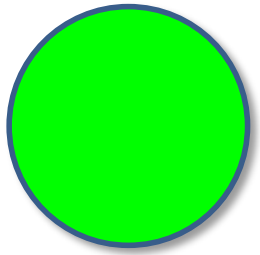
INTRODUCTION TO C#

Lesson 1.02 – C# Primitive Data Types

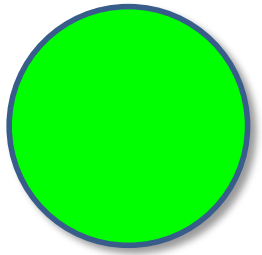


Data Types

Variables are a way for you to store information needed by your program at run-time. For example, a username could be stored in a variable, or a game's current score or length of time played. C# does not persist this information beyond the execution time of your code.

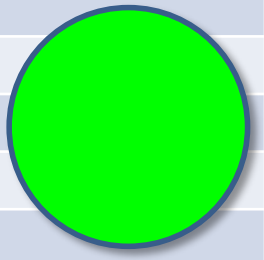
Each variable is responsible for holding a single piece of information of a particular data type. Variables are defined using the data type followed by the name you want assigned to your variable. That name is then used throughout the rest of your code to refer to that value.

Variables do have different lifespans or scope, depending on where and how they are created and used.



Data Types

Type	Range
byte	0 .. 255
sbyte	-128 .. 127
short	-32,768 .. 32,767
ushort	0 .. 65,535
int	-2,147,483,648 .. 2,147,483,647
uint	0 .. 4,294,967,295
long	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
ulong	0 .. 18,446,744,073,709,551,615
float	-3.402823e38 .. 3.402823e38
double	-1.79769313486232e308 .. 1.79769313486232e308
decimal	-79228162514264337593543950335 .. 79228162514264337593543950335
char	A Unicode character.
string	A string of Unicode characters.
bool	True or False.
object	An object.



Declaring Variables

Variables can be created anywhere in your class. However, keep in mind that the location of where the variable is created also determines which parts of your code can work with the value being stored. We'll discuss this in more detail when we talk about variable scope.

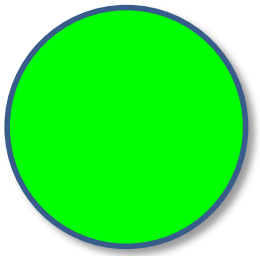
To declare a variable, use the data type followed by the name you want to give your variable. After the declaration, you can optionally give that variable a value as well. It is always a good idea to provide an initial value for the variables you are working with before using them. Note the single quotes around the char variable, and the double-quotes around the string variable.

Variables names cannot start with a number and should be meaningful to their purpose.

```
bool result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
string sentence = "This is a string of text";

int iNumber;
iNumber = 100;
```

```
var result = true;
var capitalC = 'C';
var b = 100;
var s = 10000;
var i = 100000;
var sentence = "This is a string of text";
```



Casting & Converting

Variables can be converted from one data type to another. There are two types of conversions supported by C#:

Implicit – Implicit conversions perform a direct conversion from one data type to another but follow a set of rules. These rules are in place to prevent any data loss during the conversion.

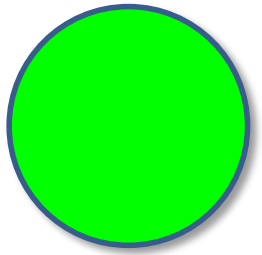
Example:

```
int a = 100;
long b = a;  //a is cast as a long
```

Explicit – Explicit conversion requires you to specify the target data type before a conversion can take place. This may produce an error, if the values are not valid for the specified type.

Example:

```
long a = 100;
short b = (short)a;  //a is cast as an int
```



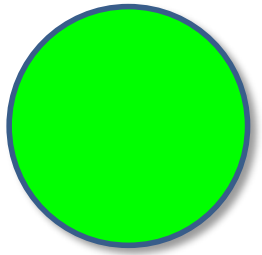
Casting Rules

The basic rules are:

Implicit: char → int → long → float → double

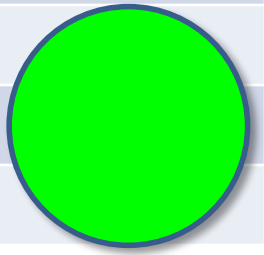
Explicit: double → float → long → int → char

Rule of thumb: a smaller variable is likely to implicitly convert into a larger variable (int to long, for example). Whereas a larger value needs to be explicitly converted down to a smaller variable type (double to float).



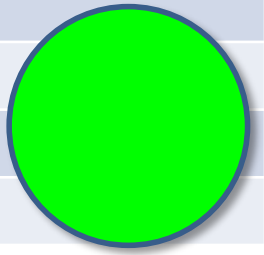
Casting Rules - Implicit

Source Type	Can Be Converted To
Byte	short, ushort, int, uint, long, ulong, float, double, or decimal
Sbyte	short, int, long, float, double, or decimal
Int	long, float, double, or decimal
Uint	long, ulong, float, double, or decimal
Short	int, long, float, double, or decimal
Ushort	int, uint, long, ulong, float, double, or decimal
Long	float, double, or decimal
Ulong	float, double, or decimal
Float	double
Char	ushort, int, uint, long, ulong, float, double, or decimal



Casting Rules - Explicit

Source Type	Can Be Converted To
Byte	sbyte or char
Sbyte	byte, ushort, uint, ulong, or char
Int	sbyte, byte, short, ushort, uint, ulong, or char
Uint	sbyte, byte, short, ushort, int, or char
Short	sbyte, byte, ushort, uint, ulong, or char
Ushort	sbyte, byte, short, or char
Long	sbyte, byte, short, ushort, int, uint, ulong, or char
Ulong	sbyte, byte, short, ushort, int, uint, long, or char
Float	sbyte, byte, short, ushort, int, uint, long, ulong, char, ordecimal
Double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, or decimal
Char	sbyte, byte, or short
Decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, or double



Reference Type vs Value Type

Value Type – A data type such as an int, bool, float that stores its value directly. In memory of the PC, a specific allocation is provided for the value being stored in the value type variable. When we work with a value type variable, a copy of it is made when we make assignments.

Reference Type – A reference type variable, such as a class, string, array, and delegate, refer to their value through a reference. In memory, an allocation is made to associate the variable to a memory location. That memory location then points to another memory location that contains the actual value. When we work with this type of variable, we're editing the same object, regardless of how we pass it to our methods.

You can convert between these types using C#'s Boxing and Unboxing feature. The process of converting a Value Type to a Reference Type is called Boxing and casting a Reference Type to a Value Type is called Unboxing.