

# C# pro začátečnice

Martin Černil, Jiří Hudec  
Filip Eckstein, Jan Kratochvíla, Filip Píndej  
Jiří Michalčík, Josef Trbušek, Martina Nemethová

21.1. 2025

# Použití materiálů

Toto dílo je licencováno pod

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License



# LEKCE

Co nás čeká a jak to bude probíhat 😊



# OSNOVA

- 21. 1. Organizace, úvod, Visual Studio, Hello World, konzole
- 28. 1. Proměnné, datové typy, podmíněný příkaz if
- 4. 2. Debugging, switch a parsování
- 11. 2. **Opakování, cykly**
- 18. 2. Metody
- 25. 2. Objektově orientované programování (OOP)
- 4. 3. OOP pokračování
- 11. 3. **Opakování OOP**
- 18. 3. Pole
- 25. 3. **Opakování**
- 1. 4. Grafická kalkulačka - základy
- 8. 4. Grafická kalkulačka - pokračování, konzultace, rady a tipy, co dál

# STRUKTURA LEKCE

## Časově

- Začínáme v 18:00, končíme ve 20:30
- Plánujeme vždy dvě přestávky – zhruba v 18:50 a 19:50

## Obsahově

- Rekapitulace úkolů z minulé lekce
- Výklad nové látky
- Breakout rooms - samostatná práce na úkolech s kouči

# ÚKOLY

Ale budou samé jednoduché, můžete nám  
věřit 😊



# ÚKOLY

- Úkoly jsou **povinné/nepovinné**, nepovinné jsou silně doporučené.
- Odevzdávají se do portálu [moje.czechitas.cz](https://moje.czechitas.cz) formou odkazu na [dotnetfiddle.net](https://dotnetfiddle.net), dostaneš od nás **individuální zpětnou vazbu**.
- Snažíme se opravit opravdu vše a odpovědět na všechny dotazy. Využij toho.
- Pro získání včasné odpovědi **odevzdávej úkoly během víkendu**.

# BUDE TO BOLET?

- To, že jsi na úkolu strávila xx hodin, vůbec nevypovídá o tom, jestli na to máš nebo ne, jestli jsi chytrá, nebo ne, atd. Je to běžná součást učení, znamená to, že pracuješ opravdu intenzivně na tom, aby ses posunula dál a jde ti to!
- Neboj se tomu věnovat čas, je to úplně nová věc a na její vstřebání je potřeba zažít i “nepříjemné chvíle zoufalství”.
- Nezapomeň ale na pravidelný odpočinek.
- Odměnou ti bude funkční program, pochopení, jak ho vytvořit a hlavně skvělý pocit!



**DOTAZY**

The background is a solid pink color. In the top-left corner, there is a rectangular area with a light pink dot grid pattern. In the bottom-left corner, there is a circular area with a light pink dot grid pattern. In the bottom-center, there is a circular area with a light pink dot grid pattern. In the bottom-right corner, there is a circular area with a light pink dot grid pattern. In the top-right corner, there is a circular area with a light pink dot grid pattern. In the center-right, there is a large, light pink circular area containing a white line-art illustration of a woman's head and shoulders, facing right. The woman has short, wavy hair and is wearing a simple top. The overall design is modern and minimalist.

# ŽÁDNÝ DOTAZ NENÍ HLOUPÝ

- Vždy se najde v místnosti někdo, kdo se na to chce také zeptat, ale nemá odvahu.
- Nezapomeň, že jsou tu i **koučové**, kteří sem přišli, aby ti byli k dispozici.
- Vyplňuj prosím zpětnou vazbu.

# HOSPODA?

Aneb zpětnou vazbu si rádi vyslechneme i u piva ústně.



# ÚVOD DO PROGRAMOVÁNÍ

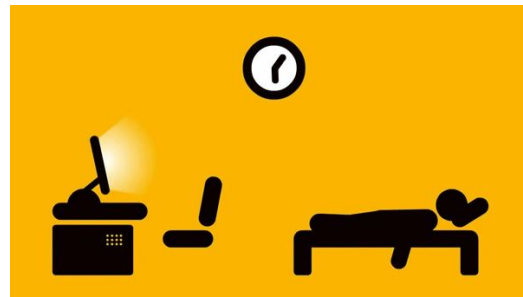


# ÚVOD DO PROGRAMOVÁNÍ

- Mýty o programování
- Proč se učit programovat?
- IT pozice
- O čem je programování
- Programovací jazyk
- Jak to funguje
- Programovací hra
- Nastavení prostředí

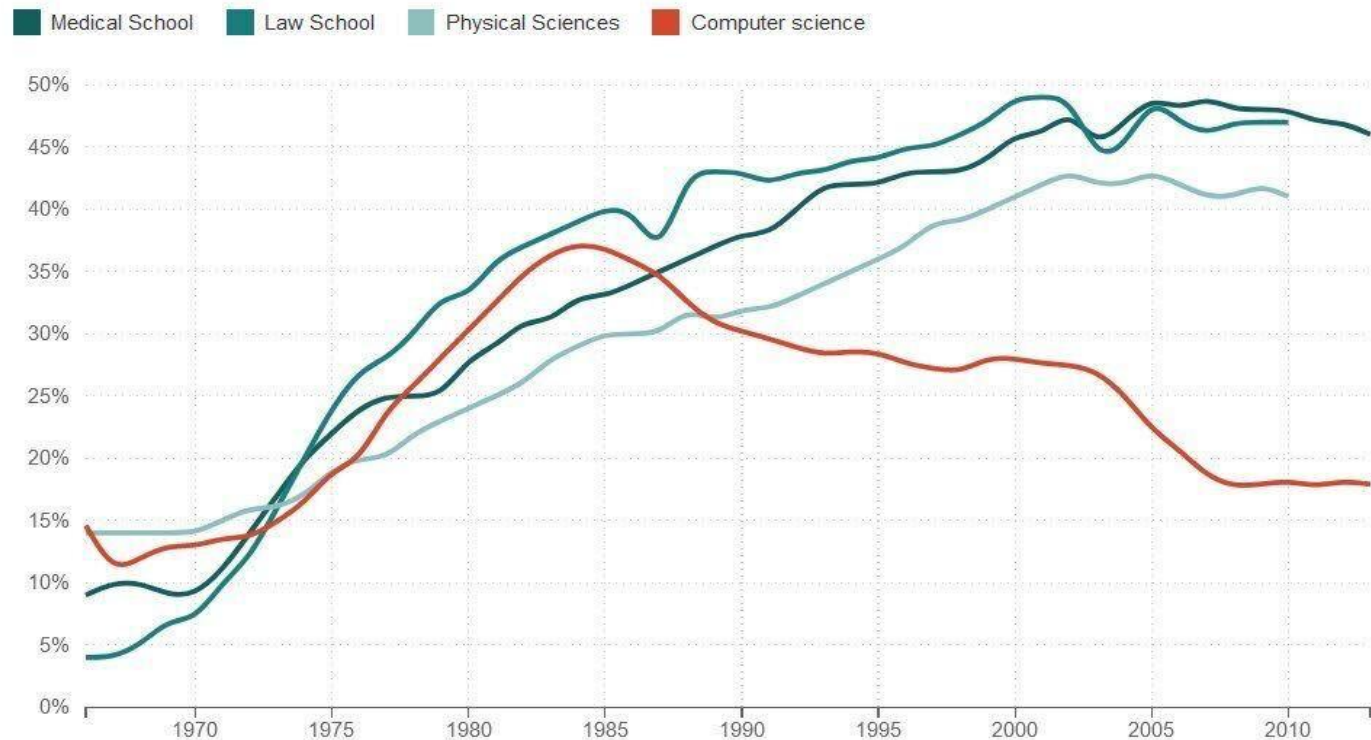
# MÝTY O PROGRAMOVÁNÍ

- Ženy se na programování nehodí
- S programováním se musí začít v mládí
- K programování je potřeba mít vysokou školu
- K programování je potřeba matematika
- Programátoři jsou asociálové zavření někde ve sklepě
- IT je jen o programování



## What Happened To Women In Computer Science?

% Of Women Majors, By Field



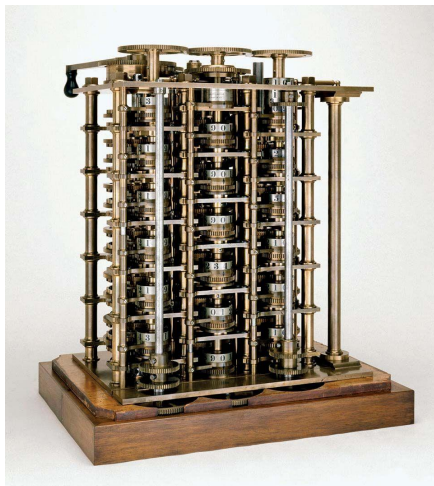
Source: National Science Foundation, American Bar Association, American Association of Medical Colleges

Credit: Quoc Trung Bui/NPR

# MÝTY – ŽENY SE NA PROGRAMOVÁNÍ NEHODÍ

## Ada Lovelace

- 1815 - 1852 Londýn
- Babbage, analytical engine
- Zavedla pojmy např.
  - podmíněný skok
  - cyklus
  - podprogram





# MÝTY – ŽENY SE NA PROGRAMOVÁNÍ NEHODÍ

## Barbara Liskov

- 1939 -
- Teoretické práce vedoucí k definici **objektového programování**
- [https://en.wikipedia.org/wiki/Barbara\\_Liskov](https://en.wikipedia.org/wiki/Barbara_Liskov)
- <https://www.quantamagazine.org/barbara-liskov-is-the-architect-of-modern-algorithms-20191120/>



# MÝTY – ŽENY SE NA PROGRAMOVÁNÍ NEHODÍ

## Margaret Hamilton

- 17.8.1936, USA
- vedoucí vývoje navigačního software pro let a přistání na Měsíci
- ocenění NASA (2003), Prezidentská medaile svobody (2016)

*„V počátcích nebyli programátoři bráni úplně vážně, nebyla to svébytná disciplína, byla to spíš taková vedlejší odnož, hlavní roli hrál hardware. Programování se také považovalo spíše za umění a kouzlení, nikoli za vědu.“*

- Originál zdrojového kódu Apollo 11 je k nahlédnutí na [GitHubu](#)



# CHCI DO IT, MUSÍM UMĚT PROGRAMOVAT?

- Ne nutně, ale ...
- Sebevědomí
- Lepší pocit v týmu s programátory
- **Znalosti se opravdu hodí!**
- Znat základy programování patří k “základní gramotnosti” ve světě IT, která tě posune z pozice “běžného uživatele”

# IT POZICE



# IT POZICE

Computing-Core Disciplines	Computing-Intensive Fields	Computing-Infrastructure Occupations
Artificial intelligence	Aerospace engineering	Blockchain administrator
Cloud computing	Autonomous systems	Computer technician
Computer science	Bioinformatics	Data analyst
Computer engineering	Cognitive science	Data engineer
Computational science	Cryptography	Database administrator
Database engineering	Computational science	Help desk technician
Computer graphics	Data science	Identity theft recovery agent
Cyber security	Digital library science	Network technician
Human-computer interaction	E-commerce	Professional IT trainer
Network engineering	Genetic engineering	Reputation manager
Programming languages	Information science	Security specialist
Programming methods	Information systems	System administrator
Operating systems	Public Policy and Privacy	Web identity designer
Performance engineering	Instructional design	Web programmer
Robotics	Knowledge engineering	Web services designer
Scientific computing	Management information systems	
Software architecture	Network science	
Software engineering	Multimedia design	
	Telecommunications	

**ZPĚT K PROGRAMOVÁNÍ** 😊



# O ČEM JE PROGRAMOVÁNÍ?

- Způsob, jak říci stroji, co má dělat.
- Sada instrukcí jdoucích logicky po sobě tak, aby mohl stroj vyřešit zadaný úkol.

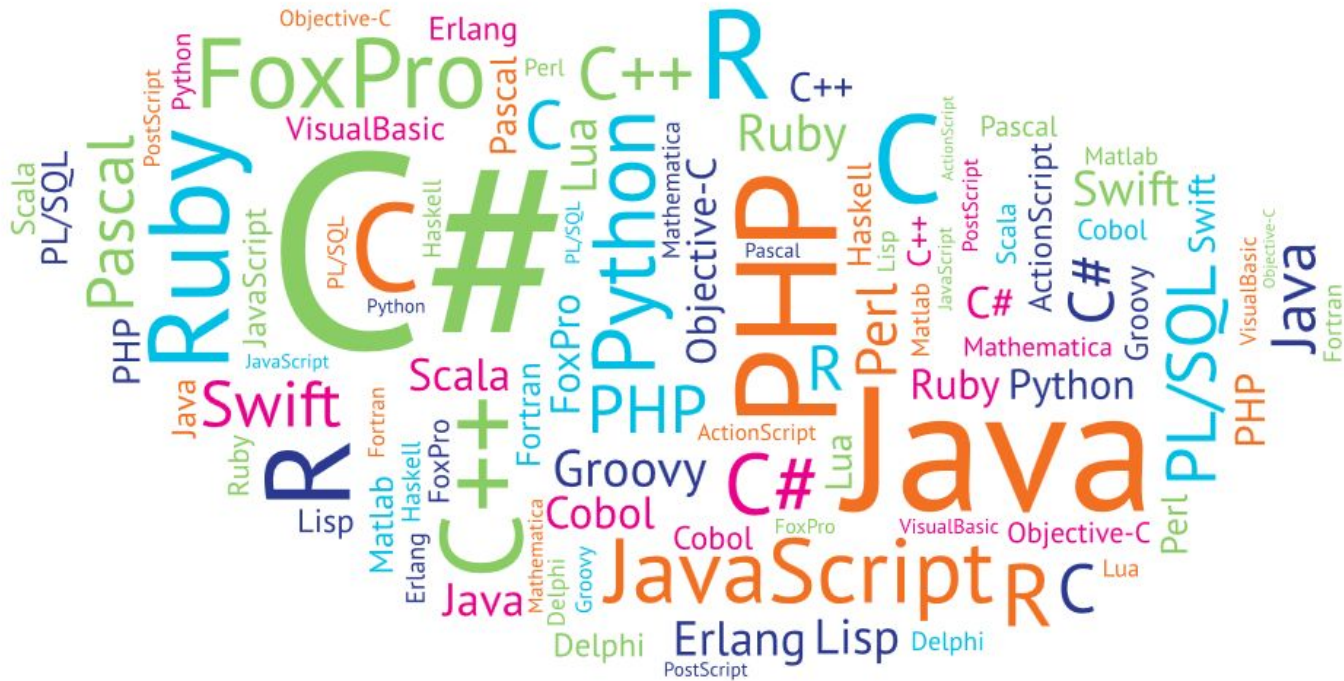


# HRA NA PROGRAMOVÁNÍ





# PROGRAMOVACÍ JAZYKY



# PROGRAMOVACÍ JAZYK

- Formalizovaný jazyk pro komunikaci s počítačem
- Obrovské množství (a další vznikají)
- Různé specializace: desktopové aplikace, weby, databáze, mobilní aplikace
- Pro často používané funkcionality vznikají **knihovny** a **frameworky**.

# PROGRAMOVACÍ JAZYK

- Programovací jazyk je mnohem jednodušší než přirozený jazyk → menší slovní zásoba v řádu desítek slov
- Důležité je porozumět principům, ty se nemění, programovací jazyky ano

# PROČ C#

- Jazyk vyšší úrovně - snadněji se učí
  - Objektově orientovaný
  - Intuitivní vývojové prostředí Visual Studio
  - Žádaný na trhu práce
  - Pravidla jazyka jsou méně benevolentní než např. u PHP pro tvorbu webu, začátečník lépe získá správné návyky psaní kódu
- 
- BONUS: syntaxe je v základu téměř identická s Javou

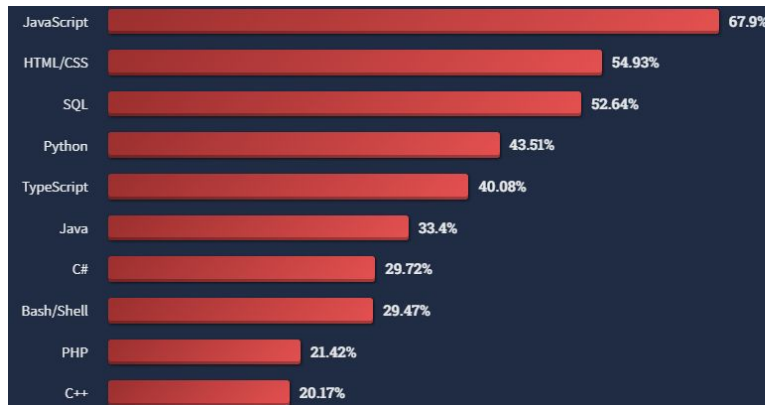
# PROČ C#

## Popularity of Programming Language

Worldwide, Sept 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.48 %	-2.4 %
2		Java	17.18 %	+0.7 %
3		JavaScript	9.14 %	+0.8 %
4		C#	6.94 %	+0.6 %
5		PHP	6.49 %	+0.4 %
6		C/C++	6.49 %	+0.9 %
7		R	3.59 %	-0.5 %
8	↑↑↑	TypeScript	2.18 %	+0.3 %
9		Swift	2.1 %	-0.4 %
10	↓↓	Objective-C	2.06 %	-0.6 %

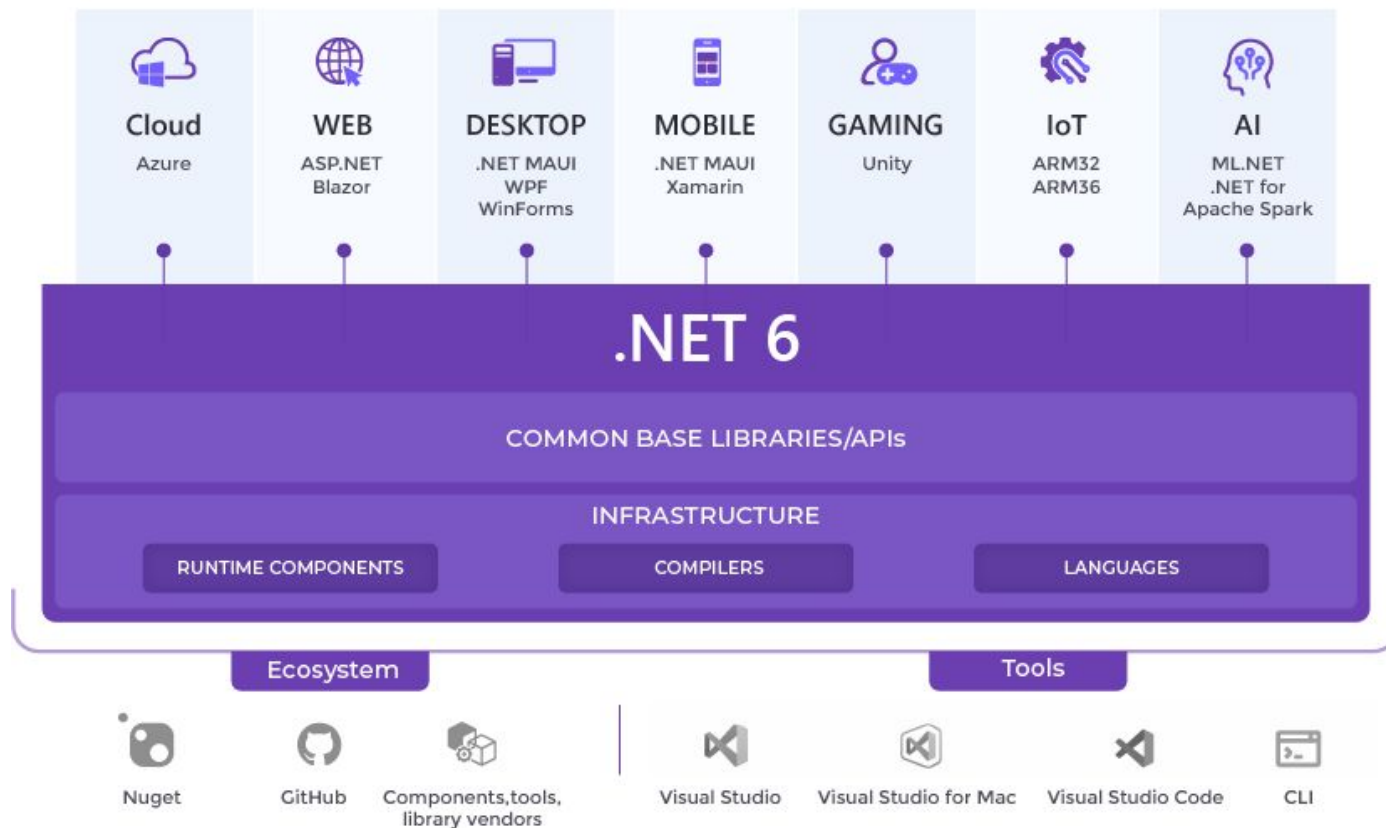
## StackOverflow 2022 Most Popular Technologies



## Top 10: Most In-Demand Programming Languages 2021

1. **JavaScript** (62%)
2. **Java** (59%)
3. **Python** (48%)
4. **C#** (40%)
5. **PHP** (32%)
6. **C++** (27%)
7. **Typescript** (24%)
8. **C** (15%)
9. **Kotlin** (15%)
10. **Swift** (14%)

# Na co vše se dá .NET použít



# JAK TO FUNGUJE



# JAK TO FUNGUJE

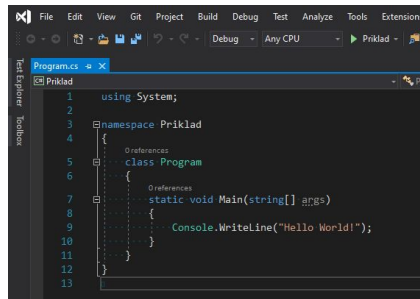
- **CPU** = processor
  - Provádí výpočty
  - Zpracovává strojový kód
- **Paměť RAM** = krátkodobé úložiště dat
  - Drží v paměti data a procesy dokud je PC zapnuté
  - Rychlý přístup, malá kapacita
- **Harddisk** = dlouhodobé úložiště dat
  - Data zůstanou zachována i po vypnutí PC
  - Pomalejší přístup, velká kapacita



# JAK TO FUNGUJE

## Překladač

- Přeloží programátorem napsaný kód tak, aby mu „rozuměl“ procesor



01001000  
01100101  
01101100  
01101100  
01101111



Kompilátor

2104  
1105  
3106  
7001  
0053  
FFFE  
0000

ORG 100  
LDA A  
ADD B  
STA C  
HLT  
DEC 83  
DEC -2  
DEC 0  
END

```
static void Main(string[] args)
{
    int a = 83;
    int b = -2;
    int c = a + b;
}
```

# BLOKOVÉ PROGRAMOVÁNÍ

<https://studio.code.org/hoc/1>



# NASTAVENÍ PROSTŘEDÍ

- Visual Studio Code



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

- .NET Fiddle



- [dotnetfiddle.net](https://dotnetfiddle.net)

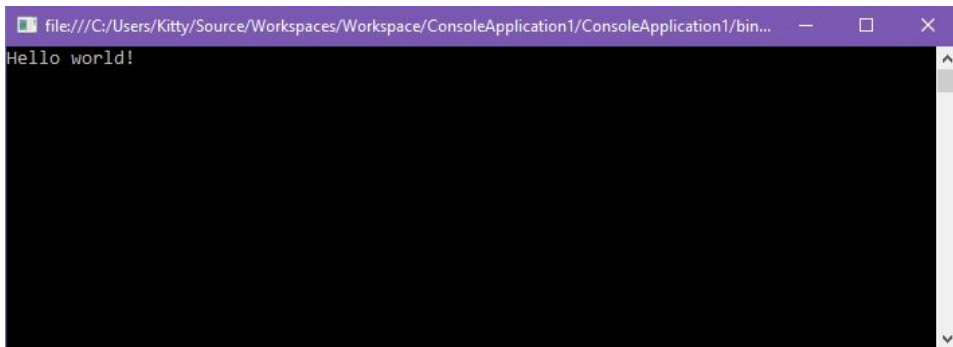
# PRVNÍ PROGRAM

Hello World 😊



# KONZOLE

- Slouží pro zadávání vstupních hodnot a zobrazování výsledků (výpis hodnot) u konzolových aplikací
- Konzole je aktivní po celou dobu běhu programu



# KOMENTÁŘE V KÓDU

- Super na psaní poznámek přímo do kódu
- Jednořádkový komentář

```
// tohle je jednořádkový komentář
```

- Víceřádkový komentář

```
/* tohle je víceřádkový komentář  
aniž by bylo potřeba na každém řádku  
psát lomítka */
```

## LEKCE 2



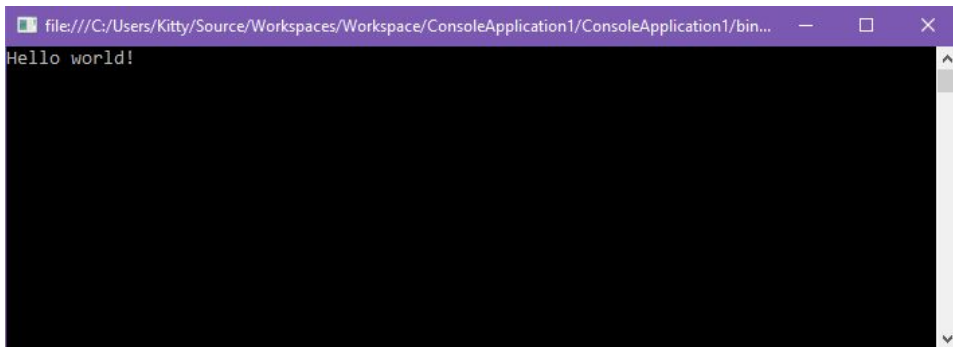
**OPAKOVÁNÍ**





# KONZOLE

- Slouží pro zadávání vstupních hodnot a zobrazování výsledků (výpis hodnot) u konzolových aplikací
- Konzole je aktivní po celou dobu běhu programu



# KOMENTÁŘE V KÓDU

- Super na psaní poznámek přímo do kódu
- Jednořádkový komentář

```
// tohle je jednořádkový komentář
```

- Víceřádkový komentář

```
/* tohle je víceřádkový komentář  
aniž by bylo potřeba na každém řádku  
psát lomítka */
```

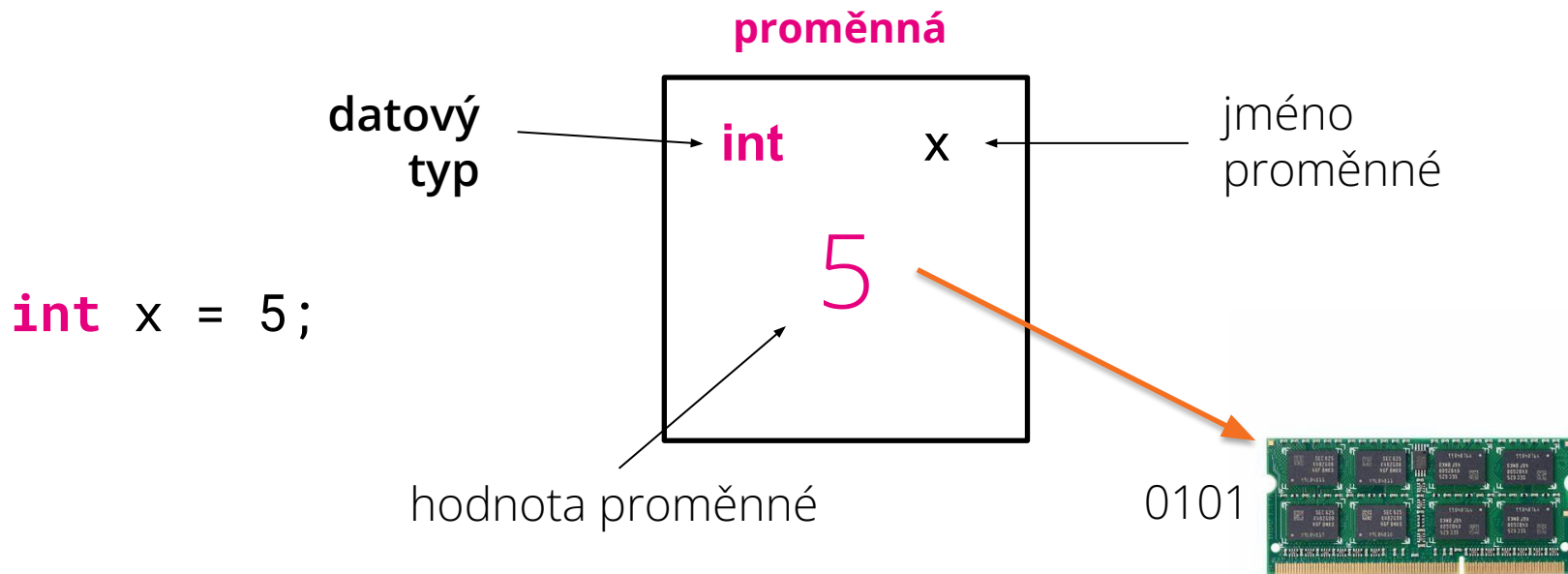
# PROMĚNNÉ A DATOVÉ TYPY



# PROČ PROMĚNNÁ?

- Způsob, jak krátkodobě uchovat data a jak s nimi pracovat v programu
- Hodnota se zadává/definuje na jednom místě, není nutné přepisovat na více místech v kódu -> eliminuje chyby, urychluje práce, kód má univerzální použití

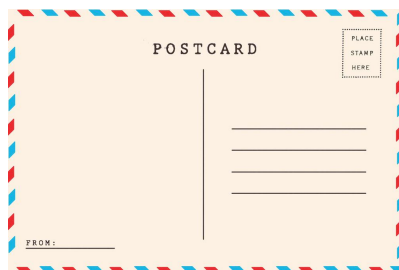
# PROMĚNNÁ A DATOVÝ TYP



# PROČ DATOVÉ TYPY?

- Důležité pro zápis v paměti – kvůli kapacitě i rychlosti
- Každý datový typ umožňuje jiné operace -> nutno dodržovat při programování – patří k základním principům
- C# je **silně typovaný jazyk** (nelze míchat “jablka” s “hruškami”), typy musí “sedět”

# PŘIROVNÁNÍ – POHLEDY, OBÁLKY, BALÍKY



# JAK TO VYPADÁ V KÓDU

**Deklarace** (vytvoření)  
proměnné

```
int cislo;  
string text;
```

Použije se, pokud chceme proměnnou  
vytvořit bez hodnoty

**Deklarace** (vytvoření)  
proměnné **s přiřazením** hodnoty

```
int cislo = 10;  
string text = "Ahoj!";
```

**Přiřazení** hodnoty (přepsání)

```
cislo = 25;  
text = "Jak je?";
```



# PŘEHLED DATOVÝCH TYPŮ

- **bool** = logická hodnota (true/false)
- **int** = celé číslo (-10, 0, 12, ...)
- **double** = desetinné číslo (-10.01, 0.0, 12.5, ...)
- **string** = řetězec znaků ("Czechitas", "x", "123" , "😊" , ...)
- **char** = znak  
( 'C', 'z', 'e', 'c', 'h', 'i', 't', 'a', 's', 'x', '1', '2', '3' )



# POJMENOVÁNÍ PROMĚNNÝCH

- Lepší víceslovný název, než nicneříkající a, b, c, ...
- Jak na to:
  - Název proměnné začíná v C# vždy malým písmenem, je bez mezer a nesmí začínat číslem
  - Pozor – **nepoužívat diakritiku!**
  - Pokud má název více slov, má každé slovo kromě prvního velké počáteční písmeno:

nazevPromenne1  
totoJeDalsiPromenna



# VÝPIS HODNOTY

- Tato funkce je pro usnadnění již naprogramována
- Vypsát hodnotu proto můžeme pomocí jediného příkazu:

```
Console.WriteLine("Hello World!");  
Console.WriteLine(123);
```

- Výpis proměnné do konzole:

```
int cislo = 789;  
Console.WriteLine(cislo);
```

# DATOVÝ TYP int

- **int** = celé číslo
- hodnoty: -10, 0, 12, ...
- základní (matematické) operace:
  - sčítání  $1+2$
  - odčítání  $3-1$
  - násobení  $2*2$
  - dělení  $4/2$  (celočíselné dělení)  
v případě, že vyjde desetinné číslo, desetinná místa se useknou bez zaokrouhlení!  $4/3 = 1$

# DATOVÝ TYP **double**

- **double** = desetinné číslo
- hodnoty: -10.01, 0.0, 12.5, ...
  - POZOR na desetinnou tečku a čárku, záleží na nastaveném jazyku
  - Proč 0.0? Samotná 0 je automaticky považována za celé číslo
- základní (matematické) operace jsou stejné jako u **intu**

# DATOVÝ TYP string

- **string** = řetězec znaků, text
- hodnoty: "Czechitas", "x", "123" , "😊" , ...
- základní operace - skládání textu

"a" + "hoj" = "ahoj"

"12" + "34" = "1234"

# DATOVÝ TYP bool

- **bool** = logická hodnota
- hodnoty: **true**, **false**
- základní logické operace
  - **AND** (součin)    **true && false**
  - **OR** (součet)    **true || false**
  - výsledky logických operací viz tabulka
- **&&** a **||** jsou tzv. logické operátory a používají se stejně jako např. + a -

vstup1	vstup2	výsledek operace	
		AND	OR
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

# PŘEHLED DATOVÝCH TYPŮ

- **bool** = logická hodnota (true/false)
- **int** = celé číslo (-10, 0, 12, ...)
- **double** = desetinné číslo (-10.01, 0.0, 12.5, ...)
- **string** = řetězec znaků ("Czechitas", "x", "123" , "😊" , ...)
- **char** = znak  
( 'C', 'z', 'e', 'c', 'h', 'i', 't', 'a', 's', 'x', '1', '2', '3' )





# NAČTENÍ VSTUPU

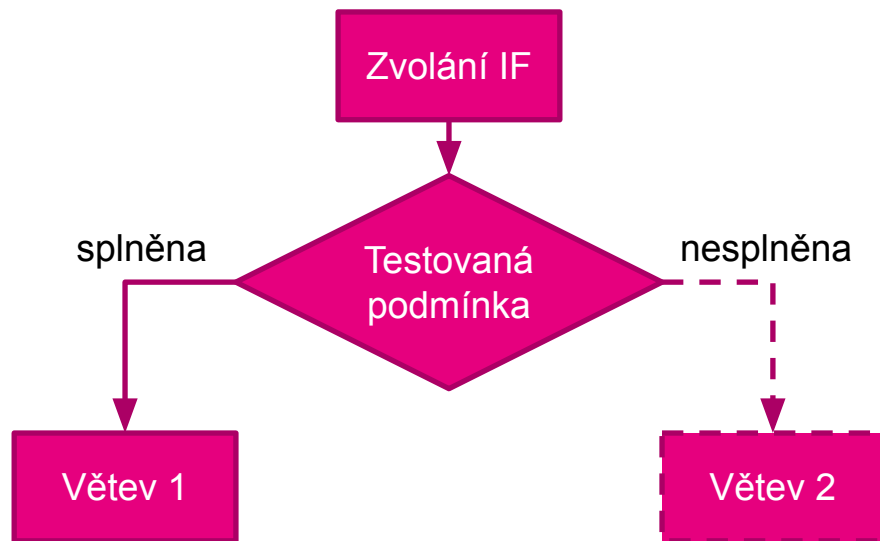
- Opět hotová funkce pro programátora.
- Použijeme v případě, že chceme, aby uživatel zadal hodnotu nějaké proměnné sám.
- Může se jednat např. o jméno, nebo o počet položek, se kterými má program pracovat.
- **Vstup vždy ukládáme do proměnné datového typu `string`.**

```
string vstupUzivatele = Console.ReadLine();
```

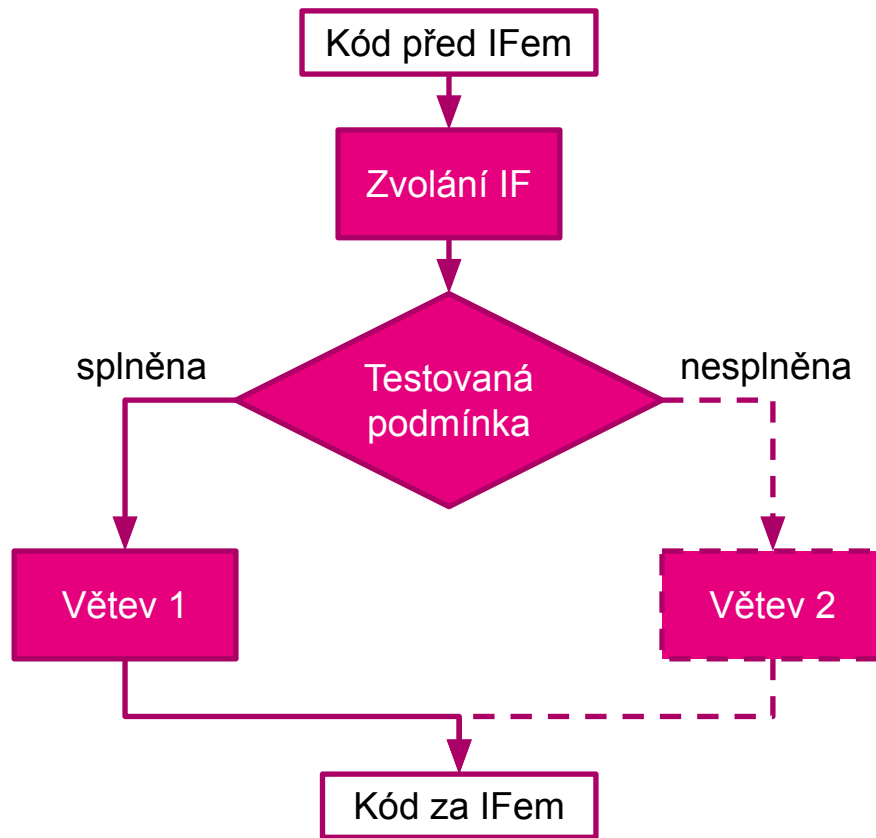
# PODMÍNĚNÝ PŘÍKAZ



# PODMÍNĚNÝ PŘÍKAZ if/else



# PODMÍNĚNÝ PŘÍKAZ if/else



// Kód před IFem

```
if (testovanaPodminka)
{
    //Větev 1
}
else // nepovinné
{
    //Větev 2
}
```

// Kód za IFem

# SLOŽENÉ ZÁVORKY

Ohraničují logický celek



# PODMÍNĚNÝ PŘÍKAZ if/else

```
int vek = 15;
if (vek >= 18)
{
    // Větev 1 - podmínka splněna
    Console.WriteLine("Na zdraví!");
}
else
{
    // Větev 2 - podmínka nesplněna
    Console.WriteLine("Mladistvým nenaléváme.");
}
Console.WriteLine("Konec programu.");
```

# PODMÍNĚNÝ PŘÍKAZ if/else

- Umožňuje větvit program pro různé možnosti podle splněných/nesplněných podmínek
- Podmínka je buď splněna nebo nesplněna -> existují jen dvě možnosti jejího vyhodnocení -> nový datový typ **bool**

# OPERÁTORY PRO POROVNÁVÁNÍ HODNOT

- Podobně jako v matematice, pozor na správný zápis:
  - větší než  $10 > 10$  ... **false**
  - větší nebo rovno  $10 >= 10$  ... **true**
  - menší než  $5 < 10$  ... **true**
  - menší nebo rovno  $9 <= 10$  ... **true**
  - rovno  $"a" == "b"$  ... **false**
  - nerovno  $"a" != "b"$  ... **true**
- Výsledkem porovnání je pak **bool**. Porovnání se používá v podmíněných příkazech a dá se skládat pomocí logických operátorů (&&, ||)



# PODMÍNĚNÝ PŘÍKAZ if/else

- **else** není povinné, použije se pouze v případě, že chceme vykonat nějaké speciální příkazy v případě nesplnění podmínky
- složené závorky mohou obsahovat více příkazů dle potřeby

```
int cislo = 12;  
if (cislo > 10 && cislo < 25)  
{  
    Console.WriteLine("Jsi ve správném intervalu mezi 10 a 25.");  
    cislo = cislo + 5;  
}  
Console.WriteLine(cislo);
```

# PODMÍNĚNÝ PŘÍKAZ `else if`

- Jedná se o rozšíření podmíněného příkazu v případě, že chceme vyhodnotit více podmínek zvlášť -> vytvoříme tím více možných větví programu.

# PODMÍNĚNÝ PŘÍKAZ else if

```
int cislo = 12;  
if (cislo > 10)  
{  
    Console.WriteLine("Číslo je větší než 10");  
}  
else if (cislo == 10)  
{  
    Console.WriteLine("Číslo je rovno 10");  
}  
else  
{  
    Console.WriteLine("Číslo je menší než 10");  
}
```

# LEKCE 3



# PŘÍKAZ SWITCH



# SWITCH

K čemu je dobrý?

- Máte proměnnou, která nabývá **konečného počtu hodnot** a chcete, aby se program pro každou tuto hodnotu zachoval rozdílně.
- Jedná se o elegantnější alternativu **else if**.
- Využijete v kalkulačce 😊

# SWITCH

```
switch (zvire)
{
    case "pes":
        Console.WriteLine("haf haf");
        break;
    case "kocka":
        Console.WriteLine("mňau mňau");
        jeZvireKocka = true;
        break;
    default:
        Console.WriteLine("neznámé zvíře");
        break;
}
```

**BOOLEAN**





# DATOVÝ TYP bool

- **bool** = logická hodnota
- hodnoty: **true**, **false**
- základní logické operace
  - **AND** (součin)    **true && false**
  - **OR** (součet)    **true || false**
  - výsledky logických operací viz tabulka
- **&&** a **||** jsou tzv. logické operátory a používají se stejně jako např. + a -

vstup1	vstup2	výsledek operace	
		AND	OR
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

# OPERÁTOR NEGACE

- Pokud chceme otočit hodnotu logického výrazu, použijeme logický operátor negace "!" - píše se před požadovanou hodnotu:

```
bool jeCislo = false;  
if (!jeCislo)  
{  
    Console.WriteLine("Bohužel, není to číslo.");  
}
```

- V tomto případě se kód pod podmínkou vykoná, protože se z původní **false** hodnoty stane **true** a podmínka je tedy splněna.

# KONTROLA VSTUPU



# KONTROLA VSTUPU

- V konzolové aplikaci načítáme vstup vždy do proměnné **string**, protože uživatel může na klávesnici natukat cokoli.
- Pokud potřebujeme od uživatele číslo, musíme vstup na číslo převést.
- K tomu slouží dvě různé funkce a proces převodu na jiný datový typ se nazývá **parsování**.

# PARSE

- Převeďte vstup na požadovaný datový typ

```
string vstup = Console.ReadLine();
```

```
int cislo = int.Parse(vstup);
```

- Do proměnné **cislo** se uloží hodnota z proměnné vstup převedená na **int**.
- To samé funguje pro desetinné číslo:

```
double desetinneCislo = double.Parse(vstup);
```

- Pokud uživatel zadá špatně vstup, funkce vyhodí chybu a program končí.

# TRYPARSE

- Uloží do **bool** proměnné hodnotu, zda se převod povedl a pokud ano, uloží do připravené proměnné hodnotu v požadovaném datovém typu.

```
string textovyVstup = Console.ReadLine();  
int prevedeneNaCislo;  
bool jeCislo = int.TryParse(textovyVstup, out prevedeneNaCislo);  
if (!jeCislo)  
{  
    Console.WriteLine("Zadaný vstup není číslo.");  
}
```

- Pokud uživatel zadá špatně vstup, program nevyhodí chybu a můžeme s ním dále pracovat.

## LEKCE 4



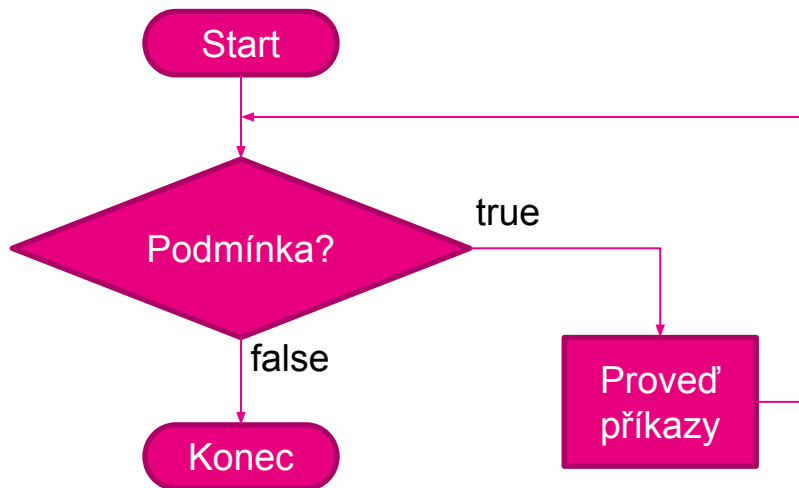
**CYKLY**

The background is a solid pink color. In the top-left corner, there is a rectangular area with a light pink dotted pattern. In the bottom-left corner, there is a circular area with a light pink dotted pattern. In the bottom-center, there is a circular area with a light pink dotted pattern. In the bottom-right corner, there is a circular area with a light pink dotted pattern. In the top-right corner, there is a circular area with a light pink dotted pattern. In the center-right, there is a large, light pink circular area. Inside this large circle, there is a line-art illustration of a woman's head and shoulders, facing right. The woman has short, wavy hair and is wearing a top with a high collar. The line art is a simple outline in a light pink color.



# CYKLUS while

- Opakuje sadu příkazů **dokud platí zadaná podmínka**
- Pokud bude podmínka platit pořád, vznikne nekonečný cyklus -> program se tzv. **zacyklí** -> to nechceme 😊



# CYKLUS while

```
int cislo = 0;  
while (cislo < 5)  
{  
    Console.WriteLine("Ahoj");  
    cislo = cislo + 1;  
}
```

# ÚLOHA

Jak umožnit uživateli zadávat vstup opakovaně, dokud nebude správně, aniž by program skončil?

# CYKLUS for

- Narozdíl od **while** se použije v případě, pokud je **předem znám požadovaný počet opakování cyklu**
- Veškerá opakovací logika se zapisuje do hlavičky:
  - vytvoření proměnné - tzv. čítače a přiřazení počáteční hodnoty
  - ukončovací podmínka (hraniční hodnota čítače)
  - změna čítače, která postupně vede k dosažení ukončovací podmínky (obvykle se hodnota čítače zvyšuje nebo snižuje o 1)

# CYKLUS for

```
for (int i = 0; i < 5; i++)  
{  
    Console.WriteLine(i);  
}
```

# UKONČOVACÍ PŘÍKAZY



# UKONČOVACÍ PŘÍKAZY

- **break**; ukončí cyklus **while/for** a příkaz **switch**
- **continue**; ukončí aktuální běh (iteraci) cyklu a přeskočí na další iteraci
- **return**; ukončí metodu, pozor pokud je použit v metodě Main, ukončí celý program

Ukončovací příkazy fungují jen pro danou úroveň, pokud máme cyklus zanořený v cyklu, tak se ukončí jen ten zanořený.

# UKONČOVACÍ PŘÍKAZY

```
int a = 0;
while (true)
{
    Console.WriteLine(a);
    a++;
    if (a == 5) break;
}
```



# UKONČOVACÍ PŘÍKAZY

```
int y = 0;  
for (int x = 1; x < 101; x++)  
{  
    if ((x % 7) == 0) continue;  
    y++;  
}
```

```
Console.WriteLine(y);
```

# LEKCE 5

## Metody



**CO JE DOBRÉ ZNÁT**



# UŽITEČNÉ ZKRATKY

`cislo = cislo + 1;`      **`== cislo++;`**

`cislo = cislo - 1;`      **`== cislo--;`**

`cislo = cislo + 5;`      **`== cislo += 5;`**

`text = text + "pokračování"`      **`== text += "pokračování"`**

`if (promenna == true)`      **`== if (promenna)`**

`if (promenna == false)`      **`== if (!promenna)`**

# OPERÁTOR modulo

- Zbytek po celočíselném dělení

$$7 \% 5 = 2$$

$$10 \% 5 = 0$$

$$6 \% 4 = 2$$

- Hodí se např., pokud se má nějaký příkaz vykonat každý x-násobek (= modulo x je nula) průchodu cyklem.

# OPERÁTOR modulo

```
for (int i = 1; i < 21; i++)  
{  
    if (i % 5 == 0)  
    {  
        Console.Write("!");  
    }  
    else  
    {  
        Console.Write(".");  
    }  
}
```

# VÝPIS DO KONZOLE

- `Console.Write()` neodřádkuje příkazový řádek na konci výpisu
- Lze tak zapisovat výstupy na jeden řádek

```
Console.Write(i + " ");
```

```
// ve for cyklu pro i od 0 do 4 včetně vypíše: 0 1 2 3 4
```

- Pokud potřebujeme jednorázově odřádkovat, použijeme `Console.WriteLine` s prázdným argumentem:

```
Console.WriteLine();
```

# VÝPIS DO KONZOLE

- Interpolovaný text lze použít ve výpisu v textu na místě, kde má být hodnota proměnné:

```
string jmeno = "Pepa";
```

```
int vek = 25;
```

```
Console.WriteLine($"Ahoj, jsem {jmeno} a je mi {vek} let.");
```



**MAIN**



# JAK SKUTEČNĚ FUNGUJE NÁŠ PROGRAM?

Lekce01.Vyklad.Konzole > C# ConsoleApp.cs

```
1 Console.WriteLine("Ahoj svete!"); // tento radek vypise do konzole text
2 |
```

- Schované přepnutí stylu:

Lekce01.Vyklad.Konzole > C# ConsoleApp.cs

```
1 Console.WriteLine("Ahoj svete!"); // tento radek vypise do konzole text
2 |
```

More Actions...

- 💡 Introduce local for 'Console.WriteLine("Ahoj svete!")'
- 💡 Introduce local for all occurrences of 'Console.WriteLine("Ahoj svete!")'
- 💡 **Convert to 'Program.Main' style program**

# JAK SKUTEČNĚ FUNGUJE NÁŠ PROGRAM?

Lekce01.Vyklad.Konzole > C# ConsoleApp.cs > Program

0 references

```
1 internal class Program
```

```
2 {
```

0 references

```
3 private static void Main(string[] args)
```

```
4 {
```

```
5     Console.WriteLine("Ahoj svete!"); // tento radek vypise do konzole text
```

```
6 }
```

```
7 }
```

```
8
```

# JAK SKUTEČNĚ FUNGUJE NÁŠ PROGRAM?

File Explorer view of the project directory: `csharp1-template > src > Lekce01.Vyklad.Konzole`

Name	Date modified	Type	Size
bin	1/15/2025 22:33	File folder	
obj	1/15/2025 22:33	File folder	
Lekce01.Vyklad.Konzole.csproj	1/15/2025 22:32	C# Project file	1 KB
Program.cs	1/28/2025 18:09	C# Source File	1 KB

File Explorer view of the output directory: `csharp1-template > src > Lekce01.Vyklad.Konzole > bin > Debug > net8.0`

Name	Date modified	Type	Size
Lekce01.Vyklad.Konzole.deps.json	1/16/2025 17:33	JSON File	1 KB
Lekce01.Vyklad.Konzole.dll	2/17/2025 21:38	Application extension...	5 KB
Lekce01.Vyklad.Konzole.exe	2/17/2025 21:38	Application	140 KB
Lekce01.Vyklad.Konzole.pdb	2/17/2025 21:38	Program Debug D...	12 KB
Lekce01.Vyklad.Konzole.runtimeconfig.js...	1/16/2025 17:33	JSON File	1 KB

# METODY

The background is a solid pink color. In the top-left corner, there is a rectangular area with a light pink dotted pattern. In the bottom-left corner, there is a circular area with a light pink dotted pattern. In the bottom-center, there is a circular area with a light pink dotted pattern. In the bottom-right corner, there is a circular area with a light pink dotted pattern. In the top-right corner, there is a circular area with a light pink dotted pattern. In the center-right, there is a large, light pink circular area. Inside this large circle, there is a line-art figure of a person with long hair, wearing a dress, and holding a long, thin object. The figure is drawn with simple black lines.

# METODY

- Jedná se o funkce, které vykonávají sadu příkazů
- Metoda může mít libovolný počet **vstupů**  
(píšeme do kulatých závorek za název)
- Metoda může a nemusí vracet **výstup**

# METODY

Zápis hlavičky metody:

výstup



[modifikátor přístupu] [typ návratové hodnoty]  
[Název Funkce]([typ parametru][název parametru],...)

vstupy

př. :

```
public double VydelCisla(double cislo1, int cislo2)
```

# METODY

- Metoda bez vstupu i výstupu

```
public void VypisVlastnosti() //prázdné závorky značí žádný vstup
```

- **void** = nic = klíčové slovo pro metodu, která nevrací žádný výstup
- Metoda se vstupem a výstupem

```
public double VydelCisla(double cislo1, int cislo2)  
//vstupem jsou proměnné cislo1 a cislo2
```

- **double** = datový typ proměnné, kterou metoda vrací
  - Metoda s výstupem musí obsahovat příkaz **return** s definovaným typem výstupu
- Lze i kombinace - metoda s výstupem bez vstupu a naopak

```
public string VypisVlastnosti(); //uloží vlastnosti do stringu
```



# METODY

Ukázka použití return:

```
public double VydelCisla(double cislo1, int cislo2)
{
    return cislo1/cislo2;
}
```

Jak zavolat metodu s výstupem

```
double vysledek = VydelCisla(7, 3)
// výstup se uloží do proměnné vysledek
Console.WriteLine(VydelCisla(7, 3))
// výstup se rovnou vypíše do konzole
// místo konkrétních čísel lze samozřejmě dosadit proměnné
```

# UKONČOVACÍ PŘÍKAZY

```
public double VydelCisla(double cislo1, double cislo2)
{
    return cislo1/cislo2;
}
```

```
public string PredstavSe(string jmeno)
{
    return "Ahoj, jmenuji se " + jmeno;
}
```

# LEKCE 6

## OBJEKTIVĚ ORIENTOVANÉ PROGRAMOVÁNÍ



# OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

Objekty, instance, třídy



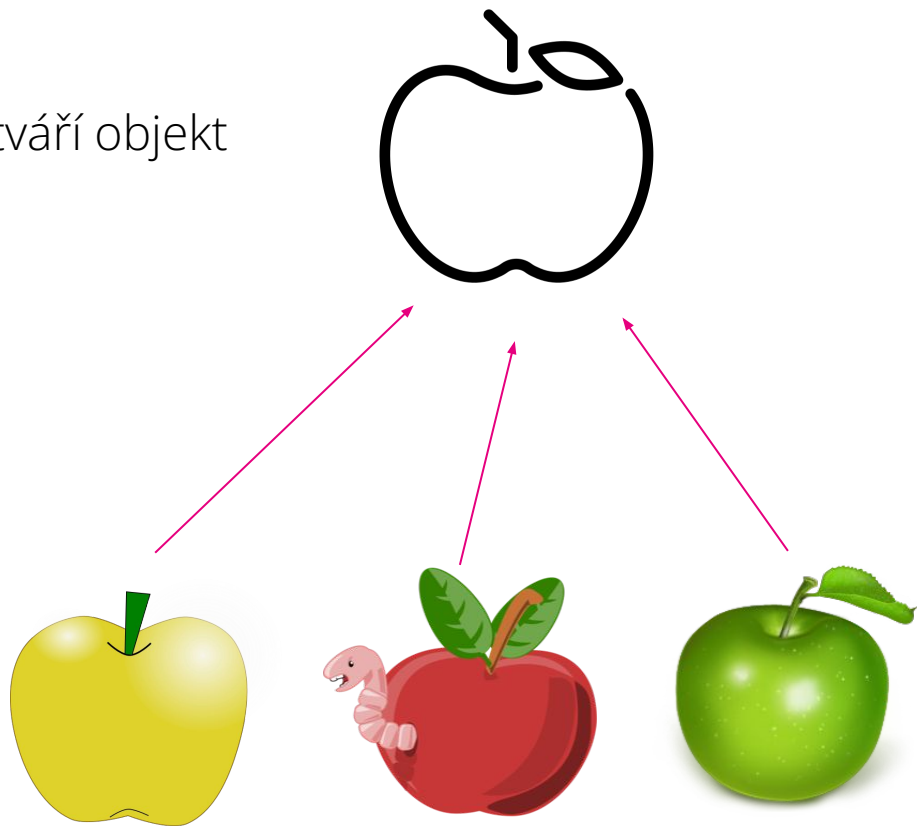
# OBJEKT

- Instance, **konkrétní** existující předmět/entita
- V programu existuje jen po dobu jeho běhu
- Vzniká na základě definice třídy



# TŘÍDA

- Předpis, na základě kterého se vytváří objekt
- Definuje **vlastnosti** objektu
- Definuje **metody** objektu



# TŘÍDA

```
public class Jablko
{
    public string Barva;
    public string Tvar;
    public bool MaStopku;
    public int Velikost;

    public void VypisVlastnosti()
    {
        Console.WriteLine("Moje barva je {0}, jsem {1}, moje
velikost je {2}.", Barva, Tvar, Velikost);
    }
}
```

# PRÁCE S OBJEKTEM

- objekt nejprve vytvoříme

```
Jablko pepa = new Jablko();
```

```
// vytvořili jsme proměnnou pepa datového typu Jablko
```

- existujícímu objektu můžeme nastavit konkrétní hodnoty do **vlastností**

```
pepa.Barva = "zelená";
```

```
pepa.Velikost = 10;
```

- na existujícím objektu můžeme zavolat **metodu** definovanou v jeho třídě

```
pepa.VypisVlastnosti();
```



# LEKCE 7

## OBJEKTIVĚ ORIENTOVANÉ PROGRAMOVÁNÍ



# KONSTRUKTOR

- Metoda pro vytvoření objektu
- Existuje v každé třídě automaticky - defaultně je bez vstupů
- Pokud chceme již při vytváření objektu nadefinovat konkrétní vlastnosti, je nutné vytvořit vlastní konstruktor
- Třída může mít více než jeden konstruktor - tj. více než jeden způsob vytváření objektů

# KONSTRUKTOR

- Vytvoříme objekt, kterému rovnou přiřadíme hodnoty vlastností

```
Tiskarna canon = new Tiskarna("Canon", 2018);
```

- Ve třídě jsme předtím nadefinovali vlastní konstruktor, který umí přijmout proměnné na vstupu a přiřadit je k požadovaným vlastnostem objektu

```
public Tiskarna(string znacka, int rokVyroby)
{
    Znacka = znacka;
    RokVyroby = rokVyroby;
}
```

# MODIFIKÁTOR PŘÍSTUPU

- **public** = daná vlastnost nebo metoda je přístupná odkudkoli z kódu
- **private** = daná vlastnost nebo metoda je přístupná pouze uvnitř třídy, kde byla definovaná

Na kurzu pro začátečnice si bohatě vystačíme s přístupem **public** 😊

Musíme ho používat, protože výchozí je **private**.

# LEKCE 8

## OBJEKTIVĚ ORIENTO VANÉ PROGRAMOVÁNÍ POKRAČOVÁNÍ



# STATICKÉ METODY

- Tyto metody lze použít/volat i bez existujícího objektu
- V definici musíme použít klíčové slovo static

```
public static double VydelCisla(double cislo1, int cislo2)
```

- Metoda se volá stejně, akorát bez objektu

```
double vysledek = VydelCisla(7, 3)
```

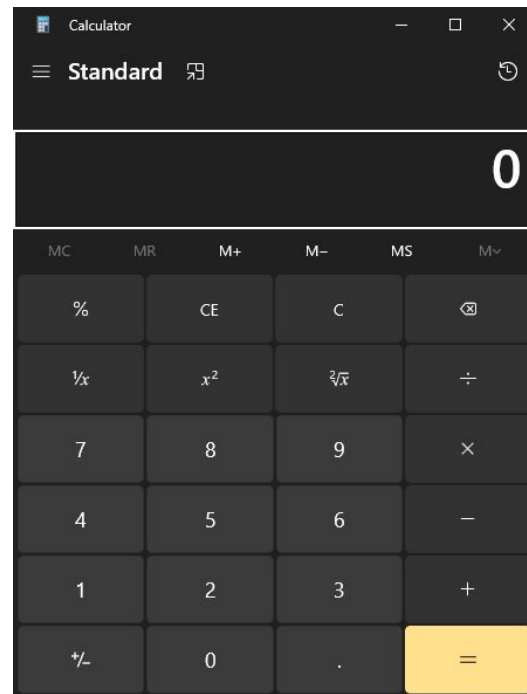
- Využijete např. v metodě Main pro **opakované operace**.
- **POZOR:** pokud je statická metoda v jiné třídě než tam, kde ji voláte, musíte místo objektu použít název té třídy:

```
double vysledek = Trida.VydelCisla(7, 3)
```

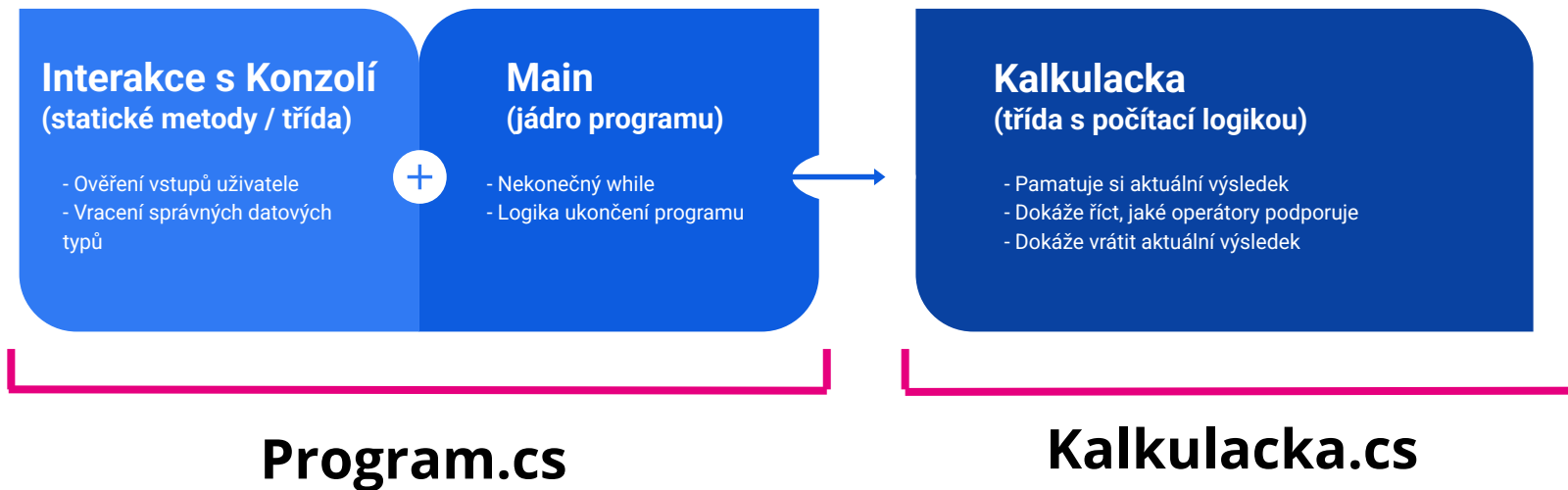
# ZÁVĚREČNÝ ÚKOL - CO OČEKÁVAT

Kalkulačka v6

- Otevíráme s nulovou hodnotou
- Střídavě zadáváme čísla a operátory
- Dokážeme získat výsledek
- **Nepovinné:**  
Grafické rozhraní Blazor

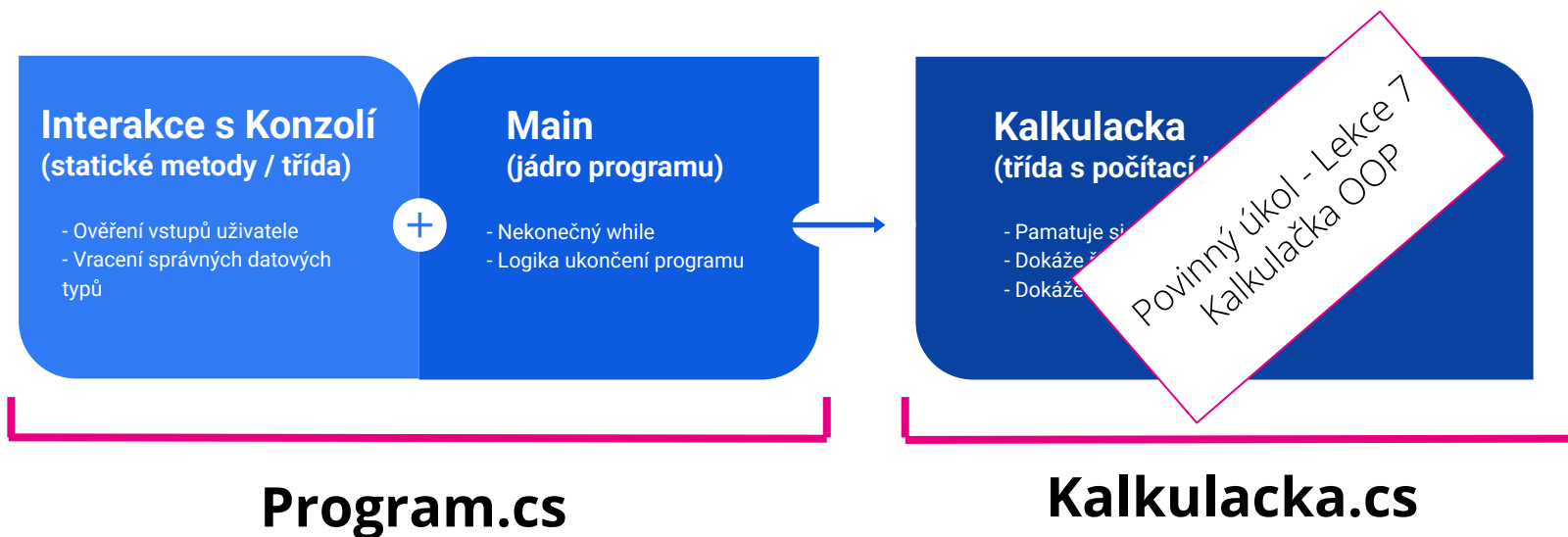


# ZÁVĚREČNÝ ÚKOL - NÁSTIN ZADÁNÍ

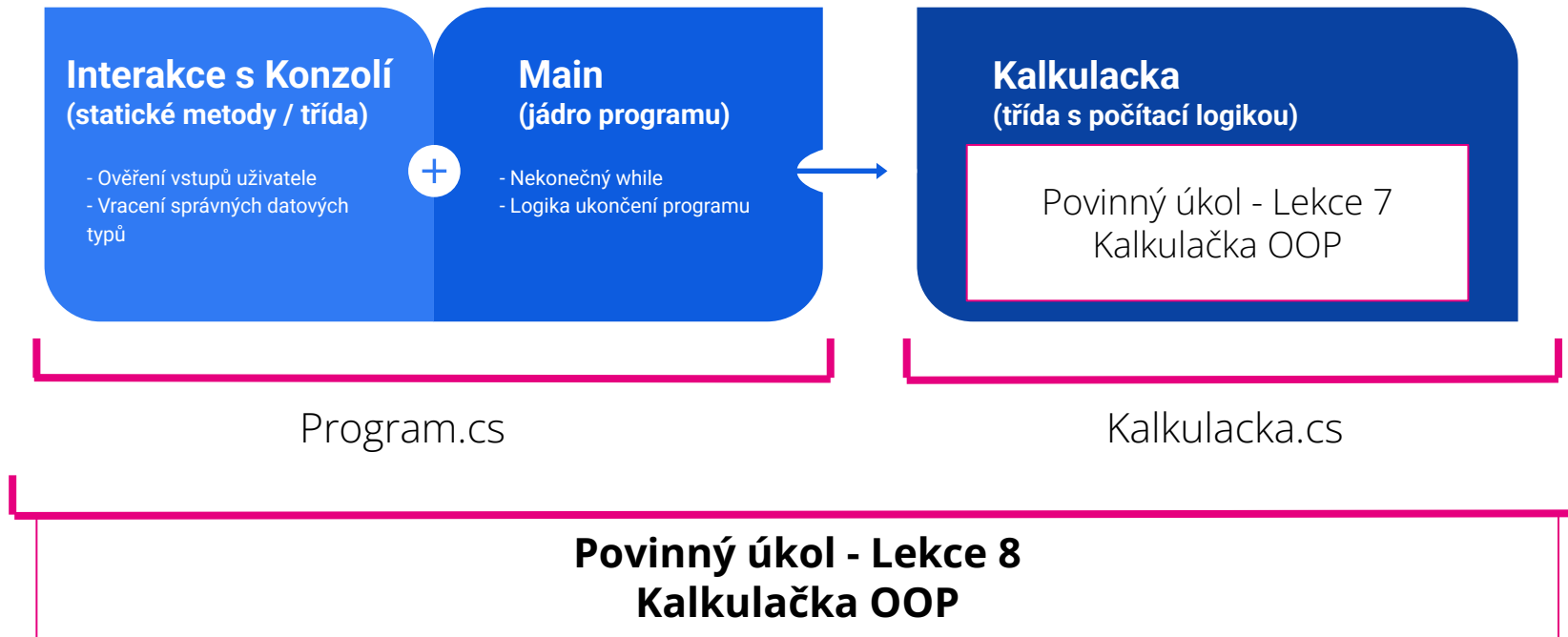




# ZÁVĚREČNÝ ÚKOL - NÁSTIN ZADÁNÍ



# ZÁVĚREČNÝ ÚKOL - NÁSTIN ZADÁNÍ



# ZÁVĚREČNÝ POVINNÝ ÚKOL - FLOW ODEVZDÁVÁNÍ

Kalkulačka OOP - Alpha

18.3.

Kalkulačka OOP - Beta

25.3.

Kalkulačka OOP - Release

1.4

Nástřel, tak abychom získali co nejdříve zpětnou vazbu.

- Vyjasnění požadavků (**Requirements**)
- **Design** - první nástřel
- Logika může mít chyby
- **Proof of Concept**  
(jak si představuju, že by to mělo fungovat)

Logika funguje, najdou se ještě nějaké chybičky.

- Ověření funkčnosti (**Validate**)
- **Testování** očekávaného chování programu

Prostor pro vycytání chyb a vylepšení programu a kódu.

- Hezčí výstup do konzole
- Vylepšení použití pro uživatele
- **Refactoring**
  - jednoduchost
  - přehlednost
  - pojmenování

# ZÁVĚREČNÝ POVINNÝ ÚKOL - NEPOVINNÉ

VYBRÁNÍ

**Interakce s Konzolí**  
(statické metody / třída)

**Main**  
(jádro programu)



**Kalkulacka**  
(třída s počítací logikou)

Povinný úkol - Lekce 7  
OOP Kalkulačka

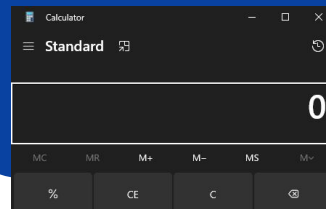
Program.cs

Kalkulacka.cs

Povinný úkol - Lekce 8  
Kalkulačka OOP

**Nepovinný úkol**  
**Grafická kalkulačka**

**Window - GUI**  
(grafické okno s tlačítky)



**KalkulackaWindow.cs**



# LEKCE 9

KOLEKCE



# KOLEKCE

- Datová struktura, která může obsahovat více hodnot najednou
- Každá hodnota má svůj unikátní **index**, podle kterého ji lze v kolekci jednoznačně identifikovat
- Kolekce je i objekt - obsahuje vlastnosti a metody

# Seznam - List



# KOLEKCE List

- `List<int>`, `List<string>[ ]`, ... = seznam, anglicky `list`
- seznam hodnot jednoho datového typu
- hodnoty: `{0}`, `{4, -1}`, `{"pes", "člověk", "321@", "b"}`
- vlastnost **Count** = počet prvků v listu
- **index**: celé číslo, začíná vždy od nuly
  - tj. list s ***n*** prvky má indexy **0** až ***n-1***
  - index se ohraničuje hranatými závorkami, např. třetí prvek pole s názvem `listCisel` značíme takto: `listCisel[2]`  
**POZOR!** Počítáme od nuly! Prvním prvkem je tak `listCisel[0]`!



# KOLEKCE List

- Vytvoření prázdného listu

```
List<int> listCisel = new List<int>();
```

- Přiřazení hodnot do existujícího pole:

```
listCisel.Add(3);  
listCisel.Add(0);
```

- Vytvoření listu rovnou s přiřazením hodnot

```
List<int> listCisel2 = new List<int>() { 3, 0, 1, -30, 1000 };
```

# KOLEKCE List

Výpis všech hodnot v listu

```
for (int i = 0; i < listStringu.Count; i++)  
{  
    Console.WriteLine(listStringu[i]);  
}  
foreach (string s in listStringu)  
{  
    Console.WriteLine(s);  
}
```

# KOLEKCE List

Hromadný zápis do listu pomocí cyklu

```
List<int> nasobkyDvojky = new List<int>();  
for (int i = 0; i < 100; i++)  
{  
    nasobkyDvojky.Add(i * 2);  
}
```

# Pole - Arrays



# KOLEKCE Pole

- `int[]`, `string[]`, ... = tzv. pole, anglicky **array**
- kolekce s definovaným počtem hodnot jednoho datového typu
- hodnoty: `{0}`, `{4, -1}`, `{"pes", "člověk", "321@"}`, `"b"`
- vlastnost **Length** = délka pole = počet prvků v poli
- **index**: celé číslo, začíná vždy od nuly
  - tj. pole délky ***n*** má indexy **0** až ***n-1***
  - index se ohraničuje hranatými závorkami, např. **třetí** prvek pole s názvem `poleCisel` značíme takto: `poleCisel[2]`  
POZOR! Počítáme od nuly! **Prvním** prvkem je tak `poleCisel[0]`!

# KOLEKCE Pole

- Vytvoření pole délky 5

```
int[] poleCisel = new int[5];
```

- Přiřazení hodnot do existujícího pole:






```
poleCisel[0] = 3;  
poleCisel[1] = 0;
```

- atd. až do indexu 4

- Vytvoření pole rovnou s přiřazením hodnot

```
int[] poleCisel2 = new int[] { 3, 0, 1, -30, 1000 };
```

# Kdy použít Array oproti List?

-  Počet prvků je neměnný
-  Potřebuješ výkon (rychlost/paměť)
-  Chceš vyjádřit pevnou velikost vstupu/výstupu
-  Potřebuješ interoperabilitu s jinými systémy
-  Pracuješ s vícerozměrnými strukturami

# LEKCE 10

Blazor - intro





# ZÁVĚREČNÝ POVINNÝ ÚKOL - FLOW ODEVZDÁVÁNÍ

Kalkulačka OOP - Alpha

18.3.

Kalkulačka OOP - Beta

25.3.

Kalkulačka OOP - Release

1.4

Nástřel, tak abychom získali co nejdříve zpětnou vazbu.

- Vyjasnění požadavků (**Requirements**)
- **Design** - první nástřel
- Logika může mít chyby
- **Proof of Concept**  
(jak si představuju, že by to mělo fungovat)

Logika funguje, najdou se ještě nějaké chybičky.

- Ověření funkčnosti (**Validate**)
- **Testování** očekávaného chování programu

Prostor pro vycytání chyb a vylepšení programu a kódu.

- Hezčí výstup do konzole
- Vylepšení použití pro uživatele
- **Refactoring**
  - jednoduchost
  - přehlednost
  - pojmenování

# ZÁVĚREČNÝ POVINNÝ ÚKOL - NEPOVINNÉ

## VYHODNĚNÍ

Interakce s Konzolí  
(statické metody / třída)

Main  
(jádro programu)

Kalkulacka  
(třída s počítací logikou)

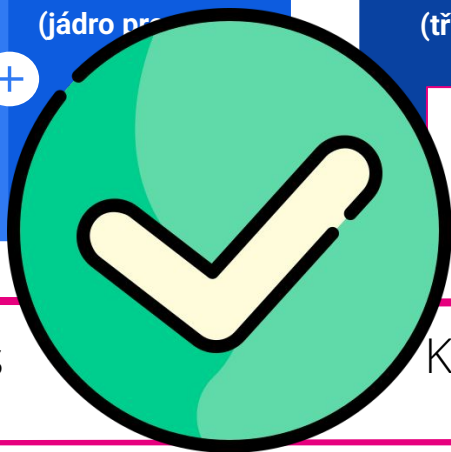


Povinný úkol - Lekce 7  
OOP Kalkulačka

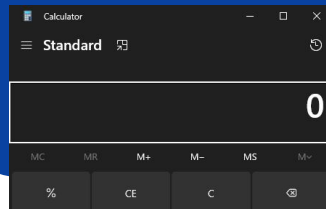
Program.cs

Kalkulacka.cs

Povinný úkol - Lekce 8  
Kalkulačka OOP



Window - GUI  
(grafické okno s tlačítky)



KalkulackaWindow.cs



Blazor  
(Lekce 10, 11, 12)

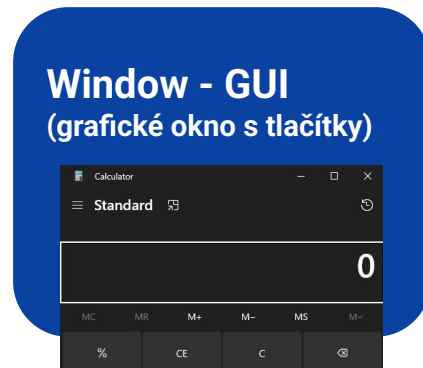
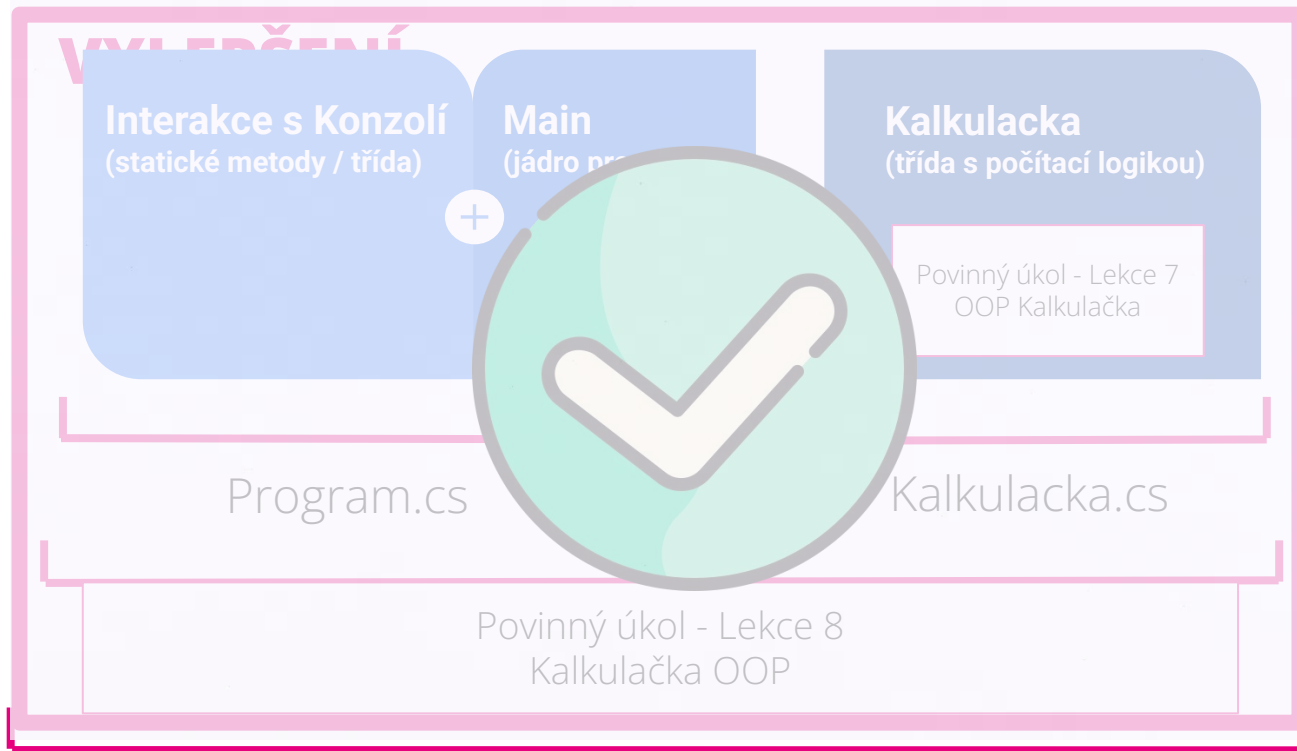
**Nepovinný úkol**  
**Grafická kalkulačka**

# LEKCE 10

Blazor - intro



# ZÁVĚREČNÝ POVINNÝ ÚKOL - NEPOVINNÉ



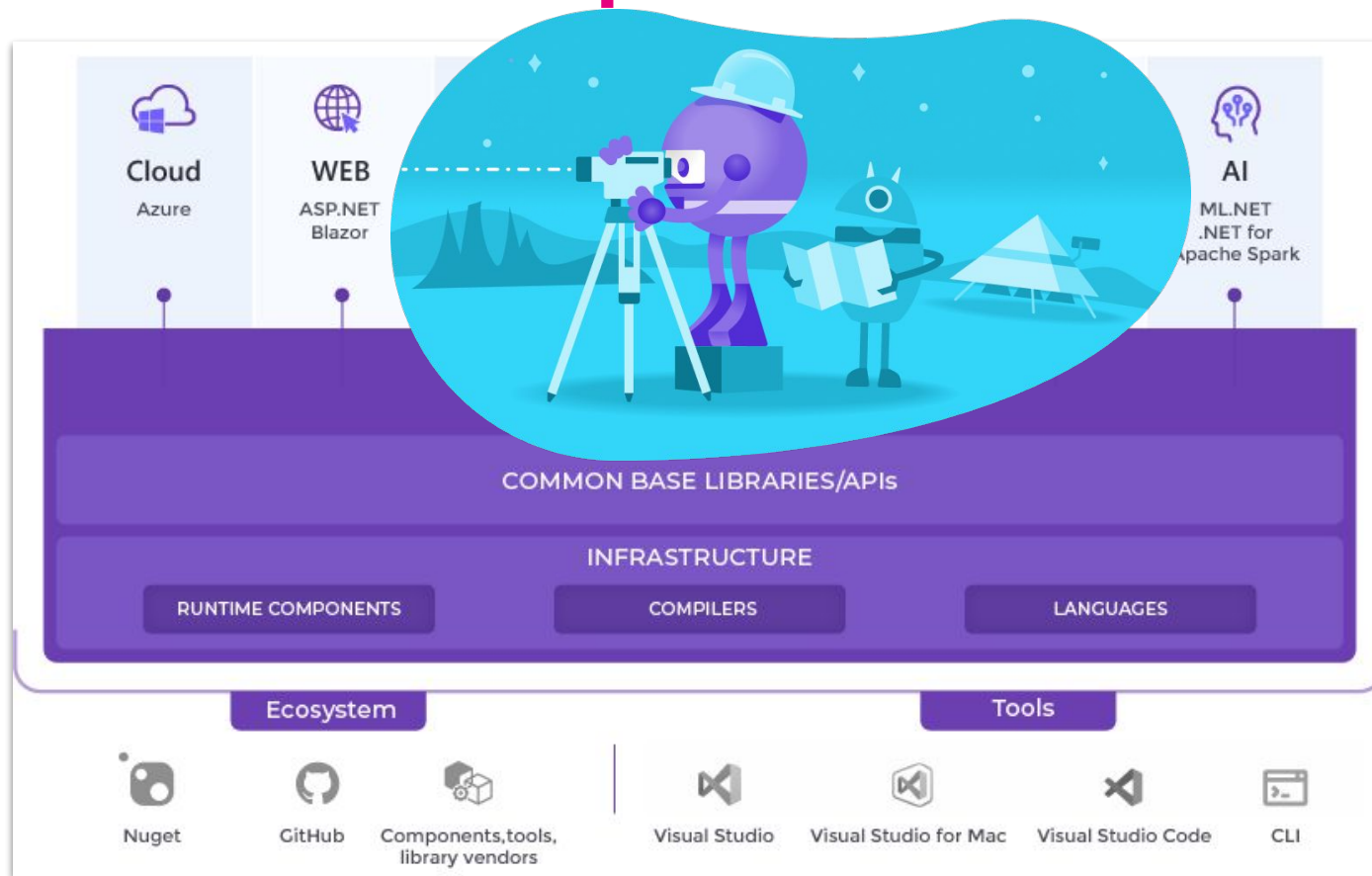
**KalkulackaWindow.cs**



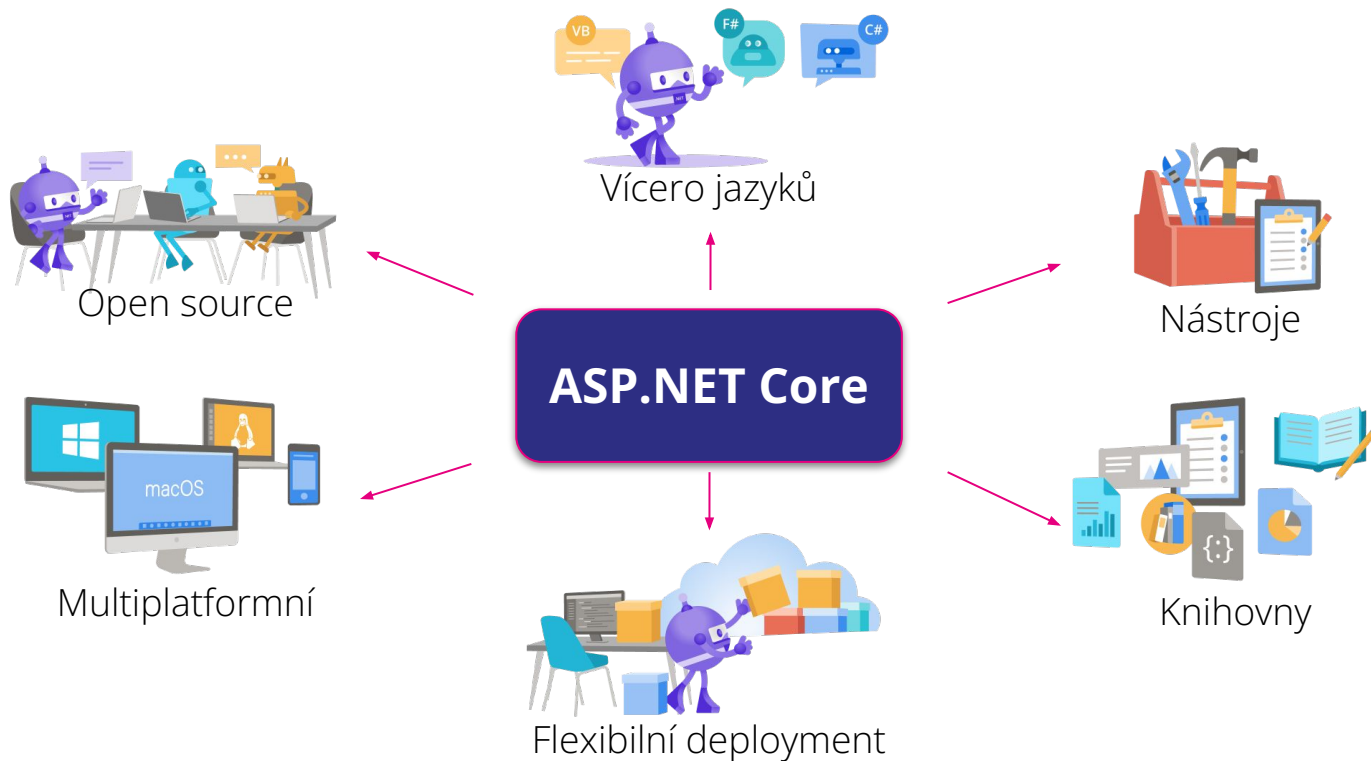
Blazor  
(Lekce 10, 11, 12)

**Nepovinný úkol**  
**Grafická kalkulačka**

# Na co vše se dá .NET použít



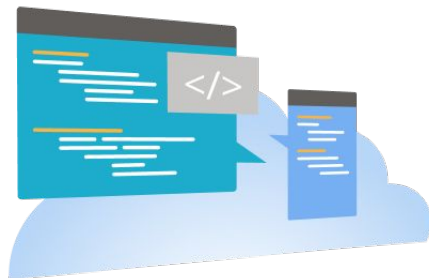
# Co je to ASP.NET Core?



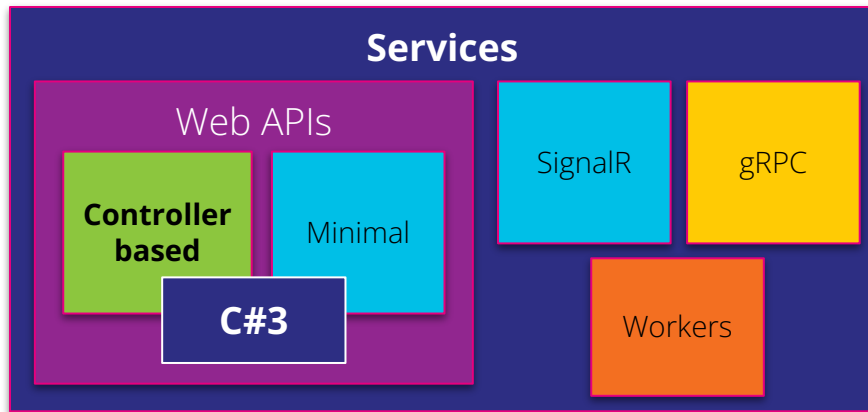
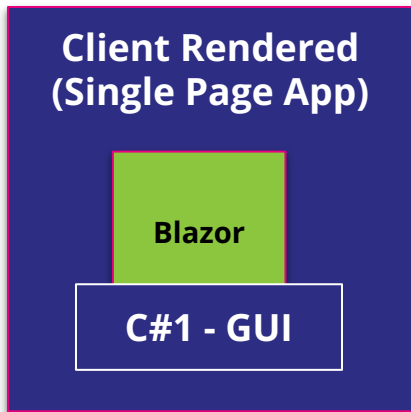
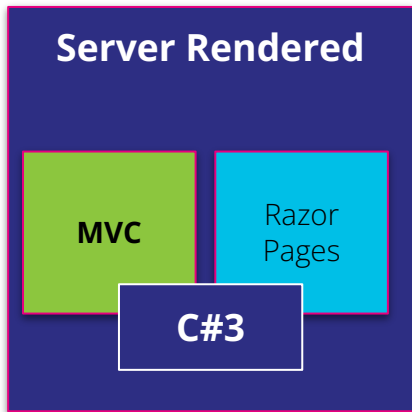
# Co se dá s ASP.NET Core vytvořit?



**Web UI**



**Backend Services**



# Co se dá s ASP.NET Core vytvořit?



**Web UI**



**Backend Services**

## Server Rendered

MVC

Razor  
Pages

C#3

## Client Rendered (Single Page App)

Blazor

C#1 - GUI

## Services

### Web APIs

Controller  
based

Minimal

C#3

SignalR

gRPC

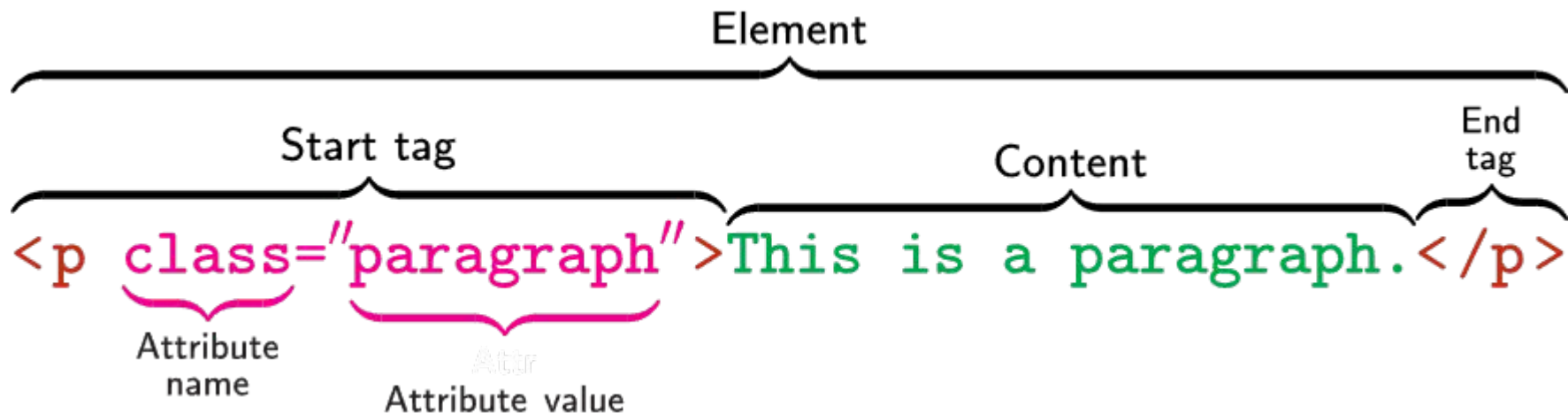
Workers



**Hello World v Blazoru**



# HTML syntax



# Základní HTML tagy

- **<h1>**Titulek**</h1>**
- **<p>**Paragraf**</p>**
- **<br>** - Nový řádek



# Základní Razor syntax

- **@page** "/relativniCestaKeStrance"
- **@code**  
{  
    zde píšeme vlastní C# kód, jak jsme zvyklí  
}
- **@nazevPromenne**
- **@if, @for, @foreach**

# Tlačítka

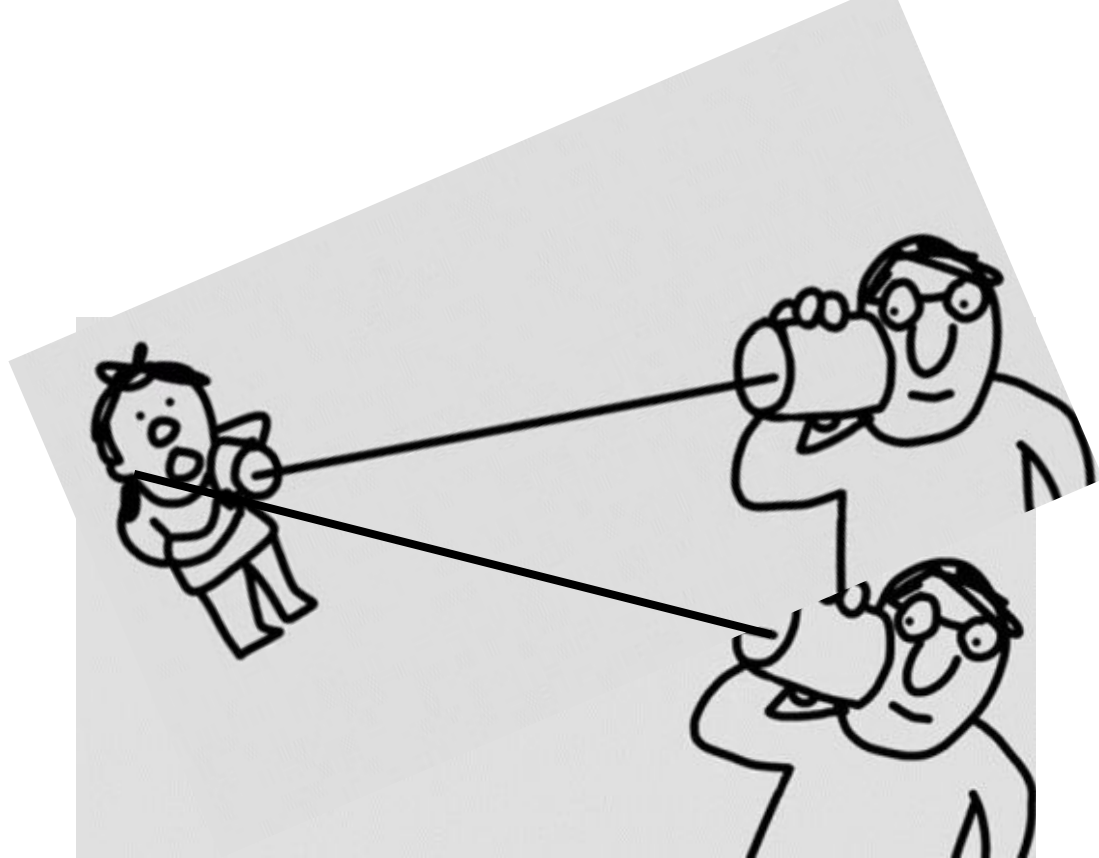
<button>

**@onclick="() => MetodaCoSeMaZavolatPoKliknuti()"**

# EVENTY

Ale jen velmi zjednodušeně ;)



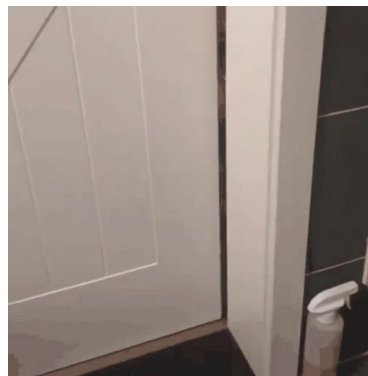




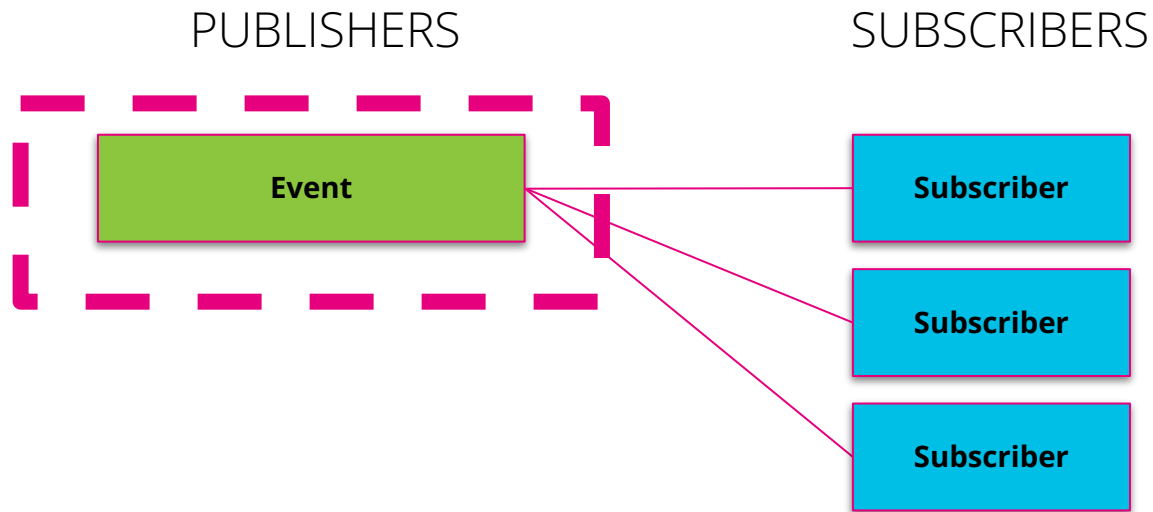
**Nikdo není doma**



## Doma se koná narozeninová oslava

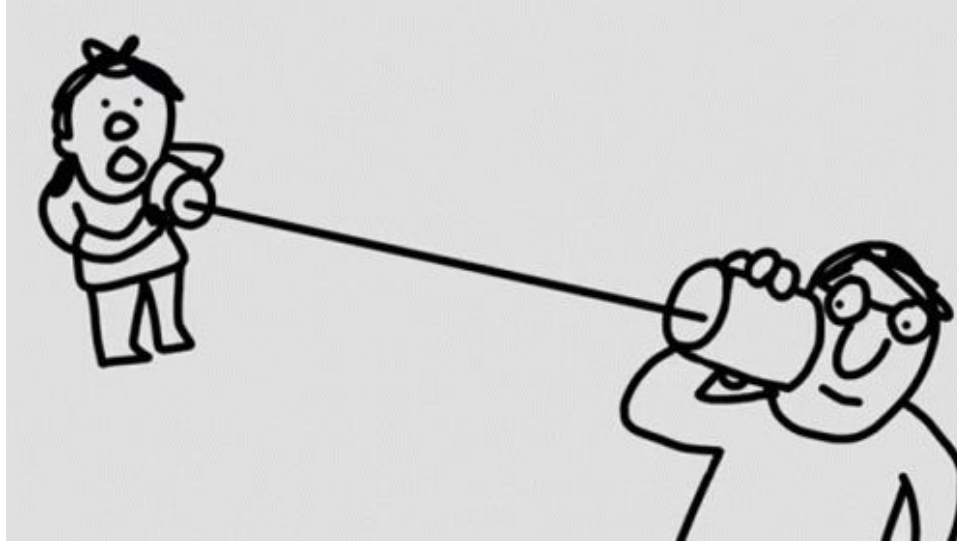


# Co je to event, subscriber, publisher

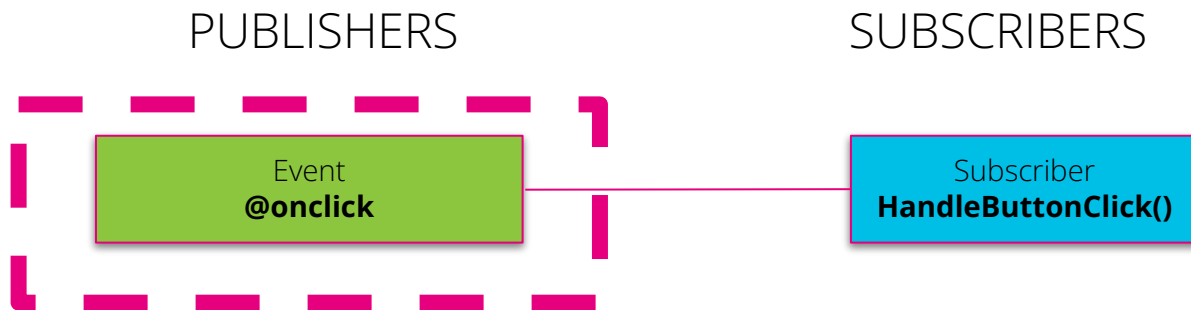


PUBLISHERS

SUBSCRIBERS



# Co je to event, subscriber, publisher



# K čemu je to dobré?



# Zajímavé projekty

- [Diablo Blazor](#)
- [Trains.NET](#)
- [Explain Face Recognition](#)
- [Habit Tracker](#)

# Extra zdroje na závěr

- [Interaktivní HTML editor](#)
- [Awesome Blazor](#)
- [Blazor school](#)



# LEKCE 12

Závěrečná



# HODNOTOVÉ A REFERENČNÍ TYPY

## Hodnotové typy:

**int, double, char, bool**

- Kopírují hodnotu.
- Každý objekt má vlastní kopii (např. **int**, **double**).
- Změny v jedné proměnné neovlivní jinou.

## Referenční typy:

**class, string, List<>, int[]**

- Sdílejí stejný objekt v paměti.
- Když změníte obsah objektu, všechny proměnné, které na něj ukazují, vidí tuto změnu (např. **class**, **int[]**).

# UŽITEČNÉ ODKAZY

# CZ

- Visual Studio
  - <https://visualstudio.microsoft.com/cs/vs/getting-started/>
- Oficiální MS dokumentace k C#
  - <https://learn.microsoft.com/cs-cz/dotnet/csharp/>

# EN

- Oficiální MS dokumentace k C#
  - <https://learn.microsoft.com/cs-cz/dotnet/csharp/>
  - Oficiální MS kurz [Start with C#](#)
  - Další MS kurzy [Learning Paths .NET](#)
- Opakování do mobilu
  - <https://www.sololearn.com/en/learn/courses/c-sharp-introduction>
- Online kurzy
  - [Microsoft Virtual Academy](#), [Pluralsight](#), [Udemy](#), [LinkedIn Learning](#), atd.
- Procvičení algoritmizace
  - [HackerRank](#), [LeetCode](#)

# Developer roadmaps

- Verzování - Git, GitHub
  - <https://roadmap.sh/git-github>
- Opakování
  - <https://www.w3schools.com/CS/index.php>
- ASP.NET Core
  - <https://roadmap.sh/aspnet-core>
- Build your own X
  - <https://github.com/codecrafters-io/build-your-own-x>
- Příprava CV a pohovorů

# Poděkování