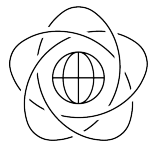
Programming 3 Advanced

C# Basics

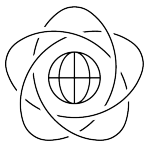
Tomasz Herman

Faculty of Mathematics and Information Science
Warsaw University of Technology

Lecture 2, 22 października 2024

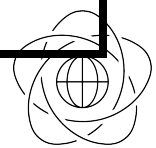


Outline



Syntax

```
1  class Person
2  {
3      private string _name;
4      public int Age { get; set; }
5
6      public Person(string name, int age)
7      {
8          (_name, Age) = (name, age);
9      }
10
11     public void Show()
12     {
13         Console.WriteLine($"{_name}, {Age}");
14     }
15 }
```



Naming convention

PascalCase

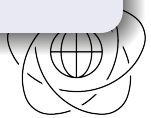
- class/struct names
- namespaces
- all public members

camelCase

- method parameters
- local variables
- private instance fields (sometimes starting with an underscore)

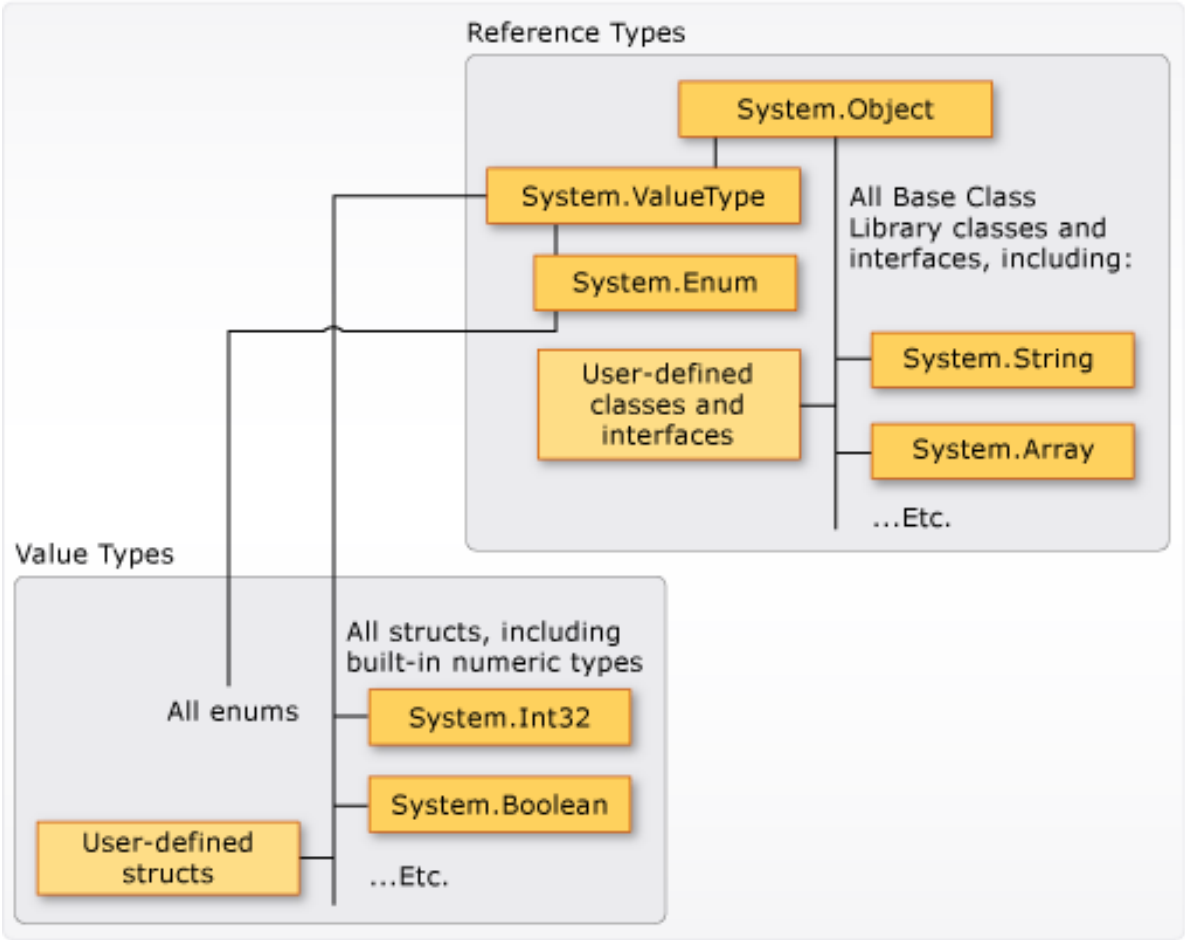
Documentation

`https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/identifier-names#naming-conventions`



C# Types

- Value Types
- Reference Types
- Pointer Types



Documentation

`https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/#the-common-type-system`

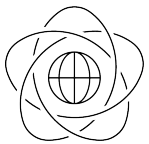
Value Types

Built-in value types:

- numeric types (int, uint, float, etc.)
- char
- bool

Custom value types:

- struct
- enum



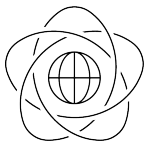
Reference Types

Built-in reference types:

- arrays
- string
- object

Custom reference types:

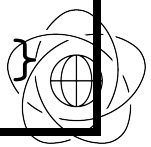
- class



Value vs Reference Types

Assignment semantics

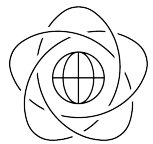
```
1  Point p1 = new Point();
2  p1.X = 3; p1.Y = -5;
3  Point p2 = p1;
4  p2.X = 1; p2.Y = -1;
5  Console.WriteLine($"P1: {p1.X}, {p1.Y}");
6  Console.WriteLine($"P2: {p2.X}, {p2.Y}");
7
8  Person anne = new Person();
9  anne.Name = "Anne"; anne.Age = 16;
10 Person bob = anne;
11 bob.Name = "Bob"; bob.Age = 18;
12 Console.WriteLine($"Anne: {anne.Name}, {anne.Age}");
13 Console.WriteLine($"Bob: {bob.Name}, {bob.Age}");
14
15 public struct Point { public float X, Y; }
16 public class Person { public string Name; public int Age; }
```



Value vs Reference Types

`null` values

```
1  Point p = new Point();
2  p.X = 3; p.Y = -5;
3  p = null; // Compilation error
4
5  Person anne = new Person();
6  anne.Name = "Anne"; anne.Age = 16;
7  anne = null; // OK
8
9  public struct Point { public float X, Y; }
10 public class Person { public string Name; public int Age; }
```



Numeric Types

Signed Integers Types

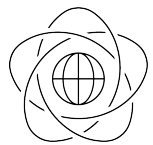
- sbyte
- short
- int
- long
- nint

Unsigned Integers

- byte
- ushort
- uint
- ulong
- nuint

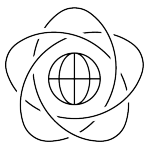
Real Types

- float
- double
- decimal



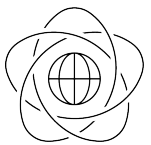
Singed Integer Types

Type	System Type	Size	Range	Suffix
sbyte	SByte	8 bits	-2^7 to $2^7 - 1$	
short	Int16	16 bits	-2^{15} to $2^{15} - 1$	
int	Int32	32 bits	-2^{31} to $2^{31} - 1$	
long	Int64	64 bits	-2^{63} to $2^{63} - 1$	l, L
	Int128	128 bits	-2^{127} to $2^{127} - 1$	
nint	IntPtr	32/64 bits		



Unsinged Integer Types

Type	System Type	Size	Range	Suffix
byte	Byte	8 bits	0 to $2^8 - 1$	
ushort	UInt16	16 bits	0 to $2^{16} - 1$	
uint	UInt32	32 bits	0 to $2^{32} - 1$	u, U
ulong	UInt64	64 bits	0 to $2^{64} - 1$	ul, UL
	UInt128	128 bits	0 to $2^{128} - 1$	
nuint	UIntPtr	32/64 bits		

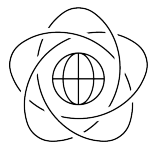


Real Types

Type	System Type	Size	Range	Suffix
	Half	16 bits	± 65504	
float	Single	32 bits	$\pm 3.4 \cdot 10^{38}$	f, F
double	Double	64 bits	$\pm 1.7 \cdot 10^{308}$	d, D
decimal	Decimal	128 bits	$\pm 7.9 \cdot 10^{28}$	m, M

Documentation

`https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types`



Numeric Literals

```
1  int i = 1000;
2  long hex = 0x4F_00_00_00;
3  int million = 1_000_000;
4  int binary = 0b0000_1001_0101_1010;
5  double d = 1.034;
6  float x = 1.234f;
7  double scientific = 1.4e-3;
8  decimal money = 16.99M;
```

Documentation - Integer Literals

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types#integer-literals

Documentation - Float Literals

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types#real-literals

Numeric Conversions

```
1  int i = 1000;
2  long l = i; // implicit conversion
3  short s = (short) i; // explicit conversion
4
5  float f = 1000.25f;
6  double d = f;
7  decimal m1 = i;
8  decimal m2 = (decimal)f;
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/numeric-conversions

Documentation - Parse and Convert

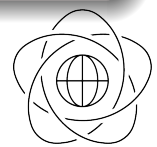
https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/types/how-to-convert-a-string-to-a-number

Bool Type

```
1  bool t = true;
2  bool f = false;
3  bool equal = (1 == 2);
4  bool s1 = t && f || !equal;
5  bool s2 = t & f | !equal;
6
7  static int Min(int a, int b)
8  {
9      return (a < b) ? a : b;
10 }
```

Conversions

There are no conversions between `bool` and numeric types.



char type

The `char` type keyword is an alias for the `System.Char` type.

- It represents a Unicode `UTF-16` character (2 bytes)
- `char` literal is specified within single quotes
- The default value of the `char` type is `'\0'`
- Characters that need escaping:
  `\'` `\"` `\\` `\0` `\a` `\b` `\f` `\n` `\r` `\t` `\v`

```
1  char a = 'a';                // char literal
2  char newLine = '\n';         // escaped character
3  char unicode = '\u00A9';     // unicode sequence
4  char hex = '\x005C'          // hexadecimal sequence
```

Conversions

An implicit conversion from a char to a numeric type works for the numeric types that can accommodate an unsigned short. For other numeric types, an explicit conversion is required.

string

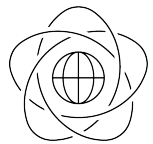
```
1 string str = "Just a text\n";
2 Console.WriteLine(str == "Just a text\n"); // true
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/strings/

Immutability

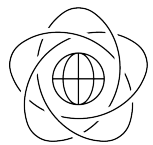
String objects are immutable: they can't be changed after they're created. All of the string methods and C# operators that appear to modify a string actually return the results in a new string object.



string

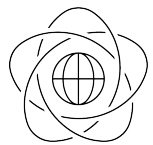
constructors and properties

```
1  char[] chars = new {'w', 'o', 'r', 'l', 'd'};
2  string fromLiteral = "Hello";
3  string fromArray = new string(chars);
4  string fromSubArray = new string(chars, 1, 2);
5  string repeatedChar = new string(' ', 4);
6  string concatenated = fromLiteral + ' ' + fromArray;
7  Console.WriteLine(concatenated);
8  Console.WriteLine("string Length: " + concatenated.Length);
9  char space = concatenated[5];
10 concatenated[5] = '_'; // compilation error, read-only
```



Verbatim string literals

```
1  string filePath = @"C:\Users\scoleridge\Documents\";
2  //Output: C:\Users\scoleridge\Documents\
3  string text = @"My pensive SARA ! thy soft cheek reclined
4      Thus on mine arm, most soothing sweet it is
5      To sit beside our Cot,...";
6  /* Output:
7  My pensive SARA ! thy soft cheek reclined
8      Thus on mine arm, most soothing sweet it is
9      To sit beside our Cot,...*/
10 string quote = @"Her name was ""Sara.""";
11 //Output: Her name was "Sara."
```



Raw string literals (C# 11)

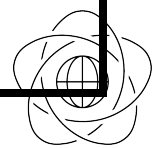
```
1  var str1 = """This is a "raw string literal"."""";
2  var str2 = """It can contain characters like \, ' and ".""";
3  var xml = """
4            <element attr="content">
5                <body>
6                </body>
7            </element>
8            """;
9  var str3 = """"
10             """Raw string literals""" can start
11             and end with more than three
12             double-quotes when needed.
13             """";
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/raw-string

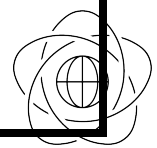
String interpolation

```
1  string author = "George Orwell";
2  string book = "Nineteen Eighty-Four";
3  int year = 1949;
4  decimal price = 19.50m;
5  string description = $"{author} is the author of {book}. \n" +
6  $"The book price is {price:C}, it was published in {year}.";
7  Console.WriteLine(description);
8
9  Console.WriteLine($"Number 1: {1.0,10:C}");
10 Console.WriteLine($"Number 2: {12.5,10:C}");
11 Console.WriteLine($"Number 3: {123.45m,10:C}");
12
13 var random = new Random();
14 Console.WriteLine(
15 $"Coin flip: {(random.NextDouble() < 0.5 ? "heads" : "tails")}"
16 );
```



String methods

```
1  string example = "Showcasing C\# strings";
2  string sub = example.Substring(11, 2);
3  Console.WriteLine($"Substring: {sub}");
4  bool contains = example.Contains("C#");
5  Console.WriteLine($"Contains 'C#': {contains}");
6  string replaced = example.Replace("Showcasing", "Demo of");
7  Console.WriteLine($"Replace: {replaced}");
8  string upper = example.ToUpper();
9  Console.WriteLine($"Uppercase: {upper}");
10 string[] words = example.Split(' ');
11 Console.WriteLine("Split:");
12 foreach (string word in words)
13 {
14     Console.WriteLine(word);
15 }
16 string joined = string.Join(", ", words);
17 Console.WriteLine($"Join: {joined}");
```

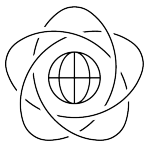


StringBuilder

```
1 StringBuilder stringBuilder = new StringBuilder("Hello, ");
2
3 stringBuilder.Append("this is ");
4 stringBuilder.Append("a simple ");
5 stringBuilder.Append("StringBuilder demo.");
6
7 Console.WriteLine(stringBuilder.ToString());
```

Documentation

https://learn.microsoft.com/en-us/dotnet/standard/base-types/stringbuilder



String - further reading

System.String class

https://learn.microsoft.com/en-us/dotnet/api/system.string?view=net-8.0

Supplementary remarks

https://learn.microsoft.com/en-us/dotnet/fundamentals/runtime-libraries/system-string

Strings and literals

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/strings/

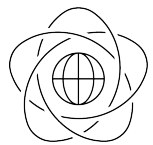
String constructors

https://learn.microsoft.com/en-us/dotnet/fundamentals/runtime-libraries/system-string-ctor

The object type

- Universal Base Type
- Reference Type

```
1 object anInt = 3;
2 object aString = "I'm an object"
3 object[] array = new object[] {3, DateTime.Now, "string"};
```

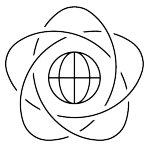


Boxing and Unboxing

```
1  int num = 42;
2  object obj = num; // Boxing
3
4  int unboxedNum = (int)obj; // Unboxing
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing

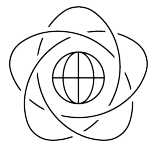


Common methods

Since all types inherit from object, they inherit the following methods:

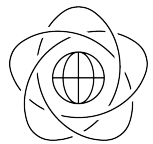
- ToString()
- GetHashCode()
- Equals(object obj)
- GetType()

```
1 int i = 15;
2 Console.WriteLine(i.ToString());
3 Console.WriteLine(i.GetType());
4 Console.WriteLine(i.GetHashCode());
5 Console.WriteLine(3.Equals(i));
```



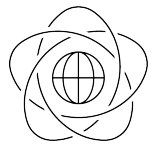
GetType and typeof operator

```
1  DateTime now = DateTime.Now;
2
3  Console.WriteLine(now.GetType()); // evaluated at runtime
4  Console.WriteLine(typeof(now)); // evaluated at compile time
5
6  Console.WriteLine(typeof(now) == now.GetType());
```



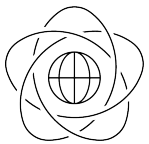
ToString

```
1  public class Person
2  {
3      public string Name { get; set; }
4
5      public override string ToString() => Name;
6  }
7
8  Person person = new Person {Name = "Alice"};
9  Console.WriteLine(person);
```



All object members

```
1  public class Object
2  {
3      public Object();
4      public extern Type GetType();
5      public virtual bool Equals(object obj);
6      public static bool Equals(object objA, object objB);
7      public static bool ReferenceEquals(object objA,
8                                         object objB);
9      public virtual int GetHashCode();
10     public virtual string ToString();
11     protected virtual void Finalize();
12     protected extern object MemberwiseClone();
13 }
```

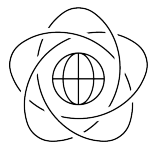


Types of Arrays

- Single Dimensional
- Multidimensional
  - Rectangular
  - Jagged

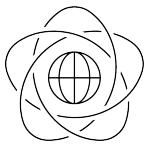
Documentation

`https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/arrays#implicitly-typed-arrays`



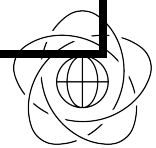
Single Dimensional Arrays

```
1  int[] primes = new int[] {2, 3, 5, 7, 11};
2  char[] vowels = {'a', 'e', 'i', 'o', 'u'};
3  uint[] even = [0, 2, 4, 6, 8]; // C# 12
4  float[] data = new float[10];
5  Array array = primes;
6
7  Console.WriteLine($"Primes array length: {primes.Length}");
8  for (int i = 0; i < primes.Length; i++)
9  {
10     Console.WriteLine(primes[i]);
11 }
```



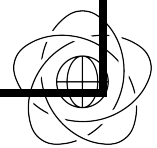
Rectangular Arrays

```
1  float[,] matrix = new float[,]
2  {
3      {1.0f, 0.0f, 0.0f},
4      {0.0f, 1.0f, 0.0f},
5      {0.0f, 0.0f, 1.0f}
6  };
7  float[,] matrix3x4 = new float[3, 4];
8
9  for (int i = 0; i < matrix.Length(0); i++)
10 {
11     for (int j = 0; j < matrix.Length(1); j++)
12     {
13         Console.WriteLine($"m[{i}, {j}] = {matrix[i, j]}");
14     }
15 }
```



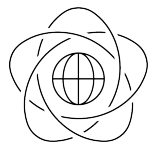
Jagged Arrays

```
1  float[][] matrix = new float[][]
2  {
3      new float[] {1.0f, 0.0f, 0.0f},
4      new float[] {0.0f, 1.0f, 0.0f},
5      new float[] {0.0f, 0.0f, 1.0f}
6  };
7  float[][] matrix3x4 = new float[3][];
8  for (int i = 0; i < matrix3x4.Length; i++)
9  {
10     matrix3x4[i] = new float[4];
11 }
12 float[][] matrix2x3 =
13 {
14     new float[] {1.0f, 0.0f, 2.0f},
15     new float[] {0.0f, 1.0f, 3.0f}
16 };
```



Array of Values vs Array of References

```
1  Point[] points = new Points[4];
2  Person[] people = new Person[4];
3
4  Console.WriteLine($"points[0] = {points[0].X}, " +
5                    $"{points[0].Y}");
6  Console.WriteLine($"people[0] = {people[0]}");
7
8  points[0].X = 10.5f; // OK
9  people[0].Name = "Anne"; // Runtime error,
10                          // NullReference exception
11
12 public struct Point { public float X, Y; }
13 public class Person { public string Name; public int Age; }
```



Indices and Ranges

```
1  int[] primes = new int[] {2, 3, 5, 7};
2  int firstElem = primes[0], secondElem = primes[1];
3  int lastElem = primes[^1], secondToLastElem = primes[^2];
4  Index first = 0;
5  Index last = ^1;
6  firstElem = primes[first]; lastElem = primes[last];
7
8  int[] firstTwo = primes[..2]; //exclusive end
9  int[] withoutFirst = primes[1..]; // inclusive start
10 int[] withoutLast = primes[..^1];
11 int[] withoutFirstAndLast = primes[1..^1];
12 int[] all = primes[..];
13 Range lastTwoRange = ^2..;
14 int[] lastTwo = primes[lastTwoRange];
```

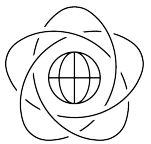
Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/tutorials/ranges-indexes

Statements

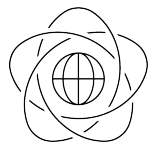
Types of statements

- Statement block
- Declaration statement
- Expression statement
- Selection statements
- Iteration statements
- Jump statements
- Other statements



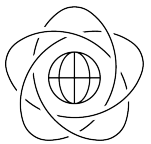
Statement block

```
1  {
2      int x = 1;
3      int y = x + 1;
4      Console.WriteLine(x + y);
5  }
6  {
7      float x, y; // OK, new scope
8  }
```



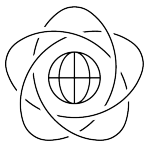
Declaration statement

```
1  int i;
2  float x, y = 1.0f, z;
3  string s = "Declaration statement"
4  const float Pi = 3.1415926f;
5  Pi += 1; // Compile-time error
```



Expression statements

```
1  // Declare variables with declaration statements:
2  string s;
3  int x, y;
4  StringBuilder sb;
5
6  // Expression statements
7  x = 1 + 2;                    // Assignment expression
8  x++;                          // Increment expression
9  y = Math.Max(x, 5);           // Assignment expression
10 Console.WriteLine(y);         // Method call expression
11 sb = new StringBuilder();     // Assignment expression
12 new StringBuilder();          // Object instantiation
13                               // expression
```



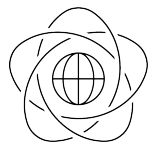
Selection statement

Available selection statements

- if statement
- switch statement
- switch expression statement (C# 8)

Documentation

`https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/selection-statements`



if statement

An if statement can be any of the two forms:

- with an else counterpart

```
1 if (condition)
2     statement1
3 else
4     statement2
```

- or without it

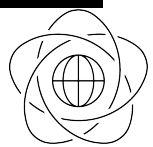
```
1 if (condition)
2     statement
```

Condition type

Unlike in C/C++ the condition must evaluate to `bool` type. Note that there is no conversion between `bool` and numeric types.

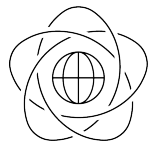
if statement - example

```
1  void DisplayWeatherReport(double tempInCelsius)
2  {
3      if (tempInCelsius < -273.15)
4      {
5          Console.WriteLine("Damn Physics.");
6          throw new ArgumentOutOfRangeException();
7      }
8      if (tempInCelsius < 15.0)
9          Console.WriteLine("Cold.");
10     else if (tempInCelsius > 25.0)
11         Console.WriteLine("Hot!");
12     else
13         Console.WriteLine("Perfect.");
14 }
```



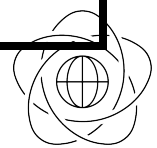
switch statement

```
1  switch (expression)
2  {
3      case pattern:
4          statement list
5      case pattern:
6      case pattern when condition: // optional case guard
7          statement list
8      default:
9          statement list
10 }
```



switch statement - example

```
1  void TellMeAboutTheObject(object obj)
2  {
3      switch (obj)
4      {
5          case 0: // constant pattern
6              Console.WriteLine("It's a zero.");
7              break;
8          case string str: // type pattern
9              Console.WriteLine($"It's a string: {str}");
10             break;
11         // type pattern with case guard
12         case DateTime dt when dt.DayOfWeek == DayOfWeek.Monday:
13             Console.WriteLine("It's a Monday");
14             break;
15         default:
16             Console.WriteLine("IDK");
17             break;
18     }
19 }
```

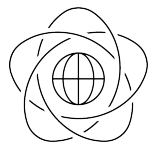


switch expression statement

```
1 type variable = input_expression switch
2 {
3     pattern => candidate_expression,
4     pattern when condition => candidate_expression,
5     pattern => candidate_expression
6 }
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/switch-expression



switch expression statement - example

```
1  string cardName = cardNumber switch
2  {
3      13 => "King", // constant pattern
4      12 => "Queen",
5      11 => "Jack",
6      > 1 and < 11 => "Pip card", // relational pattern
7      1 => "Ace",
8      // discard pattern, equivalent of default:
9      _ => throw new ArgumentOutOfRangeException()
10 };
```

Non-exhaustive switch expressions

If none of a switch expression's patterns matches an input value, the runtime throws an exception.

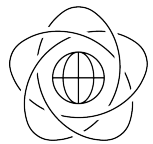
Iteration statements

Available iteration statements

- for loop
- foreach loop
- while loop
- do-while loop

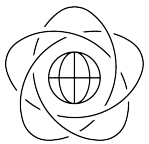
Documentation

`https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/iteration-statements`



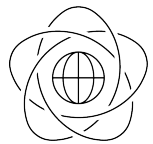
for loop

```
1 for (initializer; condition; iterator)
2     body
```



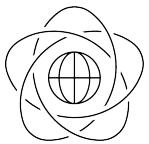
while and do-while loop

```
1 // while loop:
2 while (condition)
3     body
4 // do-while loop:
5 do
6     body
7 while (condition); // <- note the semicolon here
```



foreach loop

```
1  Span<int> span = new int[] {0, 1, 2, 3, 4};
2  foreach (ref int i in span)
3      i++;
4  foreach (int i in span)
5      Console.WriteLine(i);
6  foreach (char c in "foreach")
7      Console.WriteLine(c);
```



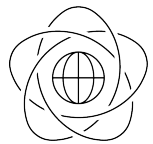
Jump statements

Available jump statements

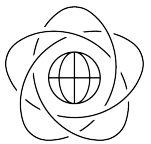
- break
- continue
- goto
- return
- yield

Documentation

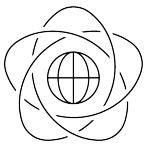
`https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/jump-statements`



break and continue statements

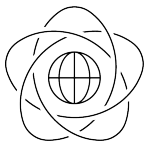


return statement



goto statement

```
1  int i = 1;
2  loop: // label
3  if (i <= 5)
4  {
5      Console.Write ($" {i}");
6      i++;
7      goto loop;
8  }
```



Other statements

- Empty statement
- Exception-handling statements
- checked/unchecked statements
- await statement
- yield return statement
- fixed statement
- lock statement