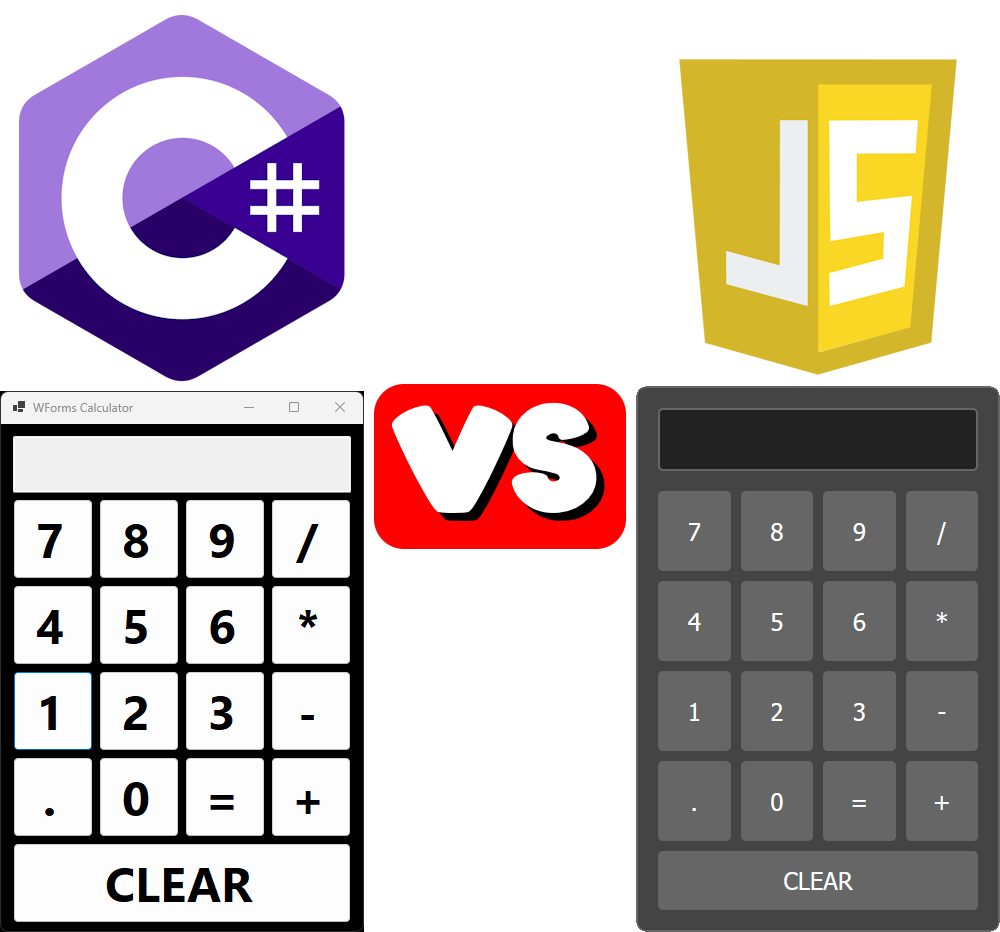
**C# VS JavaScript**

En jämförelse av konstruering, kodstruktur och exekvering av program inom C# och JavaScript.



*Skriven av Simon Lönn*

*27 Maj - 2 Aug*

[**0. Abstract 3**](#_rj7yjg776ivr)

[**1. Inledning 4**](#_o7g6se2lyv3r)

[1.1 Bakgrund 4](#_qminn4siflmp)

[1.1.1 Frontend och Backend 5](#_vag50uvzic1m)

[1.2 .NET, C# och Windows Forms 5](#_258drc5kzu5e)

[1.2.1 .NET-ramverket 5](#_oqjrb36dz64i)

[Kodbibliotek 5](#_7649wylbt3el)

[Verktyg och tjänster 6](#_yqtedv22ir6a)

[1.2.2 C#-programmeringsspråket 6](#_gw4y077lyeu7)

[Objektorienterad programmering (OOP) 6](#_nhyqmtw280hs)

[Abstraktion 6](#_8pwsxy4b6glx)

[Inkapsling 7](#_28z6qnpkt0qs)

[Arv 7](#_xk0pag35q7m9)

[Polymorfism 7](#_xjcuqhc3m3lb)

[Några exempel på variabler som används i C# 8](#_fdq23tbc34h3)

[Klasser 8](#_w2rhboupclhj)

[Objekt 9](#_aw01ievt5pia)

[Konstruktorer 9](#_bo7ryj9jn496)

[Hårt typat och mjukt typat, vad är skillnaden? 10](#_27r9o187da46)

[Garbage collection 11](#_jis5ctg53wle)

[Felhantering/try-catch 12](#_72fmdnroubjk)

[Kopplingen mellan C# och .NET 12](#_a8hi5s2rutkb)

[1.2.3 Windows Forms 13](#_pxocdj1ksynu)

[1.2.4 Visual Studio 13](#_xtzchuhizoj8)

[1.3 HTML, CSS och JavaScript 13](#_vppci3kmn62x)

[1.3.1 HTML 13](#_7dsfrtiresyx)

[1.3.2 CSS 14](#_y7q0f5tlbf8i)

[1.3.3 JavaScript 14](#_r4jz12afxce5)

[1.4 Syfte och frågeställning 14](#_q2ko1hw3pf7k)

[**2. Metod 15**](#_ke2gneqwymr)

[**3. Resultat 16**](#_24t2sh36tokn)

[3.1 Användargränssnitt 16](#_tjco4aclcyj7)

[Sammanfattning 17](#_1khd2j8n4rmx)

[3.2 Klickhändelser 17](#_fmnydclc9i4b)

[Sammanfattning 17](#_v0bug4yh6tyz)

[3.3 Matematisk uträkning 17](#_wafcshsnjedx)

[Sammanfattning 18](#_e2t9a4n63s14)

[3.4 Felhantering 18](#_h7mhywsxw6k4)

[Sammanfattning 18](#_awhlomt7z5qy)

[**4. Diskussion 19**](#_4oj2hqqucmkx)

[4.1 Skillnader i källkod/backend 19](#_uabg1p2jat6u)

[4.2 Skillnader i design/frontend 20](#_wa6hfiesdw7w)

[4.3 Svårighetsgrad för dig som utvecklare 21](#_c9nb82h5u9r1)

[4.4 Tillgänglighet för användare 22](#_tcuhgjiw519e)

[4.5 Sammanfattning 22](#_f0xid6ss4z2z)

[**5. Källförteckning 23**](#_l7x55ihbx5e)

[Underlag för kapitel 1.2 24](#_tu4uwk8r5kdx)

[Underlag för kapitel 1.3 25](#_duptcem7yic5)

[Underlag för kapitel 2 25](#_ruo7uf9j7qj3)

[**6. Källkritik 25**](#_c7bi1e2rwsbc)

[Microsoft 25](#_hjwxic1rrbup)

[LinkedIn 26](#_dvli7viov9qm)

[W3Schools 26](#_m68uagp3jnzr)

[CareerFoundry 26](#_a56go0zgqi1o)

[GitHub 27](#_qjny2jfvamrb)

[**7. Bilagor 28**](#_9go0hre96uun)

[Bilaga 1. 28](#_cje6alxbdbsi)

[Bilaga 1 Bildtext 28](#_p02is2qblkxw)

[Bilaga 2. 29](#_xeb5ly65aol)

[Bilaga 2 bildtext 29](#_4ogcdbgw42gk)

[Bilaga 3. 30](#_2prvah6fr1ta)

[Bilaga 3 Bildtext 30](#_38ae8ecf2yw0)

[Bilaga 4. 31](#_11l22o2a6pt7)

[Bilaga 4 Bildtext 31](#_w3wghmtwpe4a)

# 0. Abstract

This report documents the creation of two calculators using two different programming languages, C# and JavaScript, comparing their differences. The analysis covers the source code, user interface creation, and mathematical calculations. Both calculators share similar backend principles but differ in certain aspects too.

C# is strongly-typed, requiring explicit type conversions and has more robust error-checking, ensuring precision for things like type conversions and minimizing the risk for errors. But JavaScript, being loosely-typed, offers more flexible and shorter code but increases the risk of runtime errors due to its automatic type conversions and thus lower precision.

In frontend design, C# with Windows Forms provides a user-friendly drag-and-drop interface, simplifying the design process with easily accessible and understandable elements and styling. JavaScript with HTML and CSS requires more manual writing of code, making it more challenging, especially for beginners.

Regarding usability, the JavaScript calculator can be published online, accessible from any internet-enabled device, while the C# calculator needs to be downloaded and installed, posing a potential accessibility barrier for non-technical users. However, C# applications are more stable and secure, having less risks of harmful user modifications that break certain features or the entire programs ability to run.

This report serves as a reference to understand the differences between these languages and to guide developers in choosing the appropriate language based on what their project requires and what fits them as a developer.

# 1. Inledning

## 1.1 Bakgrund

I dagens digitala era är effektiv backend-programmering och intuitiva användargränssnitt/frontend avgörande komponenter för att skapa förståeliga mjukvarulösningar för alla våra tekniska enheter. Allteftersom digitaliseringen kryper längre och längre in i vår vardag, och allt mer känslig information skall skrivas in och sparas digitalt blir det ännu viktigare att som utvecklare ha kunskapen att välja rätt digitala redskap.

Det finns många val när det gäller val av programmeringsspråk och plattform en applikation skall ligga på såsom självständiga program eller sidor på internet och det kan kännas väldigt överväldigande att välja någonstans att börja med sitt projekt, eftersom att området är så brett med så många vägar att välja. Att se och förstå skillnader/likheter mellan två språk kan förenkla denna process och göra valet av plattform lättare. Det är viktigt att göra rätt val när man ska skapa ett program för att inte göra arbetet onödigt svårt genom att välja en plattform som inte passar ändamålet. Detta kan också vara en bra resurs om en person redan kan ett av språken, och då enklare kan förstå det andra genom att se skillnaderna och likheterna i hur de fungerar.

Att till exempel vara medveten om de redan existerande ramverken och biblioteken inom de olika språken/plattformarna gör de första stegen ännu mer stabila då man vet att de mest grundliga delarna av sitt planerade projekt kan använda de redan existerande “bitarna” för att göra arbetet mer effektivt. Sedan utöver detta finns det andra ting som bör inkluderas i valet och en av dessa är den tidigare nämnda säkerheten. När känslig information skall hanteras och sparas är det viktigt att plattformen har de säkerhetsspärrar som hindrar dataläckage eller intrång från hackergrupper eller andra oetiska församlingar som stjäler informationen antingen för att sälja den på den svarta marknaden för stora summor pengar eller själva använda för utpressning eller liknande tekniker för att tvinga oskyldiga individer att göra otrevliga saker.

Sedan finns det också prestanda och effektivitet. Allteftersom de teknologier vi använder blir snabbare och snabbare, blir kraven på utvecklare också större. De ska inneha förmågan att faktiskt ta nytta av dessa teknologiska framsteg, och kunna underhålla samt uppdatera sina program så de inte hamnar efter konkurrenterna på marknaden.

### 1.1.1 Frontend och Backend

Två ord som nämns i detta dokument är “*Frontend*” och “*Backend*” och de kan enkelt förklaras som att *Frontend* är framsidan som användaren ser när hen interagerar med programmet, och *Backend* är den delen av programmet som användaren inte kan se, men som gör saker i bakgrunden.

**Exempel på frontend** för en digital miniräknare är knapparna och textrutan som visar siffrorna,

**Exempel på backend** för en digital miniräknare är de matematiska uträkningarna som uppdaterar textrutan med svaret.

## 1.2 .NET, C# och Windows Forms

### 1.2.1 .NET-ramverket

.NET förklaras av Microsoft (2024) som “A Free and open-source application platform supported by Microsoft”. En sk. “Application Platform” är en samling (ofta kallad “ramverk”) som består av ett flertal saker:

#### Kodbibliotek

Ett kodbibliotek är en samling färdiga kodbitar som en utvecklare kan infoga i sitt program för att underlätta skapandet. Tänk dig följande exempel, du skapar en applikation / ett program som ger användare möjlighet att skapa ett personligt konto. Istället för att själv skriva ihop funktionaliteten för konto-systemet kan du använda en redan färdig kodbit från ett bibliotek och spara tid. (CareerFoundry 2023)

De bibliotek jag använt för min miniräknare i C# är:

“***System***”, som behövs för i princip alla C#-program. Den innehåller hantering av olika variabler såsom “int” (heltal) och “double” decimaltal. Utan detta bibliotek fungerar egentligen ingenting.

“***System.Data***”, som i mitt program behövs för att utföra matematiken och returnera resultatet av t.ex. 5\*5.

“***System.Windows.Forms***”, ger tillgång till möjligheten att skapa gränssnitt och koppla gränssnittets/frontend med knappar och andra element till backend.

#### Verktyg och tjänster

Utöver dessa bibliotek finns det verktyg såsom Visual Studio (förklarad under rubrik 1.2.3) och molntjänster såsom “DevOps” som förklaras av Microsoft (2024) som ett verktyg för att underlätta gemensam utveckling av program genom molnet via DevOps **Services** där datan sparas och hanteras av Microsoft själva på deras server. Sedan finns det alternativ för de som vill ha datan på sin egen lokala server istället för i molnet. Detta alternativ heter DevOps **Server**. Microsoft (2024) skriver också om att DevOps ger möjligheten till en “kollaborativ kultur” som gör det enkelt för utvecklare att samarbeta och tillsammans skriva mjukvara.

### 1.2.2 C#-programmeringsspråket

C# är designat av Microsoft för att vara modernt, lättanvänt, och kraftfullt. Det har möjligheten att bygga allting från mobilapplikationer till servermjukvara, allt med samma språk. Det är ett objektorienterat och hårt typat språk som är helt gratis att använda för vem som helst.

#### Objektorienterad programmering (OOP)

Objektorienterad programmering handlar om att programmera kring “objekt”. Dessa objekt består av egenskaper (variabler med värden) och/eller metoder (kod som justerar/manipulerar objekten och dess värden). Microsoft (2023) förklarar att det finns fyra hörnstenar för objektorienterad programmering, och de är Abstraktion, Inkapsling, Arv och polymorfism.

#### Abstraktion

Abstraktion används för att enklare strukturera koden. Till exempel, du skapar en abstrakt klass som heter “Account” och sedan använda den som mall när du skapar till exempel "interrestEarningAccount", “LineOfCreditAccount”, "GiftCardAccount" etc. då alla tre klasserna kommer att följa samma grundläggande struktur, i detta fallet med “name” och “initialBalance”. Utöver detta kan du sedan ge dessa nya klasser egna egenskaper. (Microsoft 2023)

#### Inkapsling

Inkapsling är en annan hörnsten inom OOP, och det handlar om hur programmet hanterar egenskaper och metoder som arbetar med de egenskaperna. Inom OOP-språk finns egenskaperna, dess värden och metoderna ihoplagda i samma objekt, vilket gör det mycket enklare att kontrollera vilka delar av programmet som kan interagera med objektets innehåll och åtkomst från andra delar av programmet kan blockeras enkelt med användning av inkapsling. (Microsoft 2023)

Detta är en bra säkerhetsfunktion då det möjliggör enkel avgränsning och skyddande av data vilket minskar risken att en oetisk person/samling kan hitta “kryphål” och navigera utanför programmets normala “ramar” och t.ex. kan hitta information som hen inte borde se. Ett exempel på bristande inkapsling skulle vara att en person som använder en bank-app kan öppna ett konsolfönster och skriva ut alla registrerade personnummer. Men om dessa personnummer är inkapslade i skyddade objekt som endast kan kommas åt av specifika delar av programmet, uppstår denna kris inte överhuvudtaget då det blir praktiskt taget omöjligt att plocka fram datan.

#### Arv

Arv är en annan bra funktion för att minska hur komplicerad källkoden behöver vara. Nya klasser kan ärva egenskaper och metoder från tidigare klasser, vilket innebär att du som utvecklare inte behöver skriva samma kod om och om igen, då du kan hämta tidigare skriven kod och använda den igen. Detta kan också kopplas till abstraktionen då den använder sig av arv. (Microsoft 2023)

Både Abstraktion och Arv är bra för läsbarheten av källkoden men också programmets prestanda då samma kod inte behöver stå flera gånger och då inte köras flera gånger.

#### Polymorfism

Polymorfism ger dig möjligheten att använda metoder från ärvda klasser på olika sätt. En abstrakt klass “Account” med metoden “addMoney” kan användas på olika sätt inom de olika ärvda klasserna vi diskuterade tidigare som är “interrestEarningAccount”, “LineOfCreditAccount”, "GiftCardAccount" då dessa ärvda klasser kommer att inkludera metoden i sig. Sedan kan du som utvecklare bygga ut metoden i underklassen. (Microsoft 2024)

#### Några exempel på variabler som används i C#

**Integer** förkortas Int och tar emot heltal (1, 2, 3, etc.)

Exempel för att skapa ett objekt med en int-variabel:

*int mittTal = 8*

**Double** tar emot både heltal och decimaltal (1.4, 2.5, 5 etc.)

Exempel för att skapa ett objekt med en double-variabel:

*Double mittDecimaltal = 8.5*

**String** tar emot text (“Hello World!” är ett exempel)

Exempel för att skapa ett objekt med en string-variabel:

*string minText = “Hello World!”*

**Bool** har två lägen (True och False)

Exempel för att skapa ett objekt med en bool-variabel:

*Bool ärDetSant = True*

*Mönstret är alltså Typ Namn = Variabelns innehåll*

Dessa fyra variabler ovan skapar alla sina egna objekt och de har alla sina egna namn. Sedan säger programmet till datorn att “hej, den här variabeln heter X och jag vill att du kommer ihåg att den innehåller X.

#### Klasser

En klass deklareras genom att först skriva en “access modifier", sedan ordet “class” och sist namnet du vill ge klassen. Detta namnet fungerar som en identifikation. I detta fallet används “public” som modifier, så

**Exempel:**

*public class minKlass*

*{*

*Kod skrivs inuti dessa måsvingar*

*}*

#### Objekt

Ett objekt är en instans av en klass, alltså en kopia av klassen. Till skillnad från klassen kan objektet inte användas för att skapa klasser, det är därför klassen finns så att objekten kan skapas utifrån hur klassen ser ut. Återigen så är det så att objekten innehåller klassens egenskaper, med värden på variablerna som finns. (Microsoft 2023)

**Exempel:**

*minKlass mittObjekt = new minKlass*

Detta exempel skapar ett objekt av klassen “minKlass” och objektet heter “mittObjekt”

#### Konstruktorer

Konstruktorn finns för att konstruera objekt av en klass. Den har också jobbet att initiera och ge egenskaperna värden. Om du inte själv skriver en konstruktor, används den automatiska konstruktorn som inte syns i koden. Den fyller egenskaperna med standardvärden, som blir “0” för numeriska egenskaper, och “null” (betyder “ingenting”) för till exempel string och andra text-värden. Detta för att egenskaperna måste ha någon sorts värde inuti ett objekt, det kan inte vara tomt. (Microsoft 2023)

**Exempel:**

*public class minKlass*

*{*

*public string Namn*

*public int Age*

*public minKlass(string namn, int age)*

*{*

*namn = Namn*

*age = Age*

*}*

*}*

Detta exempel har nu en konstruktor som tar emot ett string-värde för namn och sedan ett int-värde för åldern.

För att anropa konstruktorn används nästan samma kod som för att skapa objekt, men nu måste vi också ge egenskaperna värden. Återigen måste de ha värden, de kan inte vara tomma.

Exempel:

*minKlass objekt1 = new minKlass(Simon, 21)*

Detta exempel skapar objekt1 genom att anropa konstruktorn och säga till den vad egenskaperna skall ha för värden.

#### Hårt typat och mjukt typat, vad är skillnaden?

Ett hårt typat språk som C# har strängare regler kring hur variabler och deras typer hanteras. Vid arbete med hårt typade språk är det inte möjligt att direkt använda en variabel som en annan typ än den den är deklarerad som. Om programmet förväntar sig en viss typ och du ger det en annan typ, kommer programmet få fel vid kompilering, då det inte kan automatiskt hantera konverteringen från variabeltyp till variabeltyp. (LinkedIn, 2024)

Till exempel, om du har en int-variabel (heltal) med värdet 7, kan du inte använda denna int-variabel som en double-variabel utan att först konvertera den. Detta trots att siffran 7 logiskt får plats i en double-variabel (double-variabler tar emot heltal OCH decimaltal). I ett hårt typat språk måste du själv skriva kod som utför denna konvertering, till exempel genom att använda metoder som “Convert.ToDouble” som gör precis vad den säger, och konverterar till Double.

Ett mjukt typat språk som JavaScript (mer info om JS under rubrik 1.3.3) är motsatsen. Det är flexibelt med typer av variabler och kommer automatiskt att konvertera dessa om det behövs. Detta innebär att exemplet ovan faktiskt fungerar för mjukt typade språk. Så programmet kan hämta en int-variabel som en double-variabel, trots att den inte är sparad som en double. (LinkedIn 2024)

#### Garbage collection

Garbage collection är ett system som automatiskt frigör minne på datorn medan ett program är igång (Microsoft 2024). Med “minne” syftas det på RAM-minnet. RAM är "primärminne”, och en hårddisk är "sekundärminne". Största skillnaden är att primärminne inte permanent sparar någon information i sig, utan program som körs sparar temporärt information där för att snabbt kunna hämta den senare. Sekundärminne är bättre refererat till som “lagring” och består av hårddiskar. Dessa används för att permanent spara information såsom foton och andra filer. Den största skillnaden mellan RAM-minne och t.ex. en hårddisk är att RAM-minnet är ofantligt mycket snabbare. Därför behövs RAM, annars hade programmen behövt spara även temporär data på hårddisken, och då hade allting gått väldigt mycket långsammare. Ytterligare en stor skillnad är att allting i primärminne försvinner när enheten stängs av, medan det i sekundärminne sparas.

När ett program startas ger datorn en mängd minne som det skall hålla sig inom, detta kallas att “allokera” minne. Allteftersom fler och fler saker händer i programmet och det arbetar med objekt och funktioner, fylls minnet med allt mer data. (Microsoft 2023)

Funktionen analyserar vad programmet gör just nu genom att kolla vad i minnet som faktiskt används. Om minnet börjar ta slut, “sopar” garbage collection automatiskt ut objekt som inte längre används. Denna process sker genom att identifiera objekt som inte längre har några referenser till sig från programmet. Objekt utan referenser (de objekt som programmet inte längre har någon koppling till och inte behöver) anses vara "sopor" och minnet detta onödiga objekt tar upp kan tömmas och återanvändas för nya objekt. (Microsoft 2023)

Garbage collection bidrar till att minska risken för vanliga programmeringsproblem som minnesläckor, där programmet glömmer att frigöra minne och endast tar upp nytt minne. Detta leder då till en dominoeffekt där ett simpelt program ändå kan sluka i sig 20GB RAM. Genom denna moderniserade och automatiska minneshantering kan utvecklare fokusera mer på själva programmet och behöver inte tänka lika mycket på hur mycket minne som används, då programmet sköter sig själv. (Microsoft 2023)

#### Felhantering/try-catch

Om ett fel uppstår medan programmet körs måste det finnas någon form av felhantering så en användare men också en utvecklare får reda på vad som inte fungerade. Utan felhantering så kommer programmet helt enkelt att krascha utan felmeddelande.

Microsoft (2023) förklarar “The try-catch statement” som någonting du kan lägga din kod inuti, och om något fel uppstår fångas det upp och ett felmeddelande kan skrivas ut. Placera din kod inuti “try”-delen och lägg sedan in vad som händer vid fel i “catch”-delen. Om inget fel inträffar, körs endast “try”-delen och programmet fortsätter som vanligt. Men om ett fel inträffar fångar “catch” och koden inuti körs.

**Exempel:**

*try*

*{*

*Kod som kan bli fel här*

*}*

*catch*

*{*

*Kod som körs vid fel här*

*}*

#### Kopplingen mellan C# och .NET

Microsoft (2024) förklarar att C# är det mest använda programmeringsspråket för .NET.

NET-funktionerna effektiviserar skapandet och användningen av program genom att göra det möjligt att exekvera (betyder köra/starta) samma program på flera typer av enheter med samma källkod. Olika digitala enheter har olika inre struktur/arkitektur. Till exempel så har en Android-telefon och en Apple-telefon olika arkitektur men också olika operativsystem (Android eller Apples egna iOS). På grund av dessa strukturella skillnader behövs olika instruktioner för att till exempel utföra matematik. När ett program skrivet med C# startas, körs det igenom .NET-funktionerna installerade på din enhet och det är detta som gör det möjligt att köra samma kod på olika typer av enheter. Du behöver alltså inte skriva flera olika program med olika kod för till exempel Windows och Linux, då C#-källkod översätts till samma sk. “Common Intermediate Language” (förkortas CIL) när den kompileras. CIL kan liknas med ett universellt språk som kan översättas till andra språk. Din egna maskin utför översättningen från CIL-koden till sin egna kod så den fungerar för just din dator, telefon etc. när programmet väl startas. Denna typ av sista-sekunden-kompilering har det passande namnet “JIT” som står för “Just In Time”.

En mer kortfattad sammanfattning av detta skulle vara följande:  
> Projektets källkod

> Kompileras till CIL-kod

> Du startar programmet med CIL-koden på din enhet

> Din enhet kompilerar om CIL-koden “On the fly” till kod som fungerar för din enhet

> Du kan använda programmet.

### 1.2.3 Windows Forms

Windows Forms (förkortat till WForms på antal ställen i rapporten) är en del av .NET. Wforms används för att skapa ett “Graphical User Interface” (förkortas till “GUI”) för C#-applikationer (Microsoft 2024). Ett GUI är den synliga delen av programmet, alltså det som visas för användaren. Detta inkluderar allting från knappar till textlådor som en användare kan behöva för att interagera med programmet. För att skapa WForms-program används “Visual Studio”. Se bilaga 1 för exempel på hur Windows Forms designern ser ut.

### 1.2.4 Visual Studio

Visual Studio (förkortas till VS på antal ställen i rapporten) förklaras som en “Creative Launching Pad” för de som vill skapa sina egna program. Det har stöd för ett flertal språk och plattformar och inkluderar alla funktioner för att skriva, debugga, och kompilera program för dessa språk/plattformar (Microsoft 2024). I detta fall ligger fokuset på WForms, som använder C# som programmeringsspråk. Vid design av program i WForms används ett sk. “Drag and Drop”-system som finns tillgängligt i Visual Studio. För att lägga till nya element i sitt program, klickar och håller användare på ett element de vill lägga till från listan, flyttar muspekaren dit hen vill ha den placerad i programfönstret, och släpper musknappen. Då hamnar till exempel knappen inuti det simulerade fönstret och tillhörande kod genereras automatiskt inuti projektet. Denna kod innehåller elementets storlek, position och styling mm. (Se bilaga 1 för exempel på hur VS-designern ser ut). Denna automatiska kod uppdateras allteftersom du som utvecklare justerar element genom att dra runt det med muspekaren eller sträcker ut det horisontellt eller vertikalt för att ändra dess storlek. Allting inklusive själva fönstrets egenskaper kan flyttas och organiseras om, och VS uppdaterar koden helt av sig självt.

Det finns också möjlighet att läsa samt redigera den autogenererade styling-koden direkt i VS om man vill, men detta behövs sällan då alla alternativ finns tillgängliga i “properties”-fönstret när ett element klickas på/ligger i fokus. Se bilaga 2 för mer information.

## 1.3 HTML, CSS och JavaScript

### 1.3.1 HTML

HTML (HyperText Markup Language) används för att skapa strukturen som CSS och JavaScript sedan arbetar inom. HTML-koden tar hand om själva innehållet och dess element. Detta är allting från rubriker till brödtext, och alla dessa innehållstyper har egna sk. “Taggar” och det är start-tagg och slut-tagg som används för att skapa elementen. Slut-taggen är samma ord som start-taggen, men med slash infogat framför. Ett exempel är “<p>” och “</p>” som innehåller sidans brödtext. Sedan när du öppnar webbsidan så säger HTML till din enhet vad innehållet är och det visas sedan på din skärm. (W3Schools u.å)

### 1.3.2 CSS

CSS (Cascading Style Sheet) bestämmer hur HTML-innehållet visuellt presenteras i webbläsaren. Med de tidigare nämnda HTML-taggarna finns det redan inbyggda funktioner i webbläsaren som t.ex. ger vissa element större text. Men detta är en väldigt primitiv funktion, och skapar något liknande ett Word-dokument. Detta blir mest märkbart när media infogas, man kan enkelt se att HTML endast infogar innehållet på sidan, utan att t.ex. positionera det effektivt. För mer unik styling behövs CSS. CSS-koden “hämtar” taggarna och lägger till attribut till dem. Exempel är att hämta taggen “p” som innehåller brödtext som nämnt innan. När man väl hämtat denna och infogat den i sitt CSS-dokument finns det en enorm mängd olika sätt att ändra stil på innehållet inom taggen. Som exempel tar vi “font-size” som låter dig ändra storlek på texten. Denna variabel tar emot olika typer av enheter. Du kan skriva in ett exakt värde (12px t.ex.) eller göra nytta av “rem” eller “em” som tar hänsyn till andra element på sidan och dess storlek för att ge detta nya element en passande storlek. (W3Schools u.å)

### 1.3.3 JavaScript

Användningen av JavaScript (förkortas JS på antal ställen i rapporten) möjliggör mer avancerad funktionalitet på en webbsida. JS är kod som körs på klientens maskin när hen besöker en webbsida. W3Schools (u.å) förklarar att “JavaScript can change HTML Content” och det första exemplet diskuterar “getElementById” som fungerar liknande till CSS. Det hittar alltså innehållet med matchande ID/tagg och sedan kan man ändra innehållet. Det tredje exemplet visar en bild på en släckt lampa med två knappar bredvid. När man trycker på knapparna tänds eller släcks lampan. Detta utförs genom att JS hämtar “myImage” och ändrar dess innehåll när du trycker på knapparna. I detta fall ändras bilden på lampan till släckt/tänd. (W3Schools u.å)

Precis som med C#, används try-catch-block för att kontrollera att potentiella fel blir tydligare för användare och utvecklare. Se denna rubrik under C#-delen för en bredare förklaring!

## 1.4 Syfte och frågeställning

Denna rapport dokumenterar skapandet av två olika versioner av en miniräknare: en med C# Windows Forms och en med HTML/CSS/JavaScript. Denna rapport syftar till att inte bara demonstrera hur gränssnittsdesign och funktionalitet skiljer sig åt mellan dessa två programmeringsspråk/plattformar, utan också att analysera vilka designval som ligger bakom programmens arbetssätt och hur de på sitt egna vis framställer samma resultat. Rapporten jämför processen för att bygga en backend och koppla den till frontend för båda språken. Genom denna steg-för-steg-analys kan man sedan dra slutsatser om både språkens likheter/skillnader, men också hur man engagerar sig i god praxis när man konstruerar dessa program och vilket av dem som man bör välja om man skall genomföra sitt eget projekt.

**Det finns två frågeställningar:**

1. **Vilka vägar/metoder använder programmeringsspråken för att nå fram till samma resultat, hur ser de vägarna/metoderna ut jämfört med varandra?**
2. **Vilket språk bör man välja för sitt ändamål?**

# 2. Metod

För undersökningen använder jag mig av två program jag skapat och som finns offentligt läsbara/nedladdningsbara på Github. Github (u.å) förklarar sig som en molnbaserad plattform där du kan spara, dela och samarbeta med andra för att programmera.

Dessa två program har skrivits så de liknar varandra visuellt så nära som möjligt, så man kan analysera källkoden för att framställa vilka skillnader det finns mellan språken och dokumentera t.ex. hur knappar ändrar färg eller hur textrutans innehåll uppdateras. Koden och kommentarerna är skrivna på engelska för god praxis men också så att en läsare som inte kan svenska i alla fall kan läsa min källkod tillsammans med abstract om de är intresserade av rapporten.

För att göra det enklare att förstå dessa skillnader har jag plockat referenser från bl.a Microsoft och W3Schools. Dessa referat är till för att stödja undersökningen och förklara begrepp inom ämnet. Det är även en bra resurs för de som vill lära sig ännu mer om ett specifikt område.

[GITHUB-REPOSITORY HÄR](https://github.com/Chyddo/Calculator-Komvuxarbete)

# 3. Resultat

## 3.1 Användargränssnitt

C#-räknaren bygger sitt användargränssnitt med Windows Forms (förklaras under rubrik 1.2.3) som tillhandahåller både de interaktiva delarna (knappar etc.) men också stylingen. Genom användning av Drag-and-Drop systemet som finns tillgängligt finns det nu ett rutnät av knappar samt en textruta som visar resultatet. Alla knappar är fristående från varandra och de är fullständigt oberoende av var resten av elementen befinner sig. Du kan lägga till, flytta och ta bort element utan att någon annan del av programmet visuellt förändras. Styling-koden behövde inte manuellt redigeras under någon del av arbetet, utan alla visuella ändringar kunde utföras direkt i design-fliken (se bilaga 2 för exempel) som visar utvecklaren hur programmet kommer att se ut även innan det startas.

JavaScript-räknaren har fler separerade delar och har inte skapats med hjälp av ett visuellt assisterande system och har inte möjligheten att redigera element direkt på sidan utan allting behöver manuellt skrivas in som text/kod för att uppnå resultatet. Det finns ingen möjlighet att se hur en ändring i koden ändrar på utseende, utan programmet/sidan behöver startas/laddas om för att visa resultatet. HTML-koden är grunden för de element som behövs på sidan vilket inkluderar knappar och textruta. Sedan tar CSS över och utför stylingen/organisationen av dessa element. Utan CSS formateras alla knappar i en lång lista som endast är en rad. CSS behövs för att lägga 4st knappar per rad och för resten av den visuella organiseringen såsom storleken på knapparna. Detta är för att elementen inom HTML är beroende av varandra, och står i ett “virtuellt led”. Om ett element försvinner eller flyttas, samt om nya element läggs till, kommer ordningen ändras i ledet och då ändras hela räknarens utseende.

Sist finns JavaScript för att ge räknaren sin funktionalitet, alltså att ta in användarens indata och utföra matematiken.

### Sammanfattning

C#-räknaren och JavaScript-räknaren har liknande gränssnitt visuellt, men backend ser olika ut. Som exempel är C#-knapparna fristående och oberoende av varandra, medan JavaScript-räknarens hela visuella design kan “gå sönder” om element läggs till, byter plats eller tas bort.

## 3.2 Klickhändelser

C#-räknaren använder de automatiskt genererade klickhändelserna “*button\_click*” som är kopplade till knapparna. Knapparna har alla unika namn och unika händelser i koden. När knapparna klickas på av användaren körs koden som ligger innanför den tryckta knappens klickhändelse och programmet lägger till tecken i textrutan, räknar ut svaret på det som står i textrutan, eller rensar textrutans innehåll beroende på vilken av knapparna som trycktes på. Exempel: “*buttonEquals\_Click*” anropar funktionen “*EvaluateExpression*” som utför den matematiska uträkningen och uppdaterar textrutan med svaret. (Se bilaga 3 för mer information)

JavaScript-räknaren har en liknande “*onclick*”-händelse som kopplar knapparna till JavaScript-delen av programmet. När en knapp trycks på, körs dess “*onclick*” och beroende på knappens uppgift så gör programmet olika saker, som inkluderar att lägga till tecken, räkna ut svaret genom anrop av “*eval()*” funktionen eller rensar textrutans innehåll.

### Sammanfattning

Båda räknarna har knappar som har sina egna händelser som alla utför olika saker. När en knapp trycks på, utförs dess händelse och programmet utför en uppgift. Kopplingen mellan knappar och händelser är strukturerade på samma sätt, men språkens egenheter gör koden annorlunda.

## 3.3 Matematisk uträkning

När “=” trycks i C#-räknaren inträffar “*buttonEquals\_Click*” som anropar metoden “*EvaluateExpression*” med textrutans innehåll som en string. Denna string skickas då som den “*expression*” som skall räknas ut. “*DataTable.Compute*” är en metod inom .NET och kommer från “*System.Data*”-biblioteket. Metoden tar emot den “*expression*” (textrutans innehåll) som vi skickat, och utför den matematiska uträkningen. Uträkningen/Svaret sparas i ett objekt med namn “*result*” och eftersom att svaret kan ha decimaler och C# är hårt typat måste “*Convert.ToDouble*” användas eftersom hårt typade språk inte själv kan dynamiskt ändra typen av variabel. När “EvaluateExpression” har jobbat klart så får “*buttonEquals\_Click*” tillbaka objektet “*result*” och innehållet av “*result*” skrivs ut som en string i textrutan.

När “=” trycks i JavaScript-räknaren så inträffar “*onclick="calculate()"*” som kör funktionen “*eval()*” inbyggd i JavaScript som utför den matematiska uträkningen. Funktionen ges innehållet från “display.value” som är textrutans värde. Sedan uppdateras “display.value” med resultatet av uträkningen. Det behövs ingen konversion inskriven i koden då JavaScript är mjukt typad och automatiskt hanterar konversionen mellan olika typer av värden. En annan skillnad är att JavaScript kan hantera mycket större tal för att “*eval()*”-funktionen kan hantera mycket större tal än C#:s “*DataTable.Compute*” och kan visa ytterligare saker såsom “*NaN (Not a Number)*” och “*Infinity (oändligheten)*”. (Se bilaga 3 för att se källkod för båda språkens uträkningsfunktioner)

### Sammanfattning

Båda programmen har klickhändelser för “=”-knappen som anropar en metod/funktion som tar in textrutans innehåll, utför en matematisk uträkning av innehållet, och returnerar/uppdaterar textrutans innehåll. Största skillnaden här är att C# behöver en påtalad konversion till double för potentiella decimaltal, medan JavaScript självmant visar resultatet korrekt utan att behöva en konversion inskriven i koden. Sedan så behöver C# ett kodbibliotek inkluderat för att utföra matematiken, medan JavaScript har möjligheten att genomföra uträkningarna med inbyggda funktioner. JavaScript kan också visa mer information angående ologiska tal “NaN” och “Infinity” som är oändligheten. Se bilaga 3 och 4 för mer information kring hur uträkningarna utförs.

## 3.4 Felhantering

C#-räknaren använder try-catch-block för att hantera och visa felen som kan uppstå vid uträkning. Felmeddelandet visar sig som en pop-up som förklarar vad felet är men det står även “Error” i räknarens textruta. Programmet kraschar inte, utan användaren kan trycka på “CLEAR” och skriva in nya tal som skall räknas ut.

JavaScript-räknaren har också try-catch-block inom JavaScript som hanterar potentiella fel. Det kommer inte fram en pop-up ruta utan endast räknarens textruta ändras till “Error”. Det är dock svårare att nå detta fel, då JavaScript-räknaren hanterar till exempel stora tal mer effektivt.

### Sammanfattning

Båda språken använder try-catch för att säkerställa att programmen inte kraschar vid felaktig inmatning eller ologiska tal. Vid fel, visas “Error” i textrutan för båda räknarna, och C#-alternativet har utöver detta en pop-up som förklarar i mer detalj vad som gick fel.

# 4. Diskussion

## 4.1 Skillnader i källkod/backend

Efter att ha jämfört backend för båda språken kan mönster ses. Om man kisar tillräckligt så är språken väldigt likadana, men med några skillnader som kan vara viktiga att tänka på.

Båda räknarna har en mycket likadan koppling mellan knapparna och backend. När en knapp trycks på, har knappen ett värde som skickas till backend och så händer något. Till exempel om man trycker på “7” så tar programmen emot den och går igenom olika kontroller för att se vad det skall göra.

Vid uträkningen så sparar båda språken textrutans innehåll i en variabel som sedan körs igenom en metod/funktion som tar in variabelns innehåll, räknar ut svaret, och skickar tillbaka det till textrutan där användaren skrev in sitt egna tal. Skillnaden här är att C# behöver en metod skriven av dig som utvecklare som använder “*DataTable.Compute*” importerat från ett kodbibliotek och sedan behövs resultatet manuellt konverteras till Double och här är det väldigt tydligt att C# är hårt typat. Med JavaScript är det bara att köra på med den inbyggda “eval”-funktionen som inte behöver någon större hjälp från resten av koden, det löser problemet av sig självt då det är mjukt typat och automatiskt hanterar konvertering av variablerna till den korrekta typen för ändamålet. Ett jämförbart exempel skulle vara om du endast behövde skriva “DataTable.Compute()” och sedan lägga in variabeln innanför måsvingarna, och så räknade den ut allting och returnerade korrekt typ etc. Det är exakt så JavaScript gör. Ge det talet och så löser det resten. (Se bilaga 3 och 4 för att se hur koden ser ut för båda språken)

C# kan nu med jämförelsen upplevas som nästan för icke-flexibelt och fyrkantigt, men ett stort plus med detta är att risken för felaktig konvertering är i princip noll, då det måste stå i koden exakt vad som skall konverteras till vad. Hårt typade språk har dessa “säkerhetsnät” redan vid kompilering, så det är svårt att ens råka bygga ett program som inte hanterar variabeltyperna korrekt. Innan programmet ens kan startas säger det “Nej, vi kan inte ta emot denna integer som en double”.

Med JavaScript skjuter det lite mer från höften, och detta upplevs ju som skönare då det speciellt behövs mindre kod för matematiken i detta fallet, men snedsteget kan bli att större och mer komplicerade program med mycket fler variabler inte konverteras korrekt överallt. Så om det är viktigt att konvertering mellan variabeltyper alltid sker med maximal precision och tydligt förmedlad kod, är C# ett bättre alternativ då det måste tydligt förklaras vad som skall bli vad. För mindre program som inte till exempel skulle kunna vara en potentiell riskfaktor för stora företag etc. kan JavaScript vara ett bättre alternativ, då dess mjukt typade sätt förenklar skapandet av backenden och eventuella precisionsfel av språkets automatiska typkonvertering inte kan leda till katastrofer eller liknande.

## 4.2 Skillnader i design/frontend

C# har bonusen att vara fullt kompatibel med Windows Forms, vilket har en enorm digital verktygslåda för att visuellt designa ditt program med mycket enklare drag-and-drop system och fristående element som kan placeras och justeras utan restriktioner. Alla visuella redigeringar görs direkt i ett låtsas-programfönster som visar exakt hur det kommer att se ut. (Titta på bilden på räknaren från första sidan och sedan den från bilaga 1. De ser exakt likadana ut förutom fönstrets ram. Alla knappar är på samma ställe etc.) Du kan enkelt ändra exakt vad knapparna har för text, typsnitt och andra egenskaper såsom att ändra muspekarens markör när musen förs över textfönstret för att visa att det inte går att skriva i det. När ett element markeras i Windows Forms-designern, så visas “Properties”-fönstret och alla redigerbara egenskaper finns listade (se bilaga 1). CSS har inte denna förmån, utan det har endast “AutoComplete”/”AutoFyll” för om du skrivit något halvvägs så föreslås det olika saker. För att se en lista över CSS-attribut du kan lägga till behöver du själv göra den undersökningen för att se vad du behöver.

Att använda HTML,CSS och JS för att uppnå samma resultat kan upplevas som mycket mer komplicerat. En stor skillnad är att HTML-elementen som JS arbetar inom påverkas av varandra. Alla element ligger i ordning, och kan inte flyttas omkring fristående från varandra på samma sömlösa sätt. Ett tydligt exempel är att om du tar bort en knapp från C#-räknaren, så flyttas inga av de andra knapparna. Men om du tar bort en knapp från JavaScript-räknaren så blir det totalt kaos med knapparnas position, då de andra knapparna “flyter” uppåt för att det nu finns plats för dem i “ledet”. Att få knapparna i ordning med allting inkluderat var även det en viss utmaning då det inte går att omedelbart se vad som händer om CSS-koden ändras.

Det är mycket “testa, det gick inte, CTRL+Z, försök igen” och detta tar ju mycket mer tid än att bara dra och släppa i Windows Forms. En jämförelse här mot Windows Forms skulle kunna vara att du var tvungen att öppna ett ytterligare Visual Studio-fönster och manuellt skriva in varje elements namn och hur du vill att det ska se ut, och då har du något som närmar sig de extra stegen som krävs för samarbete mellan HTML och CSS.

Det är möjligt att skriva CSS-kod direkt i HTML-dokumentet, men detta avråds då det ofta blir väldigt stökigt i koden och att separera HTML och CSS-dokument är det bättre alternativet, speciellt om flera sidor behöver samma styling. Det är ju enklare att ha ett CSS-dokument och peka alla sidor till det istället för att skriva potentiellt 100-tals rader kod på alla sidor. Det är helt enkelt krångligare att visuellt designa med HTML,CSS jämfört med Windows Forms, speciellt för nybörjare.

## 4.3 Svårighetsgrad för dig som utvecklare

Svårighetsgraden för båda programmen kan ses på olika sätt. Att skriva kod som utför de matematiska uträkningarna är mycket enklare i JavaScript på grund av dess inbyggda funktioner och mjukt typande, men för att få en visuellt tilltalande räknare behövs mycket jobb med HTML och CSS för att få den JavaScript-räknaren att också se bra ut och vara användbar från ett frontend-perspektiv.

C# tillsammans med Windows Forms upplevs som motsatsen. Visuell design är jättelätt, men att skriva metoder som utför matematiken upplevs som svårare på grund av att C# är hårt typat, kräver import av kodbibliotek och behöver mer kod för att utföra samma matematik.

Det handlar egentligen om det du som utvecklare vill lägga mer tid på. Kan du leva med mer komplicerad backend, eller mer komplicerad frontend?

## 4.4 Tillgänglighet för användare

Vad passar en persons användarbas? Min JavaScript-räknare skulle kunna läggas upp på internet och vara tillgänglig för vem som helst från i princip vilken enhet som helst utan nedladdningar eller installationer. Men min C#-räknare skulle behöva manuellt laddas ner till en användares enhet, och sedan kanske tvinga användaren att uppdatera .NET-funktionerna på sin enhet så de kan köra mitt program (Se rubrik “Kopplingen mellan C# och .NET”). Detta låter ju enkelt för mig som är insatt, men för en person som inte är så tekniskt kunnig kan detta bli ett mycket märkbart hinder.

Ytterligare synpunkt är att en användare självklart behöver internet för att besöka en internetsida, och detta gör att räknaren som läggs upp på internet kan bli otillgänglig för de som saknar den förmågan. Det går ju att köra JavaScript-räknaren offline genom att direkt ladda ner filerna, men detta kan enkelt gå snett då det är flera filer som skall laddas ner, de måste ligga i samma mapp och filerna kan ju redigeras fritt som vilket dokument som helst efter nedladdning. Detta låter bra för de som vill bygga vidare, men det krävs inte så mycket för att råka öppna HTML,CSS eller JavaScript-filerna i notepad och kanske råka ta bort ett tecken. Och med det så blir det stora fel i hur räknaren fungerar om till exempel referensen till CSS-dokumentet blir fel och all styling försvinner.

Ett kompilerat C#-program har den bonusen att den är en exe-fil som körs utan kopplingar till andra filer eller liknande som kan misslyckas. Sedan så kan man inte öppna ett kompilerat program och redigera det på samma vis. Detta blir då motsatsen. Det är mer “idiotsäkert” men egna modifieringar och förbättringar av de som kan och vill är i princip omöjliga.

## 4.5 Sammanfattning

Sammanfattningsvis visar jämförelsen av backend-koden mellan C# och JavaScript att båda språken delar liknande grundprinciper men skiljer sig i vissa aspekter. C# är ett hårt typat språk vilket innebär att du som utvecklare måste vara tydlig med manuellt utförda typkonverteringar, vilket minimerar risken för fel men kräver mer kod och mer huvudkrafs. JavaScript, å andra sidan, är mjukt typat och använder inbyggda funktioner som "eval()" för att förenkla kodningen, vilket gör det mer flexibelt men samtidigt ökar risken för fel då den automatiska typkonverteringen inte har samma precision.

I frontend-design framstår C# med Windows Forms som mer användarvänligt för nybörjare på grund av dess “verktygslåda” och drag-and-drop-funktionalitet samt tydligare ledtrådar till vad som faktiskt kan göras visuellt med knappar etc.

HTML, CSS och JavaScript kräver mer manuellt arbete och kodning i skriftlig form för att uppnå liknande visuella resultat, vilket kan vara tidskrävande och kan upplevas som mycket svårare då det inte är lika snabbt att se hur ändringar påverkar det element du fokuserar på men även hur andra delar av sidan kan ändras samtidigt.

När det gäller svårighetsgrad beror valet av språk på vad utvecklaren prioriterar: enklare backend-kodning med JavaScript eller enklare visuell design med C# och Windows Forms.

För användare har JavaScript-räknaren fördelen att potentiellt vara tillgänglig online utan installation, vilket gör den lättillgänglig för en bredare publik. C#-applikationer kräver däremot nedladdningar och eventuella installationer/uppdateringar vilket kan vara ett hinder för de som inte har samma kunskap angående tekniken. Ett kompilerat C#-program är mer robust och innehar då en mindre risk att påverkas av användarens snedsteg, medan JavaScript-applikationer enkelt kan “gå sönder” då deras kod inte kompileras ner till oläslig maskin-kod utan den förblir fullt läslig och redigeringsbar för vem som helst, vilket öppnar möjligheter för förbättringar och modifieringar men som också ökar risken för att användaren helt enkelt slarvar och mycket väl kan göra sitt program ofunktionellt med till exempel en icke-menad redigering av en viktig del av programmets kod.

# 5. Källförteckning

## Underlag för kapitel 1.2

Microsoft (u.å). *What is .NET?*

<https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>

(Hämtad 2024-06-20)

CareerFoundry (2023). *What is a Programming Library? A Beginner’s Guide*

<https://careerfoundry.com/en/blog/web-development/programming-library-guide/>

(Hämtad 2024-07-15)

Microsoft (2024). *What is Azure DevOps?*

<https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops>

(Hämtad 2024-07-03)

Microsoft (2023). *Object-Oriented programming (C#)*

<https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop>

Microsoft (2023). *Introduction to classes*

<https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/classes>

(Hämtad 2024-07-13)

Linkedin (2024). *What is the difference between strongly and weakly typed programming languages?*

<https://www.linkedin.com/advice/0/what-difference-between-strongly-weakly-typed-eqwlc>

(Hämtad 2024-07-04)

Microsoft (2023). *Fundamentals of garbage collection.*

<https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals>

(Hämtad 2024-06-20)

Microsoft (2023). *Exception-handling statements - throw, try-catch, try-finally, and try-catch-finally*

<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/exception-handling-statements>

(Hämtad 2024-07-17)

Microsoft (2024). *A tour of the C# language.*

<https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview>

(Hämtad 2024-06-20)

Microsoft (2023). *Desktop Guide (Windows Forms .NET).*

<https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-8.0>

(Hämtad 2024-06-20)

Microsoft (2024). *What is Visual Studio?*

<https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022>

(Hämtad 2024-06-20)

## Underlag för kapitel 1.3

W3Schools (u.å). *HTML Introduction*.

<https://www.w3schools.com/html/html_intro.asp>

(Hämtad 2024-06-21)

W3Schools (u.å) *CSS Tutorial.*

<https://www.w3schools.com/css/>

(Hämtad 2024-06-21)

W3Schools (u.å) *JavaScript Introduction.*

<https://www.w3schools.com/js/js_intro.asp>

(Hämtad 2024-06-21)

## Underlag för kapitel 2

Github (u.å). About GitHub and Git

<https://docs.github.com/en/get-started/start-your-journey/about-github-and-git>

(Hämtad 2024-07-19)

# 6. Källkritik

### Microsoft

De källor jag plockat från Microsofts egna sidor anser jag som mest tillförlitliga i dokumentet. Det är Microsoft som har skapat .NET och C#-språket, så de har absolut ingen anledning att sprida falsk information förutom i de fall när de själva värderar sitt eget språk som “snabbt och flexibelt” samt andra positiva adjektiv. Det är ju viktigt att se tredje parters analys av programmeringsspråket, då det är självklart att MS kommer att skriva väldigt bra saker om något de själva skapat och försöka få napp på utvecklare som de vill ska börja använda deras system.

Men när det kommer till själva funktionaliteten som Microsoft beskriver för till exempel objektorienterad programmering är risken för felaktig information låg, då alla sidor inte är mer än ett år gamla och då alltså uppdateras ofta. Datumet är viktigt att kolla på när det gäller programmering, eftersom framsteg inom teknologi är så snabba. Sedan så vill ju Microsoft återigen inte ge ut falsk information som kan förvirra potentiella kunder och/eller få dem att välja en annan plattform för sina program om de inte förstår vad MS förklarar eller om de rakt av inte förmedlar korrekt information.

Sidans innehåll uppdateras via kollaboration på GitHub och ändringar som görs är tydligt kopplade till användarens identifikation samt tid och datum. Du kan se vem som ändrar vad och när det ändrades. Detta blir viktigare kopplat till teknologins snabba utveckling, då det behövs “många kockar i köket” för att vara ikapp och att kunna se vem som gör vad blir ännu mer kritiskt.

### LinkedIn

De källor jag plockat från LinkedIn använder en liknande kollaboration, och här är det ännu tydligare vem som skrivit vad. Enligt deras “Top experts in this article” så är “Gregoire Cousin”, “Anish Kumar Singh” och “Ateeb Asif” de som tillagt mest. Deras kollaborationer har även mest gillamarkeringar. De grundläggande kraven uppnås klart med datum, namn på författare etc. och jag anser källan som pålitlig då förklaringen som ges här stämmer med bl.a [TechTarget](https://www.techtarget.com/whatis/definition/strongly-typed) och [Denna StackOverflow-tråd](https://stackoverflow.com/questions/2690544/what-is-the-difference-between-a-strongly-typed-language-and-a-statically-typed). De säger i princip samma sak.

### W3Schools

Det jag lyft från W3Schools anser jag som starkt pålitligt då deras sidor används inom utlärning av webbdesign bland annat hos NTI-skolan. Skolor består ju av utbildade lärare, och en utbildad lärare innehar kunskapen om vilka källor inom deras ämnen som är relevanta. Alla deras kodexempel har “try it yourself”-knappar där du kan öppna ett nytt fönster och själv undersöka koden och testa att ändra färger eller liknande och se resultatet direkt i webbläsaren. Med detta verktyg de själva har lagt till kan man jämföra till exempel sin egen kod skriven i visual studio och se att koden gör samma sak på båda ställena för att verifiera W3Schools innehåll. Att göra dessa tester tillsammans med att läsa texten på sidan gör det ännu enklare att se om det som skrivits stämmer, då du kan direkt testa att till exempel skapa en ny paragraf med <p> taggen och se att den skrivna förklaringen matchar vad som händer i testfönstret.

### CareerFoundry

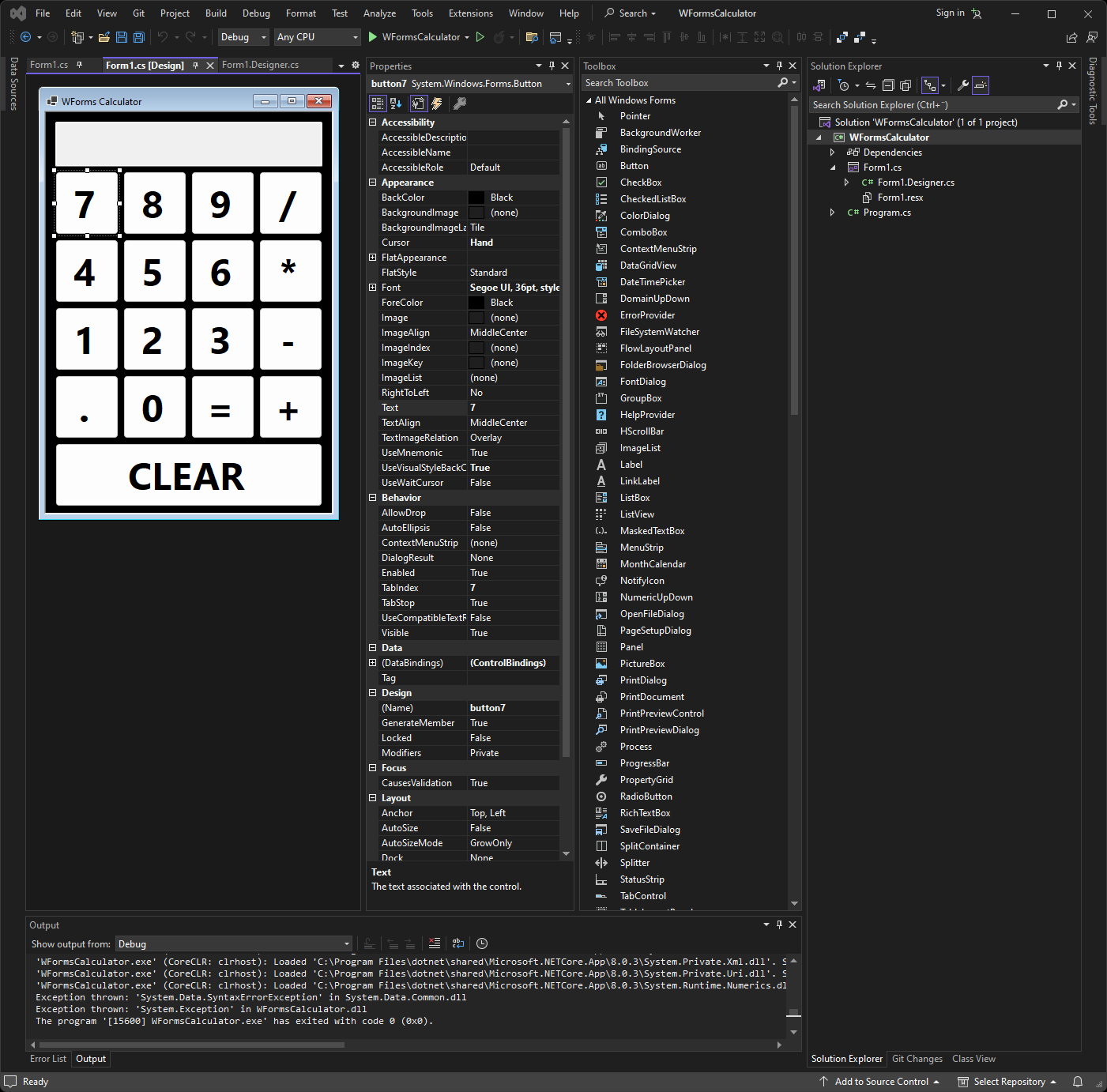
CareerFoundry är en lärningsplattform för utvecklare. De har ju då påföljande detta ganska höga krav på att vara korrekta med sin information. Den webbsida jag refererar ifrån har skribentens fulla namn, bild på skribenten, uppdateringsdatum för sidan etc. Sedan längst ner finns det en komplett beskrivning av skribenten och hennes egna hemsida som har mer information samt formulär för att kontakta henne direkt. Jag tycker att denna källa uppnår en godtagbar standard då den uppfyller dessa krav.

### GitHub

Den källa jag plockat från GitHub är en som de själva skrivit om sin egen verksamhet, och då anser jag att det mesta jag skrivit om Microsoft stämmer även här. De har ingen anledning att vilseleda läsaren angående sin egen verksamhet då detta inte skulle ha någon form av positiv förändring för deras företag. Det saknas skribentens namn, uppdateringsår etc. vilket tyvärr sänker dess trovärdighet. Men jag läser hellre om GitHub på GitHub:s egna sida än från en tredje part.

# 7. Bilagor

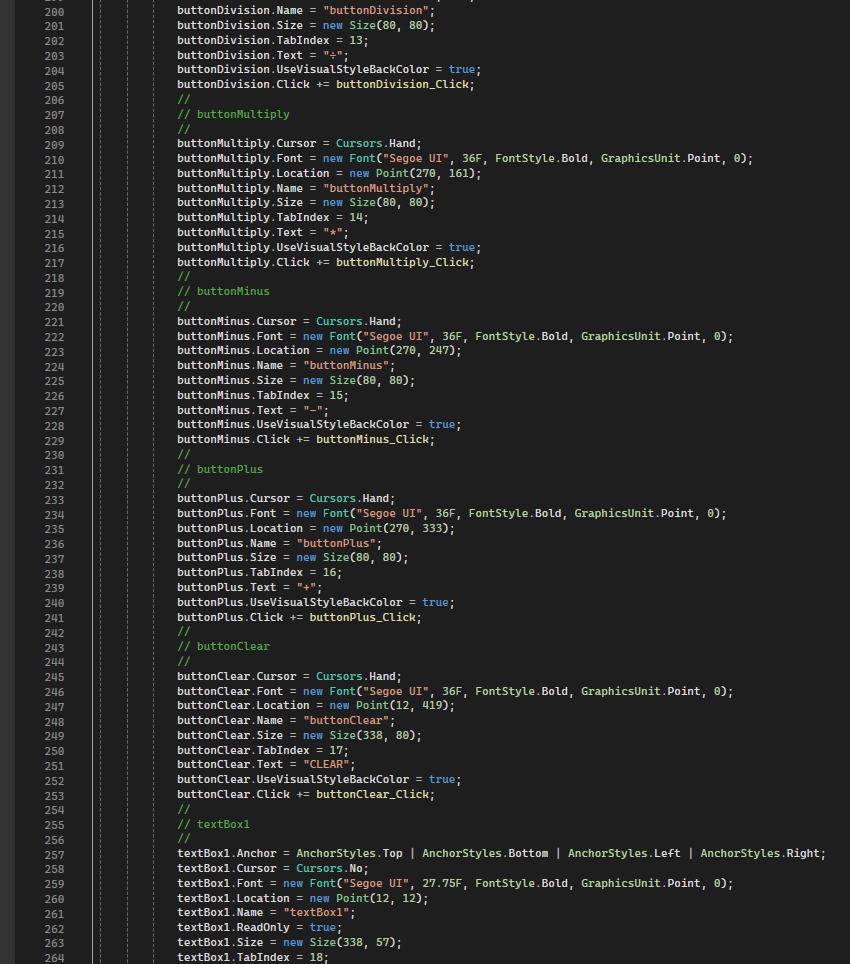
## Bilaga 1.



### Bilaga 1 Bildtext

Skärmdump från Visual Studio som visar designgränssnittet och dess funktioner såsom de olika elementen som kan läggas till i programfönstret via Drag-and-Drop-funktionalitet. Utöver detta visas även fältet med det markerade elementets egenskaper, såsom positionering, storlek, typsnitt etc.

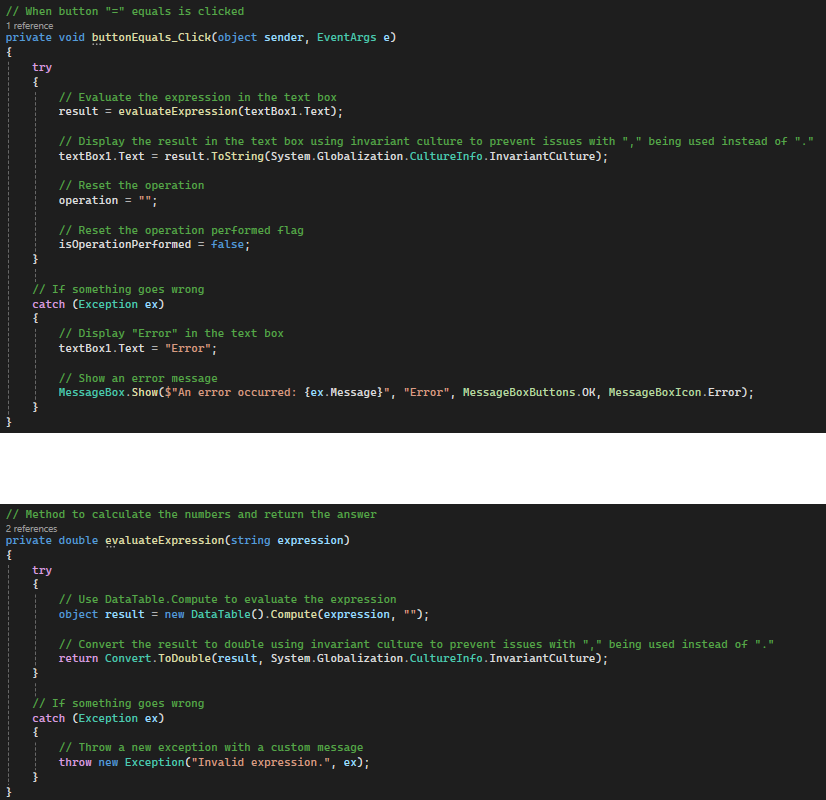
## Bilaga 2.



### Bilaga 2 bildtext

Skärmdump från källkoden till designgränssnittet i Windows Forms, som innehåller all styling för elementen. Här står det bl.a vilken storlek, position och typsnitt som programmet skall ha. Detta är kopplat direkt till bilaga 1, då ändringar som görs i designgränssnittet uppdaterar koden här. Samma process fungerar åt andra hållet också, man kan skriva direkt i koden om man vill och designgränssnittet uppdateras automatiskt.

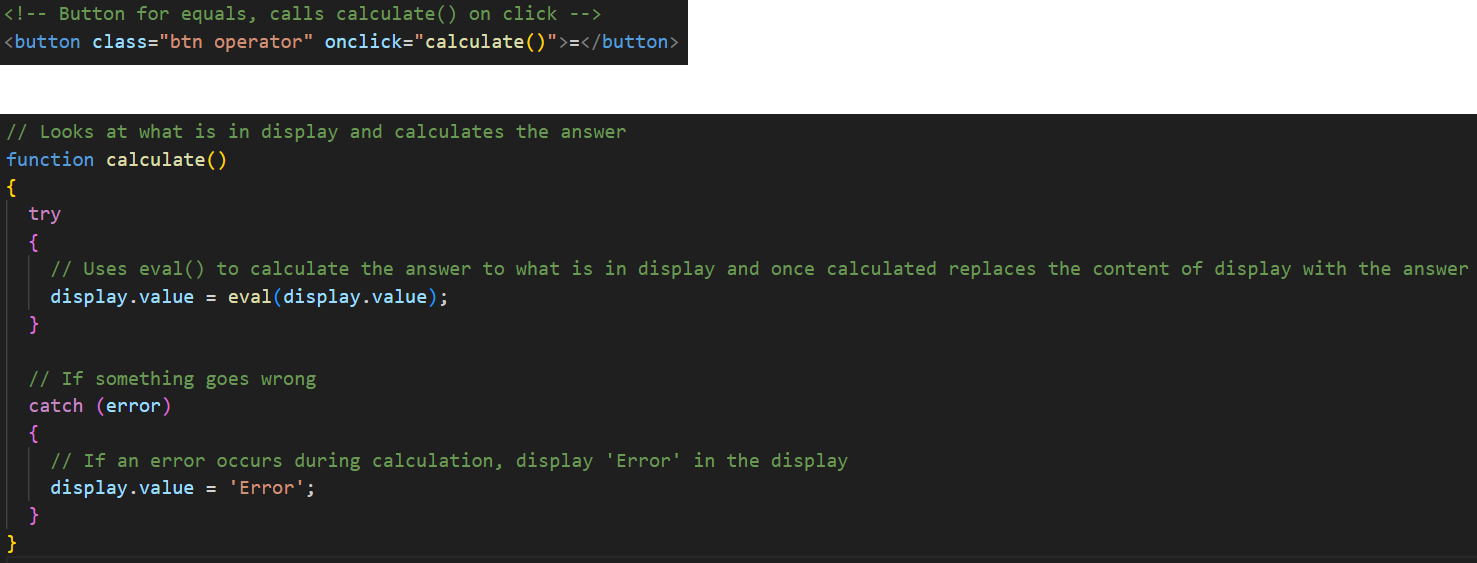
## Bilaga 3.



### Bilaga 3 Bildtext

Två skärmdumpar från C#-räknarens källkod. Den översta visar källkoden till vad som händer när “=” knappen trycks på. En av de sakerna är att anropa metoden “evaluateExpression” som syns i den undre bilden. Den undre bilden visar “evaluateExpression”

## Bilaga 4.



### Bilaga 4 Bildtext

Två skärmdumpar från JavaScript-räknarens källkod. Den översta visar källkoden till HTML-knappen som anropar funktionen som utför uträkningen. Den undre bilden är källkoden till funktionen som anropas.