

# Programování

Vlastní metody / funkce





# Opakování – funkce

- ▶ = **blok kódu, vykonávající konkrétní úlohu**
- ▶ Na funkci lze pohlížet i jako na podprogram algoritmu
- ▶ Proměnné deklarované uvnitř bloku nelze číst mimo funkci
- ▶ Výhody využívání funkcí:
  - ▶ Zamezení duplicitního kódu
  - ▶ Lepší čitelnost
  - ▶ Lepší odhalování chyb při ladění a debugingu
  - ▶ Snazší rozšiřitelnost aplikace



# Opakování - funkce

- ▶ Nově vytvářená hlavička funkce musí mít tyto náležitosti:
  - ▶ **Název funkce** – název začíná vždy velkým písmenem
  - ▶ **Viditelnost funkce** – specifikace přístupnosti
  - ▶ **Návratový datový typ** – jaký typ hodnoty očekáváme, že funkce vrátí
  - ▶ **Výčet vstupních hodnot** – specifikace datového typu (může být i prázdný)
- ▶ Další specifika funkce jsou volitelná podle potřeby
- ▶ Blok kódu se vykoná v momentě, kdy je funkce volána




# Ukázka vlastní funkce a jejího volání

```
/// <summary>  
/// Funkce pro získání náhodného desetinného čísla vyššího,  
/// než je vstupní hodnota  
/// </summary>  
/// <param name="start">základní hodnota</param>  
/// <returns>desetinné číslo vyšší než základ</returns>
```

Počet odkazů: 1

```
public double FooFrac(double start)  
{  
    return start + new Random().NextDouble();  
}
```

```
double vysledek = FooFrac(3.8);
```



# Procvičování – hlavičky funkcí

1. Funkce suma, která vrátí součet dvou načtených hodnot
2. Funkce rozhodující, zda je celočíselná hodnota sudá
3. Funkce, která zobrazí ve vyskakovacím okně jméno a věk uživatele, které přijímá jako vstup
4. Funkce, která vrátí pole součinů hodnot dvou různých, ale stejně dlouhých polí
5. Funkce, vracející aritmetický průměr hodnot pole
6. Funkce vracející pravoúhlé pole řetězců o rozměrech NxN

# Přetížení funkcí

- ▶ Vytváření funkcí se stejným názvem
  - ▶ S různým počtem atributů
  - ▶ S odlišným datovým typem atributu
- ▶ Často na přetížení funkcí narazíme u polymorfismu
- ▶ Na základě typu objektu se konkrétní funkce chová

```
public double FooFrac(double start)
{
    return start + new Random().NextDouble();
}
```

Počet odkazů: 0

```
public double FooFrac(int start)
{
    return start + new Random().NextDouble();
}
```

Počet odkazů: 0

```
public double FooFrac(double start, double end)
{
    return start / end;
}
```

# Rekurzivní funkce

- Situace kdy funkce volá samu sebe
  - Přímé volání
  - Nepřímé volání
- Při vytváření rekurzivní funkce je třeba dbát na rekurzivní zarážku
- Stav, situace, hodnota, kdy se rekurze zastaví a začne se „vynořovat“
- Rekurzivní volání musí vždy pracovat s menší sadou dat než v předchozím voláním

```
/// <summary>  
/// Určení hodnoty Fibonacciho posloupností  
/// </summary>  
/// <param name="n"> pořadí v posloupnosti</param>  
/// <returns>hodnota na dané pozici</returns>
```

Počet odkazů: 2

```
public int Fibonacci(int n)  
{  
    if (n == 0 || n == 1) return 1;  
    return Fibonacci(n - 1) + Fibonacci(n - 2);  
}
```