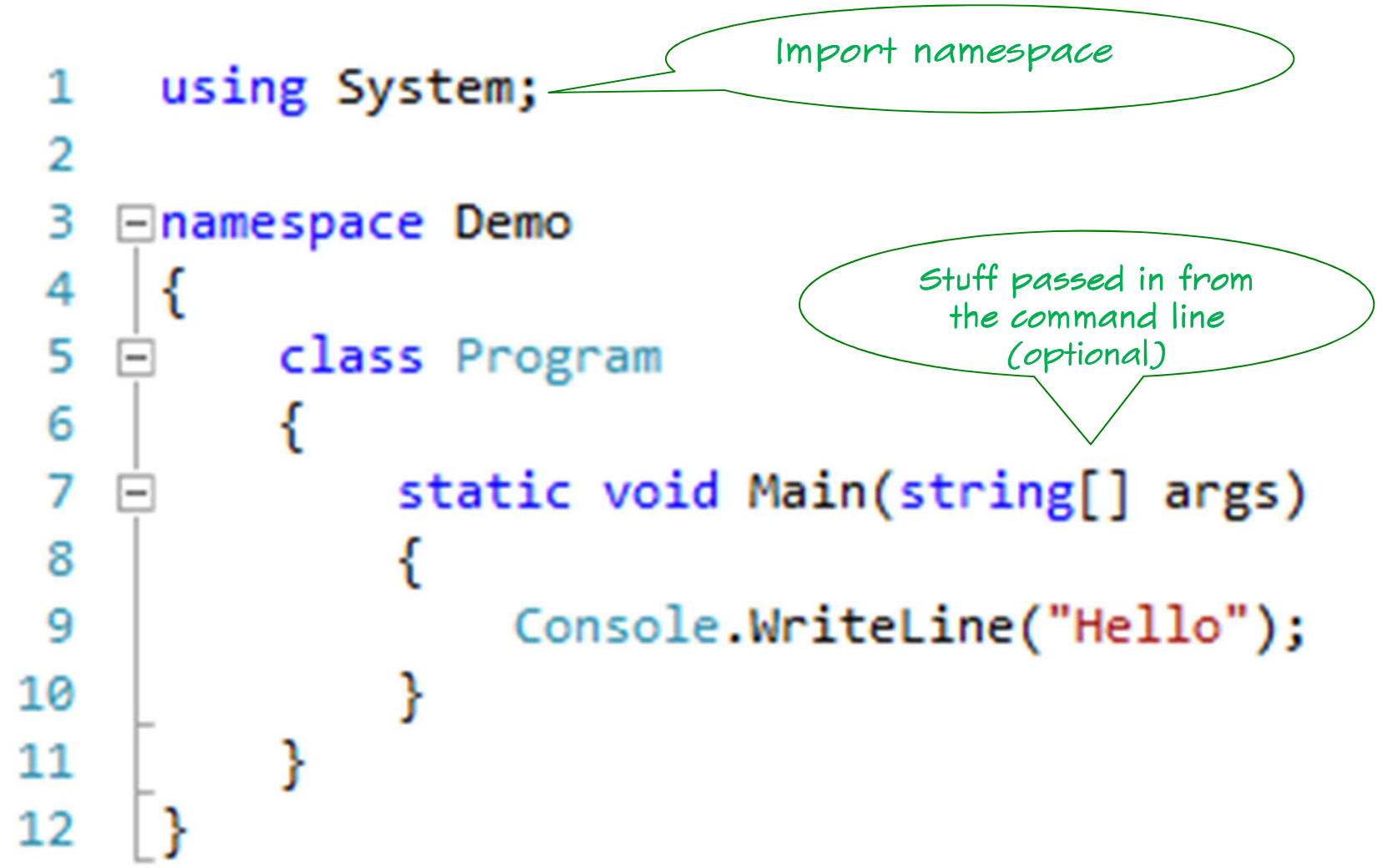
Session 2

C# Language Fundamentals

Structure of a C# Console Application



C# Syntax

- **case sensitive**
  `OBJ is not the same as obj`
- **All statements end with a semicolon;**
- **Curly braces denote blocks of statements**

```
{
    statement;
    statement;
}
```

- **Whitespace is ignored**
- **Comments are ignored:**
  - // single line comment
  - /*
    multi-line comment
    */

New Programmers

- **Remember Algebra?**

$$x = 3x + 4$$

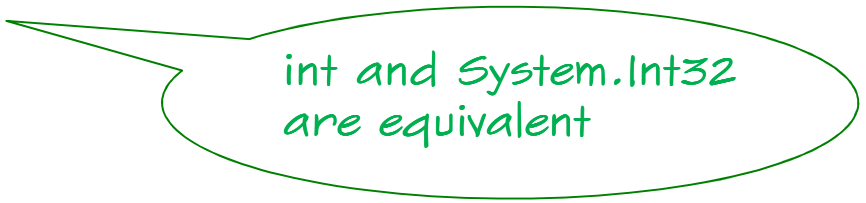
-- or --

$$F(x) = 3x + 4$$

- **x is a variable**
- **x = 3x+4 is an expression**
- **F() is a function with 1 parameter x**

Using Variables

- **Variables must have a type**
  ```
  int x = 30;
  string city = “New York”;
  ```
- **Variables must be declared before used**
- **In C#, variables must be assigned before use (rule of definite assignment)**
- **Types are provided by the .NET Framework**
  `System.Int32`
- **Some .NET types are aliased by a .NET language**
  `using int = System.Int32`
- **You can define your own types**
  - `Class`
  - `Enum`
  - `Struct`



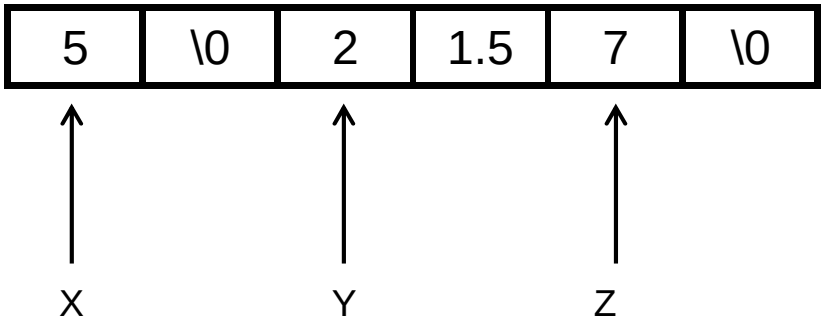
C# Types

Predefined type	Definition	# Bytes
byte	Integer between 0 and 255	1
sbyte	Integer between -128 and 127	1
short	Integer between -32,768 and 32,767	2
ushort	Integer between 0 and 65535	2
int	Integer between -2147483648 and 2147483647	4
uint	Integer between 0 and 4294967295	4
long	Integer between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8
ulong	Integer between 0 and 18,446,744,073,709,551,615	8
bool	Boolean value: true or false	1
float	Single-precision floating point value (non-whole number)	4
double	Double-precision floating point value	8
decimal	Precise decimal value to 28 significant digits	12
object	Base type of all other types	N/A
char	Single Unicode character between 0 and 65,535	2
string	An unlimited sequence of Unicode characters	N/A
Array	A data structure that contains several variables of the same type	depends

What is a variable

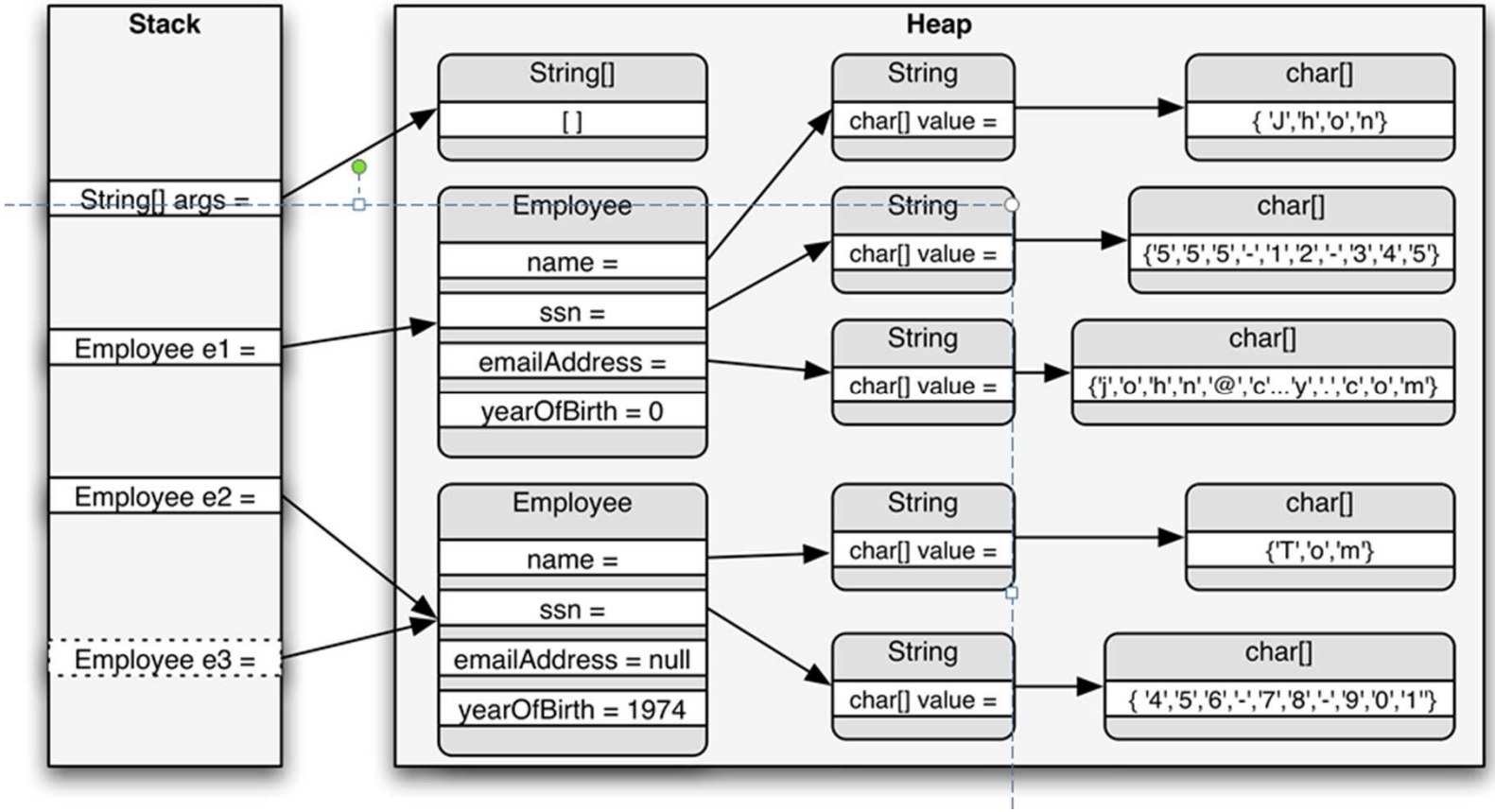
- **Named storage location in memory**
- **Variables must be declared as a certain type**
- **The type tells the runtime how much memory to claim**
- **The variable's type cannot change once declared**

```
int x = 5;
Int32 y = 2;
int z = x+y;
```



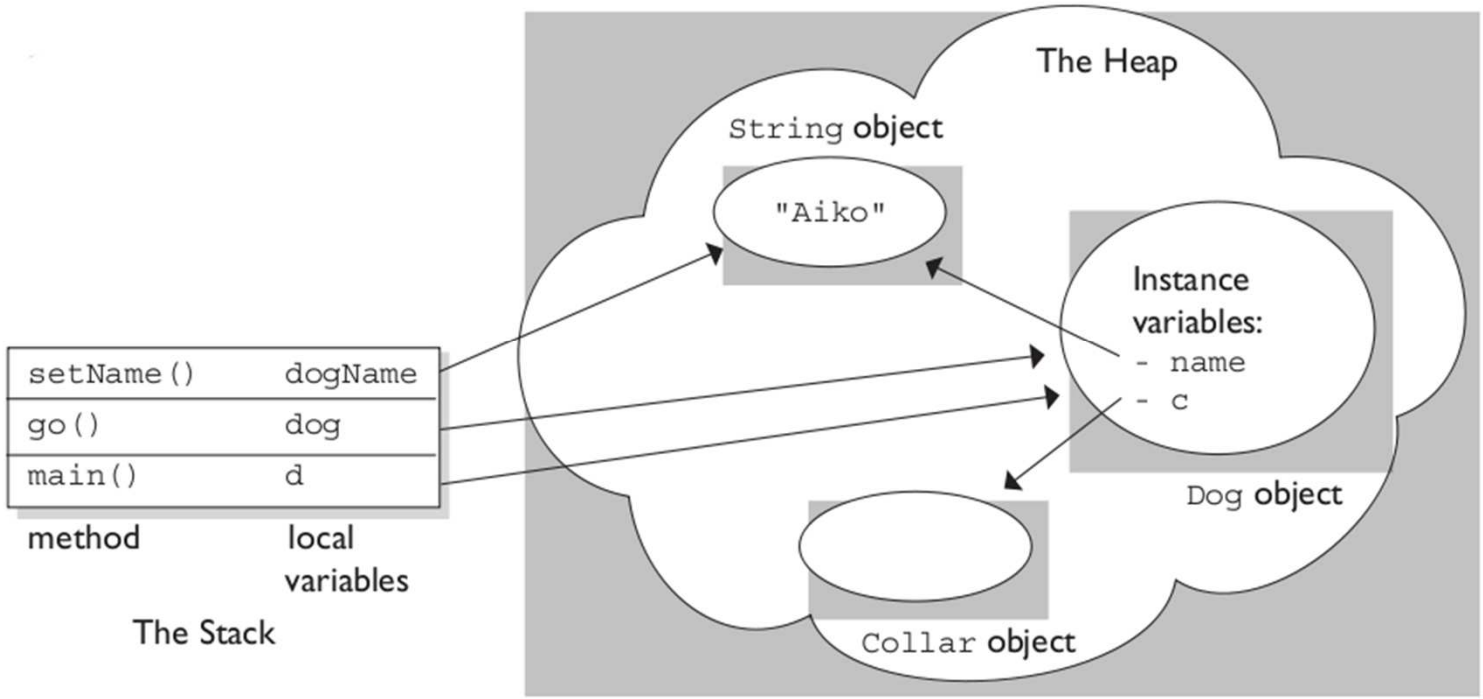
What are the 2 types of Memory

- **Stack – value types stored here**
- **Heap – objects stored here**



Memory

- **Stack – value types stored here**
- **Heap – objects stored here**



Rules for Variables

- **Declare only once (within a scope)**
- **Meaningful name**
- **Prefer camel case (randomNumber) or use _lowercase**
- **Don’t use C# keywords**
- **Don’t create variable which differ only by case**
- **Letters, numbers, and the underscore are valid characters**
- **Don’t begin with a number**

VALID	INVALID
`float f = 123.45;`	`bool continue = true;`
`bool isOpen = false;`	`string 1Cat = “boo”;`
`char vowel = ‘a’;`	`long l@ng = 123;`

Literal values

- **Sometimes the compiler needs some help**
- **1.254 is assumed to be a double (use M if decimal)**

```
decimal amt = 100.00M;
```

Category	Suffix	Description
Integer	U	Unsigned
	L	Long
	UL	Unsigned long
Real number	F	Float
	D	Double
	M	Decimal
	L	Long

Characters & Strings

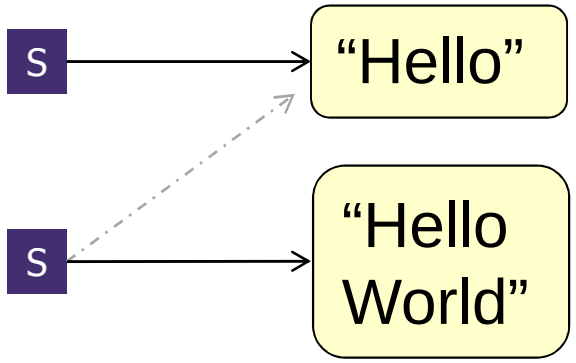
- **Use single-quotes for a character**
  `char x = ‘b’;`
- **Use double-quotes for a string**
- **A collection of characters is a string**
  `string x = “the lazy dog jumped”;`
- **Some characters cannot be represented unless escaped**

Escape sequence	Character name
\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

C# Strings

- **String literals are enclosed in double quotes**
- **Strings are made up of characters**
- **Strings are immutable (change results in new object)**

```
string s = “Hello”;
s = s + “ World”;
```



- **String instances have helpful methods such as:**
  - `ToUpper()`
  - `ToLower()`
  - `Split()`
  - `Replace()`
  - `Trim()`

Strings (cont.)

- **Strings are actually objects**
- **Considered a primitive type**
- **Can be `null` (uninitialized, no value) or empty “”**
- **The string class has some useful methods:**
  - Concat()
  - IsNullOrEmpty()
  - Compare()
  - Format()
  - Empty (a property, not a method. Returns “”)

```
string s = string.Empty; //s = “”
if (string.IsNullOrEmpty())
  Console.WriteLine(“String is empty”);
```

The char Type

- **char represents one unicode character**
- **Character literals are enclosed in single-quotes i.e. ‘X’**
- **Recall that a string is made up of characters**
- **The string Literal “Amy” has three characters ‘A’, ‘m’, ‘y’**
- **The char type has helpful methods:**
  - IsDigit()
  - IsLetter()
  - IsNumber()
  - IsPunctuation()
  - IsSeparator()

Arrays

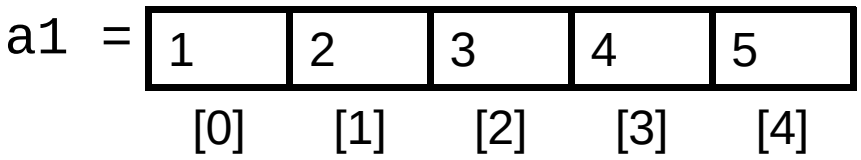
- **Collection of several values**
- **Each item is of same type**
- **Declare using [ ]**

```
int[] a1 = new int[2];
string[] names = new string[3];
```

- **Can be initialized when declared**

```
int[] a1 = new int[]{1,2,3,4,5};
```

- **Dimensions cannot change after initialization**
- **Index starts at 0**




```
a1[0] = 1
a1[3] = 4
a1[4] = 5
```

Enumerated Types

- **A group of named integral constants**
- **user-defined**
- **Makes code easier to read and maintain**
- **Allows you to define a type restricted to certain values**

```
public enum Planet
{
    Mercury ,
    Venus,
    Earth,
    Mars
}
```

Enumerated Types (cont.)

- **Any integral type, default is int**
- **When no value given, assumed to start at 0**

```
public enum Planet: int
{
    Mercury = 0,
    Venus = 1,
    Earth = 2,
    Mars = 3
}
////////////////////////////////////////
Planet p = Planet.Mercury;
```

Converting Between (non-string) Types

- **Common when using operations on values of different types**
- **Implicit conversion – when no loss of data (**compiler does it)
- **Explicit conversion – data might be lost** (you do it)
- **Use conversion syntax () or methods of System.Convert class**

```
decimal d = 1234.56M;
int x = (int)d;
int y = Convert.ToInt32(d);
```

Converting From a String

- **Cannot convert from string to numeric type directly**

```
string x = “55”;
int y = (int)x; //ERROR
```

- **Use one of these methods:**
  - Methods of the System.Convert class
  - Parse method (found on primitive types)

```
string x = “55”;
int y = int.Parse(x);
int y = Convert.ToInt32(x);
```

Converting To a String

- **Just call the ToString() method**

```
int y = 12345;
string x = y.ToString();
```

Expressions

- **Purpose is to perform an action and return a value such as**
  - Assign a value to a variable
  - Perform a mathematical calculation
- **Use operators such as +,-,=**

Operators

- **Primary: Order of precedence**
  `( )`
- **Assignment : use the equal sign**

```
int a = 1234;
```

- **Arithmetic: +, -, /,%, ***

```
int b = 5 * 5;
int x = 5 * (2+7);
int y = 5/3; // = 1 (integer division)
int z = 5%3 // = 2  (Modulo gives remainder)
```

Unary Operators

- **++, --**

```
int x = 5;
x++; //same as x = x+1
x--; //same as x = x-1
```

- **+=, -=, *=, /=, %=**

```
int x = 5;
x+=2; //same as x = x+2
x-=7; //same as x = x-7
x*=5; //same as x = x*5
x/=3; //same as x = x/3
x%=2; //same as x = x%3
```

Conditional (Logical) Operators

- **Result in a bool (**`true or false`**)**
- **==** is equal (this is not assignment)
- **!=** not equal
- **>, <, >=, <=**
- **&&** “and” (uses short-circuit evaluation)
- **||** “or” (uses short-circuit evaluation)

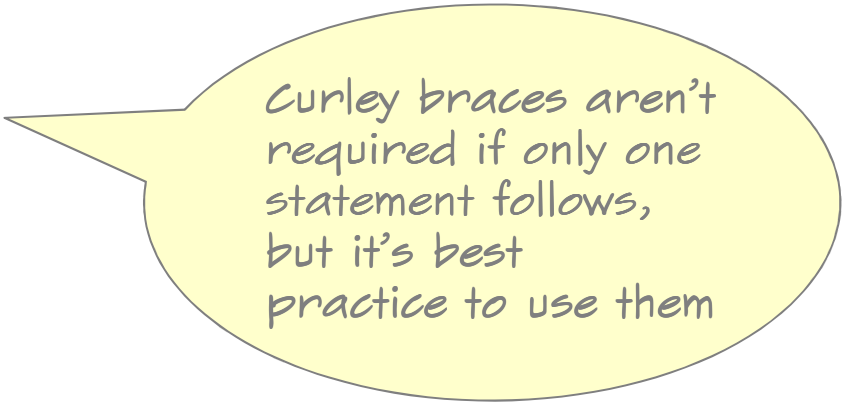
```
bool eq = (a==b);
bool isGt = (x > z && eq);
bool iGa= (a==b && b > 5);
bool isGe= (x > z || eq);
bool y = true;
bool n = false;
```

Conditional Statements

- **if or if … else is used to manage the flow of control**

```
if (x < 5)
        Console.WriteLine(“X is small”);
else
{
        Console.WriteLine(“X is large”);
}

if (y == true)
  Console.WriteLine(“y is true”);
//-- equivalent to  --
if (y)
 Console.WriteLine(“y is true”);
```



Conditional Statements

- **if or if … else is used to manage the flow of control**

```
if (sales < 100000)
{
        bonus += .05 * sales; //5% bonus if sales < 100K
}
else if(sales == 100000)
{
         bonus += .010 * sales  //10% bonus if sales is 100K
}
else
{
        bonus += .015 * sales  //15% bonus if sales > 100K
}
```

Switch Statement

- **used in place of many if..else statements**

```
switch (myPlanet)
{
    case Planet.Mercury:
          Console.WriteLine(“Hot!”);
          break;
    case Planet.Venus:
          Console.WriteLine(“Cloudy”);
          break;
    case Earth:
    case Mars:
          Console.WriteLine(“Nice place”);
          break;
    default:
          Console.WriteLine(“Far Out”);
          break;
}
```

Iteration Statements

- **C# provides several looping statements**
- **Used to execute a block of code repeatedly**
- **Stops when some condition is met**
- **3 types:**
  - for loop
  - while loop
  - do loop
  - foreach loop
- **Any loop can be made to work, but the right one produces cleaner code**

For Loop

- **Execute a set number of times**
- **Used when you know in advance how many times to loop**

```
for (int i=0; i<5; i++)
{
    Console.WriteLine(“ i = {0}”, i);
}

----------------------------------------

int[] nums = new int[] { 1, 2, 3, 4, 5 };

for (int i = 0; i < nums.Length; i++)
     Console.WriteLine(nums[i]);
```

While Loop

- **Pre-Test loop**
- **Executes 0 or more times**

```
string[] names = new []{“bob”,”keith”,”amy”};
int i = 0;

while(i < names.Length)
{
    Console.WriteLine(names[i]);
    i = i + 1;
}
```

Do Loop

- **Pre-Test loop**
- **Executes at least once**

```
string[] names = new []{“bob”,”keith”,”amy”};
int i = 0;
do
{
    Console.WriteLine(names[i]);

} while(i++ < names.Length)
```

foreach Loop

- **Easily enumerate over a collection**

```
int[] nums = new int[]{2,4,6,8,10,12,14,16,18,20};

foreach(int x in nums)
{
      Console.WriteLine(“The number is {0}”, x);
}
```

Loop keywords

- **break – exits the loop**
- **continue – goes back up to the top**

```
string[] names = new []{“bob”,”keith”,”amy”,”\n”};
int i = 0;

do
{
        if (names[i] == “bob”)
                continue;
        Console.WriteLine(names[i]);

        if (names[i] == “amy”)
                break;
        i++;
} while(names[i] != “\n”)
```

A Better Array

- **Primitive arrays are limited**
  - You have to know how many elements you want
  - You can't resize the array bounds
- **The .NET framework provides a dynamic array type:**

  List<*type*>
- **In the System.Collections.Generic namespace**
- **List is a *generic* type, you have to specify the concrete type**

```
List<string> list = new List<string>();
list.Add("Hello");
list.Add("World");
////////////////////////////////////////
List<int> nums = new List<int>();
nums.AddRange(new[]{100, 200, 300, 400, 500});
nums.RemoveAt(0); //remove the first item
```

Console Project: Getting User Input

- **Use Console.Write (or WriteLine) to prompt the user**
- **Use Console.ReadLine to get input (as a string)**

```
Console.Write(“Enter your name: “);
string input = Console.ReadLine();

Console.Write(“Enter your age: “);
string ageString = Console.ReadLine();
int age = int.Parse(ageString);
```

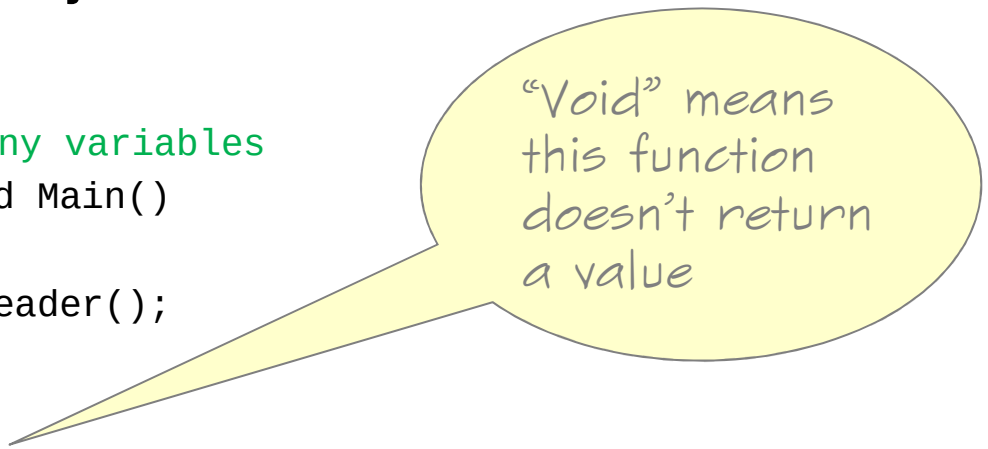
Other Console Class Methods

- `Clear()` – clear the screen
- `ForegroundColor()` – sets the text color
- `ResetColor()` – reset foreground text back to default

Console Project Best Practice

- **Don't write a lot of code in the Main() method**
- **Use helper methods to do the work**
- **Console project methods are static**

```
class program
{
   //declare any variables
    static void Main()
    {
        PrintHeader();
        //…
    }
    static void PrintHeader()
    {
         Console.WriteLine("******************");
         Console.WriteLine("**\tWELCOME!\t**");
         Console.WriteLine("******************\n");
    }
}
```



String Format

- **Some of the methods that accept string formats include:**
  - Console.Write()
  - Console.WriteLine()
  - String.Format()
- **String formats contain replaceable parameters**

```
string s = string.Format("Hello, my name is {0}", "Amy");
//Hello, my name is Amy

Datetime today = DateTime.Today;
s = string.Format("Today is {0:dddd}, {1:D}", today, today);
//Today is Tuesday, February 21, 2012
```

More Info:
http://www.dotnetperls.com/string-format

StringBuilder Class

- **System.Text namespace**
- **Mutable strings (can be changed)**
- **Can yield performance improvements**

```
StringBuilder sb = new StringBuilder();
sb.Append("Hello");
string a = "Amy";
sb.AppendFormat(" there, {0}", a);
//Hello there Amy
a = sb.ToString();
```

More Info:
http://www.dotnetperls.com/stringbuilder

Reading Text From a File

- **System.IO namespace**
- **File class can read contents of a file**
- **Use the (static) ReadAllLines method**
- **ReadAllLines returns an array of strings (one per line of text)**

```
string[] lines = File.ReadAllLines("filename.txt");
foreach(string line in lines)
{
    Console.WriteLine(line);
}
Console.WriteLine("File has {0} lines of text", lines.length);
```

Random Numbers

- **Generated by the Random class (System namespace)**

```
class Program
{
  static Random rand = new Random();

  static void Main()
  {
        int guess = 0;
        int minVal = 1;
        int maxVal = 20;
        for (int i=0; i<10; i++)
        {
           guess = rand.Next(minVal, maxVal);
           Console.WriteLine("The guess is " + guess.ToString());
        }
  }
}
```