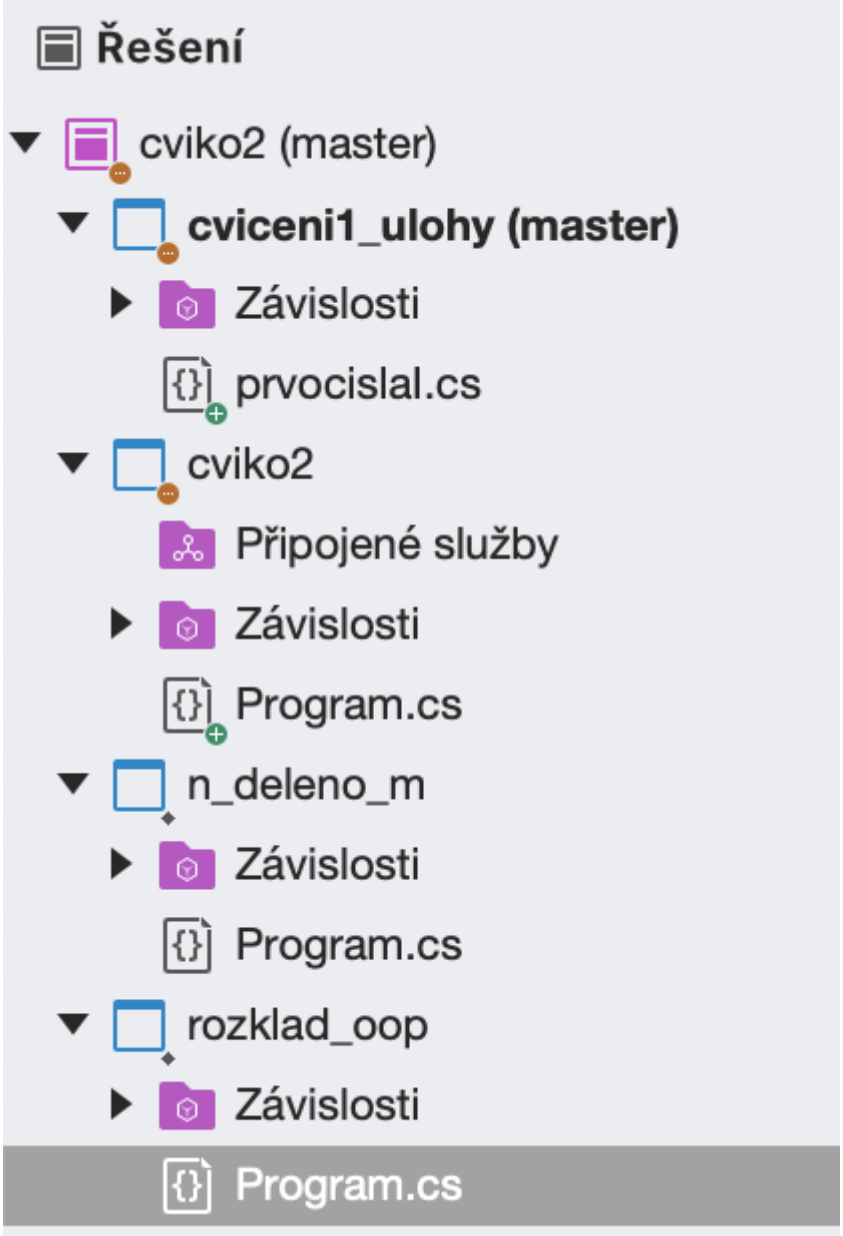
tahák - 1.cviko

Syntax

Místo odsazení (Python) se používají složené závorky `{}`, do nich je třeba uzavřít každý blok, který byste v Pythonu odsazovali.

Ve Visual Studiu si vytvoříte Solution (řešení), které může mít několik Projects (projektů).



Například já mám na každé cviko jedno solution a v něm projekt na každý příklad.
Když spustíte program, pustíte se metoda Main v třídě Program (v souboru Program.cs). Při spuštění lze vybrat, který projekt se má spusitit.

Na začátku vypadá Program.cs (soubor můžete libovolne přejmenovat) takto:

```
using System; //importujeme knihovnu system, novejsi Visual Studio nepotrebuje

//Jmeny prostor ma defualtne jmeno dle projektu. Vsechna jmena (promennych,
```

```
trid...) musi byt v ramci jednoho prostoru unikatni.
namespace cviko
{
    class Program //vsechno je objekt, i vas program
    {
        // toto se vola pri spusteni
        // metoda nic nevraci (void)
        // a zere pole stringu na vstupu (budeme probirat pozdeji jak to tam
nacpat)
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!"); //tady je to co chcete aby
program delal.
        }
    }
}
```

Načtení a výpis na konzoli

```
Console.ReadLine(); //nacte jednu radku
Console.Read();     //nacte jeden znak

Console.WriteLine(); //vypise a odradkuje
Console.Write();     //vypise bez odradkovani
```

Metody `Read()` a `ReadLine()` vrací String, ať už na konzoli zadáte cokoliv. Pokud výsledek chcete uložit do jiného datové typu, je třeba návratovou hodnotu přetypovat, v následujícím případě tedy zavolat statickou metodu Parse definovanou na třídě Int32. (int místo Int32 je jenom syntactic sugar, funguje obojí).

```
int radka_vstupu = int.Parse(Console.ReadLine());
```

Cykly

For

```
for(int i = 0; i <= 6; i++)
{
  //tady se neco deje
}
```

Pozor! za první řádkou for- a while- cyklu není středník. Kdyby tam byl, bere se to jako for-cyklus bez těla, který proběhne a nic se v něm nestane.

While

```
while(x < 5)
{
  //dokud je x mensi nez 5, neco tady delej
}
```

Podmínky (logické výrazy)

`&&` ... and
`||` ... or
`!` ... not

```
if (((x <6) && (y>= 3) && !nepocitej) || x != 3 )
{
  // proved tenhle blok kdyz
  // x je mensi nez 6, y je alespon 3 a boolean promenna nepocitej ma hodnotu
false (tedy jeji negace hodnotu true)
  // nebo kdyz aspon plati, ze x neni 3
}
```

Není třeba všechny věci při použití logických operátorů závorkovat, platí priorita operátorů jako v logice (a právě proto je závorkování někdy lepší, aby vás výsledek nepřekvapil).