VARIABLES

A variable is a name for a value stored in memory. Variables are used in programs so that values can be represented with meaningful names and because they make code easier to read, understand and modify.

- You can think of a variable as a storage box with a name for holding certain types of things:

- **Declaration Statement** – A variable must be declared before it is used. A declaration statement must include the *data type*, which determines the type of data and size (amount of memory) the variable will store and the variable name, called the *identifier*. A declaration statement takes the form:

  DataType variablename ;

```
int age;
string firstName;
double length;
decimal salePrice;
```

- You can declare multiple variables of the same type in a single statement by using the type once and separating the variable declarations with a comma.

```
int myAge, yourAge;
double mySalary, yourSalary;
```

VARIABLE NAMES

- You must **name** every variable you use in your project. Rules for naming variables are:
  - No more than 40 characters.
  - Can only use letters, numbers, and the underscore (_) character.
  - The first character must be a letter or underscore.
  - Typically variable identifiers begin with a lowercase letter. Any word after the first should begin with an uppercase letter. (example, finalCost)
  - Variables cannot contain spaces or symbols (accept underscore)
  - You cannot use a word reserved by C# Express (example, you can't have a variable named Form or one named Beep).
  - Capitals matter, age is a different variable than Age.
- ***The most important rule is to use variable names that are meaningful. You should be able to identify the information stored in a variable by looking at its name.***

ASSIGNMENT STATEMENT

- A variable assignment statement is formed with the variable name on the left side of an equal sign and the value it is to receive on the right side of the equal sign.
- The equal sign (=) operator indicates that the variable on the left is to receive the value on the right.
- The value on the right can be a literal, which is any actual value. It could also be another variable or an expression.
- In the code segment below, area was assigned the value of an expression (side * side).
- An assignment statement can be part of a variable declaration. In addition to being declared, the variable is initialized. For example, in the code segment below, variable side was assigned a value when declared.
- ***It is important to realize that a variable can store only one value at any one time.***

```
int side =3;    //side of square
int area;

area = side * side; //calculated area of square
area = 100;
```

- The value of area is 100 because this was the last value assigned to area.
- NOTE: When you type in any number with a decimal point, by default the compiler will treat it as a double data type. You must add the 'F' suffix to make it a float or the 'M' suffix to make it a decimal!

```
decimal salePrice = 15.99M;   //use the "M" suffix
```

DATA TYPES

- When choosing a data type, it is important to choose the most appropriate type for the quantity being represented. If a value could possibly have a decimal portion, then Double is the best choice. If a variable will represent only whole numbers, then Integer is the best choice even though Decimal will work. Using the most appropriate data types for variables has two benefits. First, both the compiler and the reader will understand the possible values for a variable. Second, the compiler allocates the appropriate memory for the variable.

Quick List of Data Types:

Type	Size	Range
int	(4bytes)	integers from -2,147,483,648 to 2,147,483,647
float	(4bytes)	a 7 digit number that may contain a decimal
double	(8 bytes)	a 16 digit number that may contain a decimal
decimal	(16 bytes)	large numbers possibly containing 28-29 digits
date	(8 bytes)	a date in m/d/yyyy format
char	(2 bytes)	a single character
string	NA	a set of characters
boolean	(1 byte)	True or False
object	NA	any object

CONSTANTS

- A constant is a name for a memory location that stores a value that cannot be changed from its initial assignment. Constants, like variables, are used in a program so that values can be represented with meaningful names. Constants must be assigned their values during the declaration stage.

```
const double PIE = 3.14159;
```

- Constant identifiers are typically all uppercase with an underscore (_) separating words within the identifier name.

VARIABLE CASTING

- Sometimes you need to convert variables of one data type to another. This happens with numeric data because you may have an integer that needs to be used as a real number (double or decimal) or vice-versa. The compiler will let you assign data from one variable to a variable of another data type if the assignment is "safe", meaning there is no chance you will lose or truncate data while making the conversion. For example, it's safe to convert an integer to a double because the double can always exactly represent an **int** like 42 as the **double** 42.0.
- The reverse isn't true (**double** to **int**) as the compiler will be worried that the integer data type can't hold the fractional part of the double, so will respond with an error.
- You can override the compiler's natural caution and force the conversion by casting. Casting means you explicitly tell the compiler to convert the source data to the target data type, doing whatever necessary to make the data fit. To cast, use the target data type in parentheses right before the source data, and the compiler will force it to the target data type.

```
double item = 42.3;     //start with double
int total = (int)side;  //cast it to an integer
```

  - As a general rule, only cast when you are sure the conversion is safe.

VARIABLE CONVERSION

- There are times we need to convert one data type to another. C# provides functions to do so.

```
Convert.ToBoolean(Expression)
//Converts a numerical Expression to a bool data type (if Expression is
nonzero, the result is true, otherwise the result is false)

Convert.ToChar(Expression)
//Converts any valid single character string to a char data type

Convert.DateTime(Expression)
//Converts any valid date or time Expression to a DateTime data type

Convert.ToSingle(Expression)
//Converts an Expression to a float data type

Convert.ToDouble(Expression)
//Converts an Expression to a double data type

Convert.ToInt32(Expression)
//Converts an Expression to an int datatype (any fractional part is rounded)
```