TMN: Good point. – [Bobby Tables](#) Nov 11 '10 at 23:41

2 Kev point here is that in OO you *throw an exception*. not *return*. Multiple returns can be bad. multiple exception throws aren't necessarily a code smell. –

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

been achieved even before the function started. For example, if one invokes a "Set control label" method to change a control's label to `Fred`, the window's label is already `Fred`, and setting the control's name to its present state would force a redraw (which, while potentially useful in some cases, would be annoying in the one at hand), it would be perfectly reasonable to have the set-name method early-exit if the old and new names match. – [supercat](#) Jan 13 '15 at 18:15

---

### 3 Use an If

votes In Don Knuth's book about GOTO's I read him give a reason for always having the most likely condition come first in an if statement. Under the assumption that this is still a reasonable idea (and not one out of pure consideration for the speed of the era). I'd say early returns aren't good programming practice, especially considering the fact that they're more often than not used for error handling, unless your code is more likely to fail than not fail :-)

If you follow the above advice, you'd need to put that return at the bottom of the function, and then you might as well not even call it a return there, just set the error code and return it two lines hence. Thereby achieving the 1 entry 1 exit ideal.

#### Delphi Specific...

I'm of the mind that this is a good programming practice for Delphi programmers, although I don't have any proof. Pre-D2009, we don't have an atomic way to return a value, we have `exit;` and `result := foo;` or we could just throw exceptions.

If you had to substitute

```
if (true) {  
    return foo;  
}
```

for

```
if true then  
begin  
    result := foo;  
    exit;  
end;
```

you might just get sick of seeing that at the top of every one of your functions and prefer

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

```
...
end
else
  result := foo;
```

and just avoid `exit` altogether.

edited Jul 7 '17 at 16:46

community wiki  
3 revs, 2 users 88%  
Peter Turner

2 With newer Delphis, it could be abbreviated to `if true then Exit(foo);` I use often the technique to first initialize `result` to `nil` or `FALSE` respectively, then checking all the error conditions and just `Exit;` if one is met. The success-case `result` is then (typically) set somewhere at the end of the method. – JensG May 21 '14 at 8:13

yeah, I like that new feature although it seems like it's just candy to appease Java programmers, next thing you know they'll be letting us define our variables inside the procedures. – Peter Turner May 21 '14 at 14:10

And C# programmers. Yes, to be honest, I would find that indeed useful as it reduces the number of lines between declaration and use (IIRC there is even some metric for that, forgot the name). – JensG May 21 '14 at 20:00

## 2 I agree with the following statement:

votes

I'm personally a fan of guard clauses (the second example) as it reduces the indenting of the function. Some people don't like them because it results in multiple return points from the function, but I think it's clearer with them.

Taken from [this question in stackoverflow](#).

edited May 23 '17 at 12:40

community wiki  
2 revs  
Toto

+1 For guard clauses. I also prefer a positively oriented guard eg: `if (condition = false) return` as opposed to `if (!condition) return` – JustinC Feb 18 '11 at 16:28

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

1 I use early returns almost exclusively these days, to an extreme. I write this

vote

```
self = [super init];

if (self != nil)
{
    // your code here
}

return self;
```

as

```
self = [super init];
if (!self)
    return;

// your code here

return self;
```

but it really doesn't matter. If you have more than one or two levels of nesting in your functions, they need to be busted up.

edited Jul 7 '17 at 16:48

community wiki  
2 revs, 2 users 92%  
Dan Rosenstark

---

I agree. Indentation is what causes trouble in reading. The less indentation the better. Your simple example has the same level of indentation but that first example will surely grow to more indentations which require more brain power. – [user441521](#) May 4 '17 at 17:35

---

1 I would prefer to write:

vote

```
if(someCondition)
{
    SomeFunction();
}
```

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

- 
- 2 eww. Is that pre-validation? Or is it in a validation-dedicated mehtod `DoSomeFunctionIfSomeCondition` ? – [STW](#) Nov 11 '10 at 19:27
- 
- How does this keep your concerns separated? – [justkt](#) Nov 11 '10 at 20:00
- 
- 2 You're violating encapsulation by making the function's implementation (its dependency logic) external. – [TMN](#) Nov 11 '10 at 20:45
- 
- 1 If this is run in a public method and `SomeFunction()` is a private method in the same class it could be ok. But if anyone else is calling `SomeFunction()` you would have to duplicate the checks there. I find it better to make each method take care of what it needs to do its work, no one else should have to know that. – [Per Wiklander](#) Nov 11 '10 at 23:34
- 
- 2 This is definitely the style that Robert C. Martin proposes in "Clean Code". A function should only do one thing. However in "Implementation Patterns", Kent Beck suggests, of the two options proposed in the OP, the second is better. – [Scott Whitlock](#) Nov 13 '10 at 1:20
- 

1 Like you, I usually write the first one, but prefer the last one. If i have a lot of nested checks i usually refactor to the second method.

vote I don't like how the error handling is moved away from the check.

```
if not error A
  if not error B
    if not error C
      // do something
    else handle error C
  else handle error B
else handle error A
```

I prefer this:

```
if error A
  handle error A; return
if error B
  handle error B; return
if error C
  handle error C; return

// do something
```

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

0 The conditions on the top are called "preconditions". By putting `if(!precond) return;` , you are visually listing all preconditions.

votes Using the large "if-else" block may increase indent overhead (I forgot the quote about 3-level indents).

edited Aug 3 '16 at 2:41

community wiki

2 revs

Ming-Tang

2 What? You can't do an early return in Java? C#, VB (.NET and 6), and apparently Java (which I assumed did, but had to look up since I haven't used the language in 15 years) all allow early returns. so don't accuse 'strongly-typed languages' as not having the feature. [stackoverflow.com/questions/884429/...](https://stackoverflow.com/questions/884429/...) – ps2goat Aug 2 '16 at 21:49

-1 I prefer to keep if statements small.

votes So, choosing between:

```
if condition:
    line1
    line2
    ...
    line-n
```

and

```
if not condition: return

line1
line2
...
line-n
```

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

Nested if's and for's and while's are *horrible*, avoid them at all costs.

edited Jul 7 '17 at 16:50

community wiki  
2 revs, 2 users 96%  
hasen

-2

votes

As others say, it depends. For little functions that return values, I may code early returns. But for sizeable functions, I like to always have a place in the code where I know I can put something that will get executed before it returns.

answered Nov 11 '10 at 19:37

community wiki  
Mike Dunlavey

-2

votes

I practice fail-fast at the function level. It keeps code consistent and clean (to me and those I've worked with). Because of that I always return early.

For some oft-checked conditions, you can implement aspects for those checks if you use AOP.

answered Nov 11 '10 at 19:38

community wiki  
Steven Evers