スタート
ごみ箱

ERROR HANDLING

I was thinking, if the error exists between keyboard and chair, I want the strictest failure mode to both catch it and force me to do things right the first time.

Andai @ https://news.ycombinator.com/item?id=22833601

AGENDA

- What is worse than crashing?
- Error handling in your code
- Error tolerant architectures
- War stories

WHAT IS WORSE THAN CRASHING?

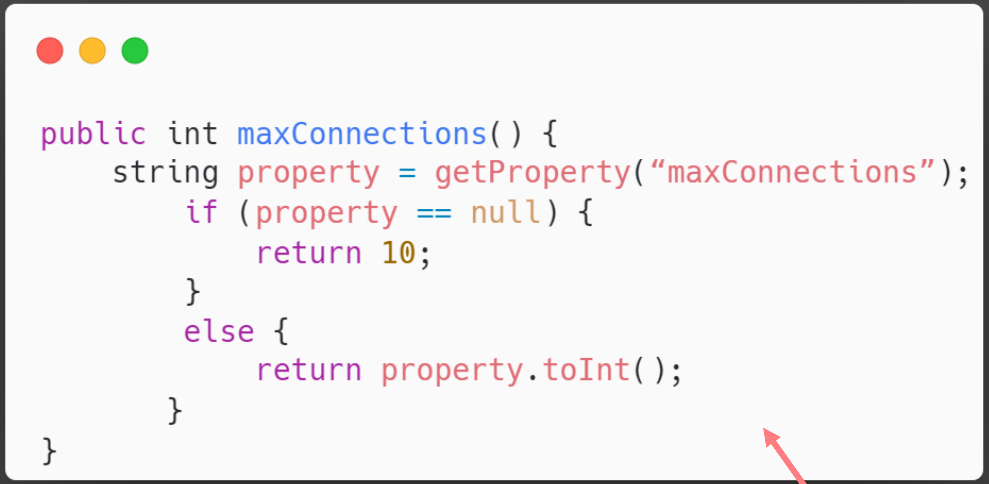
Crash scenarios, in order from best to worst:

1. Application works as expected and never crashes.
2. Application crashes due to rare bugs that nobody notices or cares about.
3. Application crashes due to a commonly encountered bug.
4. Application deadlocks and stops responding due to a common bug.
5. Application crashes long after the original bug.
6. **Application causes data loss and/or corruption.**

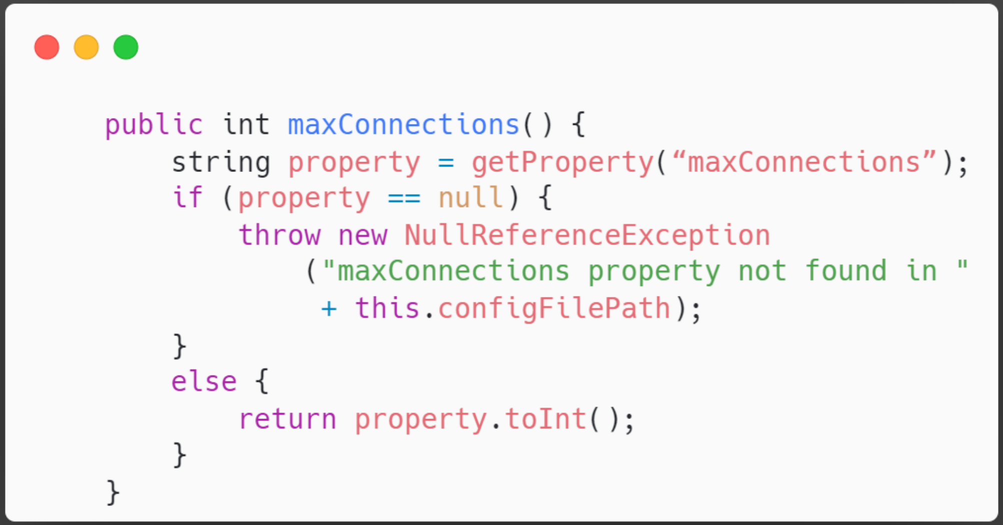
There's a natural tension between...

- failing *immediately* when your program encounters a problem, eg "fail fast"
- attempting to recover from the failure state and proceed normally

FAIL FAST



```
public int maxConnections() {
    string property = getProperty(“maxConnections”);
        if (property == null) {
            return 10;
        }
        else {
            return property.toInt();
        }
}
```



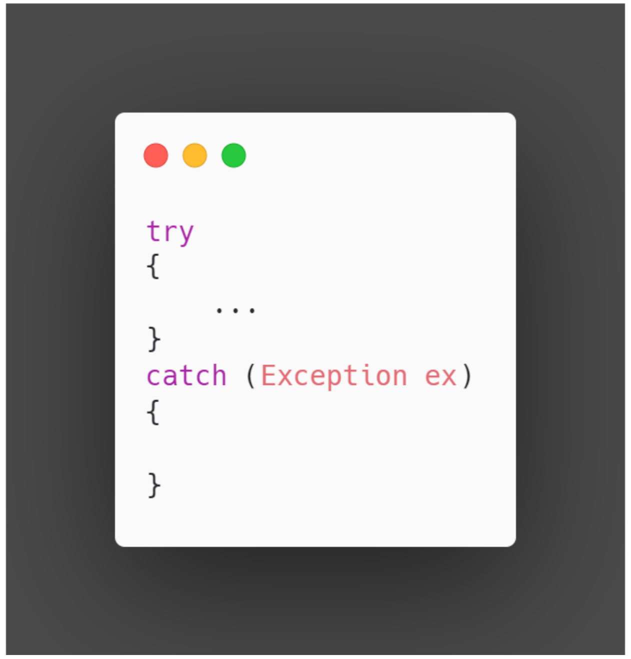
```
public int maxConnections() {
    string property = getProperty(“maxConnections”);
    if (property == null) {
        throw new NullReferenceException
            ("maxConnections property not found in "
             + this.configFilePath);
    }
    else {
        return property.toInt();
    }
}
```

This code does not fail fast. It will become hard to debug why the code might be slow.

ERROR HANDLING IN YOUR CODE

EXCEPTIONS

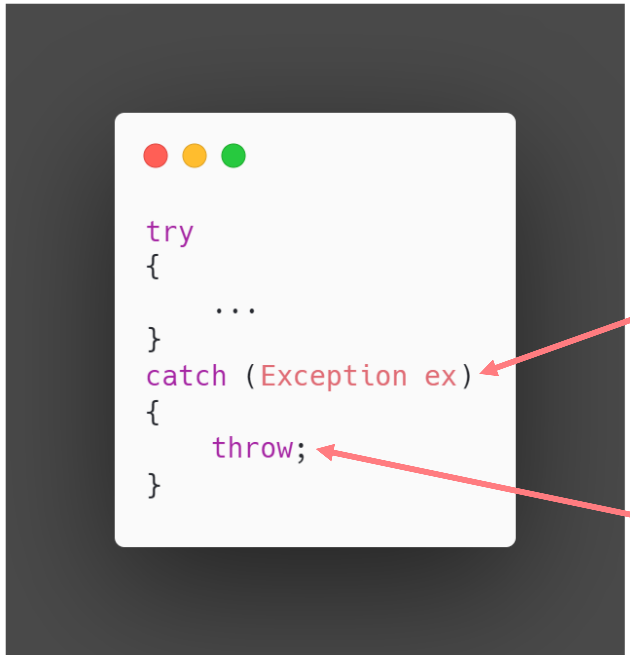
- Only catch exceptions that you can handle



```
try
{
    ...
}
catch (Exception ex)
{

}
```

EXCEPTIONS

- Only catch exceptions that you can handle
  - Unless you just log and rethrow



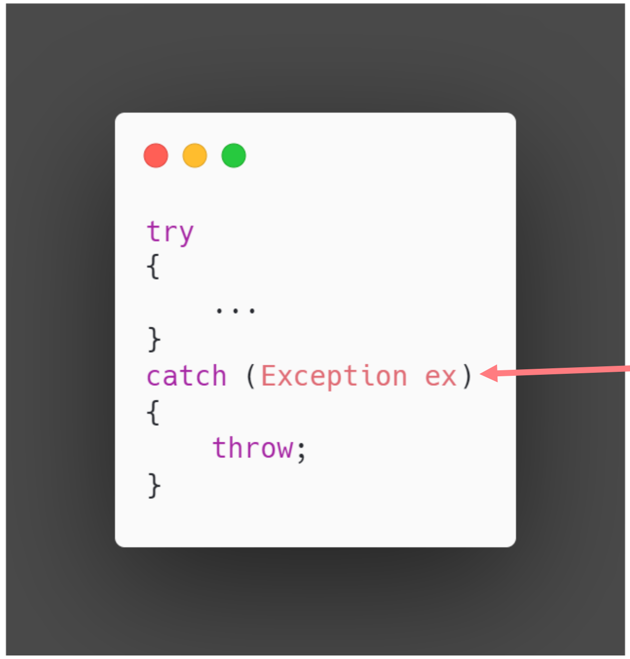
```
try
{
    ...
}
catch (Exception ex)
{
    throw;
}
```

What do you think this is called

Do not write “throw ex” - You get more stack trace information with just using “throw”

EXCEPTIONS

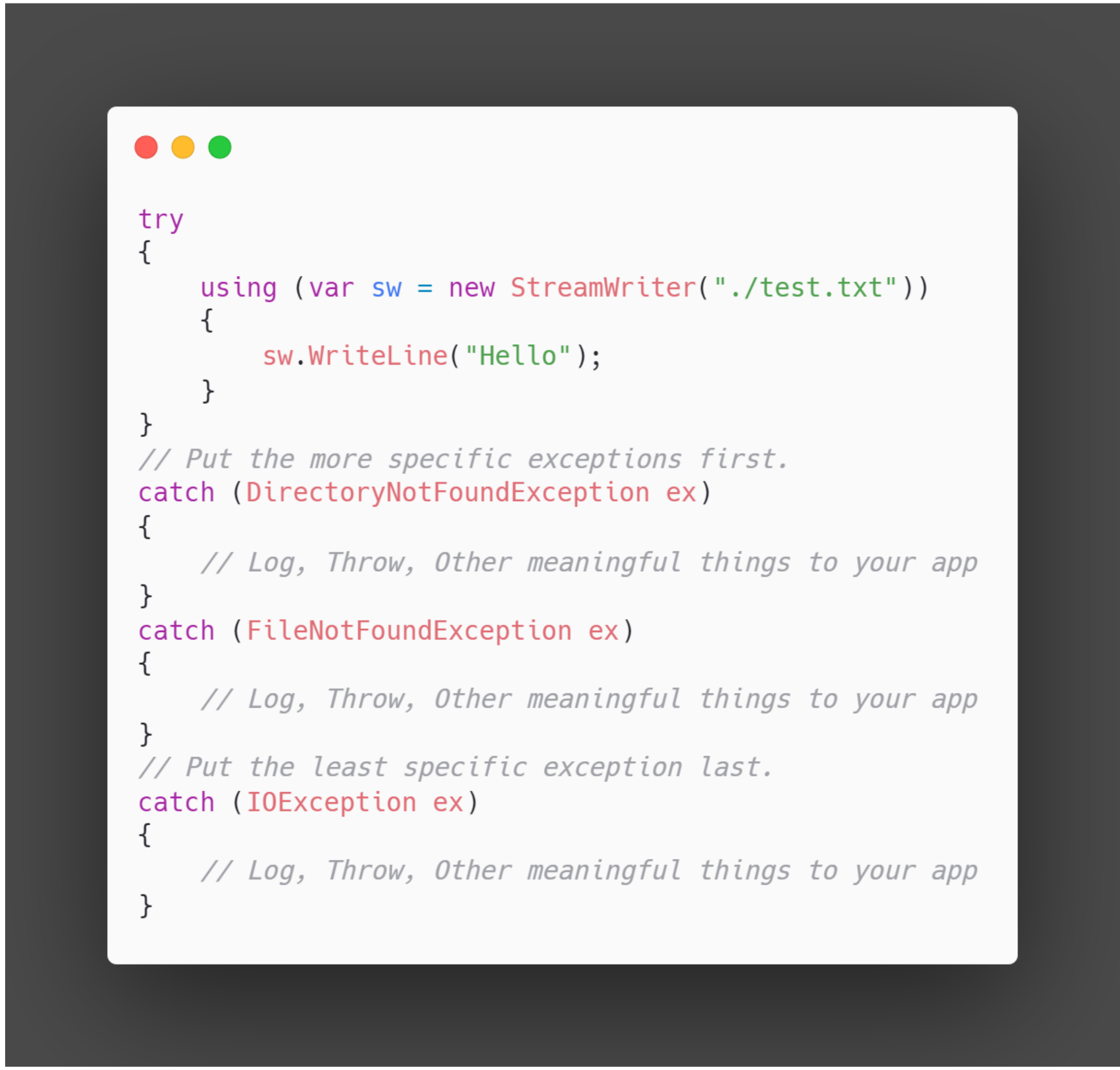
- Only catch exceptions that you can handle
  - Unless you just log and rethrow



```
try
{
    ...
}
catch (Exception ex)
{
    throw;
}
```

EXCEPTIONS

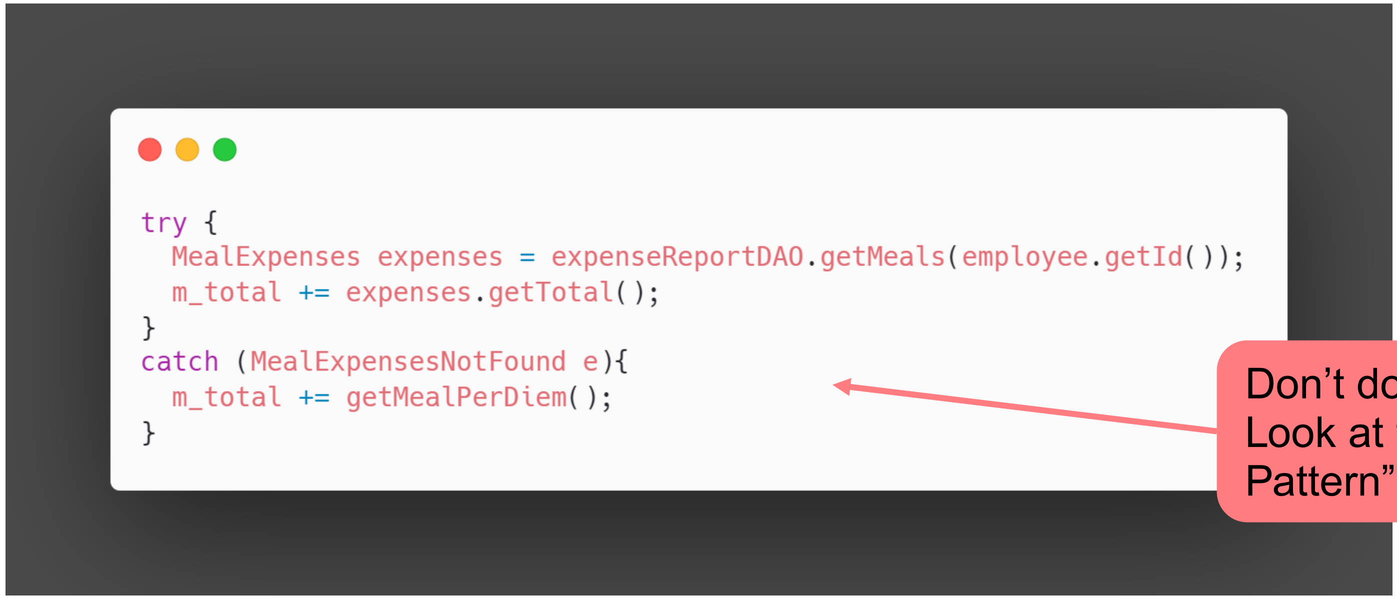
- Only catch exceptions that you can handle
  - Unless you just log and rethrow
  - Example of different types of exceptions



```
try
{
    using (var sw = new StreamWriter("./test.txt"))
    {
        sw.WriteLine("Hello");
    }
}
// Put the more specific exceptions first.
catch (DirectoryNotFoundException ex)
{
    // Log, Throw, Other meaningful things to your app
}
catch (FileNotFoundException ex)
{
    // Log, Throw, Other meaningful things to your app
}
// Put the least specific exception last.
catch (IOException ex)
{
    // Log, Throw, Other meaningful things to your app
}
```

EXCEPTIONS

- Only catch exceptions that you can handle
  - Unless you just log and rethrow
  - Example of different types of exceptions
  - Don’t use exceptions to emulate control-flow

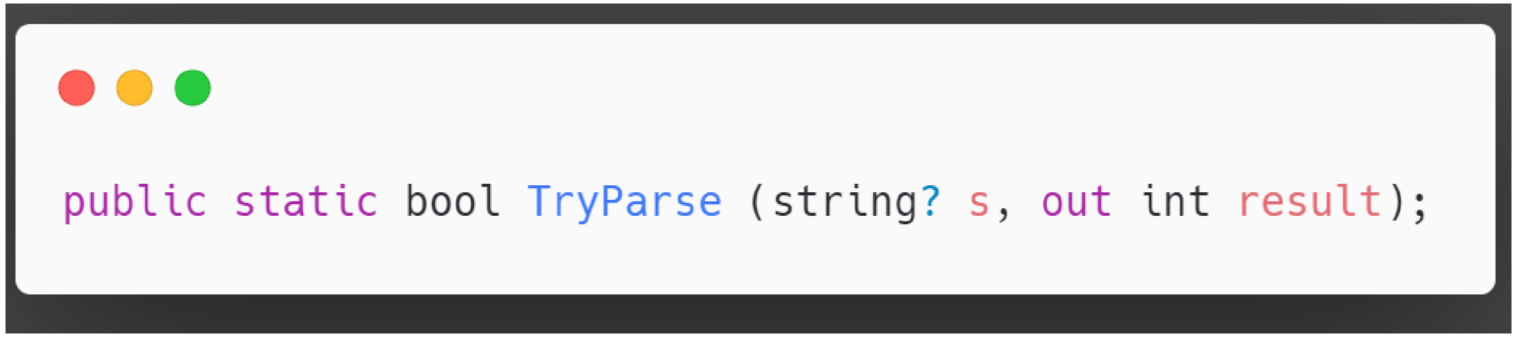


```
try {
  MealExpenses expenses = expenseReportDAO.getMeals(employee.getId());
  m_total += expenses.getTotal();
}
catch (MealExpensesNotFound e){
  m_total += getMealPerDiem();
}
```

Don’t do this!
Look at the “Special Case Pattern” if you have code like this

GET THE COMPILER TO HELP WITH ERROR HANDLING

- Out parameters in C#

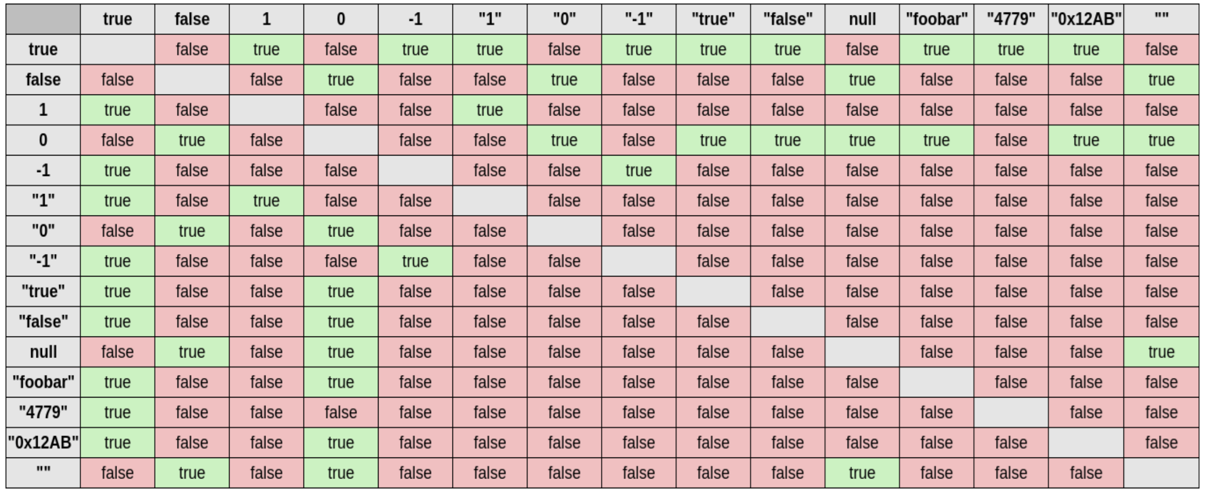


```
public static bool TryParse (string? s, out int result);
```

- Functional languages often use something called Option, Either or Maybe
  - The return value is “wrapped” with a Some and a None value
- Checked exception
  - Great idea – but isn’t really practical

DYNAMIC TYPE LANGUAGES

- Examples JavaScript, PHP, Python, …
- Variable can be assigned and re-assigned to anything
- No check for return type of function matches or are equal
- Often just try handling values in some way

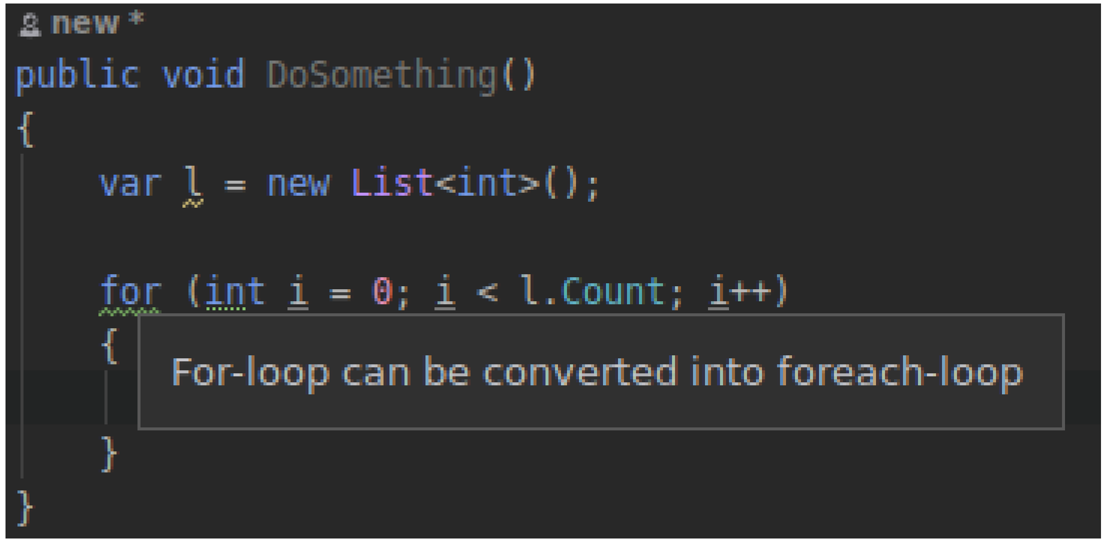


	true	false	1	0	-1	"1"	"0"	"-1"	"true"	"false"	null	"foobar"	"4779"	"0x12AB"	""
true		false	true	false	true	true	false	true	true	true	false	true	true	true	false
false	false		false	true	false	false	true	false	false	false	true	false	false	false	true
1	true	false		false	false	true	false	false	false	false	false	false	false	false	false
0	false	true	false		false	false	true	false	true	true	true	true	false	true	true
-1	true	false	false	false		false	false	true	false	false	false	false	false	false	false
"1"	true	false	true	false	false		false	false	false	false	false	false	false	false	false
"0"	false	true	false	true	false	false		false	false	false	false	false	false	false	false
"-1"	true	false	false	false	true	false	false		false	false	false	false	false	false	false
"true"	true	false	false	true	false	false	false	false		false	false	false	false	false	false
"false"	true	false	false	true	false	false	false	false	false		false	false	false	false	false
null	false	true	false	true	false	false	false	false	false	false		false	false	false	true
"foobar"	true	false	false	true	false	false	false	false	false	false	false		false	false	false
"4779"	true	false	false	false	false	false	false	false	false	false	false	false		false	false
"0x12AB"	true	false	false	true	false	false	false	false	false	false	false	false	false		false
""	false	true	false	true	false	false	false	false	false	false	true	false	false	false	

- What happens if you expect int but get object?

LINTING & WARNING

- Warnings comes from the compiler – listen to them!
- Linting is a separate tool (ex. Reshaper, Sonarlint)
- Both help us avoid errors
- **Always** treat warnings as errors – They are warnings for a reason

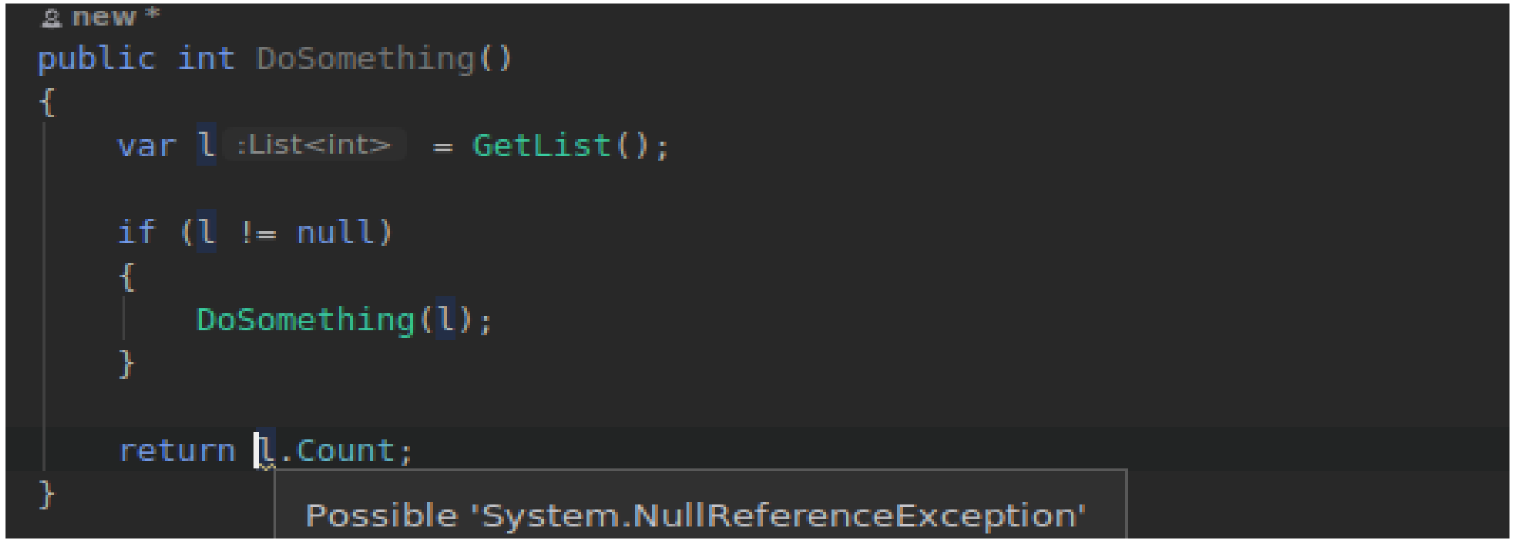


```
new *
public void DoSomething()
{
    var l = new List<int>();

    for (int i = 0; i < l.Count; i++)
    {

    }
}
```

For-loop can be converted into foreach-loop

LINTING & WARNING

- Warnings comes from the compiler – listen to them!
- Linting is a separate tool (ex. Reshaper, Sonarlint)
- Both help us avoid errors
- **Always** treat warnings as errors – They are warnings for a reason

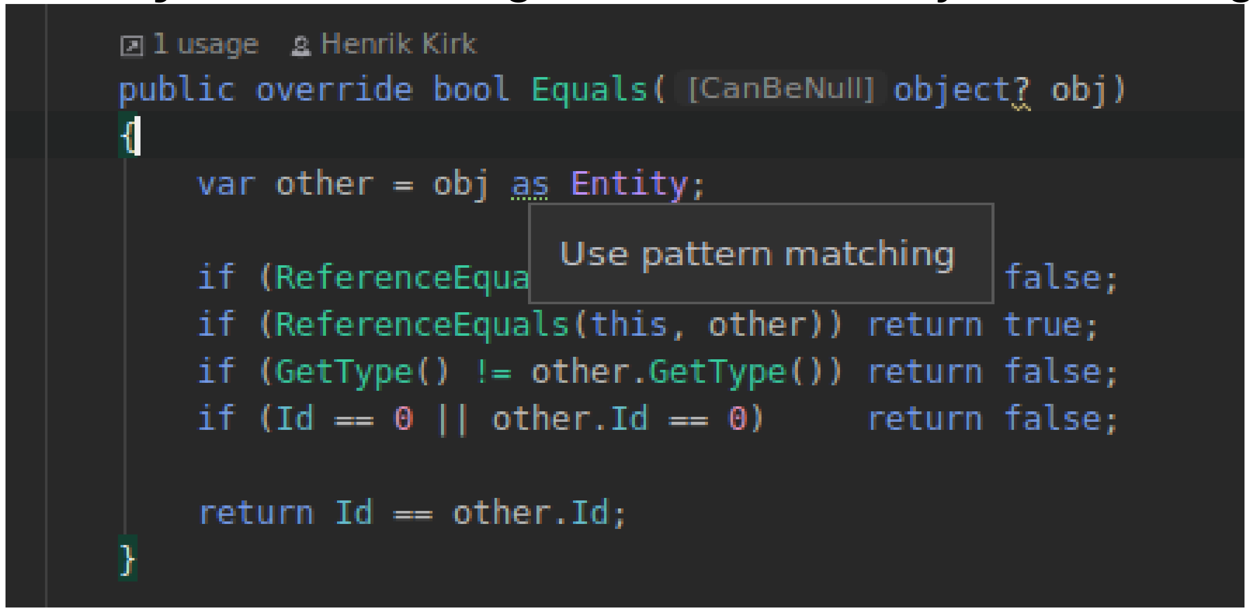


```
new *
public int DoSomething()
{
    var l :List<int>  = GetList();

    if (l != null)
    {
        DoSomething(l);
    }

    return l.Count;
}
```

Possible 'System.NullReferenceException'

LINTING & WARNING

- Warnings comes from the compiler – listen to them!
- Linting is a separate tool (ex. Reshaper, Sonarlint)
- Both help us avoid errors
- **Always** treat warnings as errors – They are warnings for a reason



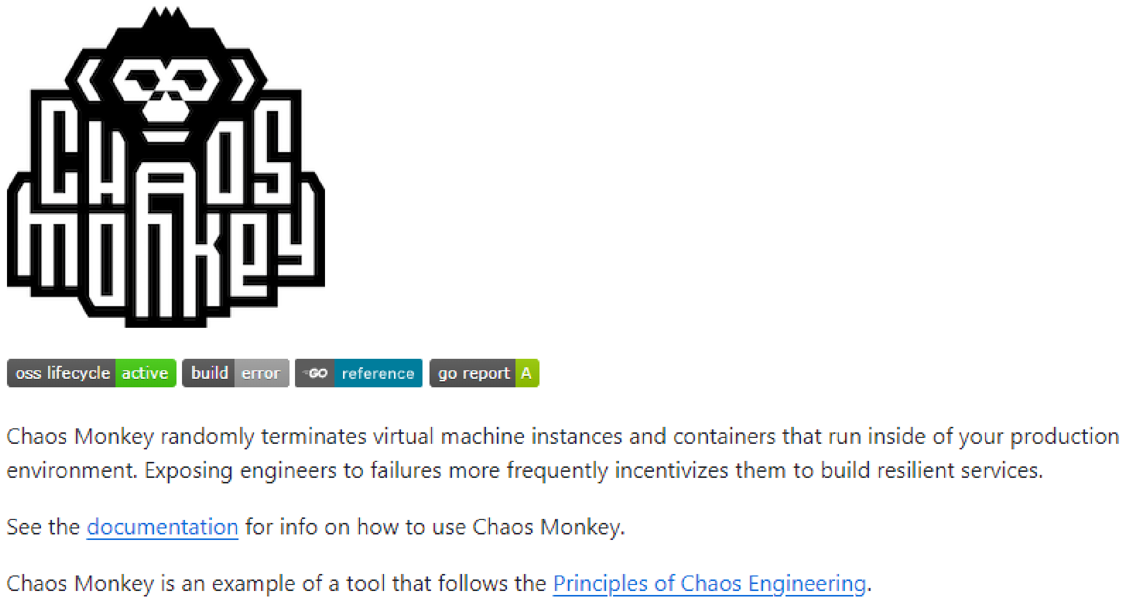
```
1 usage   Henrik Kirk
public override bool Equals( [CanBeNull] object? obj)
{
    var other = obj as Entity;

    if (ReferenceEqua                                false;
    if (ReferenceEquals(this, other)) return true;
    if (GetType() != other.GetType()) return false;
    if (Id == 0 || other.Id == 0)     return false;

    return Id == other.Id;
}
```

Use pattern matching

ERROR TOLERANT ARCHITECTURES

ARCHITECTURAL STRATEGIES

- Handle failures by:
  - Redundancy (think Netflix)
  - Reduced capability
- Testing error handling
  - Chaos Engineering



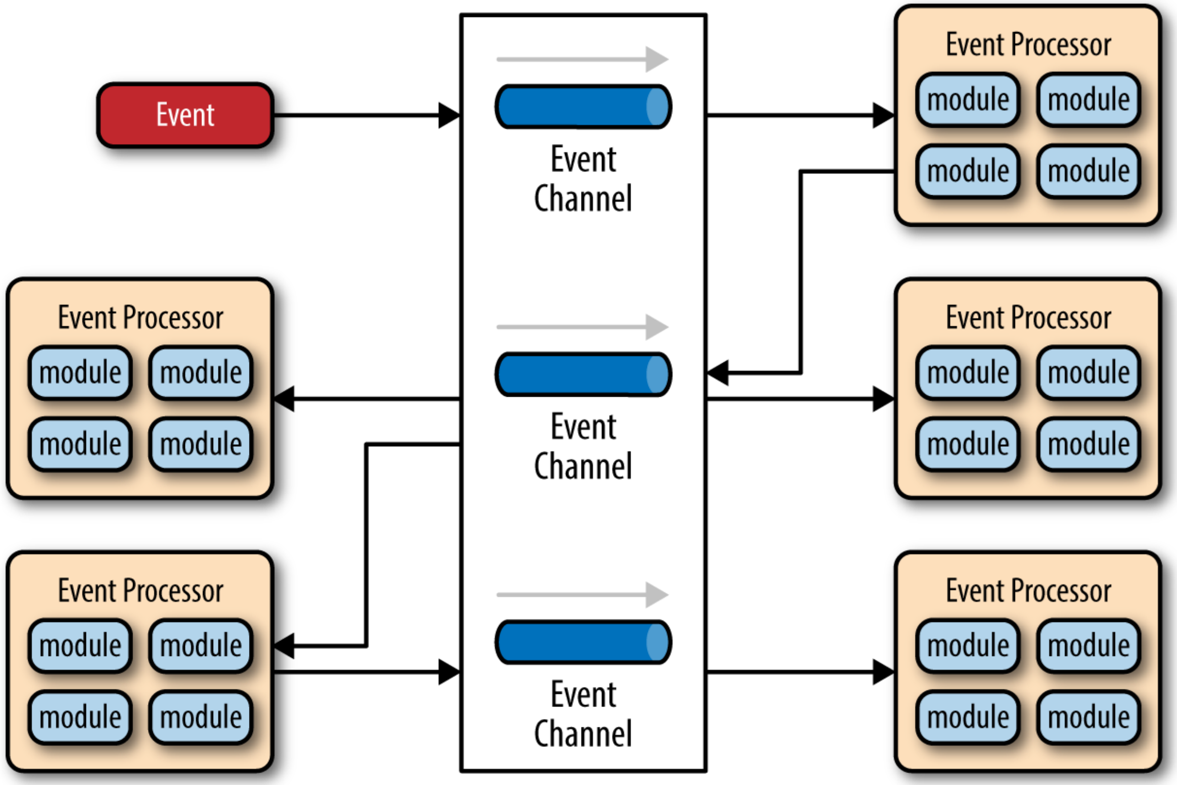
Chaos Monkey randomly terminates virtual machine instances and containers that run inside of your production environment. Exposing engineers to failures more frequently incentivizes them to build resilient services.

See the documentation for info on how to use Chaos Monkey.

Chaos Monkey is an example of a tool that follows the Principles of Chaos Engineering.

REDUCED CAPABILITY

EVENT DRIVEN ARCHITECTURE



SELF-TESTING SYSTEMS

- Power on Built in Test (PBIT)
- Initiated Built In Test (IBIT)
- Continuous Built In Test (CBIT)

WAR STORIES

EXCEPT GARBAGE

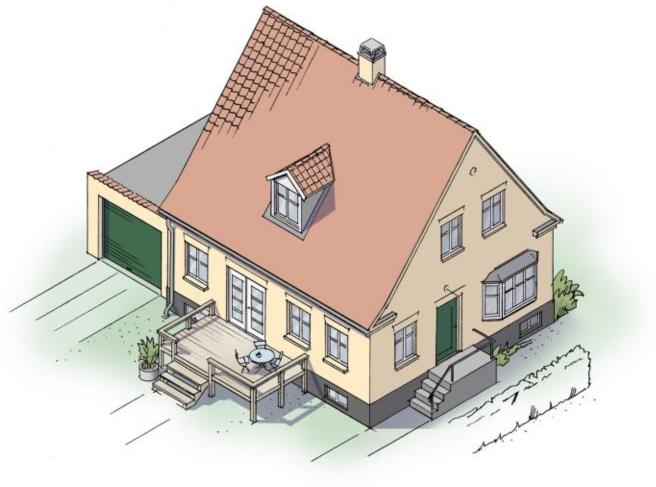
- When you interface to other systems or users, expect that you get garbage.
  - Handle it by sanitizing your input

Example:

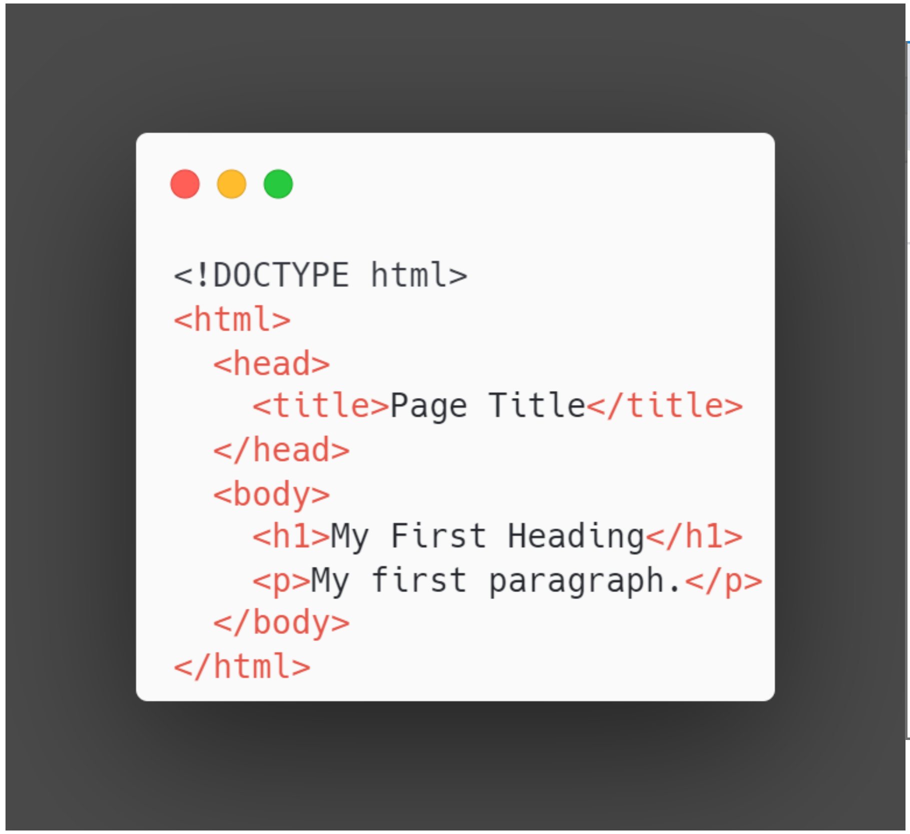
Interfacing to the BBR register (Bygnings og Bolig-registeret)

Missing data in entries

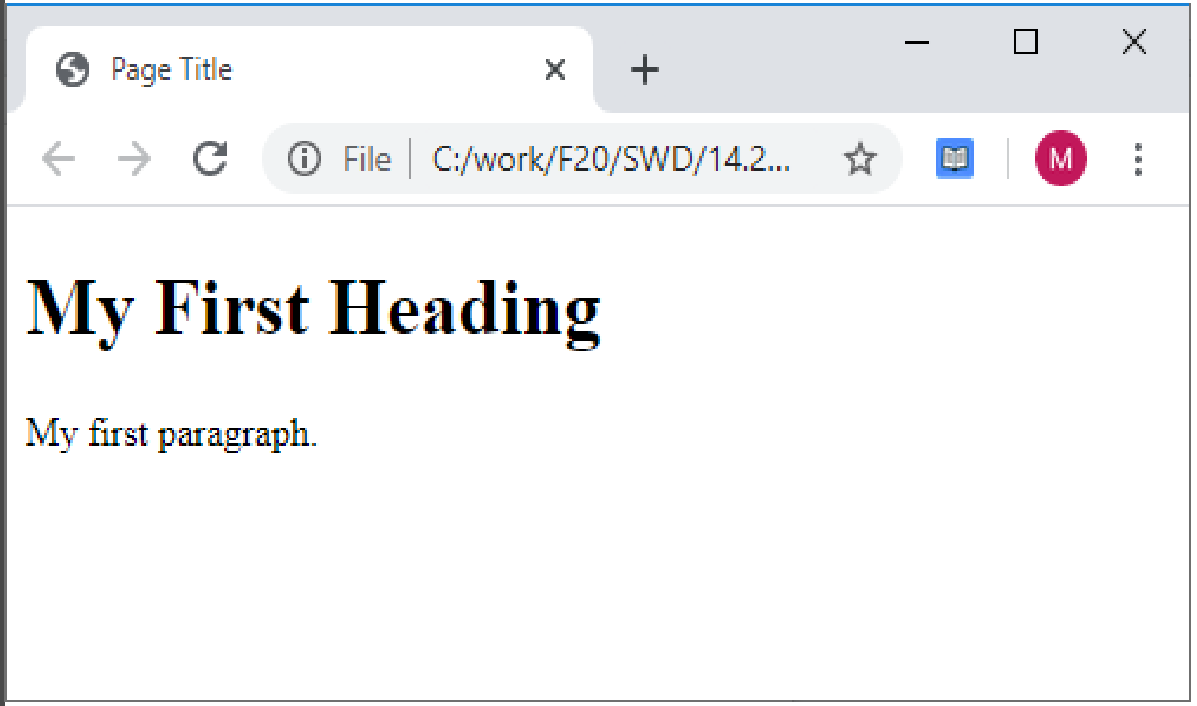
Invalid data in entries, e.g. postcode: 9999 Andeby (Ducktown)



HTML GARBAGE



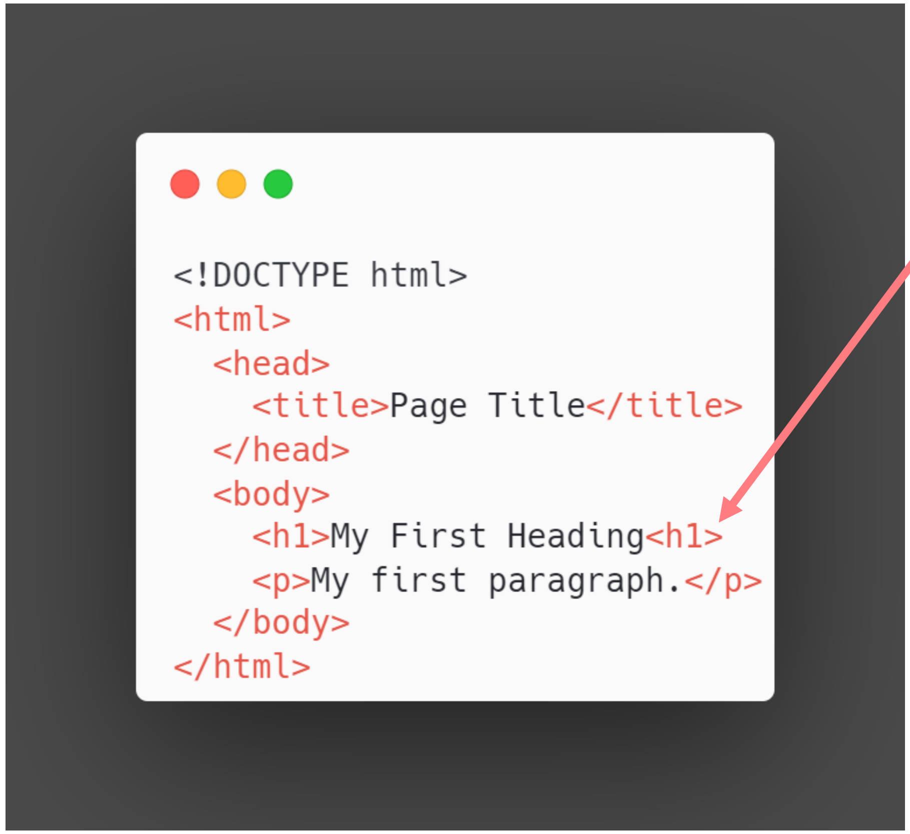
```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>My First Heading</h1>
    <p>My first paragraph.</p>
  </body>
</html>
```



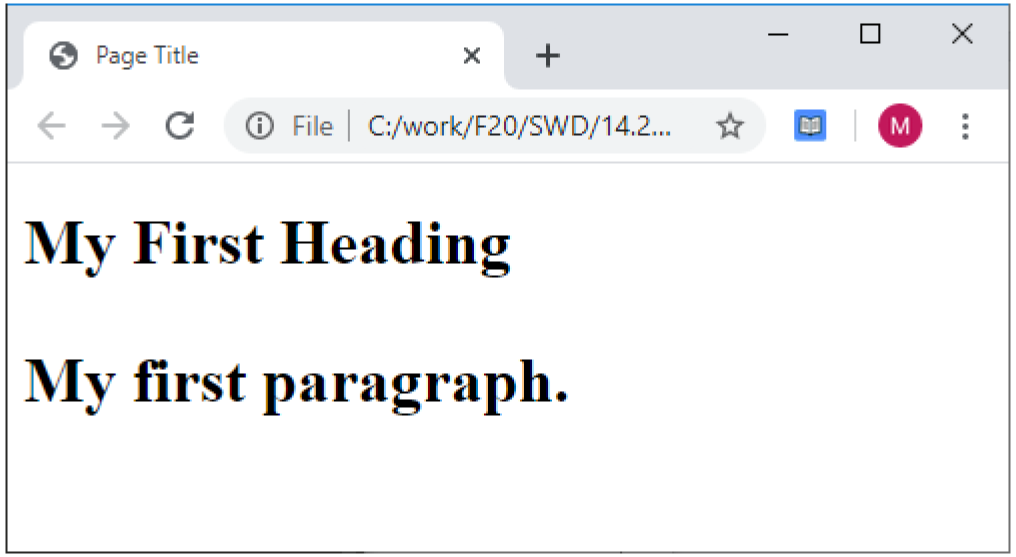
My First Heading

My first paragraph.

MORE HTML GARBAGE



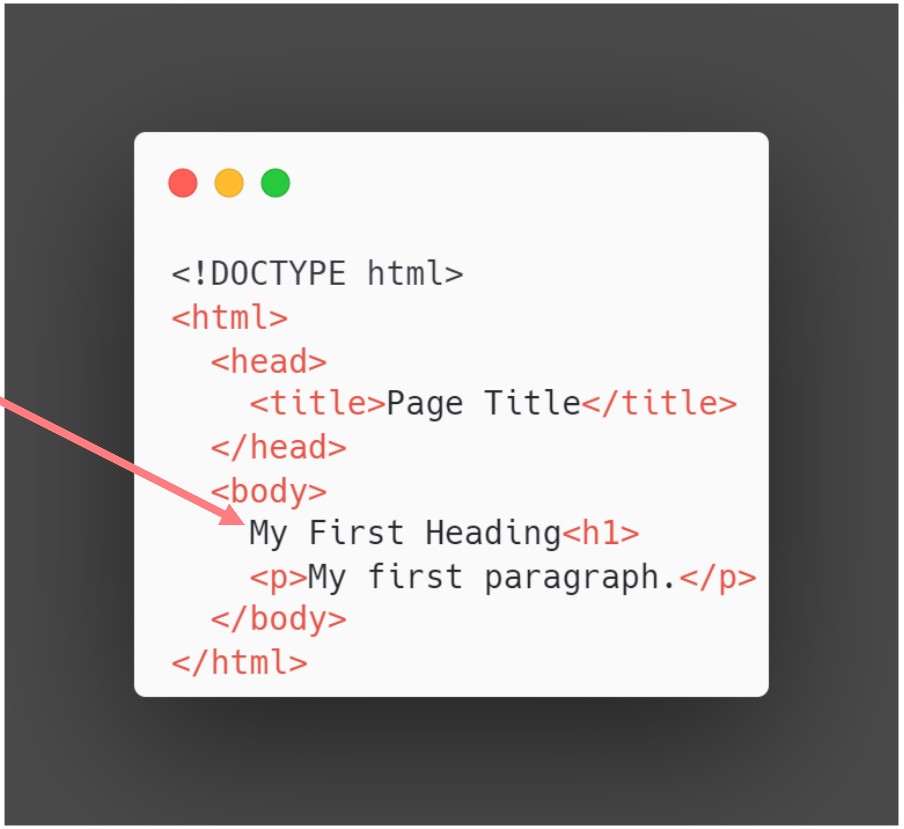
```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>My First Heading<h1>
    <p>My first paragraph.</p>
  </body>
</html>
```



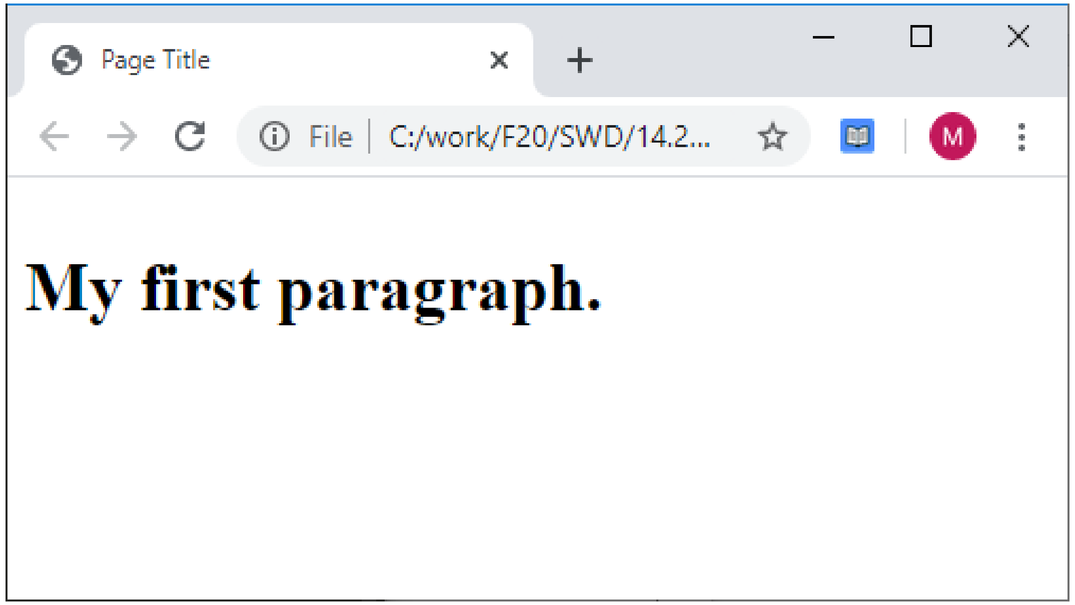
My First Heading

My first paragraph.

MORE HTML GARBAGE

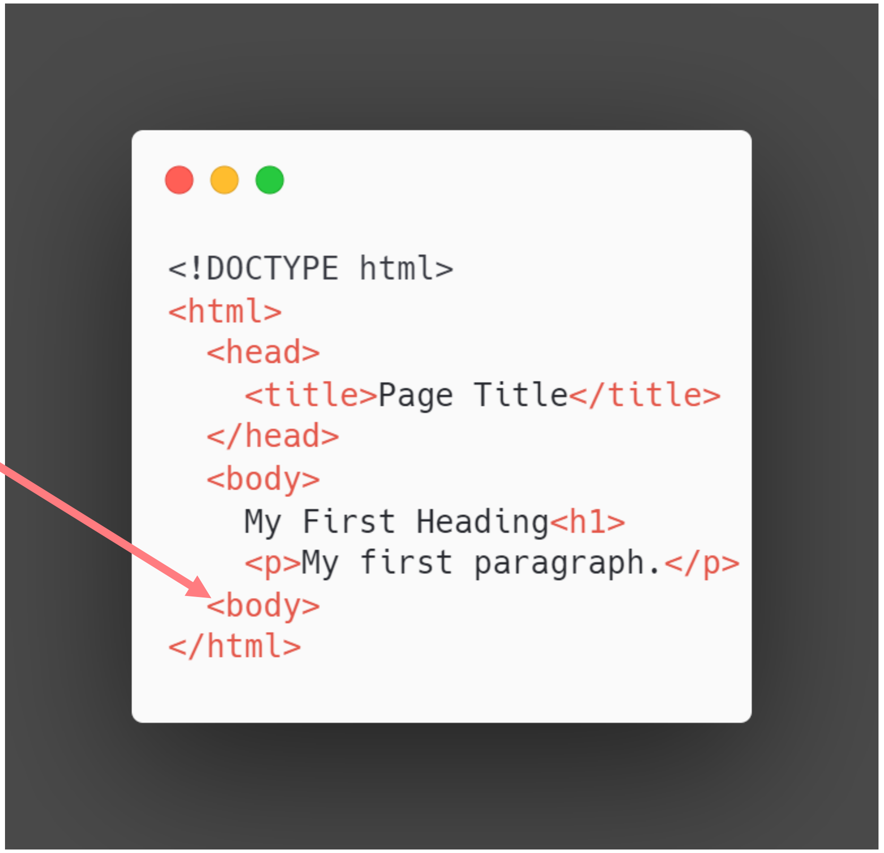


```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    My First Heading<h1>
    <p>My first paragraph.</p>
  </body>
</html>
```

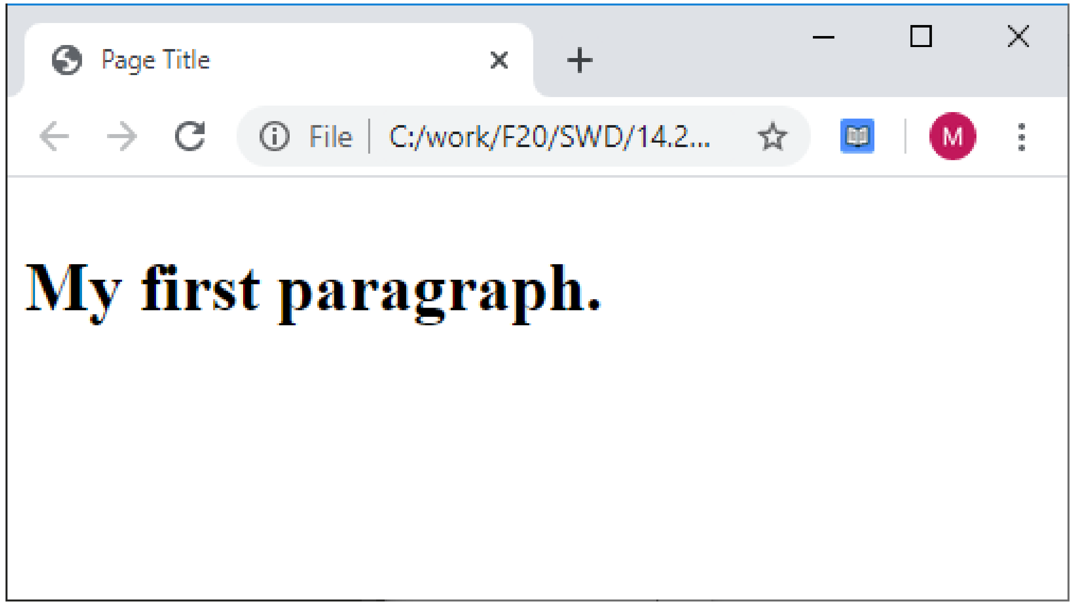


My first paragraph.

MORE HTML GARBAGE

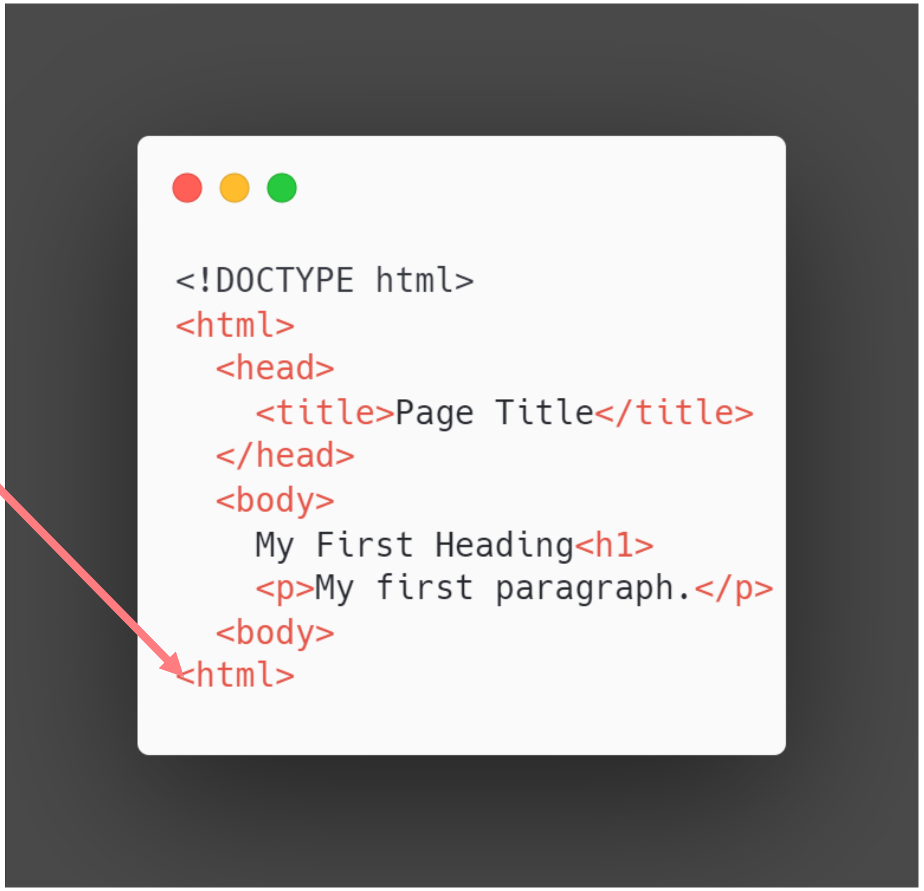


```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    My First Heading<h1>
    <p>My first paragraph.</p>
  <body>
</html>
```

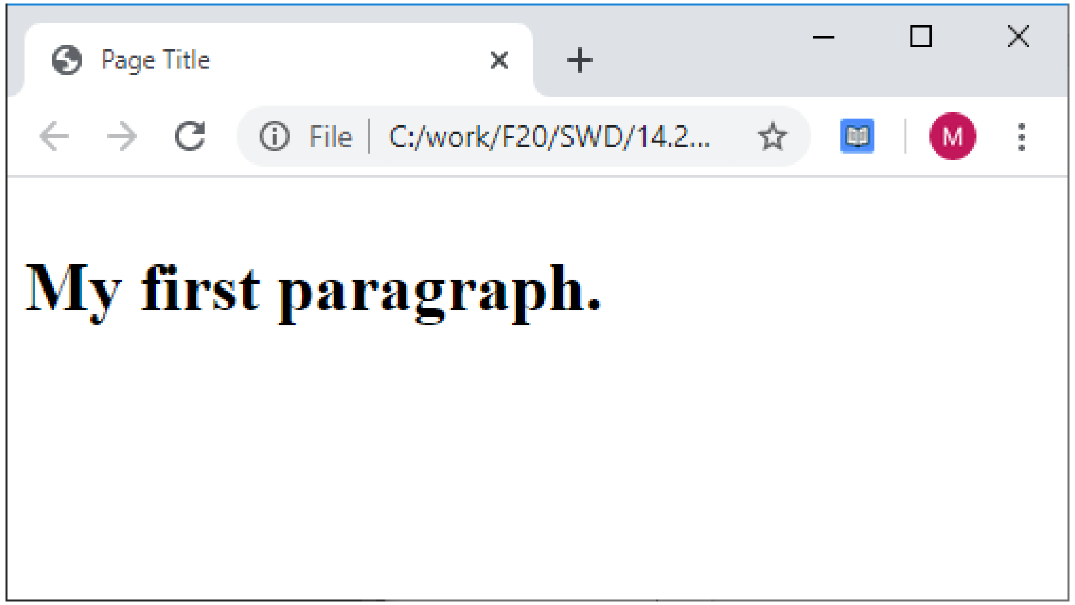


My first paragraph.

MORE HTML GARBAGE

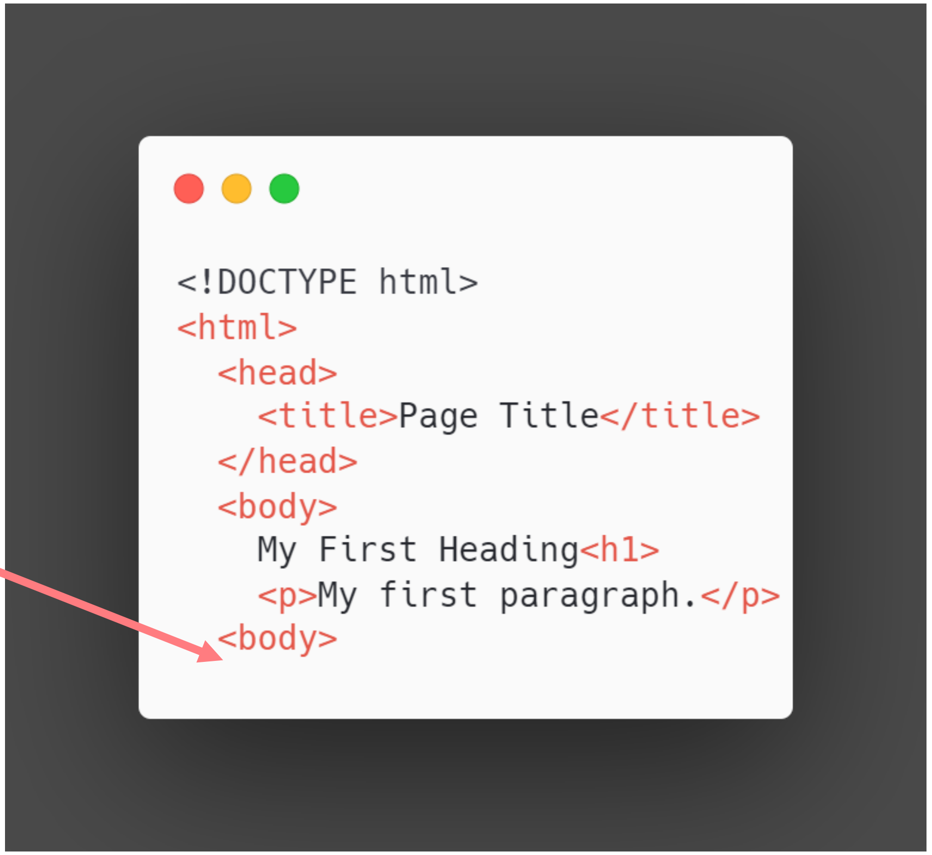


```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    My First Heading<h1>
    <p>My first paragraph.</p>
  <body>
<html>
```

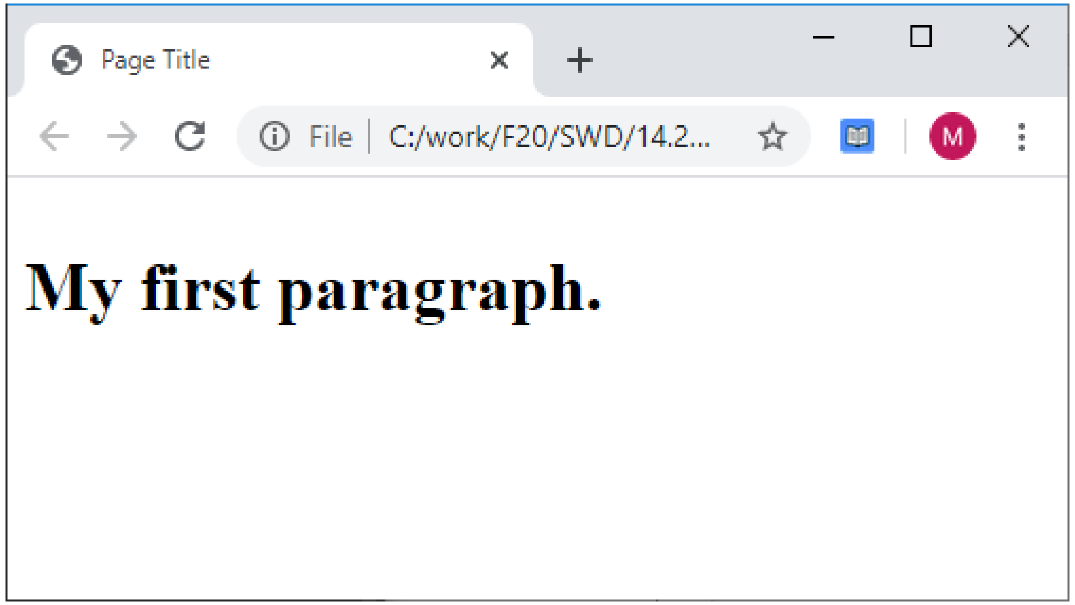


My first paragraph.

MORE HTML GARBAGE

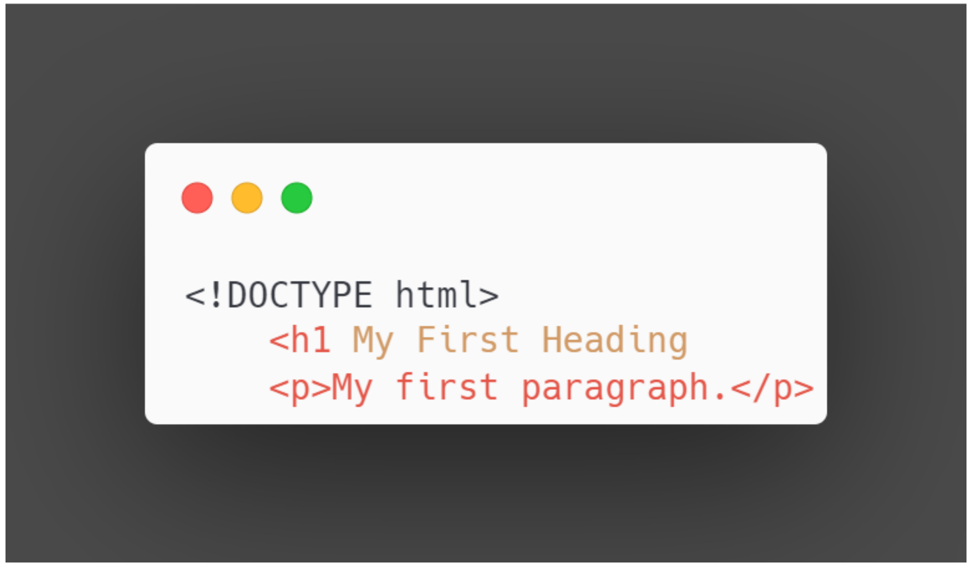


```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    My First Heading<h1>
    <p>My first paragraph.</p>
  <body>
```

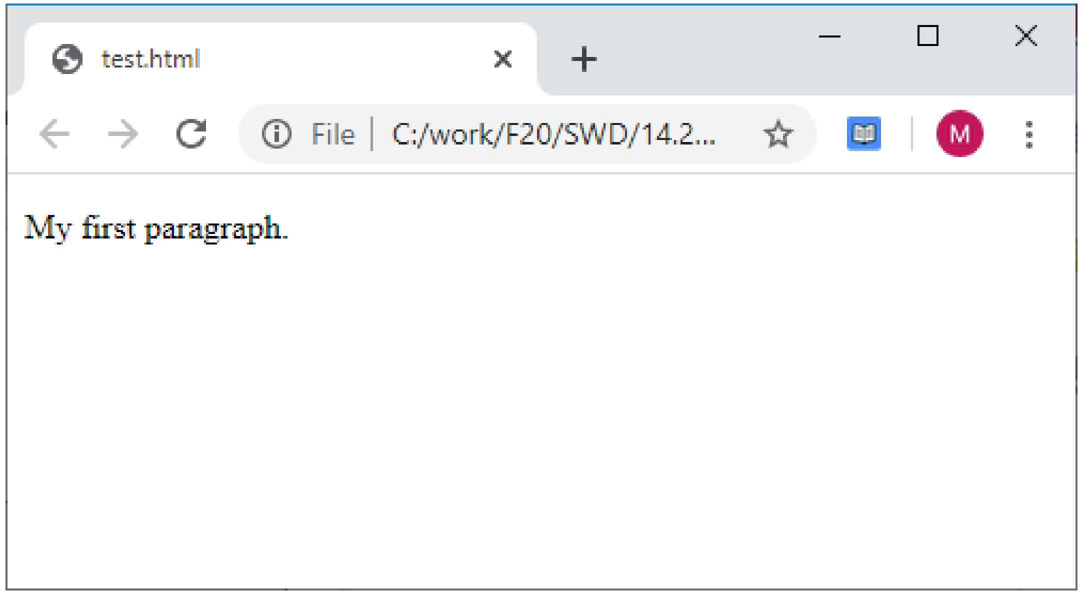


My first paragraph.

MORE HTML GARBAGE

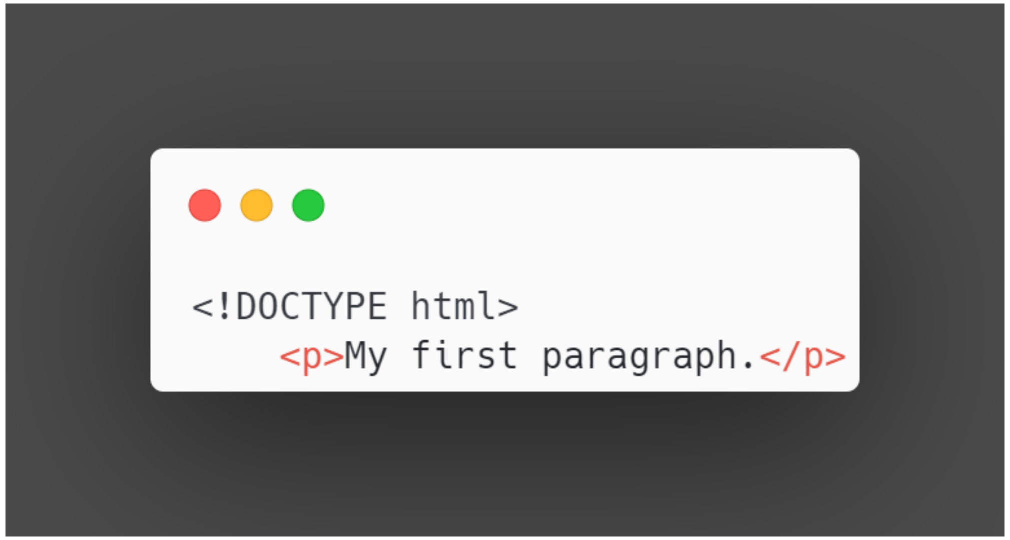


```
<!DOCTYPE html>
    <h1 My First Heading
    <p>My first paragraph.</p>
```

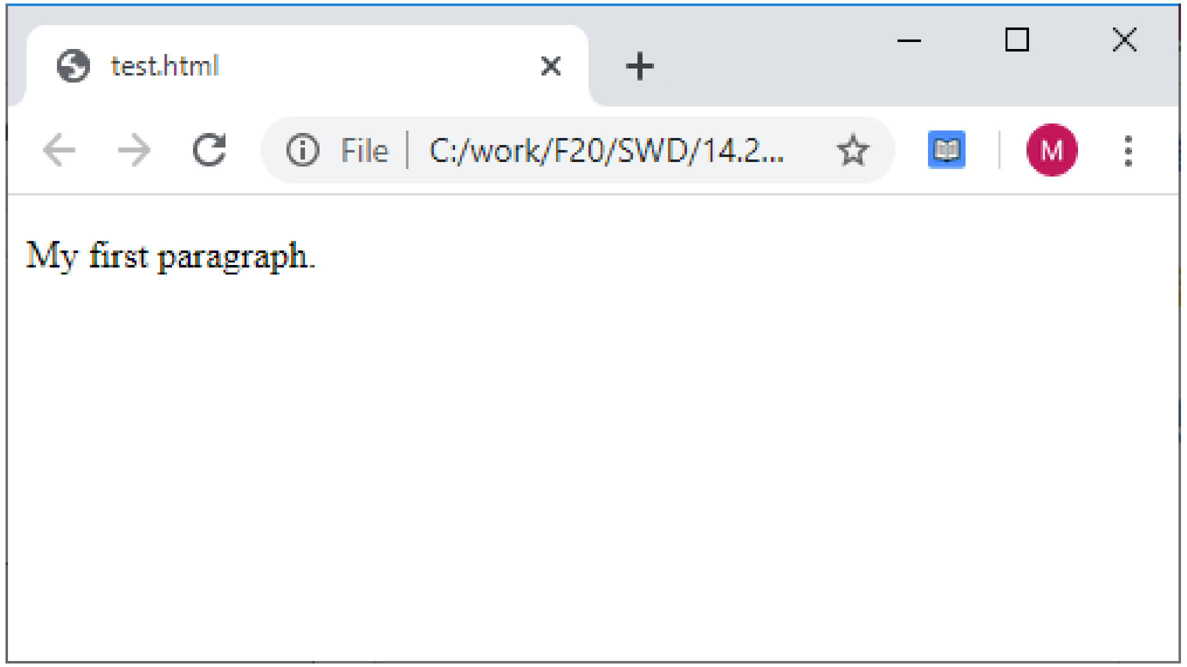


My first paragraph.

MORE HTML GARBAGE

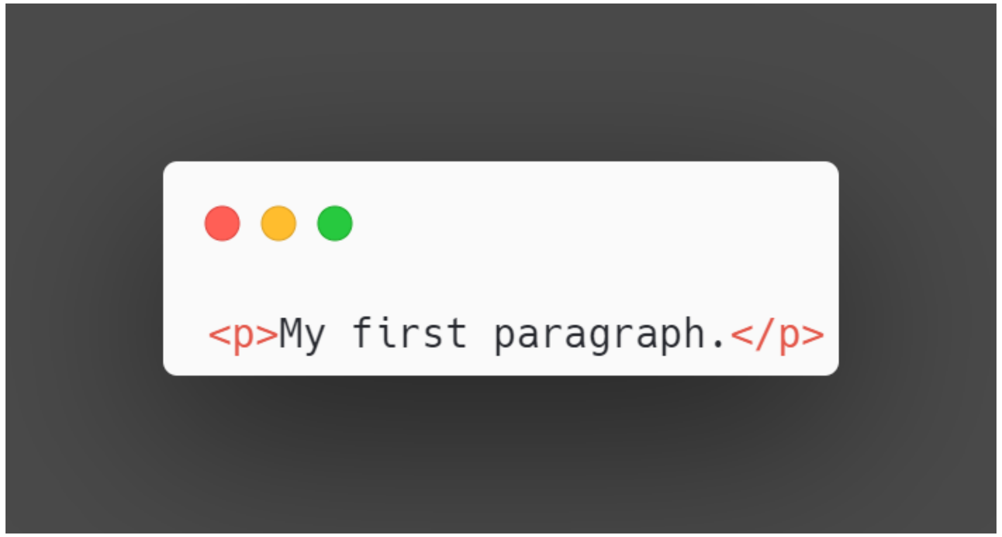


```
<!DOCTYPE html>
    <p>My first paragraph.</p>
```

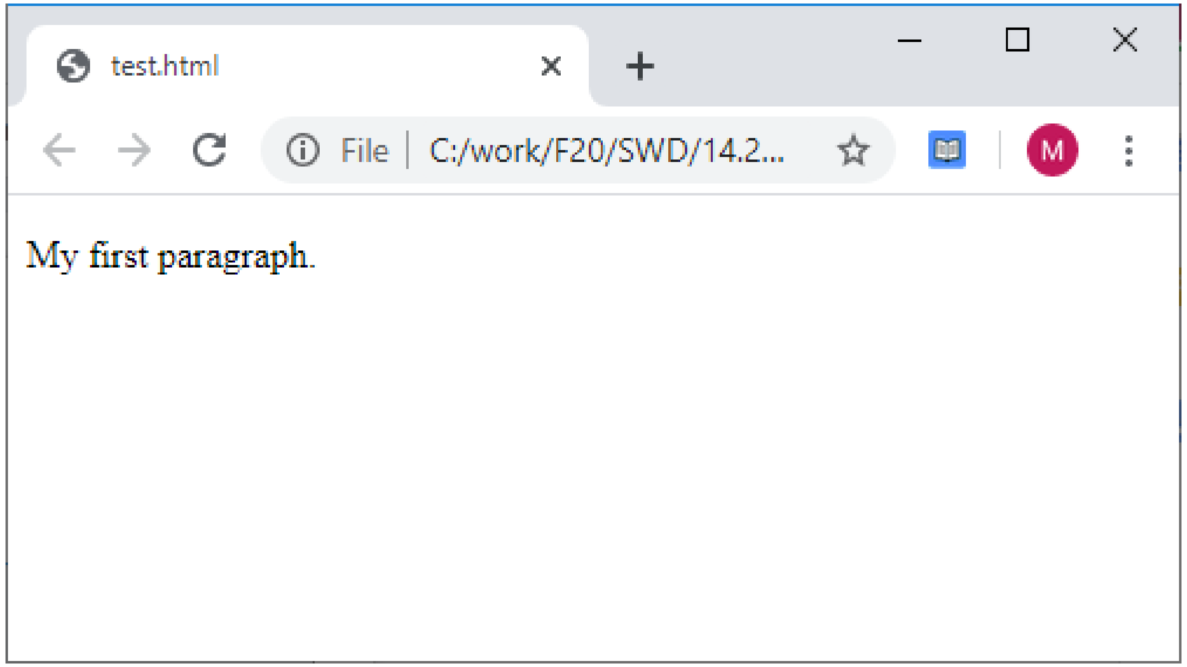


My first paragraph.

MORE HTML GARBAGE



```
<p>My first paragraph.</p>
```



My first paragraph.

AARHUS UNIVERSITY