C# Essentials - Fouten

Inhoudsopgave

1. Soorten fouten

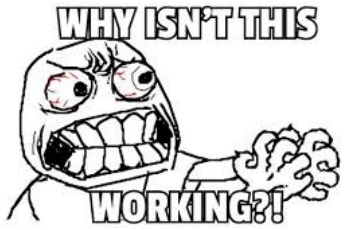
1.1. Syntaxisfouten

Syntaxisfouten zijn fouten tegen de grammatica van de programmeertaal. Je vergeet bijvoorbeeld een haakje of je schrijft een commando verkeerd. Visual Studio zal deze fouten meestal onmiddellijk aanduiden met rode golflijntjes onder de code.

```
int age = 25
Console.WriteLine(age);
```

Fout: er ontbreekt een puntkomma op het einde van de eerste regel.

1.2. Logische fouten (bugs)



Een logische fout betekent dat je programma **wel correct uitvoert**, maar niet doet wat jij bedoelde. Er komt dus geen foutmelding, maar het resultaat is fout.

Deze fouten zijn vaak het moeilijkst te vinden. Ze zitten in je manier van denken of de manier waarop je je code hebt opgebouwd.

1.2.1. Voorbeeld 1

```
int total = 100;
int percentage = 25;
int result = total * percentage / 1000;
Console.WriteLine(result); // Output: 2
```

Bedoeling was om 25% van 100 te berekenen, dus `25` als resultaat. Door een foutje in de deling (`/ 1000` in plaats van `/ 100`), krijg je 2 als resultaat. De code compileert en runt perfect, maar het antwoord is fout.

1.2.2. Voorbeeld 2

```
int score = 85;

if (score > 50)
{
    Console.WriteLine("Je bent niet geslaagd.");
}
else
{
```

```
    Console.WriteLine("Proficiat, je bent geslaagd!");
}
```

In deze code krijgt een student met 85 punten de boodschap "Je bent niet geslaagd." Dat klopt natuurlijk niet. De programmeur heeft per ongeluk de logica van het `if`-statement omgedraaid.

Het programma compileert én runt perfect, maar het gedrag is fout — typisch voor een logische fout.

1.3. Runtime-fouten

Een runtime-fout treedt op **tijdens** het uitvoeren van het programma. Alles ziet er op het eerste zicht goed uit, maar zodra je het programma draait, crasht het.

Deze fouten kunnen je programma abrupt stoppen als je ze niet op de juiste manier opvangt. Daarom leer je later in deze module werken met `try` en `catch`.

```
string text = "abc";
int number = int.Parse(text);
```

Fout: de tekst `"abc"` kan niet omgezet worden naar een getal. Dit veroorzaakt een `FormatException`.

1.4. Samenvatting

Type fout	Wanneer gebeurt het?	Voorbeeld	Gevolg
Syntaxisfout	Tijdens het schrijven	Vergeten puntkomma	Code compileert niet
Logische fout	Tijdens het uitvoeren	logica van het `if`-statement omgedraaid	Verkeerd resultaat
Runtime-fout	Tijdens het uitvoeren	Ongeldige conversie of deling door 0	Programma crasht

Soms zie je geen foutmelding, maar werkt je programma toch niet correct. In dat geval is er vaak sprake van een logische fout. Gebruik dan debugging (zie volgend hoofdstuk) om stap voor stap na te gaan wat er precies gebeurt.

2. Debugging

2.1. Wat is debuggen?

Debuggen is het proces waarbij je stap voor stap je programma uitvoert om te begrijpen wat er precies gebeurt. Je kan pauzeren op bepaalde regels en nagaan welke waardes de variabelen op dat moment hebben.

2.2. Een breakpoint instellen

Een **breakpoint** is een marker die aangeeft waar Visual Studio je code moet pauzeren tijdens het uitvoeren.

Breakpoints plaatsen kan op 3 manieren:

- Klik links van het regelnummertje in Visual Studio op de grijze balk.
- Selecteer een regel code en druk op `F9`.
- Ga naar het menu `Debug > Toggle Breakpoint`.

Wanneer je programma die regel bereikt, stopt het tijdelijk en krijg je de kans om alles van dichtbij te bekijken.

[toggle breakpoint] | *toggle-breakpoint.gif*

2.2.1. Programma stap voor stap uitvoeren

Zodra je een breakpoint hebt geplaatst, kan je je programma starten in debugmodus (`F5`). Van zodra de breakpoint wordt bereikt, pauzeert het programma en verschijnt er een extra debugbalk bovenaan je scherm.

De belangrijkste knoppen:

Sneltoets	Actie
F11	**Step Into** – Ga naar de volgende regel. Als het een methode is, spring je die methode binnen.
F10	**Step Over** – Ga naar de volgende regel, maar voer een eventuele methode volledig uit zonder er dieper op in te gaan.
Shift + F11	**Step Out** – Spring uit de huidige methode en ga terug naar waar ze werd opgeroepen.
Ctrl + F10	**Run To Cursor** – Voer alles uit tot aan de positie van je cursor.

2.2.2. Voorbeeld

```
TODO: code voorbeeld of labo met verschillende methods om in te duiken
```

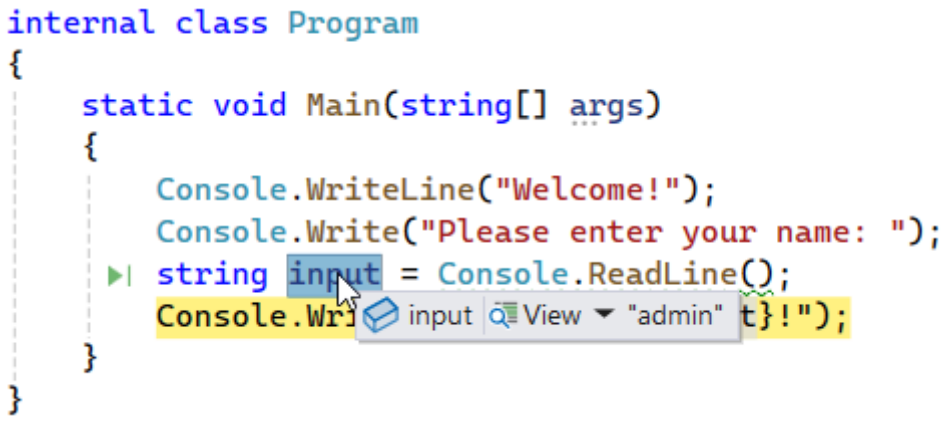
Plaats een breakpoint op de regel `...`. Als je daarna `F11` gebruikt, spring je de methode binnen. Met `F10` voer je ze gewoon uit zonder erin te duiken.

2.3. Waarden bekijken tijdens het debuggen

In dit onderdeel leer je hoe je variabelen en expressies kan bekijken tijdens het debuggen in Visual Studio. Zo zie je wat er écht gebeurt in je programma.

2.3.1. Hoveren boven variabelen

De eenvoudigste manier om een waarde te bekijken is door met je muis over een variabele te bewegen terwijl je programma gepauzeerd is. Visual Studio toont dan automatisch de huidige waarde in een pop-up.

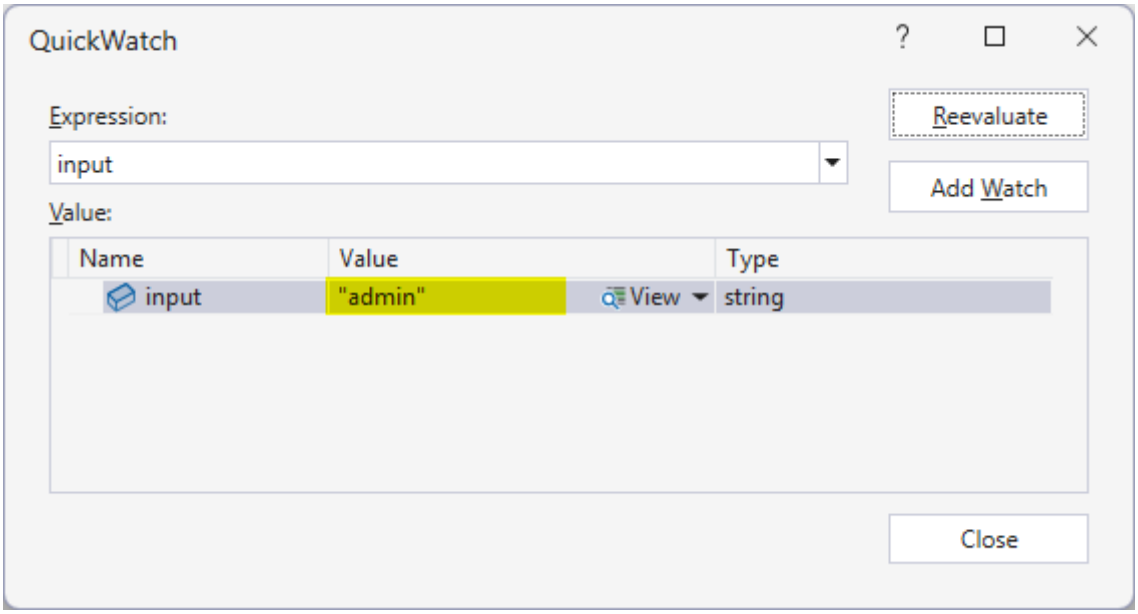


Bij objecten kan je doorklikken op het pijltje om alle eigenschappen en hun waardes te bekijken.

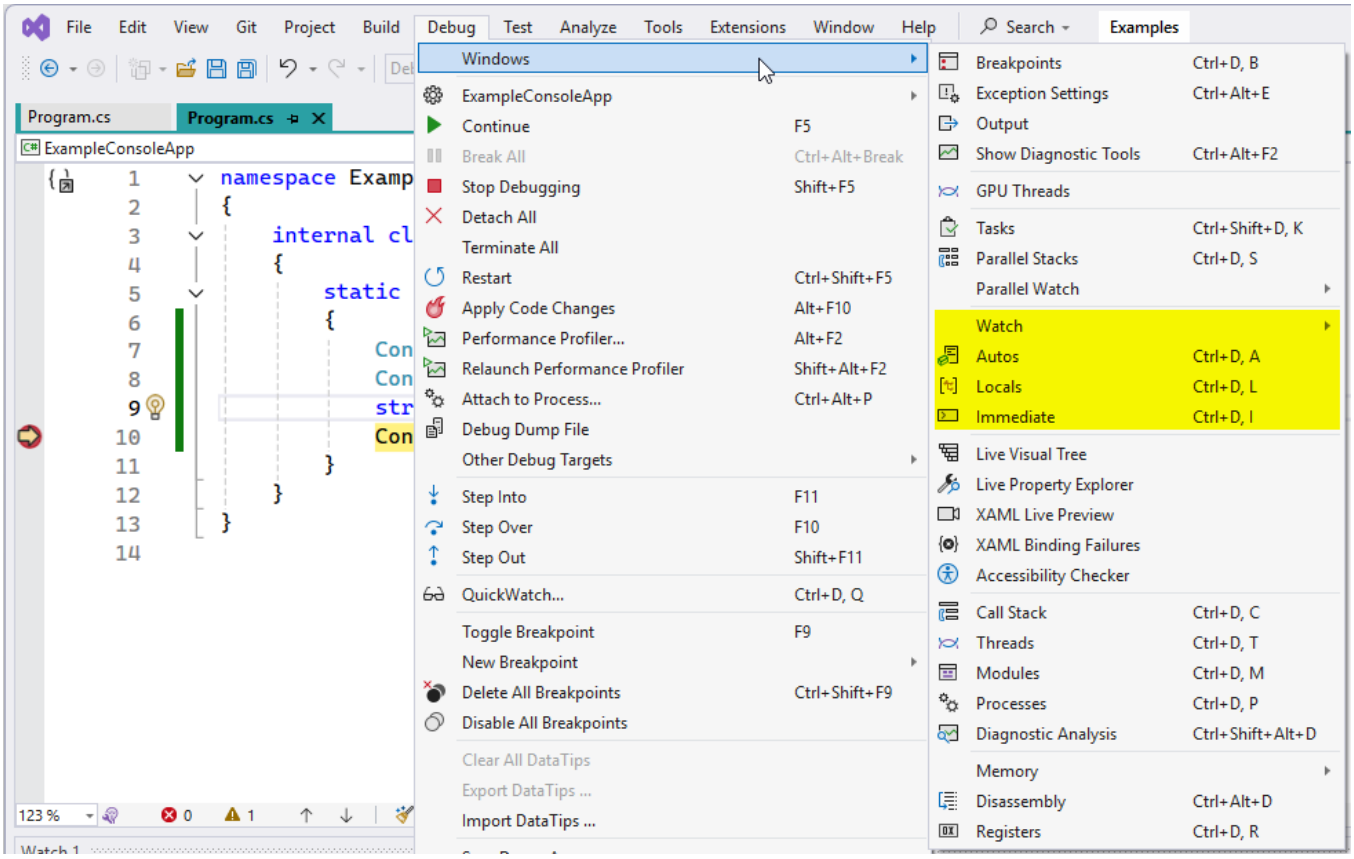
2.3.2. QuickWatch-venster

wil je de waarde van een bepaalde variabele eenmalig onderzoeken? Dan kan je ook het QuickWatch-venster gebruiken.

- Klik met de rechtermuisknop op de variabele
- Selecteer QuickWatch



2.4. Debug-vensters



2.4.1. Watch-venster

Wil je bepaalde variabelen of expressies blijvend opvolgen tijdens het debuggen? Gebruik dan het Watch-venster.

- Open het venster via `Debug > Windows > Watch > Watch 1`
- Typ de naam van een variabele of een volledige expressie, zoals `total + 1`
- Je ziet direct de huidige waarde en je kan deze ook aanpassen
- Indien de waarde wijzigt, wordt deze in het rood weergegeven

Je kan meerdere Watch-vensters tegelijk gebruiken (Watch 1, Watch 2, …).

2.4.2. Autos-venster

Het Autos-venster toont automatisch:

- De variabelen uit het huidige statement
- De variabelen uit het vorige statement

Handig als je snel wil zien wat er net veranderd is. Je opent dit venster via `Debug > Windows > Autos`.

2.4.3. Locals-venster

Wil je een overzicht van **alle lokale variabelen** in de huidige methode? Gebruik dan het Locals-venster:

- Ga naar `Debug > Windows > Locals`
- Toont alle variabelen die je in de methode gebruikt, met hun huidige waarde

2.4.4. Immediate-venster

Wil je iets testen of aanpassen tijdens het debuggen? Dan is het Immediate-venster ideaal.

- Open het venster via `Debug > Windows > Immediate`
- Typ een variabele om de waarde op te vragen
- Je kan ook opdrachten uitvoeren of waardes aanpassen

2.4.5. Call Stack-venster

Het Call Stack-venster toont de volledige oproepketen van methodes die geleid heeft tot het huidige punt in je code. Zo weet je wie wat heeft opgeroepen. Handig bij fouten in diep geneste methodes of bij recursie.

- Open via `Debug > Windows > Call Stack`
- Klik op een methode in de stack om de bijhorende code te tonen

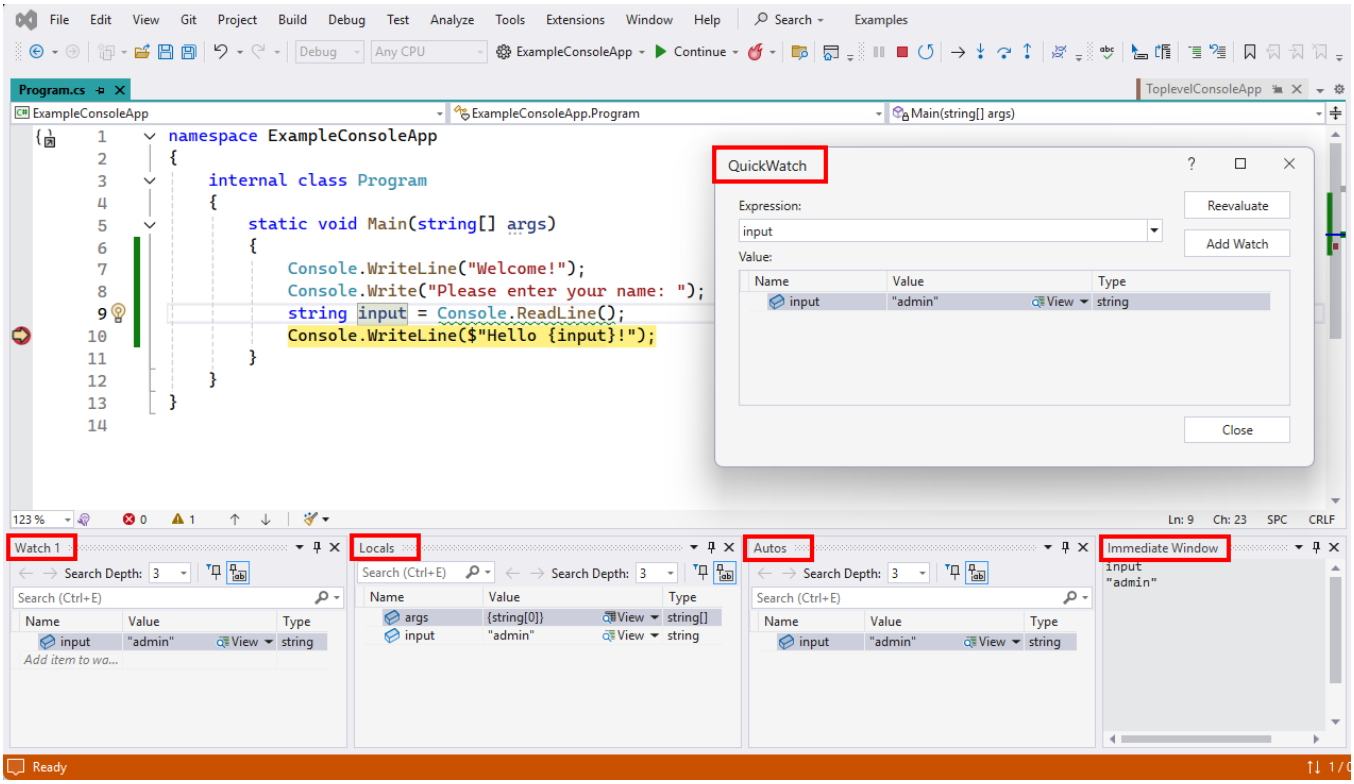
2.5. Oefening

- Maak een nieuwe console-applicatie en kopieer onderstaande code naar de `Main`-methode

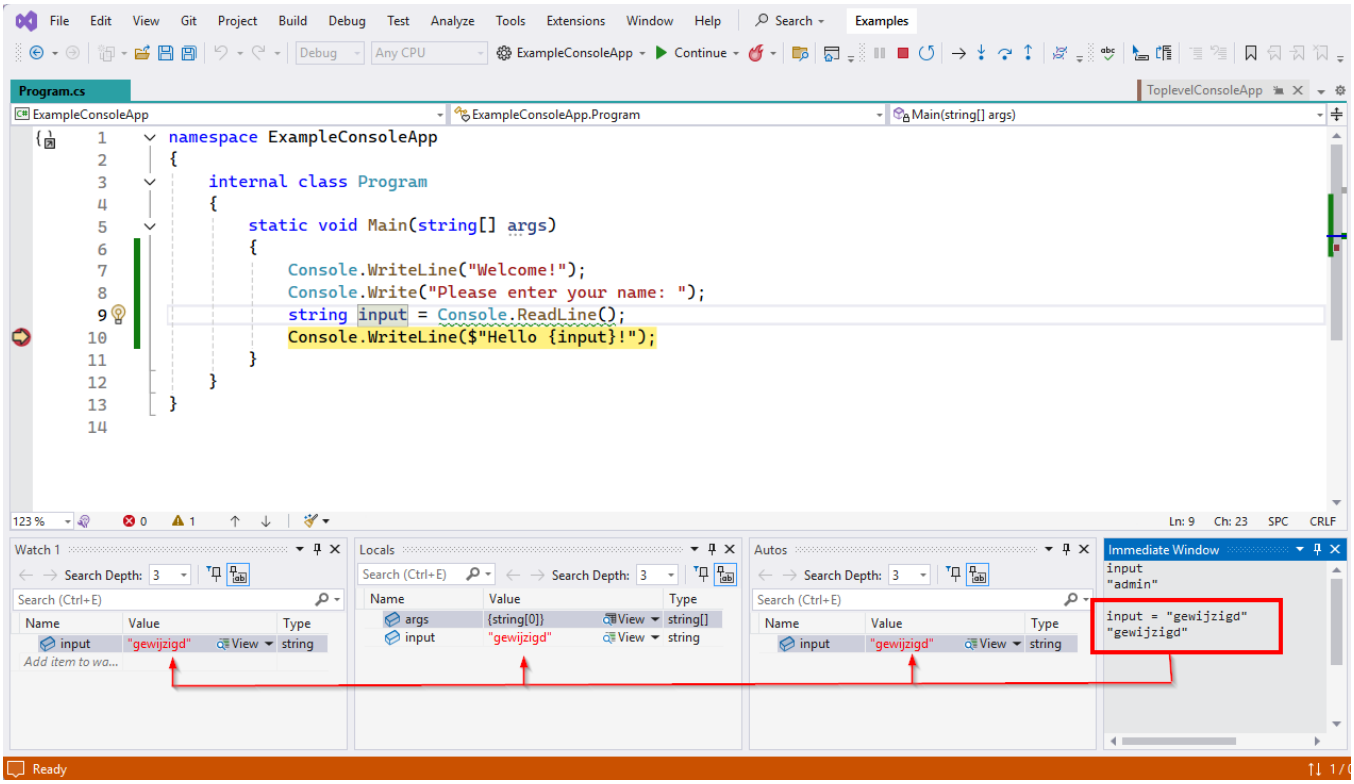
```
Console.WriteLine("Welcome!");
Console.Write("Please enter your name: ");
string input = Console.ReadLine();
Console.WriteLine($"Hello {input}!");
```

- Plaats een **breakpoint** op de laatste regel
- Bekijk de waarde van de `input`-variabele door:
  - met je muis over de variabele te **hoveren**

    - de variabele te openen in het **QuickWatch**-venster
    - de variabele toe te voegen aan het **Watch**-venster
- Onderzoek het **Locals**- en **Autos**-venster



- Wijzig de waarde van de variabele via het ***Immediate***-venster



2.6. Samenvatting

- Debuggen betekent: stap voor stap je code doorlopen om fouten op te sporen.

- Met breakpoints pauzeer je je programma op een specifieke regel.
- De debugbalk laat je toe om regels één voor één uit te voeren en methodes binnen te gaan of over te slaan.
- Je kan de waarden van variabelen of methodes bekijken tijdens het debuggen.

3. Exceptions

3.1. Wat is een exception?

Een exception is een fout die optreedt tijdens het uitvoeren van je programma. Als je deze fout niet opvangt, zal je programma crashen.

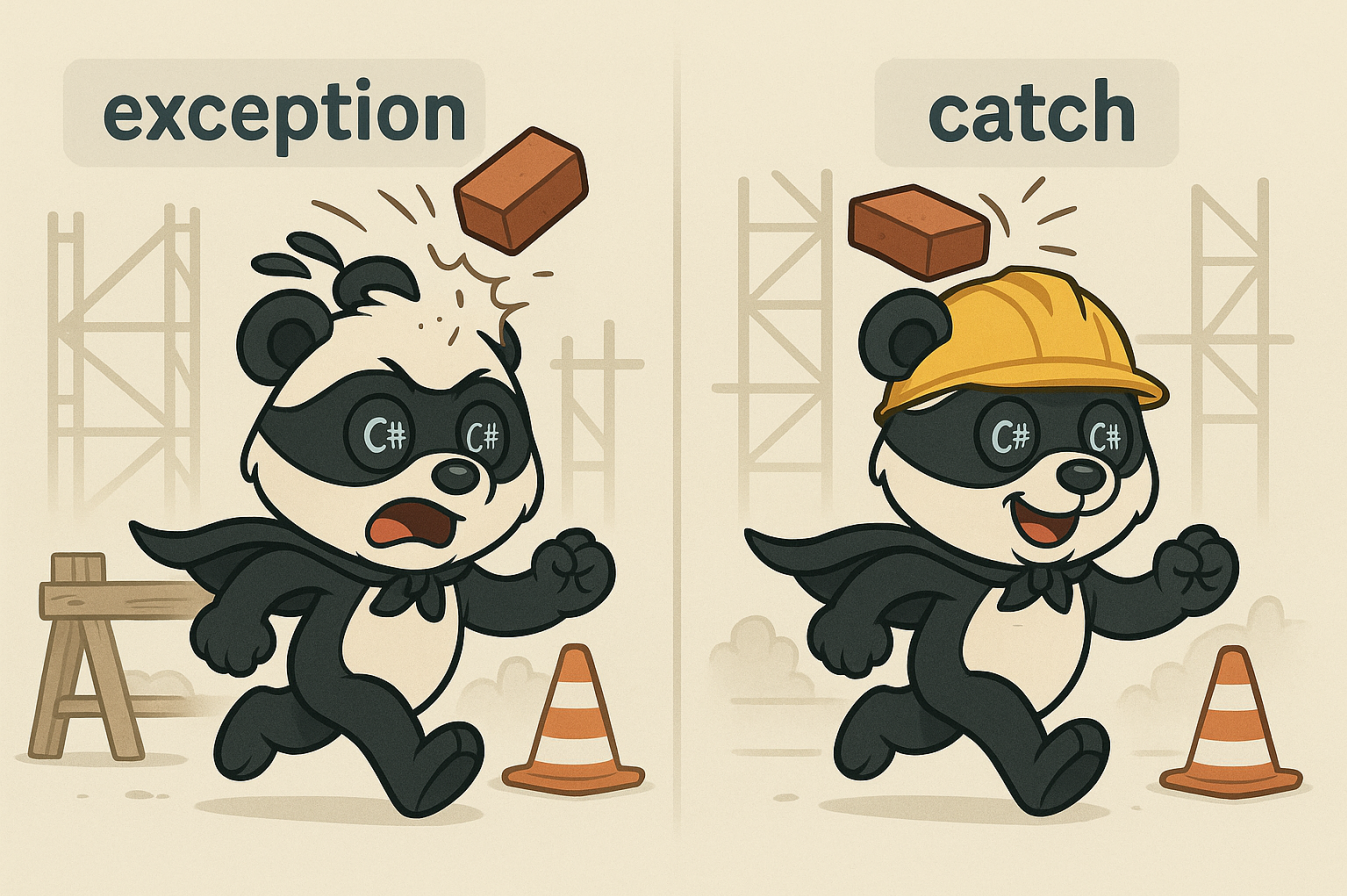
Typische voorbeelden:

- Je probeert te delen door nul
- Je probeert een string om te zetten naar een getal, maar de inhoud is ongeldig
- Je probeert een bestand te openen dat niet bestaat

3.2. Fouten afhandelen met try-catch

Je kan een fout proberen op te vangen met een `try`–`catch`-blok:

```
try
{
    string text = "vijf";
    int number = int.Parse(text); // dit veroorzaakt een fout
}
catch (Exception ex)
{
    Console.WriteLine("Er is een fout opgetreden!");
    Console.WriteLine(ex.Message); // toont de technische foutmelding
}
```



3.2.1. finally

Je kan ook een `finally`-blok toevoegen. De code hierin wordt **altijd** uitgevoerd — of er nu een fout is opgetreden of niet.

```
try
{
    Console.WriteLine("Hier kan iets fout gaan.");
}
catch (Exception)
{
    Console.WriteLine("Er is iets misgelopen.");
}
finally
{
    Console.WriteLine("Deze boodschap wordt altijd getoond.");
}
```

3.2.2. Meerdere catch-blokken

Je kan meerdere soorten fouten opvangen in aparte `catch`-blokken. Zo geef je een meer gerichte foutmelding.

```
try
{
    int numerator = int.Parse(Console.ReadLine());
```

```
    int denominator = int.Parse(Console.ReadLine());
    int result = numerator / denominator;
}
catch (DivideByZeroException)
{
    Console.WriteLine("Je mag niet delen door nul!");
}
catch (FormatException)
{
    Console.WriteLine("Gelieve enkel gehele getallen in te vullen.");
}
catch (Exception ex)
{
    Console.WriteLine($"Onverwachte fout: {ex.Message}");
}
```

Zet altijd de meest specifieke fout **eerst**, en de algemene `Exception` pas als laatste.

3.3. Veelvoorkomende exceptions

In de praktijk kom je vaak dezelfde soorten fouten tegen. Hier is een handig overzicht:

Exception	Wanneer treedt dit op?
`DivideByZeroException`	Je probeert een getal te delen door nul.
`FormatException`	Je probeert een string om te zetten naar een getal, maar de inhoud is ongeldig.
`IndexOutOfRangeException`	Je probeert een index te gebruiken buiten de grenzen van een array of lijst.
`NullReferenceException`	Je probeert een object te gebruiken dat `null` is.
`InvalidOperationException`	Een bewerking is ongeldig in de huidige toestand van het object (bv. enumerator na einde van een lijst).
`ArgumentException`	Een methode kreeg een ongeldig argument (bv. negatieve lengte).
`OverflowException`	Een numerieke bewerking resulteert in een waarde die buiten het bereik van het datatype valt.

Let altijd goed op de exacte foutmelding die Visual Studio geeft. Die bevat vaak de naam van de exception én een korte uitleg.

3.4. Zelf een exception gooien met throw

Je kan ook zelf bewust een exception veroorzaken met het `throw`-keyword. Dat is handig als je foutieve invoer wil detecteren.

```
private static int WordToNumber(string word)
{
    if (word == "ten")
        return 10;
    else if (word == "hundred")
        return 100;
    else
        throw new FormatException("Ongeldige invoer: " + word);
}
```

Gebruik `throw` enkel wanneer je de fout niet zelf kan oplossen, maar het wil melden aan de rest van het programma.

3.5. Oefening

3.5.1. Leeftijd berekenen

Schrijf een consoletoepassing die de gebruiker vraagt om zijn geboortedatum in te geven. Het programma berekent en toont daarna de leeftijd.

```
Console.Write("Geef je geboortedatum in (yyyy-MM-dd): ");
string input = Console.ReadLine();

DateTime birthDate = DateTime.Parse(input);
int age = DateTime.Now.Year - birthDate.Year;

Console.WriteLine($"Je bent ongeveer {age} jaar oud.");
```

Opdracht:

- Voeg foutafhandeling toe zodat het programma niet crasht bij:
  - ongeldige invoer (bv. “hallo”)
  - een datum in de toekomst
- Toon in elk geval een gepaste foutmelding via `Console.WriteLine()`

3.5.2. Vierkantswortel berekenen

Schrijf een programma dat de gebruiker vraagt om een positief getal. Het programma toont de vierkantswortel ervan.

```
Console.Write("Geef een positief getal: ");
string input = Console.ReadLine();
```

```
double value = double.Parse(input);
double result = Math.Sqrt(value);

Console.WriteLine($"De vierkantswortel van {value} is {result}");
```

Opdracht:

- Voeg foutafhandeling toe zodat het programma niet crasht bij:
  - ongeldige invoer
  - negatieve getallen (gooi hier zelf een `ArgumentOutOfRangeException`)

4. Defensief programmeren

4.1. Wat is defensief programmeren?

Defensief programmeren betekent dat je vooraf nadenkt over wat er fout zou kunnen gaan en dit voorkomt via extra controles of veiligere methodes.

Voorbeelden van typische defensieve checks:

- Controleren of een deler nul is voordat je deelt
- Controleren of invoer geldig is voordat je ze omzet naar een getal
- `TryParse` gebruiken in plaats van `Parse`

4.2. if-checks om fouten te vermijden

Een klassiek voorbeeld: delen door nul. In plaats van de fout op te vangen met `try-catch`, controleer je dit vooraf.

```
int number = 10;
int divisor = 0;

if (divisor == 0)
{
    Console.WriteLine("Delen door nul is niet toegestaan.");
}
else
{
    int result = number / divisor;
    Console.WriteLine($"Resultaat: {result}");
}
```

4.3. TryParse: een veilige manier om strings om te zetten

Een andere veelgebruikte vorm van defensief programmeren is `TryParse`.

Gebruik je `int.Parse()` en de invoer is foutief, dan crasht je programma. Met `TryParse()` kan je dit vermijden:

```
Console.Write("Geef een geheel getal: ");
string input = Console.ReadLine();

bool isValid = int.TryParse(input, out int number);

if (isValid)
{
    Console.WriteLine($"Je gaf het getal {number} in.");
}
else
```

```
{
    Console.WriteLine("Dit is geen geldig geheel getal.");
}
```

4.3.1. Wat doet `TryParse()`?

- Controleert of de string een geldig getal bevat
- Geeft `true` of `false` terug
- Slaat het resultaat enkel op in de `out`-parameter als de conversie lukt
- Vermijdt een `FormatException` — je hoeft geen `try-catch` te gebruiken

4.4. Wanneer kies je voor defensieve checks?

Gebruik `if`-testen of `TryParse` wanneer je weet dat een fout **vaak voorkomt** of eenvoudig te voorspellen is. Gebruik `try-catch` alleen wanneer:

- de fout zeldzaam is
- je de fout niet op voorhand kan detecteren
- je externe afhankelijkheden hebt (bv. bestand, netwerk, invoer van gebruiker)

4.5. Oefening: veilig een getal vragen

Schrijf een programma dat aan de gebruiker een geheel getal vraagt. Gebruik `TryParse()` om de invoer correct te verwerken.

```
Console.Write("Geef een geheel getal: ");
string input = Console.ReadLine();

// Voeg hier TryParse toe om de invoer veilig om te zetten
```

Toon een foutmelding als de invoer niet geldig is. Toon anders het getal op het scherm.

4.6. Oefening

Vraag de gebruiker een geheel getal in te geven tussen 1 en 100.

- Controleer of de ingegeven waarde een geldig geheel getal is
- Controleer of de waarde tussen 1 en 100 ligt

```
Console.Write("Geef een geheel getal tussen 1 en 100: ");
string input = Console.ReadLine();

// Voeg hier defensieve checks toe en controleer het bereik
```

4.7. Samenvatting

Techniek	Gebruik bij…
`if`-check	Controleer op zaken als nul, negatief, leeg, foutieve toestand
`TryParse`	Veilig omzetten van strings, zonder exceptions
`try-catch`	Als je de fout niet kan vermijden of detecteren op voorhand