How to work
with numeric
and string data

Arithmetic operators

Operator	Name	Description
+	Addition	Adds two operands.
-	Subtraction	Subtracts the right operand from the left operand.
*	Multiplication	Multiplies the right operand and the left operand.
/	Division	Divides the right operand into the left operand. If both operands are integers, the result is an integer.
%	Modulus	Returns the value that is left over after dividing the right operand into the left operand.

Arithmetic operators (cont.)

Operator	Name	Description
+	Positive sign	Returns the value of the operand.
-	Negative sign	Changes a positive value to negative, and vice versa.
++	Increment	Adds 1 to the operand (x = x + 1).
- -	Decrement	Subtracts 1 from the operand (x = x - 1).

Arithmetic expressions that use integers

```
int x = 14;
int y = 8;
int result1 = x + y;        // result1 = 22
int result2 = x - y;        // result2 = 6
int result3 = x * y;        // result3 = 112
int result4 = x / y;        // result4 = 1
int result5 = x % y;        // result5 = 6
int result6 = -y + x;       // result6 = 8
int result7 = --y;          // result7 = 7, y = 7
int result8 = ++x;          // result8 = 15, x = 15
```

Arithmetic expressions that use decimals

```
decimal a = 8.5m;
decimal b = 3.4m;
decimal result11 = a + b;     // result11 = 11.9
decimal result12 = a - b;     // result12 = 5.1
decimal result13 = a / b;     // result13 = 2.5
decimal result14 = a * b;     // result14 = 28.90
decimal result15 = a % b;     // result15 = 1.7
decimal result16 = -a;        // result16 = -8.5
decimal result17 = --a;       // result17 = 7.5, a = 7.5
decimal result18 = ++b;       // result18 = 4.4, b = 4.4
```

Assignment operators

Operator	Name
=	Assignment
+=	Addition
-=	Subtraction
*=	Multiplication
/=	Division
%=	Modulus

Statements that use the same variable on both sides of the equals sign

```
total = total + 100m;
total = total – 100m;
price = price * .8m;
```

Statements that use the shortcut assignment operators

```
total += 100m;
total -= 100m;
price *= .8m;
```

The order of precedence for arithmetic operations

1. Increment and decrement
2. Positive and negative
3. Multiplication, division, and modulus
4. Addition and subtraction

How implicit casting works

Casting from less precise to more precise data types

byte→short→int→long→decimal

int→double

short→float→double

char→int

Examples

```
double grade = 93;                 // convert int to double

int letter = 'A';                  // convert char to int

double a = 95.0;
int b = 86, c = 91;
double average = (a+b+c)/3;        // convert b and c to doubles
                                   // (average = 90.666666...)
```

How to code an explicit cast

The syntax for coding an explicit cast

```
(type) expression
```

Examples

```
int grade = (int)93.75;          // convert double to int
                                 // (grade = 93)

char letter = (char)65;          // convert int to char
                                 // (letter = 'A')

double a = 95.0;
int b = 86, c = 91;
int average = ((int)a+b+c)/3;    // convert a to int value
                                 // (average = 90)

decimal result = (decimal)b/(decimal)c; // result has decimal
                                        // places
```

Five static methods of the Math class

The syntax of the Round method

```
Math.Round(decimalNumber[, precision[, mode]])
```

The syntax of the Pow method

```
Math.Pow(number, power)
```

The syntax of the Sqrt method

```
Math.Sqrt(number)
```

The syntax of the Min and Max methods

```
Math.{Min|Max}(number1, number2)
```

Statements that use static methods of the Math class

```
int shipWeight = Math.Round(shipWeightDouble);
    // round to a whole number

double orderTotal = Math.Round(orderTotal, 2);
    // round to 2 decimal places

double area = Math.Pow(radius, 2) * Math.PI;
    // area of circle

double sqrtX = Math.Sqrt(x);

double maxSales = Math.Max(lastYearSales, thisYearSales);

int minQty = Math.Min(lastYearQty, thisYearQty);
```

Results from static methods of the Math class

Statement	Result
`Math.Round(23.75, 1)`	`23.8`
`Math.Round(23.85, 1)`	`23.8`
`Math.Round(23.744, 2)`	`23.74`
`Math.Round(23.745, 2)`	`23.74`
`Math.Round(23.745, 2, MidpointRounding.AwayFromZero)`	`23.75`
`Math.Pow(5, 2)`	`25`
`Math.Sqrt(20.25)`	`4.5`
`Math.Max(23.75, 20.25)`	`23.75`
`Math.Min(23.75, 20.25)`	`20.25`

Common methods for data conversion

Method	Description
`ToString([format])`	A method that converts the value to its equivalent string representation using the specified format. If the format is omitted, the value isn't formatted.
`Parse(string)`	A static method that converts the specified string to an equivalent data value. If the string can't be converted, an exception occurs.
`TryParse(string, result)`	A static method that converts the specified string to an equivalent data value and stores it in the result variable. Returns a true value if the string is converted. Otherwise, returns a false value.

Some of the static methods of the Convert class

Method	Description
`ToDecimal(value)`	Converts the value to the decimal data type.
`ToDouble(value)`	Converts the value to the double data type.
`ToInt32(value)`	Converts the value to the int data type.
`ToChar(value)`	Converts the value to the char data type.
`ToBool(value)`	Converts the value to the bool data type.
`ToString(value)`	Converts the value to a string object.

Statements that use ToString, Parse, and TryParse

```
decimal sales = 2574.98m;
string salesString = sales.ToString();      // decimal to string
sales = Decimal.Parse(salesString);         // string to decimal
Decimal.TryParse(salesString, out sales);   // string to decimal
```

An implicit call of the ToString method

```
double price = 49.50;
string priceString = "Price: $" + price;    // automatic ToString
call
```

A TryParse method that handles invalid data

```
string salesString = "$2574.98";
decimal sales = 0m;
Decimal.TryParse(salesString, out sales);   // sales is 0
```

Conversion statements that use the Convert class

```
decimal subtotal = Convert.ToDecimal(txtSubtotal.Text);
    // string to decimal
int years = Convert.ToInt32(txtYears.Text);
    // string to int
txtSubtotal.Text = Convert.ToString(subtotal);
    // decimal to string
int subtotalInt = Convert.ToInt32(subtotal);
    // decimal to int
```

Standard numeric formatting codes

Code	Format
C or **c**	Currency
P or **p**	Percent
N or **n**	Number
F or **f**	Float
D or **d**	Digits
E or **e**	Exponential
G or **g**	General

How to use the ToString method to format a number

Statement	Example
`string monthlyAmount = amount.ToString("c");`	$1,547.20
`string interestRate = interest.ToString("p1");`	2.3%
`string quantityString = quantity.ToString("n0");`	15,000
`string paymentString = payment.ToString("f3");`	432.818

How to work with scope

- The *scope* of a variable determines what code has access to it. If you try to refer to a variable outside of its scope, it will cause a build error.
- The scope of a variable is determined by where you declare it. If you declare a variable within a method, it has *method scope*. If you declare a variable within a class but not within a method, it has *class scope*.
- A variable with method scope can only be referred to by statements within that method. A variable with class scope can be referred to by all of the methods in the class.
- The *lifetime* of a variable is the period of time that it's available for use. A variable with method scope is only available while the method is executing. A variable with class scope is available while the class is instantiated.

The syntax for declaring an enumeration

```
enum EnumerationName [: type]
{
    ConstantName1 [= value][,
    ConstantName2 [= value]]...
}
```

An enumeration that sets the constant values to 0, 1, and 2

```
enum Terms
{
    Net30Days,
    Net60Days,
    Net90Days
}
```

An enumeration that sets the constant values to 30, 60, and 90

```
enum TermValues : short
{
    Net30Days = 30,
    Net60Days = 60,
    Net90Days = 90
}
```

Statements that use these enumerations

```
Terms t = Terms.Net30Days;
int i = (int) Terms.Net30Days;             // i is 0
int i = (int) TermValues.Net60Days;        // i is 60
string s = Terms.Net30Days.ToString();     // s is "Net30Days"
```

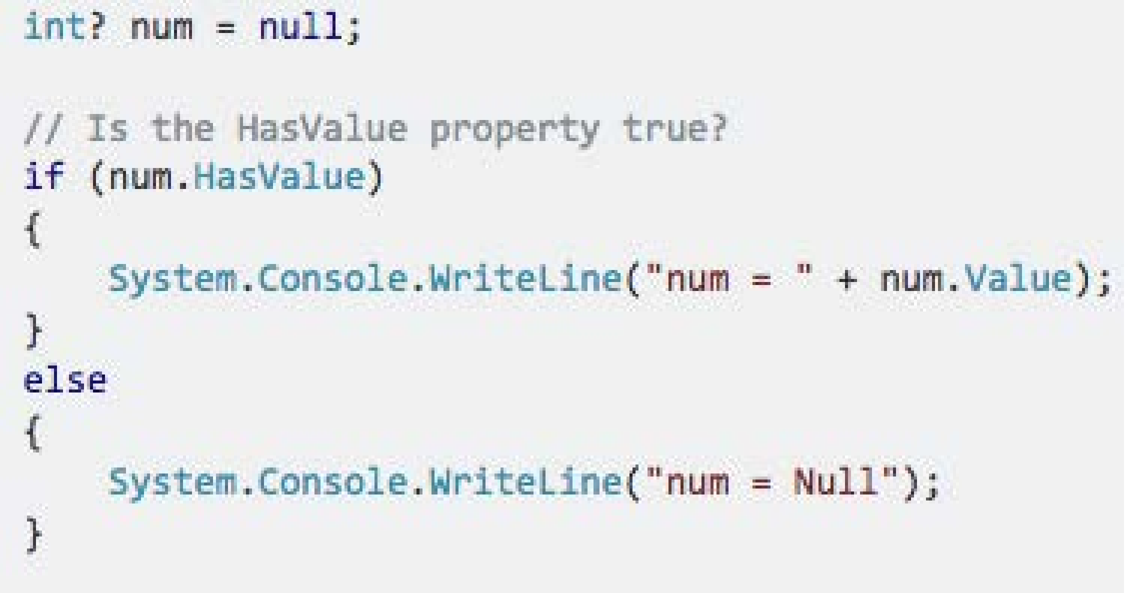
How to declare a value type that can contain null values

```
int? quantity;
quantity = null;
quantity = 0;
quantity = 20;

decimal? salesTotal = null;

bool? isValid = null;

Terms? paymentTerm = null;

// string? message = null;  // not necessary or allowed
```

Two properties for working with nullable types

Property	Description
`HasValue`	Returns a true value if the nullable type contains a value. Returns a false value if the nullable type is null.
`Value`	Returns the value of the nullable type.

working with nullable types



```
int? num = null;

// Is the HasValue property true?
if (num.HasValue)
{
    System.Console.WriteLine("num = " + num.Value);
}
else
{
    System.Console.WriteLine("num = Null");
}
```

Exercises

Write programs to:
Accept the length and width of a rectangle the program should display the area and perimeter
Example input 25 length and 100 width – area 2500 and perimeter 250

Accept a number of scores (whole numbers between 1 – 100) , the user enters -1 to end the input of the scores – the program should then display the number of scores entered, the average score, the min and the max scores.