

OBJEKTORIENTERAD PROGRAMMERING I C#.NET

.NET
Intro C#

IDAG

- Vad är .NET?
- IDE och Setup av kodmiljön
- Variabler och kodblock
- Flödeskontroll
- Loopar

VAD ÄR .NET?



.NET

.NET kan förstås som...

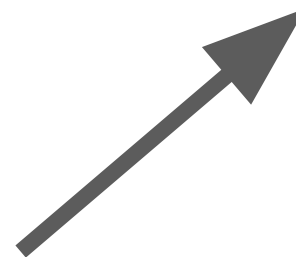
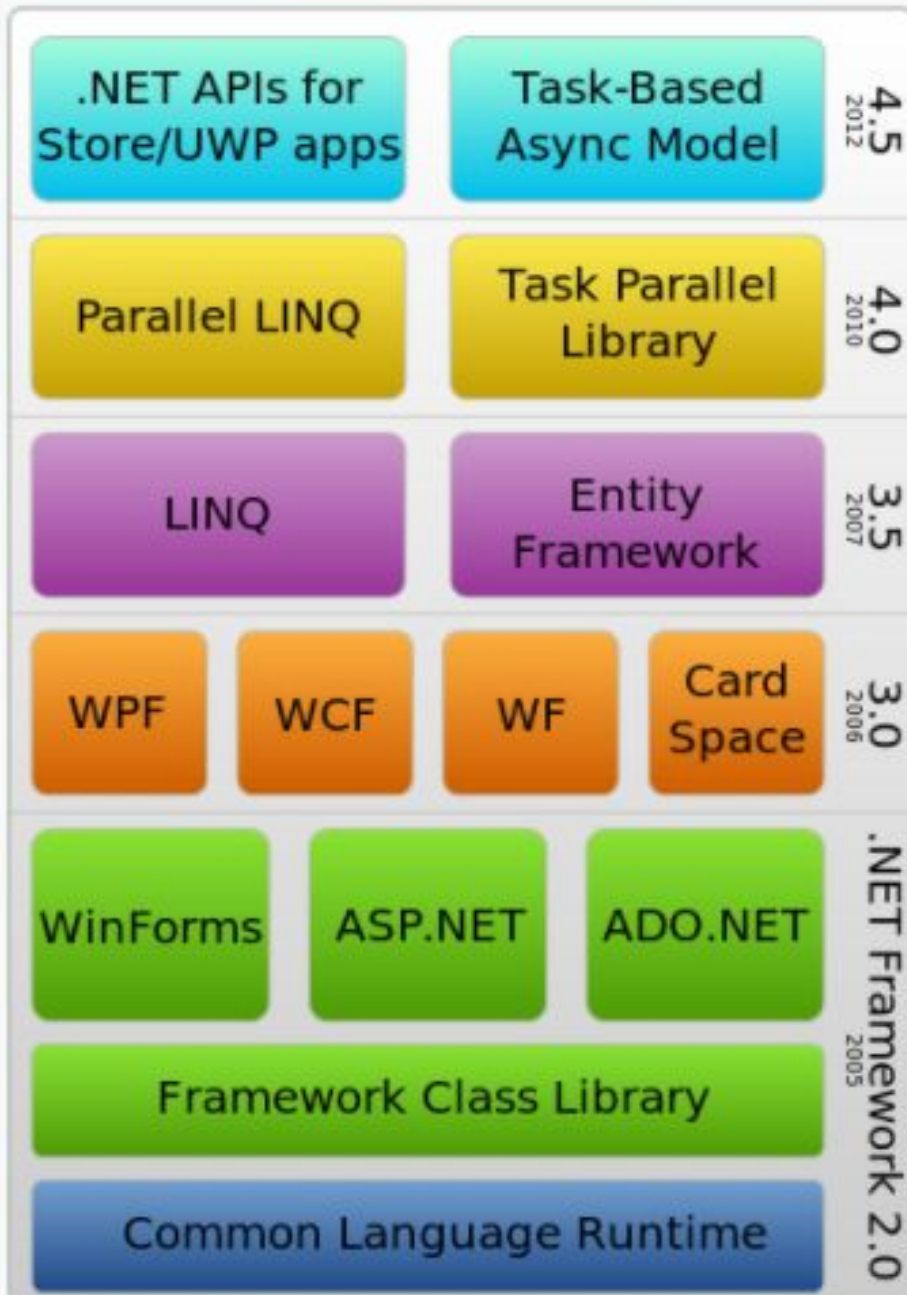
- Ett ramverk (framework) för att utveckla applikationer
- En parser för flera programmeringsspråk
- En kompilator för att köra program skrivna för .NET miljön

.NET

- Från början en del av Windows (första version i Windows Server 2008 & Windows Vista)
- Idag ett open-source framework som kan installeras i Windows- och Unix-miljöer (Linux & Mac till exempel).

.Net Framework
vs
~~.Net Core~~

.NET Framework



| Version | Release date |
|---------------|----------------------------|
| .NET Core 1.0 | 2016-06-27 ^[21] |
| .NET Core 1.1 | 2016-11-16 ^[22] |
| .NET Core 2.0 | 2017-08-14 ^[23] |
| .NET Core 2.1 | 2018-05-30 ^[24] |
| .NET Core 2.2 | 2018-12-04 ^[25] |
| .NET Core 3.0 | 2019-09-23 ^[26] |
| .NET Core 3.1 | 2019-12-03 ^[27] |
| .NET 5 | 2020-11-10 ^[28] |
| .NET 6 | 2021-11-08 ^[29] |
| .NET 7 | 2022-11 (projected) |
| .NET 8 | 2023-11 (projected) |

.NET Framework, .NET

.NET är en nystart på .NET Framework:

- Omskrivet från scratch med moderna tekniker
- Open Source (Vad innebär det i detta fall?)
- Mycket bättre prestanda
- Crossplattform på riktigt (Mac, Linux, Windows osv)
- Framtiden för .NET

Senaste versionen av .NET Framework är 4.x. **Ingen ny version kommer!**

Aktuell versionen av .NET är 6.0

VARNING FÖR FÖRVIRRING!

STANDARD BIBLIOTEK

- .NET har en stor mängd färdiga lösningar för nätverkande, filhantering, konsolapplikationer, grafiska applikationer med mera.
- Dessa är indelade i moduler och används olika beroende på vilket språk man använder.

SPRÅKOBEROENDE

- .NET Framework och C# utvecklades samtidigt av Microsoft
- .NET Framework lanserades med en öppen standard kallad *Common Language Infrastructure* (CLI) för hur andra språk kan användas för ramverket.
- Genom CLI finns det en uppsjö av språk som fungerar med .NET:
 - F#, VisualBasic, IronPython, J#, cLisp, [lua.NET](#) mm

KOMPILATOR

- All kod tänkt att användas med .NET kompileras till **Bytekod** för .NET's runtime.
- .NET kompilerar koden till maskinkod under körning, detta kallas för *Just In Time*-kompilering.
- Detta för att få en bra balans på felåterhämtning och hastighet.
- I introkursen är detta utanför ditt huvudsakliga intresseområde, men kul att veta! 😊

SAMMANFATTNING

- En del av Windows
- .NET är tillgänglig på andra plattformar
- Standardbibliotek
- Språkoberoende
- JIT kompilator

SETUP AV KODMILJÖN

KODMILJÖ - IDE

Integrated Development Environment.

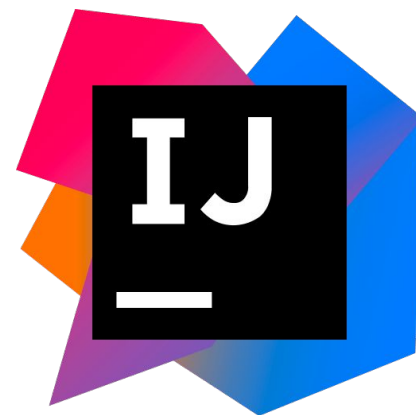
- Används för att få hjälp att skriva programmeringskod.
- Utvecklarens bästa vän
- Är en avancerad texteditor med många hjälpfunktioner.
- All kod kan skrivas i notepad men det är sjukt svårt och tar onödigt långt tid!

KODMILJÖ - IDE

Finns en uppsjö med olika IDE:er

De har alla olika features men gör i grunden samma sak

Vi kommer i den här kursen använda Visual Studio Code.



Installation av VS Code

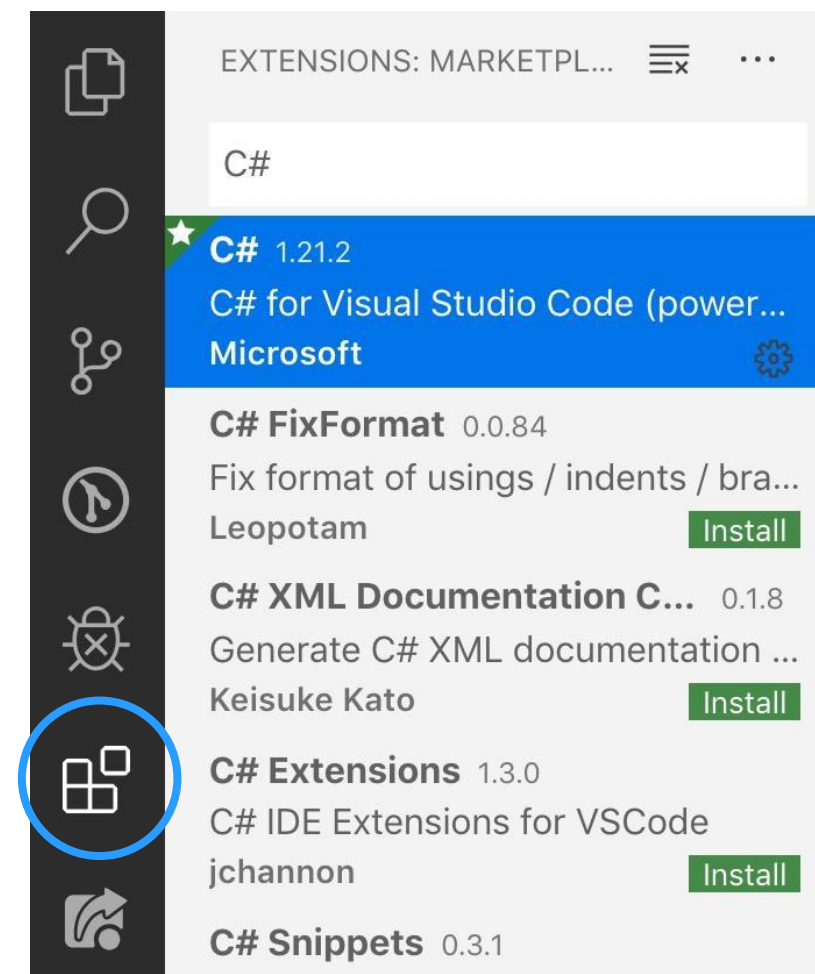
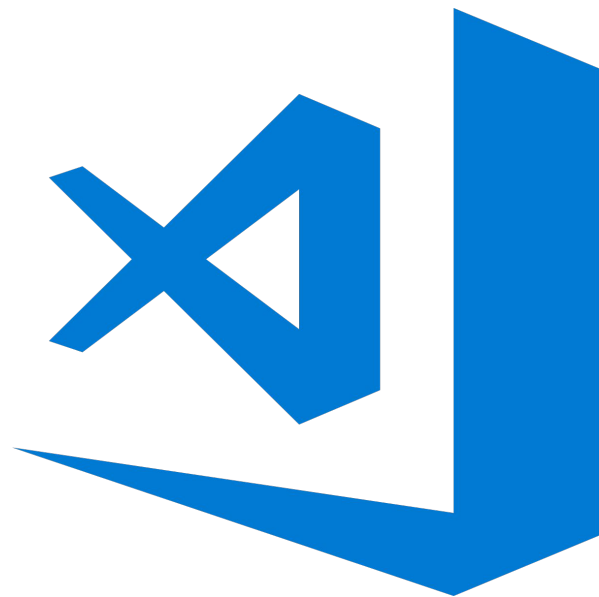
Hur har det gått?

KODMILJÖ - VS CODE

Ladda ner och installera VS-code:

<https://code.visualstudio.com/download>

Öppna sedan VS Code och installera tillägget - C#



KODMILJÖ - .NET

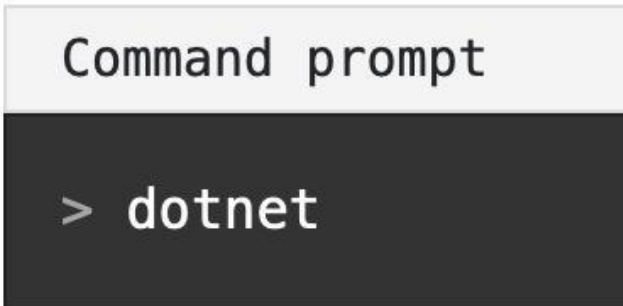
Installera .NET SDK på din dator

<https://dotnet.microsoft.com/download>

- Välj SDK

För att se så att dotnet är korrekt installerat ska det nu gå att skriva "*dotnet*" kommandot i Kommandotolken/Terminalen.

(tryck enter och se vad du får för svar)



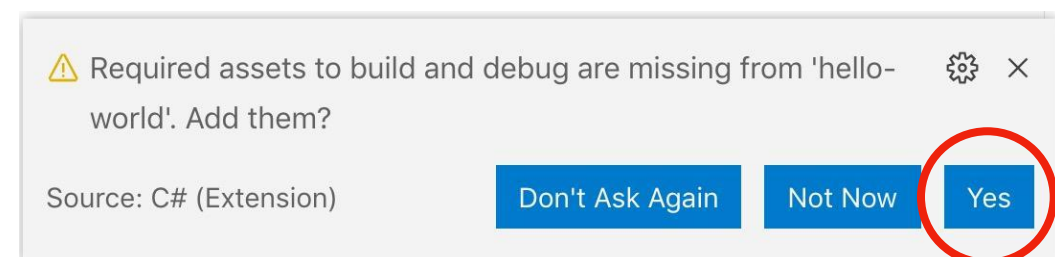
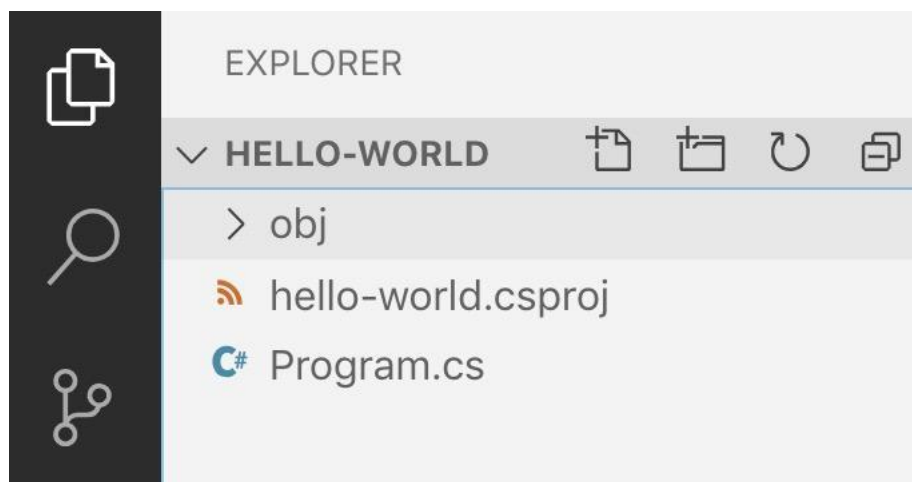
Command prompt

> dotnet

KODMILJÖ - HELLO WORLD

- Skapa en ny mapp som heter "hello-world" på din dator - lägg mappen på ett lämpligt ställe.
- Öppna sedan mappen i VS Code
- Nu är det dags att skapa ditt första program - öppna terminalfönstret i VS Code och skriv: *"dotnet new console"*

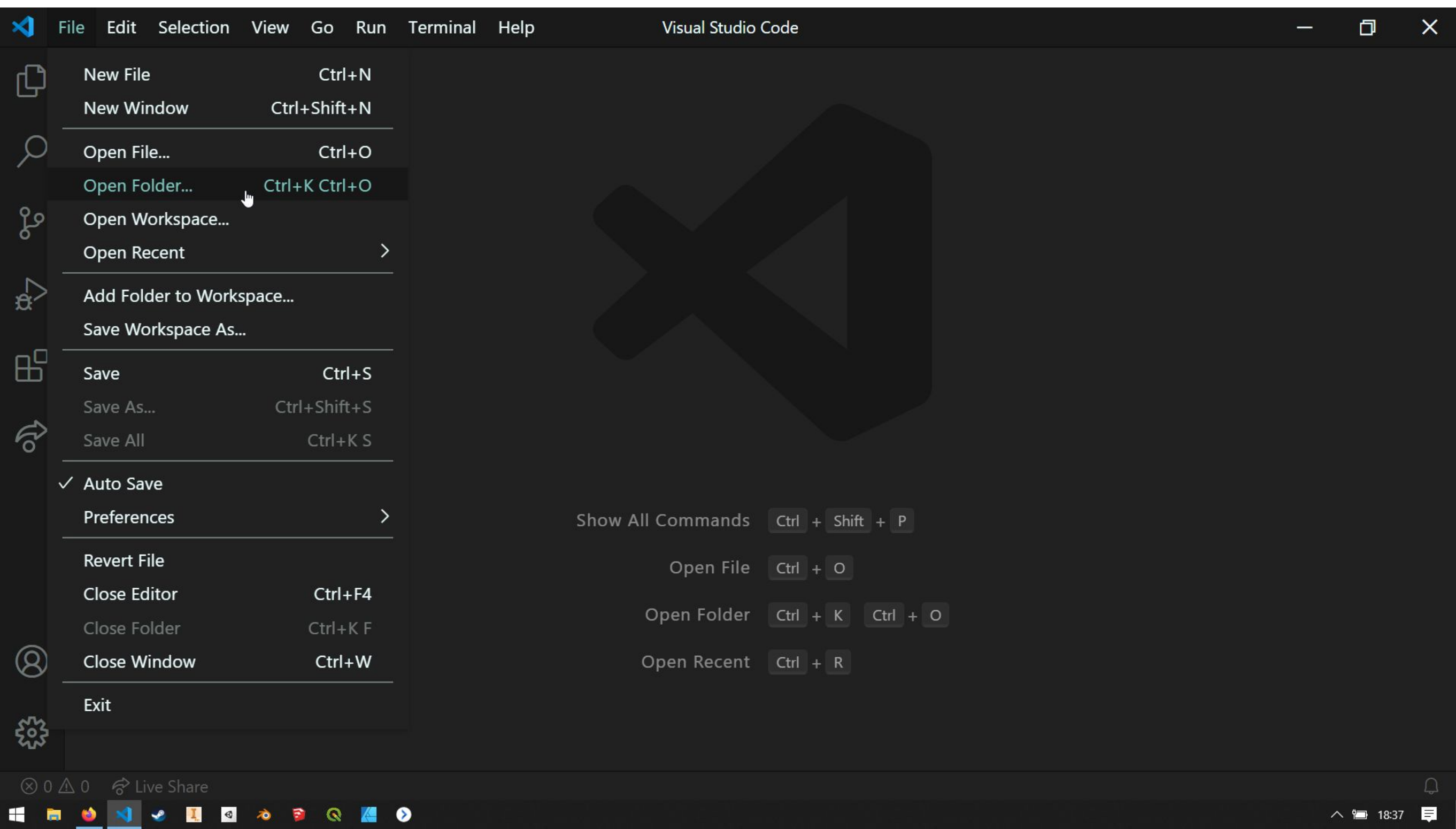
Du bör nu se följande filer i VSCode - klicka på "Program.cs"
Tryck på "Yes" om du får upp följande ruta.



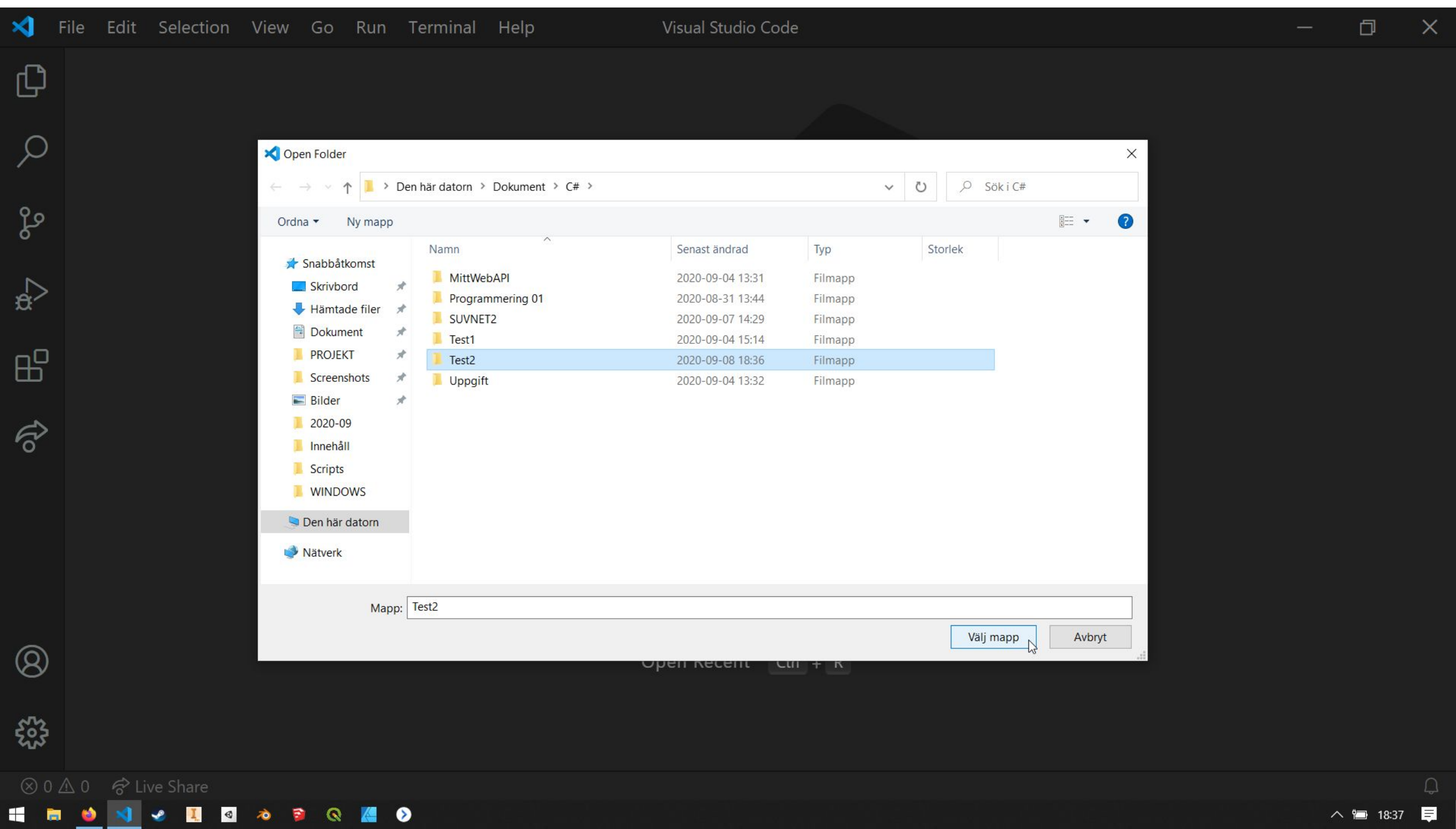
Det går nu att starta programmet genom att klicka på "play"-knappen som du hittar under debug-fliken i VS code.

Köra programmet i terminalfönstret med kommandot: *"dotnet run"*

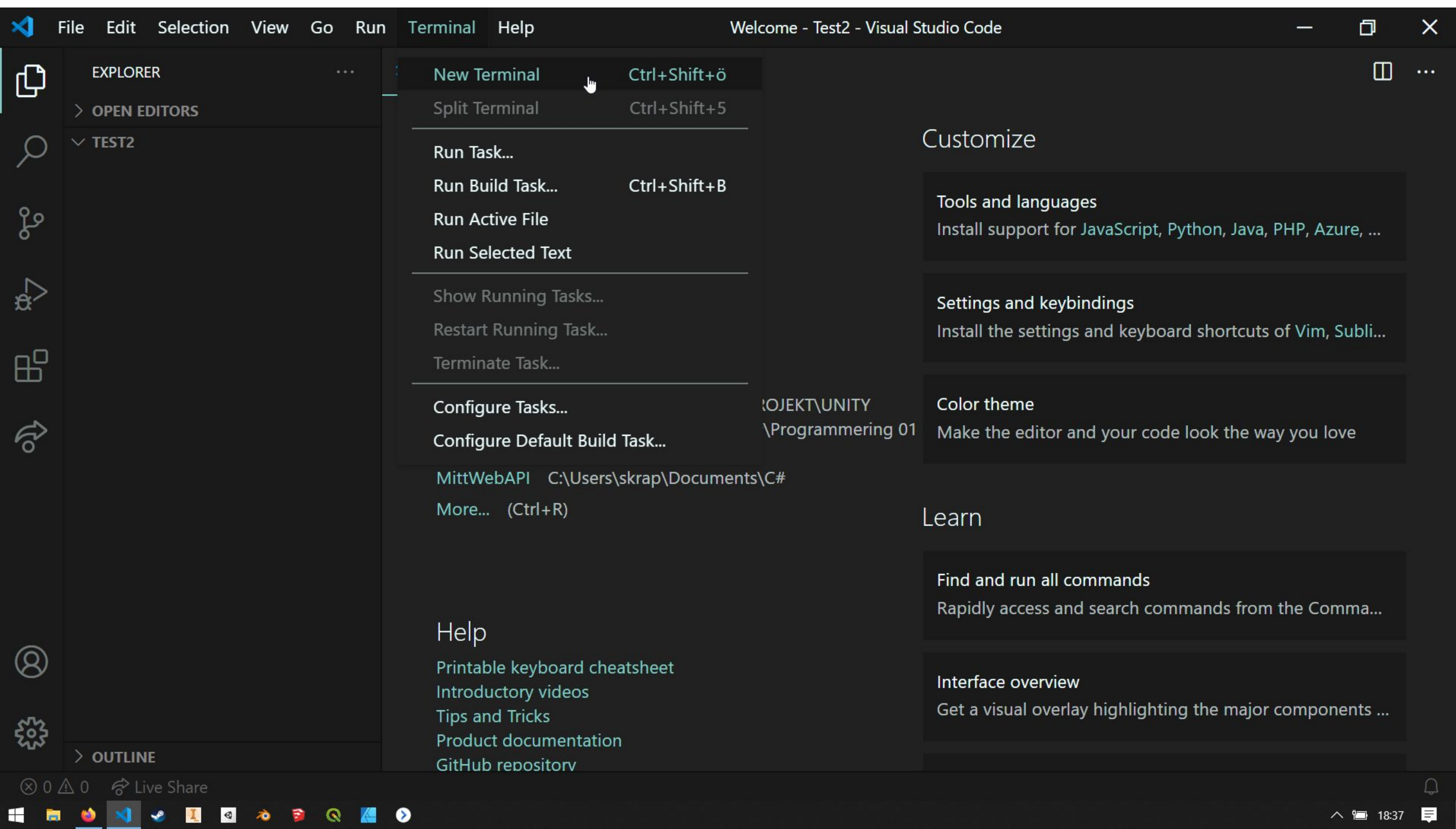
Börja med att välja “Open Folder”



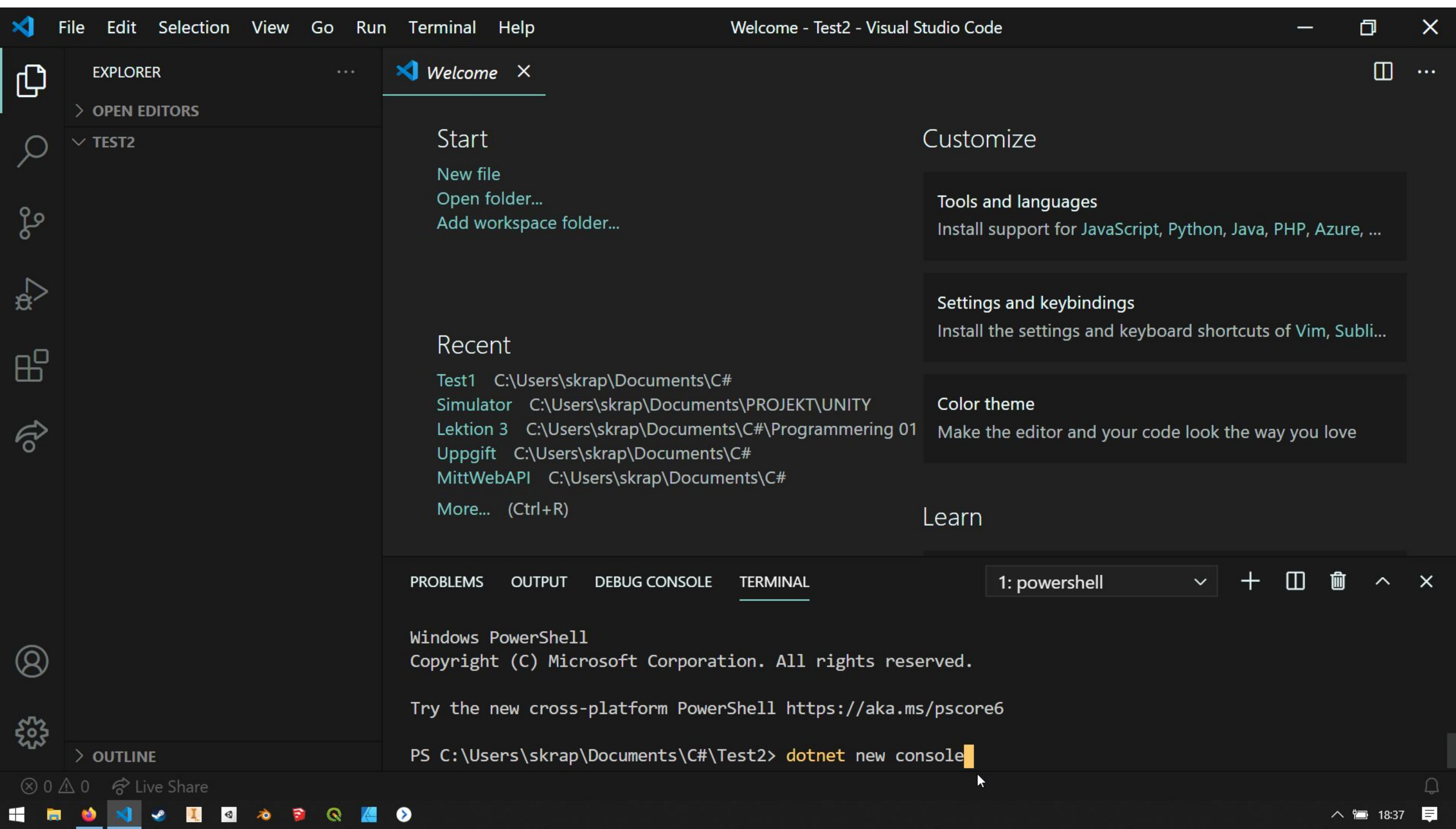
Skapa en “Ny mapp” och döp den till vad du vill.
Se till att den är markerad och klicka på “Välj mapp”



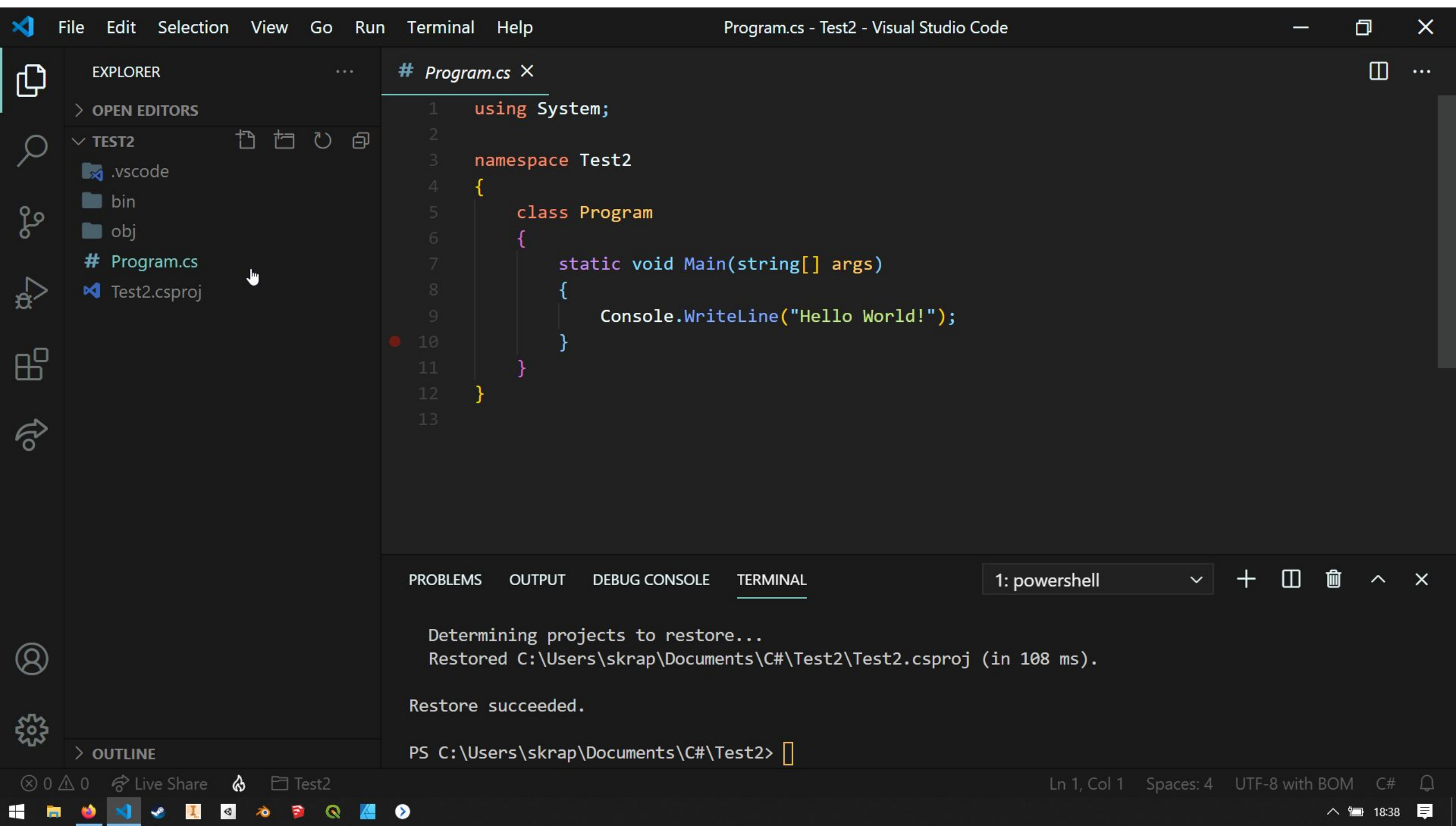
Öppna ett terminalfönster genom att klicka på Terminal -> New Terminal



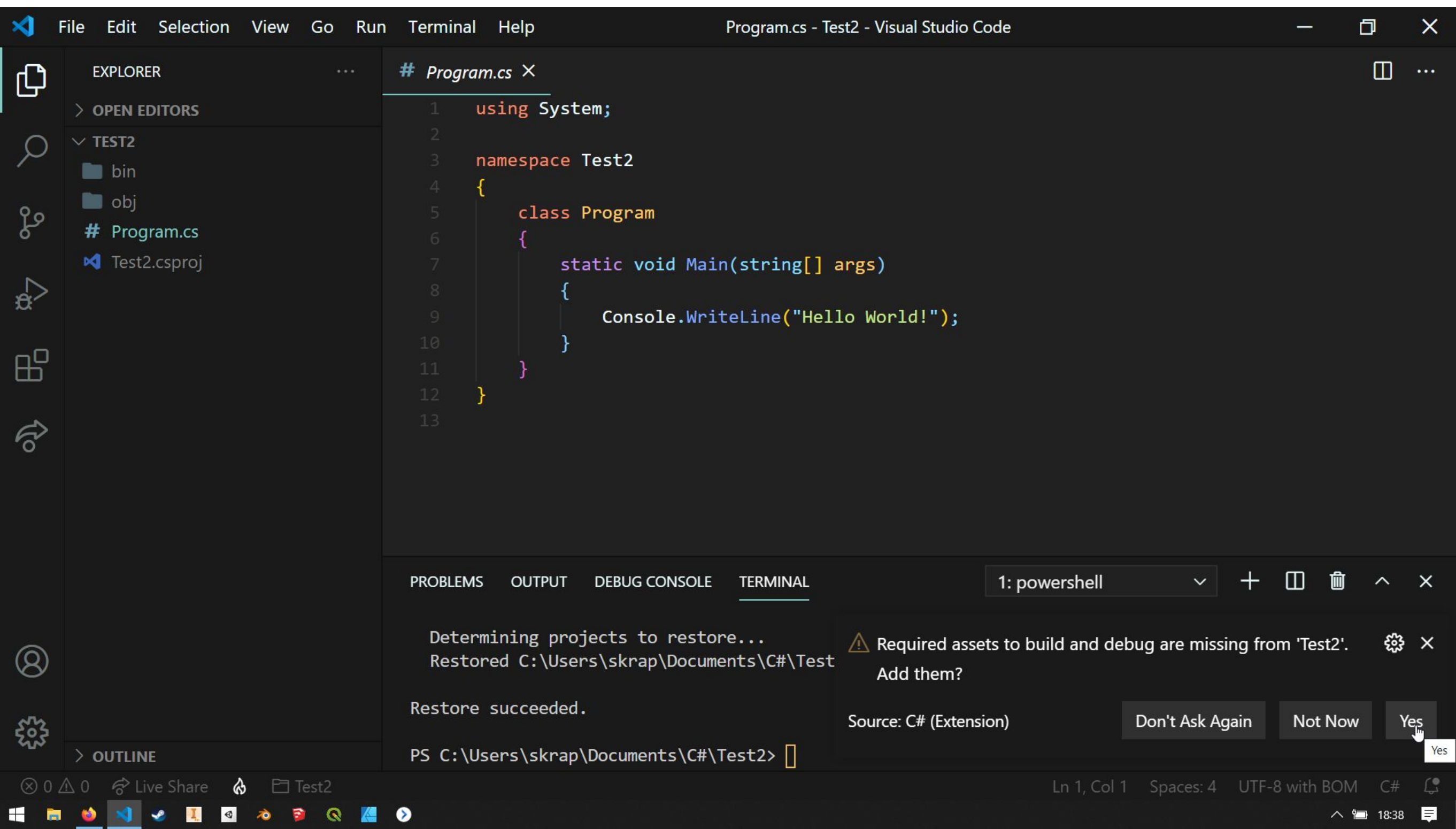
Starta ett nytt projekt genom att skriva dotnet new console



Klicka på filen Program.cs för att komma till koden.



Klicka Yes på rutan som ploppar upp nere i högra hörnet.



KOMMANDON I TERMINALEN

| | Mac/Linux | Windows |
|-----------------------------------|---------------------|---------------------|
| KOMMANDO | BASH | CMD/POWERSHELL |
| Byta mapp/katalog | <code>cd</code> | <code>cd</code> |
| Lista filer | <code>ls</code> | <code>dir</code> |
| Skapa katalog | <code>mkdir</code> | <code>mkdir</code> |
| Ta bort katalog | <code>rmdir</code> | <code>rmdir</code> |
| Ta bort fil | <code>rm</code> | <code>del</code> |
| Rensa all text | <code>clear</code> | <code>cls</code> |
| .net-kommandot | <code>dotnet</code> | <code>dotnet</code> |
| Öppna VS Code i nuvarande katalog | <code>code .</code> | <code>code .</code> |

KOMMANDON I TERMINALEN

Navigera mellan mappar:

- `.` -> Samma katalog som vi är i.
- `..` -> Den överliggande katalogen.
- `~` -> Din hemkatalog.

Exempel:

`cd ..` -> Gå tillbaka

`cd folder` -> Gå in i en katalog

`cd C:\Windows` -> Gå till Windowskatalogen på C-hårddisken

LS funkar på samma sätt!

KOMMANDON I TERMINALEN

Öva, öva, testa, gör fel, gör om, gör rätt osv.

Undersök vad som händer när du gör olika saker.

Vanligt fel: Fel katalog öppen i VS Code

Bra inställningar

Inställningar:

- Autospara:

Commonly Used

- > Text Editor
- > Workbench
- > Window
- > Features
- > Application
- > Extensions

Commonly Used

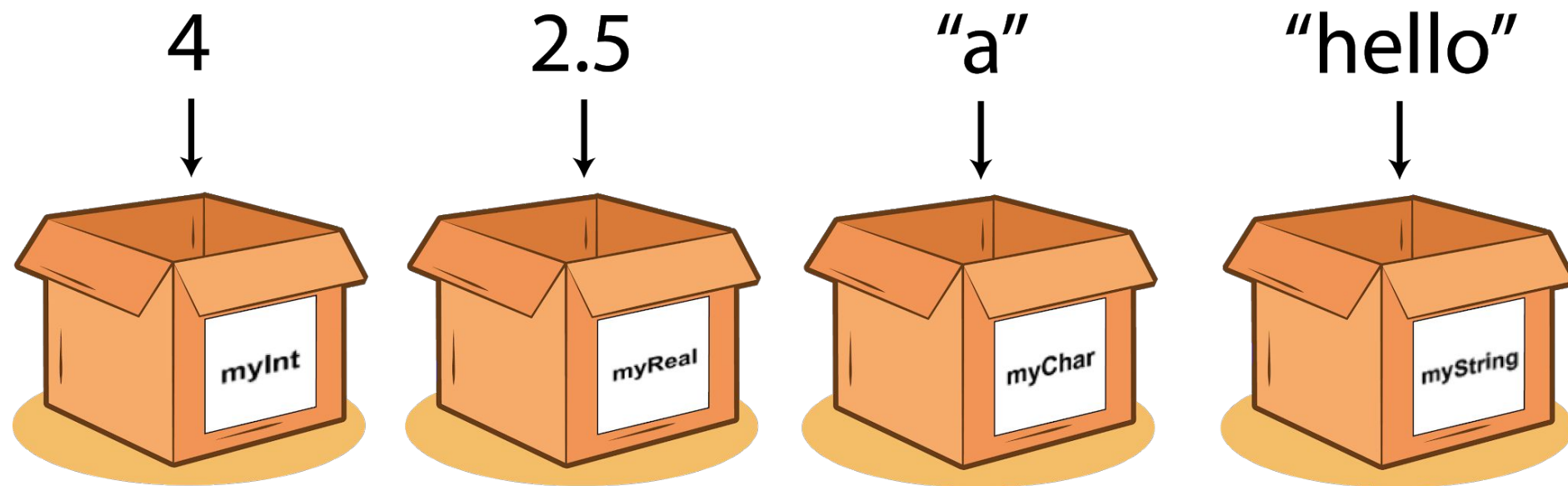
Files: Auto Save
Controls auto save of dirty editors. Read more about autosave [here](#).

onFocusChange

Files: Auto Save Delay
Controls the delay in ms after which a dirty editor is saved automatically.
Files: Auto Save is set to **afterDelay**.

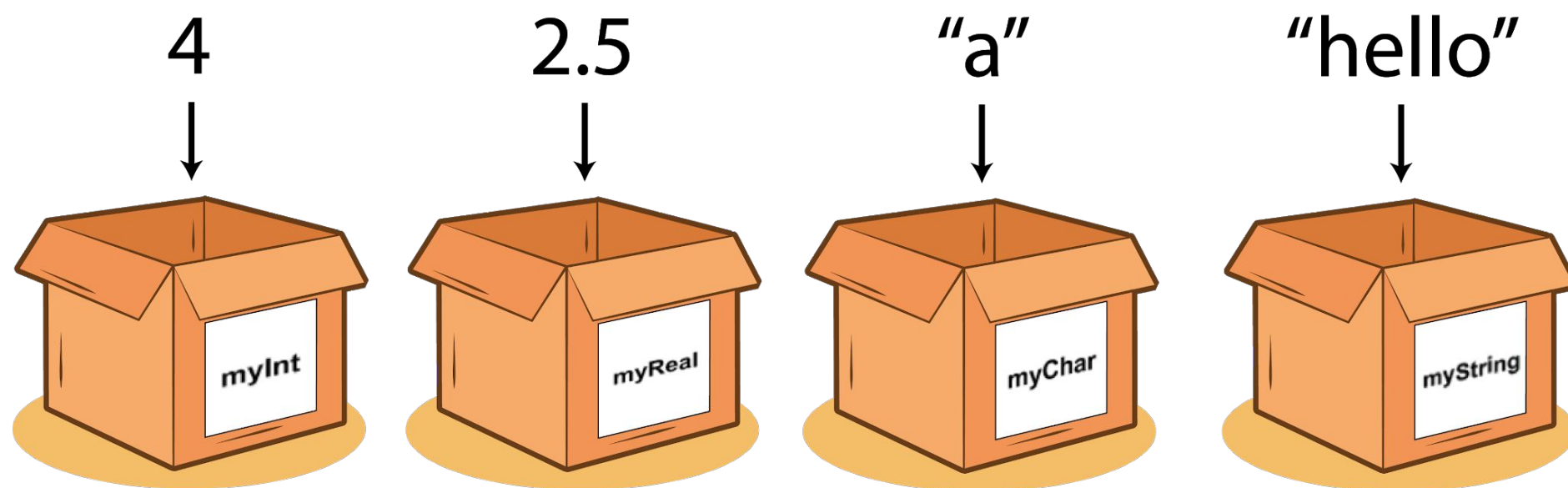
C# Intro

VARIABLE

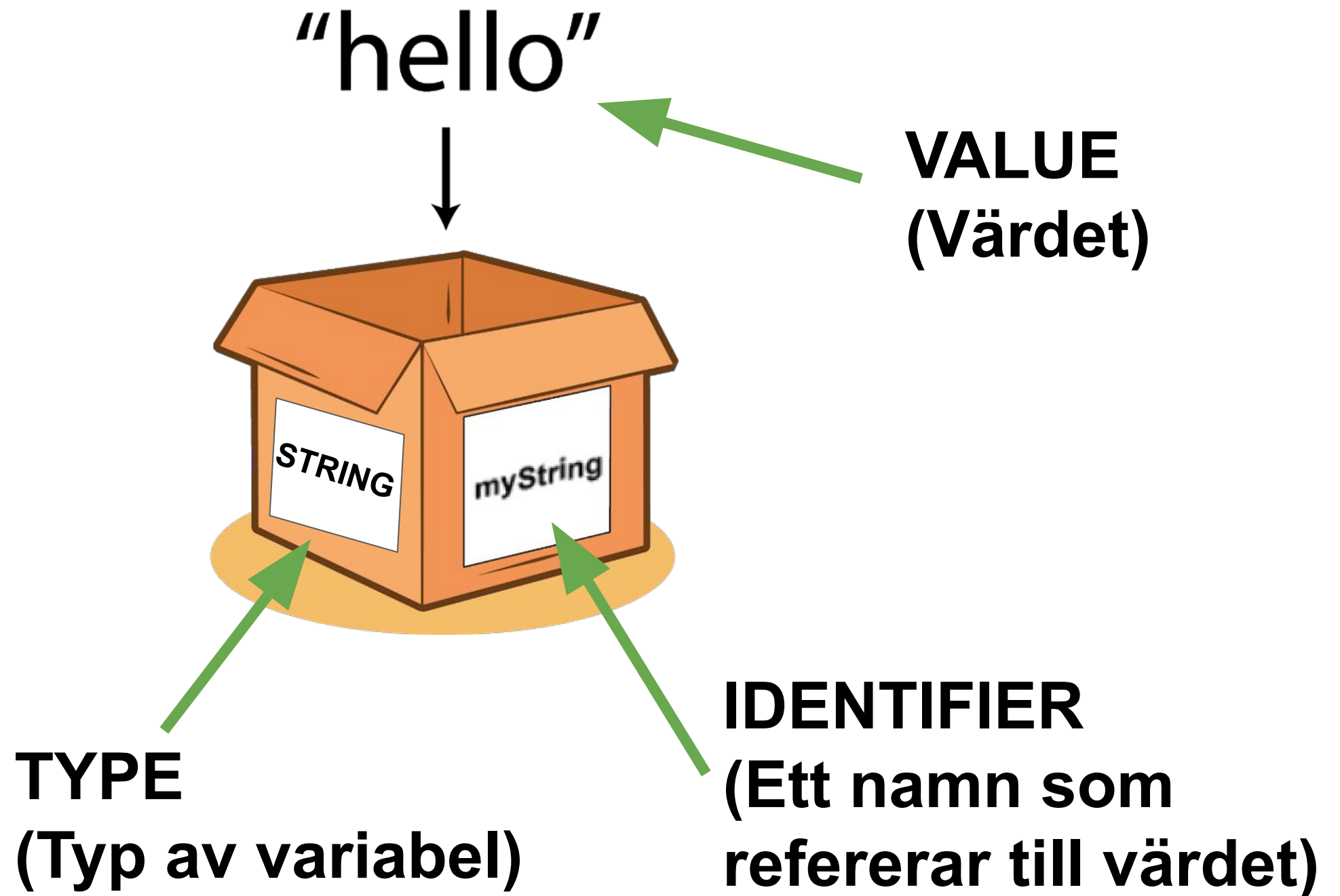


VAD ÄR EN VARIABEL?

- En variabel kan ses som en låda med en etikett på. På etiketten står det ett namn och i lådan ligger något.
- I C# måste vi bestämma vad som kan ligga i en låda. Bananer ligger i lådan för bananer och äpplen ligger i lådan för äpplen. Att lägga ner bananer i lådan med äpplen går inte. Då skulle kompilatorn klaga.



VAD ÄR EN VARIABEL?



VARIABELTYPER

| Variabeltyp | Beskrivning |
|---------------|---|
| byte | 8-bitars heltal |
| short | 16-bitars heltal |
| int | 32-bitars heltal |
| long | 64-bitars heltal |
| float | 32-bitars decimaltal |
| double | 64-bitars decimaltal |
| char | 16-bitars tecken |
| string | Textsträng |
| bool | Booleansk variabeltyp, kan bara ha värdena true eller false |

VARIABELTYPER

- I `c#` kallas variabeltypen för att hantera ett heltal `int`.
- Enkelt förklarat kan man säga att vad som skiljer 8-, 16-, 32- och 64-bitars variabler från varandra är deras storlek. Alltså hur stora tal som får plats i variabeln.
- Genom att använda `float` eller `double` kan vi lagra decimaltal i en variabel. En `float` är ett 32-bitars decimaltal och en `double` är ett 64-bitars.

INT: -2 147 483 648 **to** 2 147 483 647

VARIABELTYPER

| Variabeltyp | Beskrivning |
|-------------|---|
| byte | 8-bitars heltal |
| short | 16-bitars heltal |
| int | 32-bitars heltal |
| long | 64-bitars heltal |
| float | 32-bitars decimaltal |
| double | 64-bitars decimaltal |
| char | 16-bitars tecken |
| string | Textsträng |
| bool | Booleansk variabeltyp kan bara ha värdena true eller false |

VARIABELTYPER - STRING

- En variabel av typen **string** kan i C# helt enkelt innehålla text
- Vi kan även plussa ihop strängvariabler. Observera att det är " " mellan de två plussen. Detta för att få ett mellanslag.

```
string myString = "Detta är en textsträng!";  
Console.WriteLine(myString);
```

I konsolfönstret:

Detta är en textsträng!

```
string myString = "Detta är en textsträng!";  
string newString = "Detta är också en textsträng!";  
Console.WriteLine(myString + " " + newString);
```

VARIABELTYPER - CHAR

- Variabeln av typen `char` kan bara innehålla ett enda tecken.
- Första kodexemplet blir 'a' i konsolfönstret.
- Men vad som är intressant med `char` är att det är en bokstav som resresenteras av en siffra enligt UNICODE.
- Kodexempel 2 blir också 'a' i konsolfönstret.

```
char myChar = 'a';  
Console.WriteLine(myChar);
```

```
char myChar = (char)97;  
Console.WriteLine(myChar);
```

KORT OM : UNICODE

- En global standard för hur siffervärden ska tolkas som tecken
- Implementerat likadant i så gott som 100% i alla moderna frameworks.
- Innehåller många tusentals olika karaktärer (**char**)
- Finns i flera olika utföranden UTF-8, UTF-16, och UTF-32
- Alla emojis 🤪 är en del av unicode standarden

Grinning Face With One Large and One Small Eye

<https://emojipedia.org/grinning-face-with-one-large-and-one-small-eye/>

U+1F92A (hexadecimal unicode notering)

VARIABELTYPER - BOOL

- En variabel av typen `bool` är en variabel som endast kan innehålla ett av två värden, **true** eller **false**.
- Booleanska variabler är framförallt användbara i **if-satser** och **loopar**.
- Utskriften i exemplet blir:

myBool är nu = True

myBool är nu = False

```
bool myBool = true;  
Console.Write("myBool är nu = " + myBool);  
myBool = false;  
Console.Write("myBool är nu = " + myBool);
```


KONSTANTER

- Variabler är föränderliga. I C# kan man också arbeta med konstanter. Vi kan enklast se på dem som oföränderliga variabler.
- En variabel görs om till en konstant med hjälp av ordet **const** framför. Vi måste direkt ange variabelns värde annars klagar kompilatorn
- Vad är fördelarna med en konstant jämfört med en variabel?
- När en variabel angetts med ordet **const** vet man att den anges där och kommer aldrig att ändras.

```
const int nr = 100;
```

```
const int nr; // FEL!!  
nr = 100;
```

ANVÄNDA VARIABLER

- Först deklarerar man vilken typ av variabel & därefter namnet.
- För att spara ett värde i en variabel används tilldelningsoperatorn "="
- Nya variabler innehåller inget (**NULL**) innan vi tilldelat dem ett värde. Detta kallas för variabelns initialisering.

- `int`, lagrar heltal (t ex -2, 1, 3...)
- `double`, lagrar decimaltal (t ex 12.34, 1.872)
- `string`, lagrar text

```
int intvariable = 56;
```

```
string strvariable = "Meera Academy";
```

```
char charvariable = 'M';
```

```
float floatvariable = 56.2F;
```

```
double doublevariable = 123.56;
```

```
bool boolvariable = true;
```

```
int firstNumber;  
int secondNumber;  
int integerAnswer;
```

```
firstNumber = 10;  
secondNumber = 32;
```

```
integerAnswer = firstNumber + secondNumber;
```

Det som ligger till höger om tilldelningsoperatorn händer först.

ANVÄNDA VARIABLER

- Nu kan vi tex. Skriva ut våra variabler med hjälp av `Console.WriteLine()` metoden.
- Efter att man skapat variabeln och använt den kan man fortsätta att använda den.

```
namespace ConsoleApp1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            int nr = 10;
            Console.WriteLine(nr);
        }
    }
}
```

```
{
    int nr = 10;
    Console.WriteLine(nr);
    nr = 200;
    Console.WriteLine(nr);
}
```

LÄSA IN DATA FRÅN ANVÄNDARE

- Detta kan vi göra med hjälp av metoden **Console.ReadLine()**
- Vi sparar texten som användaren skriver in i en **string** variabel
- **Console.Write()** lägger inte till en blankrad efter det gör däremot **Console.WriteLine()**.

```
namespace ConsoleApp1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.Write("Skriv något: ");
            string readText = Console.ReadLine();
            Console.Write("Du skrev följande text: ");
            Console.Write(readText);
            Console.ReadLine();
        }
    }
}
```

LÄSA IN HELTAL FRÅN ANVÄNDAREN

Feler vi får är **Cannot implicitly convert type 'string' to 'int'**. Felet uppstår för att `Console.ReadLine()` är en metod som bara tar emot strängar. Nu måste vi konvertera *int* till *string*!

```
{  
    Console.WriteLine("Skriv ett nummer: ");  
    int nr = Console.ReadLine(); //FEL!  
    nr = nr + 100;  
    Console.WriteLine("Ditt nummer plus 100 blir: ");  
    Console.WriteLine(nr);  
}
```

Med hjälp av metoden `Convert.ToInt32()`

Har vi nu omvandlat det inmatade talet till en *string* som vi sen kan skriva ut.

```
{  
    Console.WriteLine("Skriv ett nummer: ");  
    string strNr = Console.ReadLine();  
    int nr = Convert.ToInt32(strNr);  
    nr = nr + 100;  
    Console.WriteLine("Ditt nummer plus 100 blir: ");  
    Console.WriteLine(nr);  
}
```

OPERATORER

Jämföresleoperatorer:

| Tecken | Betydelse |
|--------|--------------------------|
| == | Lika med |
| < | Mindre än |
| > | Mer än |
| <= | Mindre än eller lika med |
| >= | Mer än eller lika med |
| != | Inte lika med |

Logiska operatorer

| Tecken | Betydelse |
|--------|-----------|
| && | Och |
| | Eller |
| ! | Inte |

OPERATORER

Räkneoperatorer

| Tecken | Betydelse |
|--------|---------------------|
| + | Addition |
| - | Subtraktion |
| * | Multiplikation |
| / | Division |
| ++ | Öka med ett |
| -- | Minska med ett |
| % | Rest efter division |

Tilldelningsoperatorer

| Tecken | Betydelse |
|--------|------------------------|
| = | Tilldelning (standard) |
| += | Addition |
| -= | Subtraktion |
| *= | Multiplikation |
| /= | Division |
| = | Eller (bool) |
| &= | Och (bool) |

OPERATORER

Att skriva...

foo += bar

Är samma samma som att skriva...

foo = foo + bar

Detsamma är sant för alla tilldelnings operatorer
undantaget standardtilldelningen =

NU TESTAR VI!

Övning: Variabler Nivå 1

1. **Deklarera** och **tilldela** värden till variabler av typen
 - ☐ int
 - ☐ float
 - ☐ string
 - ☐ char
 - ☐ bool
2. **Skriv ut** samtliga värden med **Console.WriteLine();**
3. **Ta in** värden till samtliga variabler med **Console.ReadLine();**
4. Skriv ut dem.
5. Lägg ihop två variabler av typen sträng och skriv ut resultatet
6. Lägg ihop en int och en float och skriv ut resultatet
7. Testa olika situationer, vad funkar och vad funkar inte?

Övning: Svårare

1. Be användaren skriva in en mening.
2. Skriv ut varje teckens int-värde (varje char)
3. Skriv ut varje teckens binära värde separerade med mellanslag (skriv alltså inte ut mellanslag som binära värden)
4. Skapa en caesar-kryptering:
 - a. Låt användaren skriva in en mening som innan
 - b. Låt användaren skriva in ett heltal
 - c. Använd heltalet för att “skifta” varje teckens int-värde (lägg på heltalet till varje chars-värde i meningen)
 - d. Skriv ut meningen på nytt, nu i sin “krypterade” form

FLÖDESKONTROLL

IF

- En **if-sats** är en typ av villkorssats.
- Ett program behöver oftast göra olika val beroende på parametrar och värden, då kan man använda **if-satsen**.
- Med hjälp av villkorssatser kan vi jämföra olika värden med varandra.

Pseudokod:

***“OM något SÅ
gör detta.”***

IF

- Vi kan skriva en `if-sats` och tex. kontrollera hur varmt vattnet är.

OM temperature är 100 SÅ

skriv ut “Nu kokar vattnet” på skärmen.

```
if (temperature > 100)
{
    Console.WriteLine("Nu kokar vattnet!");
}
```

IF

- Att använda **if-satsen** kräver att man i satsen kollar om något är sant eller falskt
- Satsen är det som skrivs mellan paranteser ()
- Ibland vill man ha möjlighet att lägga till fler alternativ, då kan man använda **else** och **else if**.

ELSE

OM temperature är 100

Skriv ut “Nu kokar vattnet!” på skärmen.

ANNARS

Skriv ut “Vattnet har inte exakt 100 grader...”

```
if (temperature > 100)
{
    Console.WriteLine("Nu kokar vattnet!");
}
else
{
    Console.WriteLine("Vattnet kokar inte!");
}
```


ELSE IF

OM temperature är 100 eller mer

Skriv ut "Nu kokar vattnet!"

ANNARS OM temperature är mer än 85

skriv ut "Vattnet har bra-te-temperatur!"

ANNARS SÅ

Skriv ut "Vattnet kokar inte"

```
if (temperature ≥ 100)
{
    Console.WriteLine("Nu kokar vattnet!");
}
else if (temperature ≥ 85)
{
    Console.WriteLine("Vattnet har bra-te-temperatur!");
}
else
{
    Console.WriteLine("Vattnet kokar inte!");
}
```

SWITCH

- **switch** liknar en **if-sats** och är också en typ av selektion
- Tanken är att använda **switch** när vi har många utfall istället för att följa upp med en massa **else if**
- Först skapar vi en helhetsvariabel 'nr'. Vi kan mata in värdet 1 och hamnar då i case 1. Anger vi värdet 2 hamnar vi i case 2. Default tar hand om alla andra värden.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Console.Write("Ange nr: ");
        string str = Console.ReadLine();
        int nr = Convert.ToInt32(str);

        switch (nr)
        {
            case 1:
                Console.WriteLine("case 1");
                break;

            case 2:
                Console.WriteLine("case 2");
                break;

            default:
                Console.WriteLine("default case");
                break;
        }
    }
}
```

KODBLOCK

- C# kod skrivs nästan alltid inom de två tecknen: { }.
- Detta skapar i de allra flesta fall ett *scope* eller ett kodblock. All kod som ligger mellan dessa klammerparenteser (viktor: måsvingar) tillhör samma kodblock.
- Kodblock kallas beroende på sin kontext även för kropp alt. *body*.
- Ett kodblock kan innehålla flera andra kodblock som i dessa fall refereras till som underliggande kodblock, underblock eller *sub-scope*.
- Tex. Innehåller `class Program` metoden `Main()`. `Main()` är ett kodblock som kan innehålla loopar och if-satser, som i sin tur kan innehålla ytterligare loopar och if-satser.

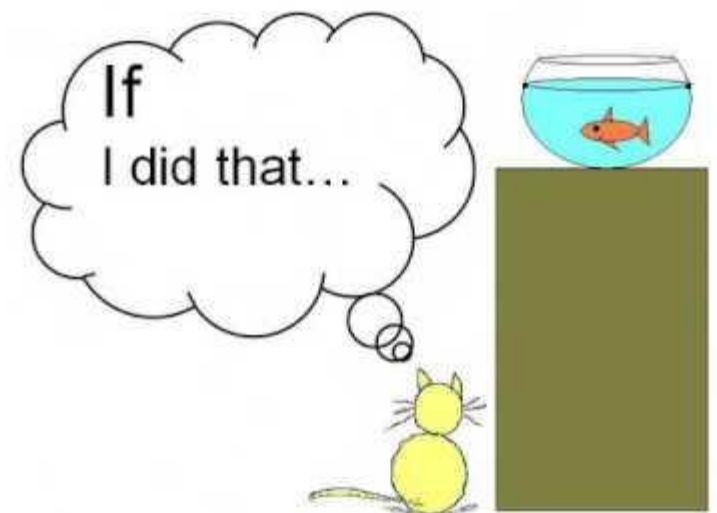
KODBLOCK

```
Kodblock 1
{
    Underkodblock 1.1
    {
        Under-underkodblock 1.1.1
        {
            Osv...
        }
    }
    Underkodblock 1.2
    {
        Osv...
    }
}
Kodblock 2
{
    Osv...
}
```

VARIABELNS LIVSLÄNGD

- En variabel finns endast efter att den har deklarerats.
- Den finns tillgänglig inom det kodblock där den deklarerats, men också i ett kodblocks underliggande kodblock. Inte i övriga kodblock.
- Varför kan inte allt bara vara tillgängligt överallt?

NU TESTAR VI FLÖDESKONTROLL!



Övning FLÖDESKONTROLL

1. Kontrollera värdet (del 1)

Skapa ett program som ställer frågan "Är det fint väder?". Om användaren svara "j" skriver programmet ut "Vi går på picknick!". Annars händer ingenting.

2. Kontrollera värdet (del 2)

Arbeta vidare på övningen ovanför men lägg till att användaren kan svara "n". Då skriver programmet ut "Vi stannar inne och läser en bok".

EXTRA: Det ska inte spela någon roll om användaren matar in stor eller liten bokstav.

Övning FLÖDESKONTROLL

3. Kontrollera värdet (del 3)

Arbeta vidare på övningen. Gör så att om användaren matar in något annat än "J", "j", "n", "N" så skriver programmet ut "Jag förstår inte!".

4. Betygprogram

Skriv ett program som tar in fem olika procentvärden. Beroende på genomsnittlig procent skall olika betyg delas ut, exempelvis 100 = A, 90 = B, 80 = C osv. Under 50 = Underkänt.

Input:
89 76 45 34 95

Output:
Genomsnitt: 67,8%
Betyg: E

Övning FLÖDESKONTROLL Svårare

Skriv ett elpriskalkuleringsprogram

Användaren matar in:

- antalet kWh förbrukade
- vilket prisområde användaren bor
- kundnummer

Programmet skriver ut:

- Användarens kundnummer
- totalpris
- eventuell ransomeringsavgift.

Om antalet förbrukade kWh är mer än 10.000kWh läggs en ransomeringsavgift till de kWh som överstiger 10.000. Ransomwarens avgiften är på 25% för de första 5000kWh och 50% för allt utöver det. Hårda tider!

Dagens spotpris på el

8 SEPTEMBER 2022

SE1

LULEÅ

111,33 öre/kWh

Imorgon: Tillgängligt kl 13:30

SE2

SUNDSVALL

111,33 öre/kWh

Imorgon: Tillgängligt kl 13:30

SE3

STOCKHOLM

305,76 öre/kWh

Imorgon: Tillgängligt kl 13:30

SE4

MALMÖ

305,76 öre/kWh

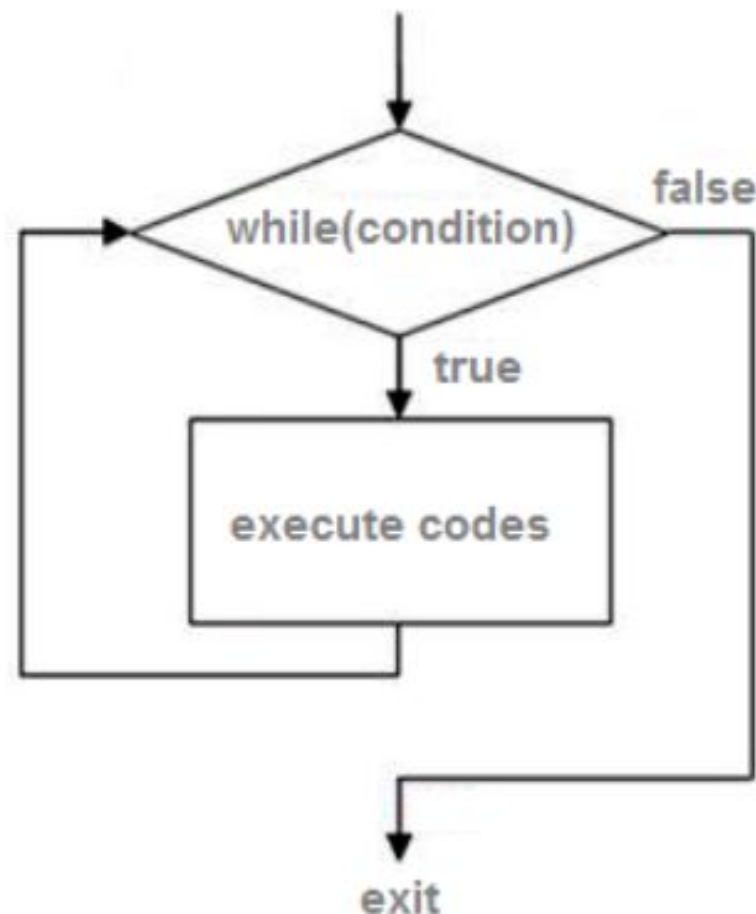
Imorgon: Tillgängligt kl 13:30

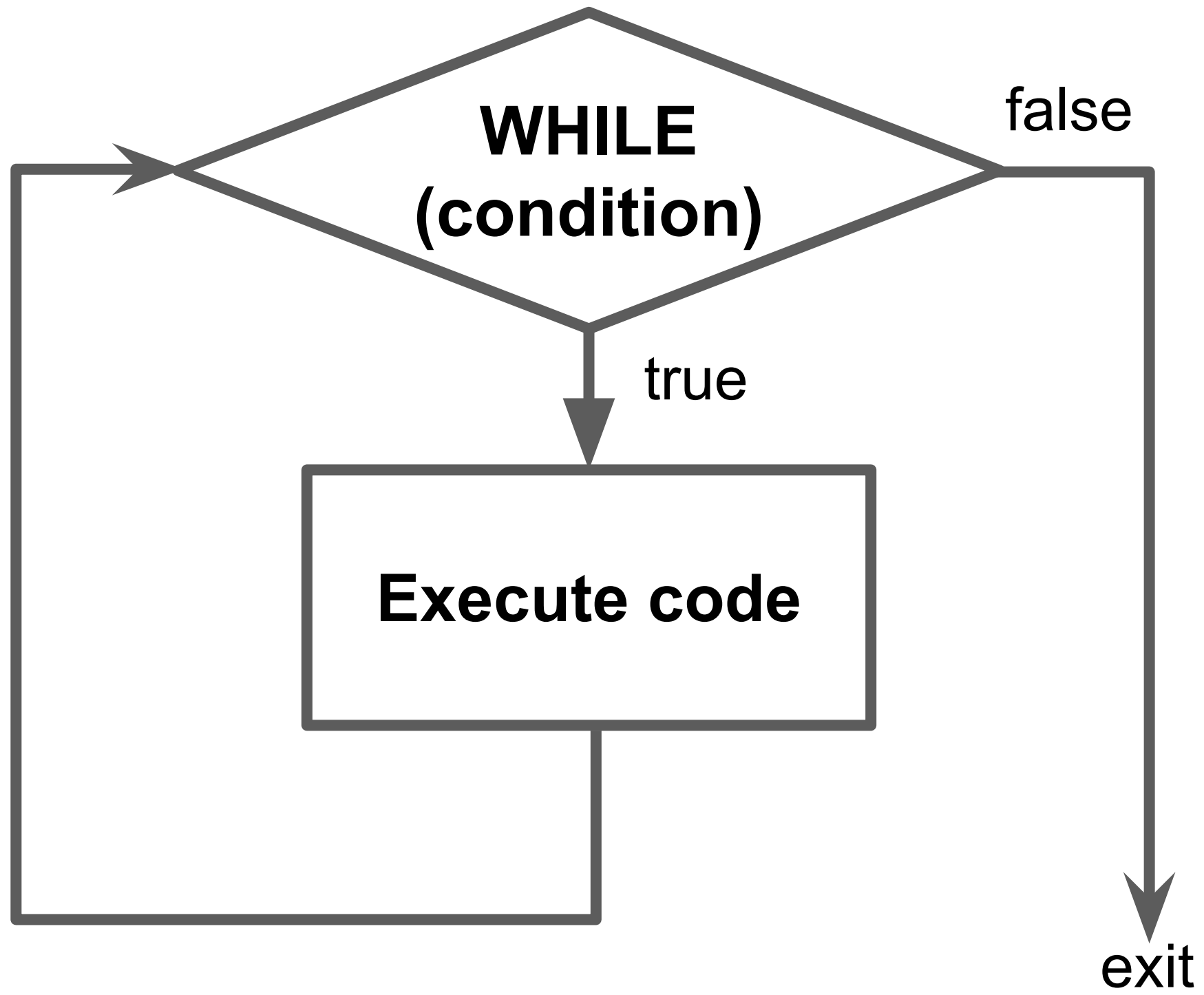
LOOPAR

(UPPREPADE)
FLÖDEN

LOOPAR

- En loop, även kallad iteration eller slinga, är ett kodstycke som upprepas om och om igen. Oftast så länge ett visst villkor är sant, precis som i en `if-sats`.
- While-loopen är egentligen den enda vi behöver kunna; Alla andra loopar är till för att förenkla för programmeraren





WHILE-LOOPEN (ITERATION)

- En **while**-loop fortsätter så länge det som står i paranteserna efter ordet **while** är sant.

```
while (true)
{
    Console.WriteLine("loop....");
}
```

- Detta kan verka dumt och det är det om vi lämnar det så här
- Men vi kan använda detta för vettiga saker! Tex. Kan man skapa en meny!

WHILE-LOOPEN (ITERATION)

Pseudokod:

MEDAN temperaturen är mindre än 100 SÅ

Öka temperaturen med ett

Skriv ut "Temperaturen är nu X"

Skriv ut Vattnet kokar

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Console.Write("Ange temperaturen: ");
        string str = Console.ReadLine();
        int temperature = Convert.ToInt32(str);
        while (temperature < 100)
        {
            temperature++;
            Console.WriteLine("Temperaturen är nu " + temperature);
        }
        Console.WriteLine("Vattnet kokar");
    }
}
```

DO WHILE

- **do while** fungerar som **while**, bara det att kontrollen huruvida loopen ska fortsätta, ligger efter koden i loopen. Med andra ord loopar **do while** minst en gång.

```
Console.WriteLine("Innan loopen");  
while ( 1 == 2 )  
{  
    Console.WriteLine("Inne i loopen");  
}
```

Vid exekvering får vi
endast: *"Innan loopen"*

```
Console.WriteLine("Innan loopen");  
do  
{  
    Console.WriteLine("Inne i loopen");  
} while (1 == 2);
```

Vid exekvering får vi:
"Innan loopen"
"Inne i loopen"

WHILE-LOOPEN (ITERATION)

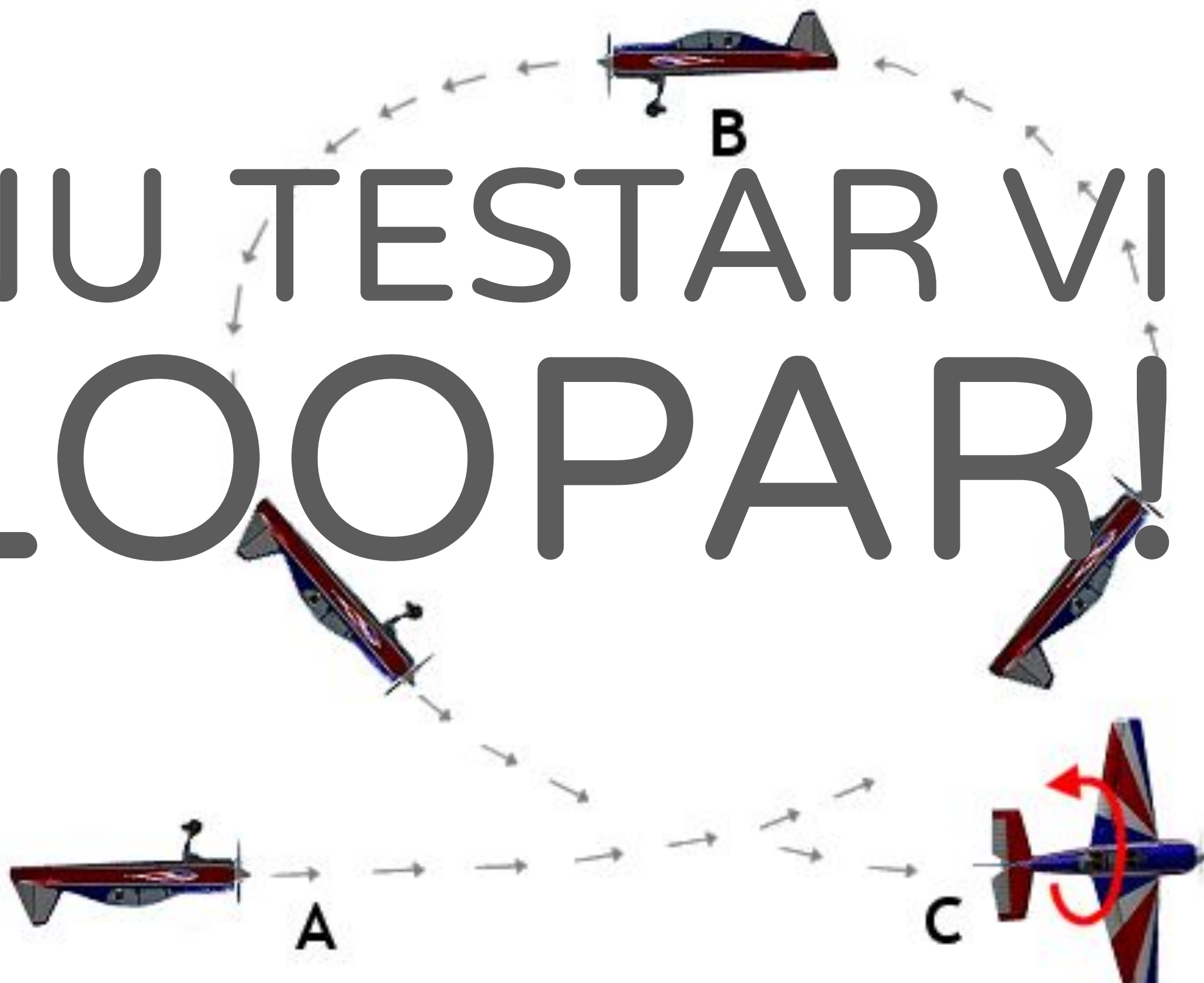
- Här har vi skapat en meny som tack vare **while**-loopen, körs om och om igen, tills att användaren trycker på Q eller q.
- När man trycker på Q/q så avbryts loopen med hjälp av ordet **break**. En **break** avslutar alltid aktuell loop.
- Närbesläktat med **break** är **continue**. Till skillnad från **break** så avslutar inte **continue** en loop. Däremot avslutar den just detta varv och går vidare till nästa varv.

```
while (true)
{
    Console.Write("Gör ditt val i menyn: ");
    Console.Write("[A] Meny val 1: ");
    Console.Write("[B] Meny val 2: ");
    Console.Write("[Q] Stäng menyn");

    string mySelection = Console.ReadLine(); //Användarens val

    if (mySelection == "A" || mySelection == "a")
    {
        Console.WriteLine("Menu val 1 är här");
    }
    else if (mySelection == "B" || mySelection == "b")
    {
        Console.WriteLine("Menu val 2 är här");
    }
    else if (mySelection == "Q" || mySelection == "q")
    {
        break; //bryter while loopen
    }
    else
    {
        Console.WriteLine("Ogiltligt val!");
    }
}
```


NU TESTAR VI
LOOPAR!



Övning While-loopar

1. Skriv ett program som skriver ut siffrorna 1 - 10 efter varandra.
2. Skriv ett program som skriver ut multiplikationstabellen för en siffra som användaren får mata in själv.
3. Skriv ett program som låter användaren skriva in en siffra. Denna siffra skall representera höjden på en stapel av tecknet #. Om användaren skriver 4 skall stapeln se ut så här:

```
  #  
 ##  
###  
####  
#####
```

Övning While-loopar

Banksimulatore (Switch och Loopövning)

Skriv ett program som simulerar en bank. Man ska kunna sätta in pengar och ta ut pengar, samt kolla saldo. Detta görs via en meny som du ska göra med en switch-sats.

Observera att du bör jobba med loopar i denna uppgift!

SAMMANFATTNING

SAMMANFATTNING

- Variabeltyper
- Variabler
- Konstanter
- Flödeskontroll
- Kodblock
- Loopar

Hemuppgift

Adressboken

I den här övningen behöver du hantera inmatning och utmatning. Programmet är en adressbok som styrs via en meny.

- ❑ Det ska gå att lägga till namn till adressboken
- ❑ Det ska gå att visa alla namn som finns i adressboken
- ❑ Det ska gå att rensa adressboken så att den blir tom
- ❑ Det ska gå att avsluta programmet genom att exempelvis trycka på Q
- ❑ Lagom längd på programmet kan vara 30-50 kodrader

För att lösa uppgiften behöver du använda dig av följande:

- Utskrift till konsolen
- Inmatning av data, spara variabler i korrekt datatyp
- Selektion, **if** eller **switch** för meny
- **Loop** som accepterar menyval tills användaren väljer att avsluta programmet
- Kodblock, **tänk på att det är viktigt var du deklarerar dina variabler**

Snabbt klar?

Vi gör adressboken mer komplex! Lägg till features från denna lista i vilken ordning du vill:

- ☐ Adressboken skall kunna visa statistik på totalt antal tecken i adressboken
- ☐ Användargränssnittet bör vara prydligt och lättanvänt, se tex till att användaren inte behöver trycka på ENTER efter att de gjort ett val i menyn
- ☐ Se till att använda fina färger, inte bara standard-grå
- ☐ Adressboken ska kunna visa statistik på totala antalet namn
- ☐ Felhantering, se till så att det inte går att mata in tomma namn!

Utmaning: Se till att bara använda den typ av C#-kod vi har gått igenom i denna slide, inga andra mer avancerade koncept såsom listor eller arrayer! Saker som finns inbyggda i string-klassen osv är ok att använda (tex Length, Split osv)

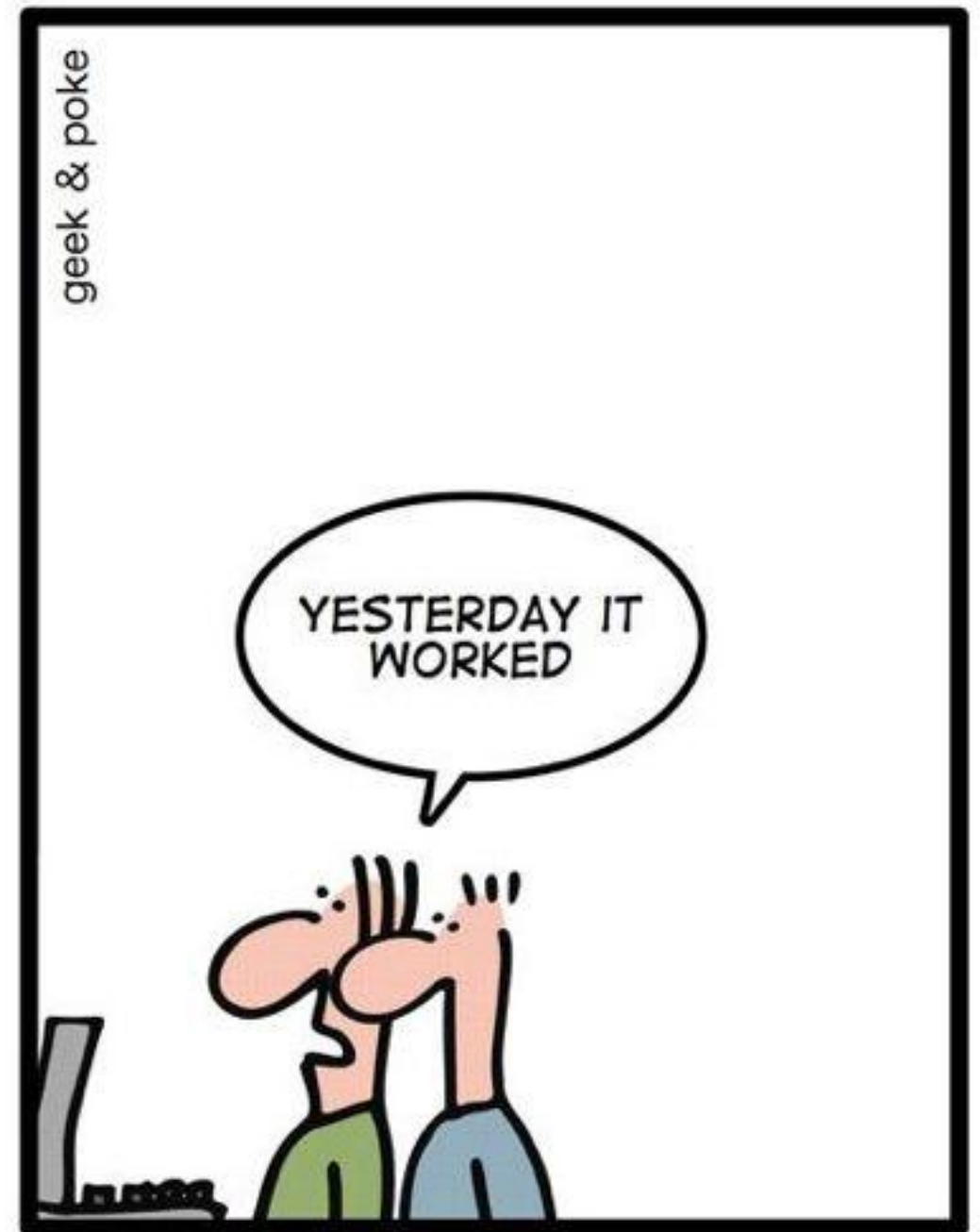
Fler övningar

Fler övningar kommer att läggas upp på Google Classroom som du kan göra fram till onsdag nästa vecka.

Nästa slide kommer också upp nästa vecka. Gå gärna igenom den och förbered dig på föreläsningen. Skriv gärna upp frågor som dyker upp och ta upp dem när vi ses!

HEJ DÅ
OCH
TACK
FÖR
IDAG!

WHEN YOU HEAR THIS:



*YOU KNOW YOU'RE IN A
SOFTWARE PROJECT*