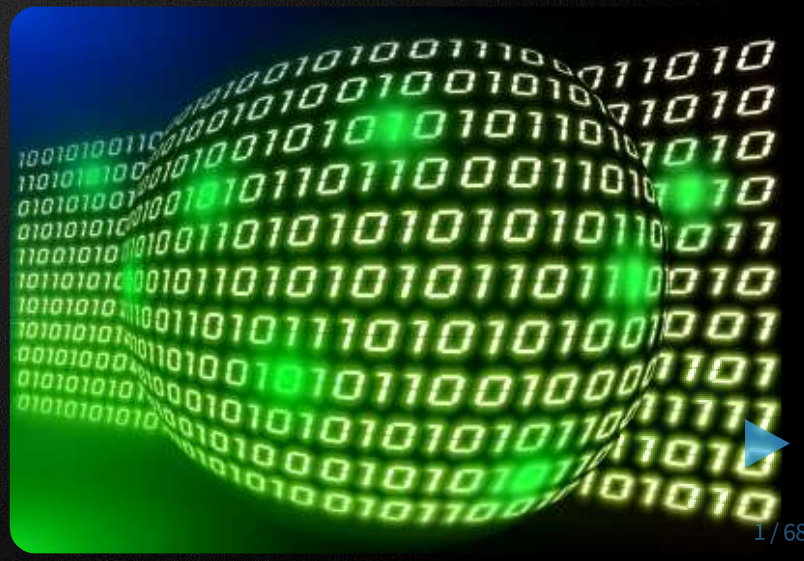
Telerik Academy

Primitive Data Types and Variables

Integer, Floating-point, Text Data, Characters, Variables, Literals



C# Fundamentals

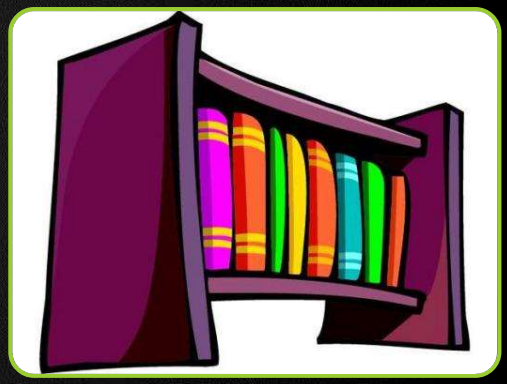
Telerik Software Academy

http://telerikacademy.com

Follow us

Telerik Academy

Table of Contents



Follow us

Telerik Academy

Primitive Data Types

Follow us

Telerik Academy

How Computing Works?

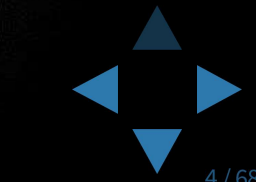
- Computers are machines that process data
  - Data is stored in the computer memory in variables
  - Variables have name, data type and value
- *Example* of variable definition and assignment in C#

```
int count = 5;
```

Variable name

Data type

Variable value

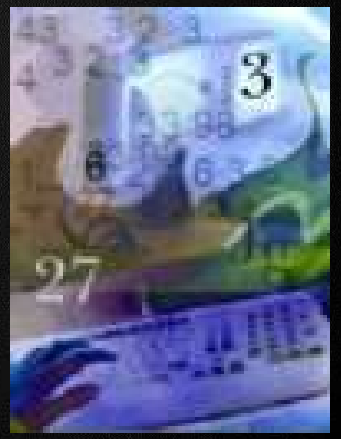


Follow us

Telerik Academy

What Is a Data Type?

- A data type:
  - Is a domain of values of similar characteristics
  - Defines the type of information stored in the computer memory (in a variable)
- *Examples*:
  - Positive integers: 1, 2, 3, ...
  - Alphabetical characters: a, b, c, ...
  - Days of week: `Monday`, `Tuesday`, ...

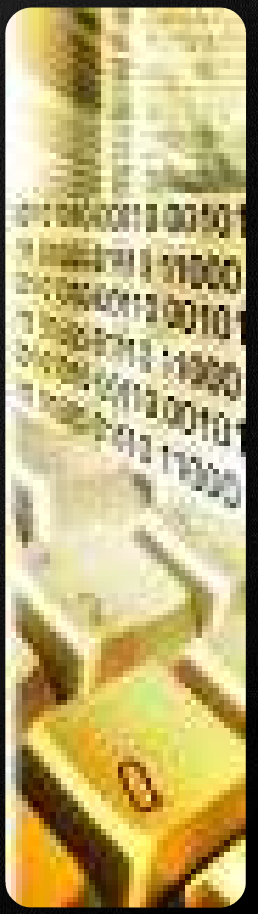


Follow us

Telerik Academy

Data Type Characteristics

- A data type has:
  - Name (C# keyword or .NET type)
  - Size (how much memory is used)
  - Default value
- *Example*:
  - Integer numbers in C#
  - Name: `int`
  - Size: 32 bits (4 bytes)
  - Default value: `0`



Follow us

Telerik Academy

Integer Types

Follow us

Telerik Academy

Integer Types

- Integer types:
  - Represent integer numbers
  - May be signed or unsigned
  - Have range of values, depending on the size of memory used
- The default value of integer types is:
  - `0` – for integer types
    - except `0L` – for the `long` type

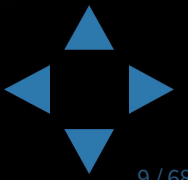
Follow us

Telerik Academy

Integer Types

- `sbyte` (signed 8-bit)
  - Min: `-128`
  - Max: `127`
- `byte` (unsigned 8-bit)
  - Min: `0`
  - Max: `255`
- `short` (signed 16-bit)
  - Min: `-32,768`
  - Max: `32,767`
- `ushort` (unsigned 16-bit)
  - Min: `0`
  - Max: `65,535`

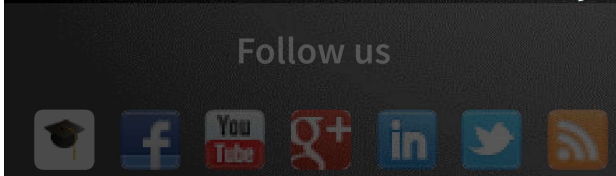
Follow us

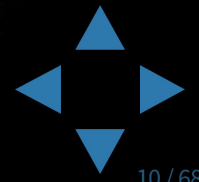


Telerik Academy

Integer Types (cont.)

- `int` (signed 32-bit)
  - Min: `-2,147,483,648`
  - Max: `2,147,483,647`
- `uint` (unsigned 32-bit)
  - Min:`0`
  - Max: `4,294,967,295`
- `long` (signed 64-bit)
  - Min: `-9,223,372,036,854,775,808`
  - Max: `9,223,372,036,854,775,807`
- `ulong` (unsigned 64-bit)
  - Min: `0`
  - Max: `18,446,744,073,709,551,615`



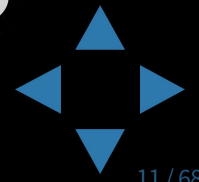


Telerik Academy

Integer Types: *Example*

- Depending on the unit of measure we may use different data types:
- *Example: Measuring times*

```
byte centuries = 20;    // Usually a small number
ushort years = 2000;
uint days = 730480;
ulong hours = 17531520; // May be a very big number
Console.WriteLine("{0} centuries is {1} years," +
                  " or {2} days, or {3} hours.",
                  centuries, years, days, hours);
```



Follow us

Telerik Academy

Integer Types

Demo

Follow us

Telerik Academy

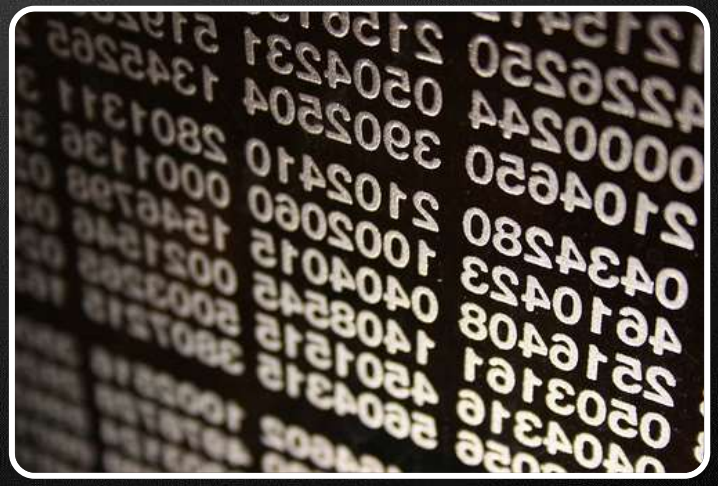
Floating-point and Decimal Floating-Point Types

Follow us

Telerik Academy

Floating-point Types

- Floating-point types:
  - Represent real numbers
  - May be signed or unsigned
  - Have range of values and different precision depending on the size of memory used
  - Can behave abnormally in the calculations



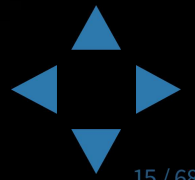
Follow us

Telerik Academy

Floating-Point Types

- Floating-point types are:
  - `float` (32-bits)
    - Min: $\pm 1.5 \times 10^{-45}$
    - Max: $\pm 3.4 \times 10^{38}$
    - Precision: 7 digits
  - `double` (64-bits)
    - Min: $\pm 5.0 \times 10^{-324}$
    - Max: $\pm 1.7 \times 10^{308}$
    - Precision: 15-16 digits
- The default value of floating-point types:
  - Is `0.0F` for the `float` type
  - Is `0.0D` for the `double` type

Follow us

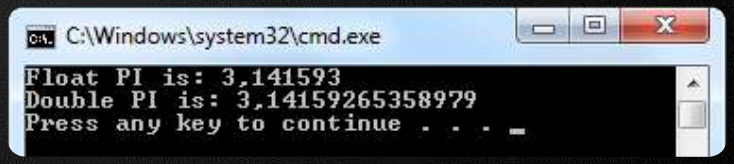


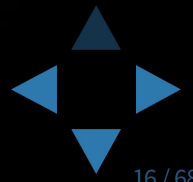
Telerik Academy

PI Precision – *Example*

- See below the difference in precision when using `float` and `double`:
- *Note*: The `f` suffix in the first statement!
  - Real numbers are by default interpreted as `double`!
  - One should explicitly convert them to `float`

```
float floatPI = 3.14159265358979238f;
double doublePI = 3.14159265358979238;
Console.WriteLine("Float PI is: {0}", floatPI);
Console.WriteLine("Double PI is: {0}", doublePI);
```





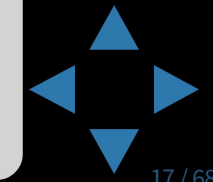
Follow us

Telerik Academy

Abnormalities in the Floating-Point Calculations

- Sometimes abnormalities can be observed when using floating-point numbers
  - Comparing floating-point numbers can not be performed directly with the == operator
- *Example*:

```
double a = 1.0f;
double b = 0.33f;
double sum = 1.33f;
bool equal = (a+b == sum); // False!!!
Console.WriteLine("a+b={0}  sum={1}  equal={2}",
                  a+b, sum, equal);
```

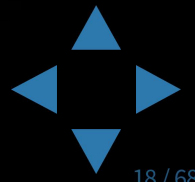


Follow us

Telerik Academy

Decimal Floating-Point Types

- There is a special decimal floating-point real number type in C#:
  - `decimal` (128-bits)
    - Min: $\pm 1,0 \times 10^{-28}$
    - Max: $\pm 7,9 \times 10^{28}$
    - Precision: 28-29 digits
  - Used for financial calculations
  - No round-off errors
  - Almost no loss of precision
- The default value of `decimal` type is:
  - `0.0M` (`M` is the suffix for decimal numbers)



Follow us

Telerik Academy

Floating-Point and Decimal Floating-Point Types

Demo

Follow us

Telerik Academy

Boolean Type

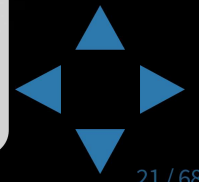
Follow us

Telerik Academy

The Boolean Data Type

- The Boolean data type:
  - Is declared by the `bool` keyword
  - Has two possible values: `true` and `false`
  - Is useful in logical expressions
- The default value is `false`
- *Example:* boolean variables with values `true` and `false`:

```
int a = 1;
int b = 2;
bool greaterAB = (a > b);
Console.WriteLine(greaterAB);  // False
bool equalA1 = (a == 1);
Console.WriteLine(equalA1);    // True
```



Follow us

Telerik Academy

Boolean Type

Demo

Follow us

Telerik Academy

Character Type

Follow us

Telerik Academy

The Character Data Type

- The character data type:
  - Represents symbolic information
  - Is declared by the `char` keyword
  - Gives each symbol a corresponding integer code
  - Has a `'\0'` default value
  - Takes 16 bits of memory (from `U+0000` to `U+FFFF`)

Follow us

Telerik Academy

Characters and Codes

- *Example*: Symbols have unique Unicode codes:

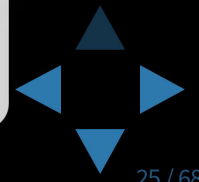
```
char symbol = 'a';
Console.WriteLine("The code of '{0}' is: {1}",
    symbol, (int) symbol);

symbol = 'b';
Console.WriteLine("The code of '{0}' is: {1}",
    symbol, (int) symbol);

symbol = 'A';
Console.WriteLine("The code of '{0}' is: {1}",
    symbol, (int) symbol);

symbol = '0';
Console.WriteLine("The code of '{0}' is: {1}",
    symbol, (int) symbol);
```

'a' has code value 97

'b' has code value 98

'A' has code value 65

'0' has code value 48



Follow us

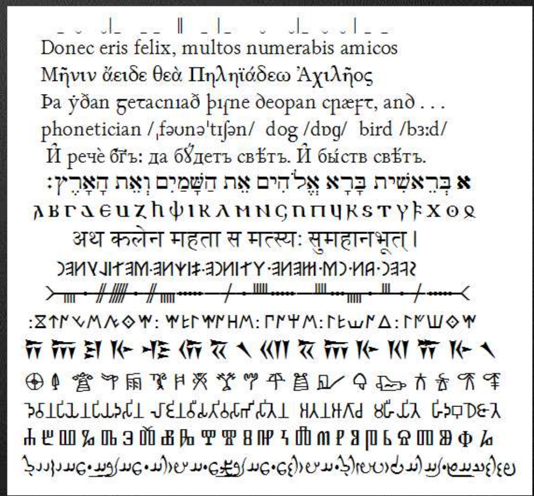
Telerik Academy

Character Type

Demo

Telerik Academy

String Type



Follow us

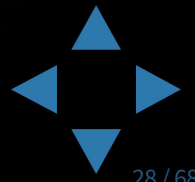
Telerik Academy

The String Data Type

- The string data type:
  - Represents a sequence of characters
  - Is declared by the `string` keyword
  - Has a default value `null` (no value)
- Strings are enclosed in quotes:

```
string s = "Microsoft .NET Framework";
```

- Strings can be concatenated
  - Using the + operator



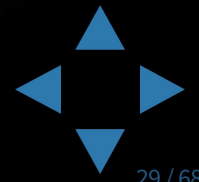
Follow us

Telerik Academy

String Concatenation: *Example*

- *Example:* Concatenating the two names of a person to obtain his full name:
  - *Note*: a space is missing between the two names! We have to add it manually

```
string firstName = "Ivan";
string lastName = "Ivanov";
Console.WriteLine("Hello, {0}!\n", firstName);

string fullName = firstName + " " + lastName;
Console.WriteLine("Your full name is {0}.",
  fullName);
```



Follow us

Telerik Academy

String Type

Demo

Follow us

Telerik Academy

Object Type

Follow us

Telerik Academy

The Object Type

- The object type:
  - Is declared by the `object` keyword
  - Is the base type of all other types
  - Can hold values of any type

Follow us

Telerik Academy

Using Objects

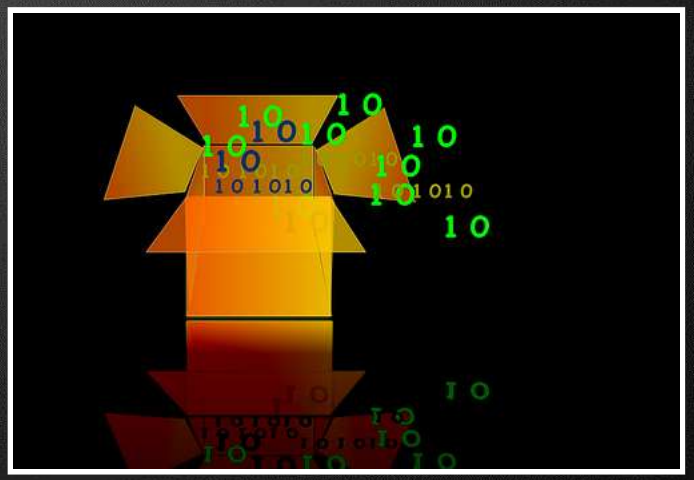
- *Example* of an object variable taking different types of data:

```
object dataContainer = 5;
Console.Write("The value of dataContainer is: ");
Console.WriteLine(dataContainer);

dataContainer = "Five";
Console.Write("The value of dataContainer is: ");
Console.WriteLine(dataContainer);
```

Follow us

Telerik Academy

Introducing Variables

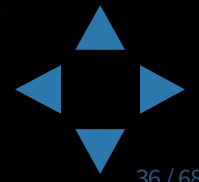


Follow us

Telerik Academy

What Is a Variable?

- A variable is a:
  - Placeholder of information that can usually be changed at run-time
- Variables allow you to:
  - Store information
  - Retrieve the stored information
  - Manipulate the stored information



Follow us

Telerik Academy

Variable Characteristics

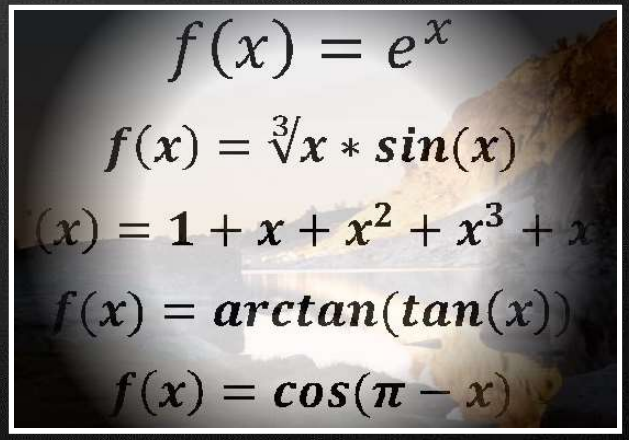
- A variable has:
  - Name
  - Type (of stored data)
  - Value
- *Example*:
  - Name: `counter`
  - Type: `int`
  - Value: `5`

```
int counter = 5;
```

Follow us

Telerik Academy

Declaring and Using Variables



Follow us

Telerik Academy

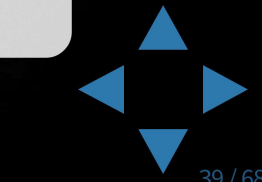
Declaring Variables

- When declaring a variable we:
  - Specify its type
  - Specify its name (called identifier)
  - May give it an initial value
- The syntax is the following:

```
<data_type> <identifier> [= <initialization>];
```

- *Example:*

```
int height = 200;
```

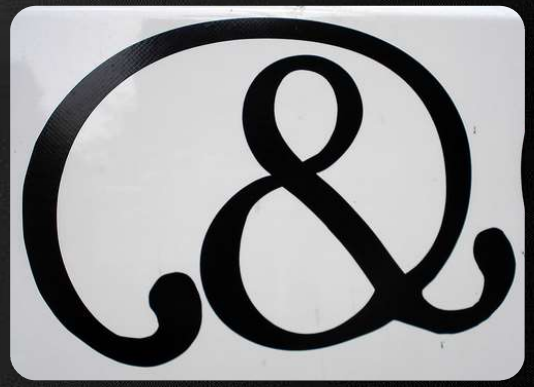


Follow us

Telerik Academy

Identifiers

- Identifiers may consist of:
  - Letters (Unicode)
  - Digits [0-9]
  - Underscore _
- Identifiers
  - Must begin with either a letter or an underscore
  - Cannot be a C# keyword

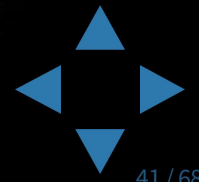


Follow us

Telerik Academy

Identifiers

- Identifiers
  - Should have a descriptive name
  - It is recommended to use only Latin letters
  - Should be neither too long nor too short
- *Note*:
  - In C# small letters are considered different than the capital letters (case sensitivity)



Follow us

Telerik Academy

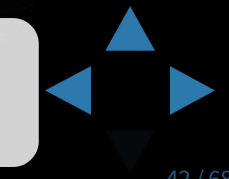
Identifiers – *Examples*

- *Examples* of correct identifiers:

```
int New = 2; // Here N is capital
int _2Pac; // This identifiers begins with _
string поздрав = "Hello"; // Unicode symbols used

// The following is more appropriate:
string greeting = "Hello";
int n = 100; // Undescriptive
int numberOfClients = 100; // Descriptive

// Overdescriptive identifier:
int numberOfPrivateClientOfTheFirm = 100;
```

- *Examples* of incorrect identifiers:

```
int new;  // new is a keyword
int 2Pac; // Cannot begin with a digit
```



Follow us

Telerik Academy

Assigning Values To Variables

Follow us

Telerik Academy

Assigning Values

- Assigning of values to variables
  - Is achieved by the = operator
- The = operator has
  - Variable identifier on the left
  - Value of the corresponding data type on the right
  - Could be used in a cascade calling, where assigning is done from right to left

```
int firstValue = 3;
int secondValue;

secondValue = firstValue;

firstValue = secondValue = 3; // Avoid this!
```

Follow us

Telerik Academy

Initializing Variables

- Initializing
  - Is assigning of initial value
  - Must be done before the variable is used!
  - By using the new keyword

```
int num = new int(); // num = 0
```

  - By using a literal expression

```
float heightInMeters = 1.74f;
```

  - By referring to an already initialized variable

```
string greeting = "Hello World!";
string message = greeting;
```

Follow us

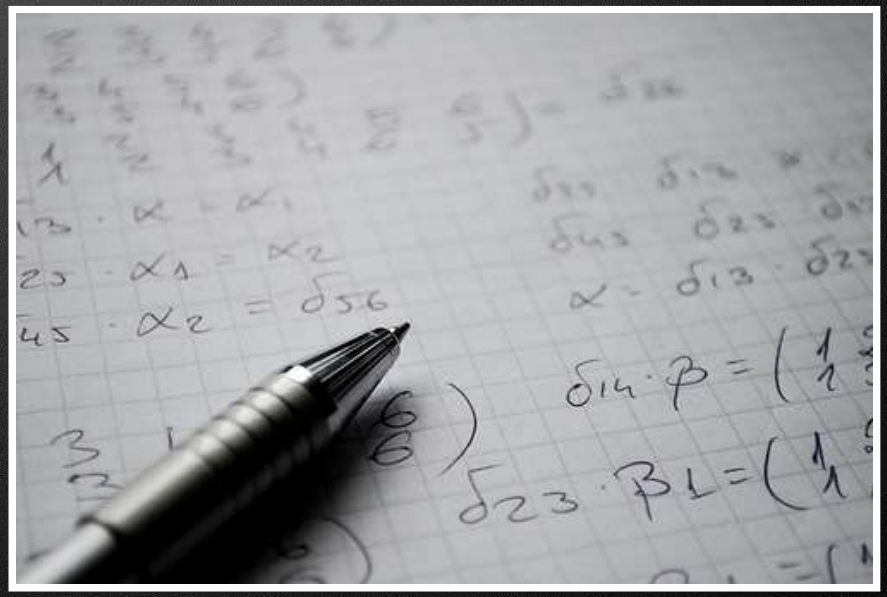
Telerik Academy

Assigning and Initializing Variables

Demo

Telerik Academy

Literals



Follow us

Telerik Academy

What are Literals?

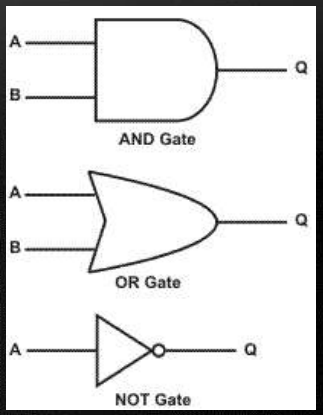
- Literals are:
  - Representations of values in the source code
- There are six types of literals
  - Boolean
  - Integer
  - Real
  - Character
  - String
  - The `null` literal

Follow us

Telerik Academy

Boolean and Integer Literals

- The boolean literals are:
  - `true`
  - `false`
- The integer literals:
  - Are used for variables of type `int`, `uint`, `long`, and `ulong`
  - Consist of digits
  - May have a sign (+, -)
  - May be in a hexadecimal format



Follow us

Telerik Academy

Integer Literals

- *Examples* of integer literals
  - The `'0x'` and `'0X'` prefixes mean a hexadecimal value

```
0xA8F1
```

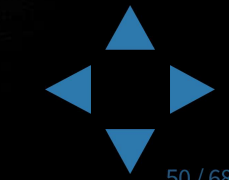
  - The `'u'` and `'U'` suffixes mean a `ulong` or `uint` type

```
12345678U
```

  - The `'l'` and `'L'` suffixes mean a `long` or `ulong` type

```
9876543L
```

Follow us

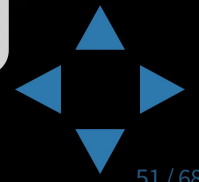


Telerik Academy

Integer Literals – *Example*

- *Note*: the letter `'l'` is easily confused with the digit 1 so it's better to use `'L'`!!!

```
// The following variables are
// initialized with the same value:
int numberInHex = -0x10;
int numberInDec = -16;

// The following causes an error,
//because 234u is of type uint
int unsignedInt = 234u;

// The following causes an error,
//because 234L is of type long
int longInt = 234L;
```

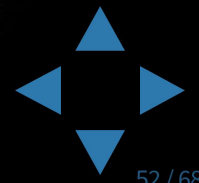


Follow us

Telerik Academy

Real Literals

- The real literals:
  - Are used for values of type `float`, `double` and `decimal`
  - May consist of digits, a sign and `.`
  - May be in exponential notation: `6.02e+23`
- The `f` and `F` suffixes mean `float`
- The `d` and `D` suffixes mean `double`
- The `m` and `M` suffixes mean `decimal`
- The default interpretation is `double`



Follow us

Telerik Academy

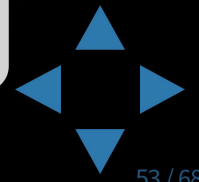
Real Literals – *Example*

- *Example* of incorrect `float` literal:

```
// The following causes an error
// because 12.5 is double by default
float realNumber = 12.5;
```

- A correct way to assign floating-point value (using also the exponential format):

```
// The following is the correct
// way of assigning the value:
float realNumber = 12.5f;
// This is the same value in exponential format:
realNumber = 1.25e+7f;
```



Follow us

Telerik Academy

Character Literals

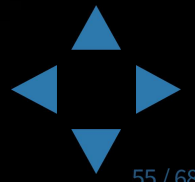
- The character literals:
  - Are used for values of the `char` type
  - Consist of two single quotes surrounding the character value: `'<value>'`
- The value may be:
  - Symbol
  - The code of the symbol
  - Escaping sequence

Follow us

Telerik Academy

Escaping Sequences

- Escaping sequences are:
  - Means of presenting a symbol that is usually interpreted otherwise (like ')
  - Means of presenting system symbols (like the new line symbol)
- Common escaping sequences are:
  - \' for single quote \" for double quote
  - \\ for backslash \n for new line
  - \uXXXX for denoting any other Unicode symbol



Follow us

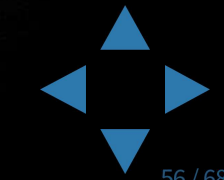
Telerik Academy

Character Literals – *Example*

- *Examples* of different character literals:

```
char symbol = 'a'; // An ordinary symbol

// Unicode symbol in hexadecimal format (letter 'o')
symbol = '\u006F';

symbol = '\u8449'; // 葉 (Leaf in Traditional Chinese)
symbol = '\''; // single quote symbol
symbol = '\\'; // backslash symbol
symbol = '\n'; // new line symbol
symbol = '\t'; // TAB symbol
symbol = "a"; // Incorrect: use single quotes
```

Follow us

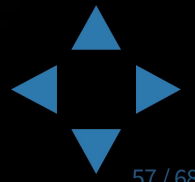


Telerik Academy

String Literals

- String literals:
  - Are used for values of the string type
  - Consist of two double quotes surrounding the value: `"<value>"`
  - May have a @ prefix which ignores the used escaping sequences: `@"<value>"`
- The value is a sequence of character literals

```
string s = "I am a sting literal";
```



Follow us

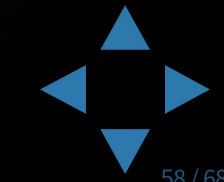
Telerik Academy

String Literals – *Example*

- Benefits of quoted strings (the @ prefix):
- In quoted strings \" is used instead of ""!

```
// Here is a string literal using escape sequences
string quotation = "\"Hello, Jude\", he said.";
string path = "C:\\WINNT\\Darts\\Darts.exe";

// Here is an example of the usage of @
quotation = @"""Hello, Jimmy!"", she answered.";
path = @"C:\WINNT\Darts\Darts.exe";

string str = @"some
        text";
```

Follow us



Telerik Academy

String Literals

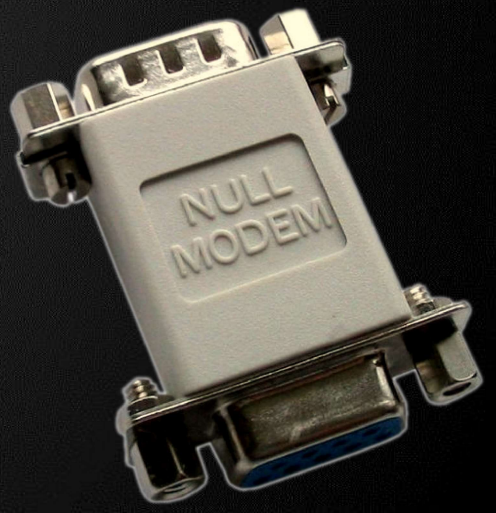
Demo

Follow us

Telerik Academy

Nullable Types

NULL

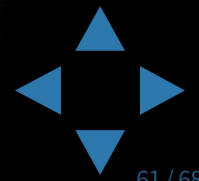


Follow us

Telerik Academy

Nullable Types

- Nullable types are instances of the `System.Nullable` struct
  - Wrapper over the primitive types
  - E.g. `int?`, `double?`, etc.
- Nullabe type can represent the normal range of values for its underlying value type, plus an additional `null` value
- Useful when dealing with Databases or other structures that have default value `null`



Follow us

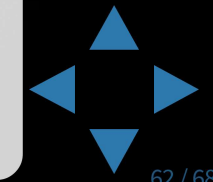
Telerik Academy

Nullable Types – *Example*

- *Example* with Integer:

```
int? someInteger = null;
Console.WriteLine(
  "This is the integer with Null value -> "
    + someInteger);
someInteger = 5;
Console.WriteLine(
  "This is the integer with value 5 -> "
    +  someInteger);
```

```
double? someDouble = null;
Console.WriteLine(
  "This is the real number with Null value -> "
  + someDouble);
someDouble = 2.5;
Console.WriteLine(
  "This is the real number with value 5 -> "
    + someDouble);
```



Follow us

Telerik Academy

Nullable Types

Demo

```
int?
```

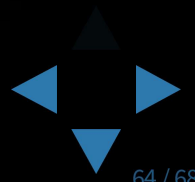
```
double?
```

Follow us

Telerik Academy

Dynamic Types in C#

Types Holding Anything & Evaluated at Runtime

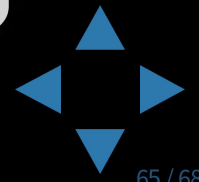


Telerik Academy

Dynamic Types

- Dynamic types in C# (keyword `dynamic`)
  - Can hold anything (string, number, object, function / method reference)
  - Operations evaluated at runtime
  - Behave like types in JavaScript / PHP

```
dynamic a = 5;
dynamic b = 3;
Console.WriteLine(a + b); // 8 (sum of integers)
a = "5";
b = 3;
Console.WriteLine(a + b); // 53 (string concatenation)
```



Follow us

Telerik Academy

Dynamic Types

Demo

Follow us

Telerik Academy

Primitive Data Types and Variables

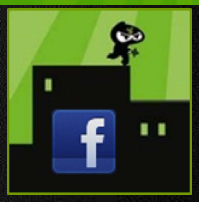
Questions?

Follow us

Telerik Academy

Free Trainings @ Telerik Academy

- Fundamentals of C# ProgrammingTrack of Courses
  - csharpfundamentals.telerik.com
    - Telerik Software Academy
  - telerikacademy.com
    - Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy
    - Telerik Academy Learning System
  - telerikacademy.com



Follow us