**C#**

* https://learn.microsoft.com/en-us/dotnet/api/system?view=net-8.0
* General formatting
  + //denotes comments
  + “ and ‘ are different
  + Breakdown Statements
    - Class.Method();
      * Class represents and object
      * Method is what you do with the object
        + Always has() these are called *method invocation operators*
        + Every method has one job.
      * . is a *member access operator*- or how you navigate from a class to method
      * ; is a *statement* operator – these tell the compiler that you are finished entering the command
  + Code blocks are typically in {} brackets and limit the scope of variables (if you can read them in or out of those brackets)
    - Variables retain value changes that occur within code blocks
  + Breakdown Variables
    - Datatype variableName = variableValue;
      * Refer to table for data types
      * variableName
        + begin with letters
        + case sensitive
        + not be a keyword
        + \_ are fine but special characters are not
        + Camel case which is lowercase first word, uppercase first letter of every other word

example; thisIsCamelCase

* + - * + Do not abbreviate so it can be legible and self-apparent to others
  + .Net Class Library
    - Contains thousands of classes which contain tens of thousands of methods.
    - Methods can be stateful or stateless
      * Stateful methods put and use data from ‘fields’ (variables). You must create an ‘instance’ of a class beforehand to create it’s field.
        + You create instances using the new operator

|  |
| --- |
| Example Code: Random dice = new Random(); [Class object = new Class();]   * Random is the class. * Dice is the object   + Object is a way to call a method without having to redefine it * New is the operator used to create an instance of the class   *New creates an address in pc’s memory to store object that is based on the class (in this case it’s storing the dice object based on the random class)* |

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Format** | **Example** | **Other Details** |
| Basics | | | |
| Comments | // | // this is a comment | Comment not run in program |
| Multi-Line Comment | /\*  Comment  \*/ |  | Comments everything between lines |
| Using | Using X | Using System | Allows us to use classes from **X** namespace |
| Namespace | Namespace X | Namespace HelloWorld |  |
| Class | Class X | Class Program |  |
| Read Line | Console.ReadLine() | var = Console.ReadLine() | Waits for user to enter something then gives that value to whatever it is interacting with. (variable, condition, etc) |
| Terminal Commands | | | |
| Create Console (project) | dotnet new console -o X/Y | dotnet new console -o ./CsharpProjects/TestProject | Creates new console at the x location with the Y name |
| Build Application | Dotnet build |  | Builds the application from what’s in the .cs file |
| Run Application | Dotnet run |  | Runs application |
| Misc | | | |
| Random | Random varName = new Random(); | Random coin = new Random();  int flip = coin.Next(0, 2);  Console.WriteLine((flip == 0) ? "heads" : "tails");  [Example](https://learn.microsoft.com/en-us/training/modules/csharp-evaluate-boolean-expressions/5-solution-1) | Randomly generates |
| Data Types | | | |
| String | “X” | Method(“X”); | Denotes a string data type. Notice “ |
| Character | ‘X’ | Method(‘A’); | Denotes a character data type. Notice ‘ |
| Int | X | Method(123) | Denotes a integer data type |
| Float | X**f** | Method(5.5**f**) | Denotes a float data type precise up to 6-9 digits |
| Double | X | Method(5.5) | Denotes a float data type precise up to 15 digits |
| Decimal | Xm | Method(5.5m) | Denotes a float data type precise up to 29 digits |
| Boolean | true or false | Method(true) | Denotes a boolean, notice lowercase and no ‘ or “ |
| Casting | variable = (datatype) var1 + (datatype)var2 | int first = 7;  int second = 5;  decimal quotient = (decimal)first / (decimal)second; | Temporarily changes data type of variable to whatever is in ( ) |
| Nullable | datatype? varName; |  | A nullable data type is a data type for which the input can be null. Best used for user entered values. |
| Data Conversions | | | |
| Implicit Conversion | int Var1=num;  string Var2=text;  string Var3= var1+var2 | int first = 2;  string second = "4";  string result = first + second; | You can implicitly convert from any data type to another if the new type holds more information. For example int to decimal or Int to string. |
| Casting | int var = (int)var2; | decimal myDecimal = 3.14m;  int myInt = (int)myDecimal; | Casting is explicitly converting from one data type to another. Used when going from a wider type of data to a narrower. |
| To String | var.ToString(); |  | Not needed in most cases because converting to string is a usually widening conversion. It helps other developers know what your code is doing. |
| String to Int | int.Parse(var); |  | Turns var into an int. Can cause error if not possible. Combine with method below to ensure no runtime errors happen. |
| Bool if a string Is an int | int.TryParse(stringVar, out intVar) |  | Checks if it is possible to turn string to int. Gives true value if it is. Assigns intVar the value.  Does not have to parse for int. Could be decimal. |
| to Int | Convert.ToInt32(var); |  | Turns any data type into an integer. Has correct rounding where casting just turnicates.  ToInt32 has 19 overloaded versions to account for most data types.  There are many methods in Convert class to cover all data types. |
| String Manipulation | | | |
| All Upper | varName =.ToUpper() |  | Turns all characters in varName string upper case |
| All Lower | varName = varName.ToLower() |  | Turns all characters in varName string Lower case |
| Trim | varName = varName.Trim() |  | Removes trailing or leading blank spaces |
| Split | varName = varName.Split(x) | string[] items = result.Split(','); | Splits variable wherever specified in x. Could be a dash like ‘ – ‘. Can turn it into an array if the var is an array |
| Parse | int intVar = int.Parse(stringVar) |  |  |
| Index | varName.IndexOf(“x”) |  | Finds the index location of X in variable. If not present returns -1. |
| Variables | | | |
| Variable | datatype varName; | String firstName; | Creates a string variable. |
| Char | char varName; | char userOption; | Creates a char variable |
| Int | int varName; | int gameScore; | Creates an integer variable |
| Integer Datatypes | sbyte  short  int  long  byte  ushort  uint  ulong |  | All of these are numeric data types. Top 4 are “signed” in that they have a bit at the front telling them if they’re negative or positive. Bottom 4 are “unsigned” in that they don’t have a negative, but because of have an extra bit that doubles their range. |
| Decimal | decimal varName; | decimal particlesPerMillion; | Creates a decimal variable |
| Decimal Datatypes | float  double  decimal | 6-9 digits of precision  15-17 digits of precision  28-29 digits of precision | Higher down the greater degree of precision. |
| Bool | bool varName; | bool processedCustomer; | Creates a bool variable |
| Var | Var varName= X; | Var myString=”string variable”; | Creates a variable where the data type is deteremind by whatever is after the =  In this case a string variable.  Once you create a variable with this method, you cannot change it to a different datatype |
| Defining Variable | varName = Value | myString=”string variable”; | Defines the variable with whatever is after the = to later retrieve |
| Substring | varName.Substring(startindex);  Or  varName.Substring(startindex, Length) |  | creates a substring of the string that starts at the index determined by the startindex value. Can set end index at length |
| Remove | varName.Remove(startIndex); |  | used to remove a specified number of characters from a string, starting at a specified position. |
| Printing Text | | | |
| Write | Console.Write(): | Console.Write(“TEXT”); | Prints TEXT in-line |
| Write Line | Console.WriteLine(); | Console.WriteLINE(“TEXT”); | Prints Text then adds line break after text |
| Character escape | \x | Console.WriteLINE(“TEXT\nTEXT”); | Adds formatting to output string. List of values;  \n – adds new line  \t – adds a tab  \” – adds “ without breaking a string  \u – adds character represented by 4 digit unicode  \\ - adds backslash (needed because \ is a break) |
| Verbatim String | Console.WriteLine(@””); | Console.WriteLine(@" c:\source\repos  (this is where your code goes)"); | Keeps all whitespace and characters (linebreaks, tabs, etc) in the string in the output |
| Concatenate | Value 1 + Value 2 | string firstName = "Bob";  string greeting = "Hello";  Console.WriteLine(greeting + " " + firstName + "!"); | Concatenates values. In this case it is concatenating variables and strings that will then be printed by WriteLine |
| Interpolation | $”{var1} text {var2} | Console.WriteLine($"{updateText} {version}!"); | Using $”x” lets you define things using multiple variable and data, anything in { } is a variable. |
| Starts With | Y.StartsWith(X) | order.StartsWith("B") | Returns true value if the variable Y starts with value X |
| Data Formatting | | | |
| Formatting | String.Format(“TEXT {x}”, XVAR); | string result = string.Format("{0} {1}!", first, second); | Formats the string being defined |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Math | | | |
| Math when concatenating |  | Console.WriteLine(“name sold " + (widgetsSold + 7) + " widgets."); | This example will add widgetsold variable and 7 and then concatenate it with the rest of the line |
| Addition | var=X+y | Int answer = 5 +5 | Adds x and y |
| Subtraction | var=X-y | Int answer = 5 - 4 | Subtracts y from x |
| Multiplication | var=X\*y | Int answer = 5 \* 5 | Multiplies x and y |
| Division | var=X/y | decimal answer = 7.0m/5 | Divides X by Y. NOTE: To get decimals, the data type of the variable needs to be decimal and one of the numbers needs to be a decimal type. |
| Remainder | X % Y | Console.writeLine =( 7 % 5)  7 / 5 : 2 | Gives the remainder when you preform integer division |
| Increment & decrement | Var += x | int value = 5  value += 5; // value is now 10. | Different increments:  += (adds by value after =)  -= (subtracts by value after =)  \*= (multiplies by value after =)  ++ (adds by 1)  -- (subtracts by 1)  You can also put increments after the value to increment after the value is retrieved  Ex; Console.WriteLine($"Second: {value++}"); |
| IF - [EXAMPLE](https://learn.microsoft.com/en-us/training/modules/csharp-if-elseif-else/5-solution) | | | |
| IF | if( X)  {  Y  } | if (total > 14)  {  Console.WriteLine("You win!");  } | Creates a conditional statement if the value of the X condition is true. The codeblock in Y runs if it is. In the example, total is a variable. |
| OR | if ((x) || (y))  {  Z  } | if ((roll1 == roll2) || (roll1 == roll3) || (roll2 == roll3))  {  total+=2;  } | OR is symbolized by the || between expressions. The expressions need to be in () |
| AND | if ((x) && (y))  {  Z  } | if ((roll1 == roll2) && (roll2 == roll3))  {  Console.WriteLine("You rolled triples! +6 bonus to total!");  total += 6;  } | && are the AND operators |
| Else | if ( X)  {  Y  }  else  {  Z  } |  |  |
| Else IF | if ( X)  {  Y  }  else if  {  Z  }  else  {  A  } |  | Lets you write a series of if statements |
| Nesting | if ( X)  {  if (Y)  {  ZED  }  else  {  ABC  } |  | If X is true, if Y is true then ZED, else ABC |
| Switch Cases – [Example](https://learn.microsoft.com/en-us/training/modules/csharp-switch-case/4-solution) | | | |
| Switch | Switch (var)  {  case “val1”:  Codeblock;  break;  } | switch (fruit)  {  case "apple":  Console.WriteLine($"App will display information for apple.");  break;  default:  Console.WriteLine($"App will display information for item.");  break;  } | Creates a condition that checks if the value of the variable provided matches any of the cases. If it does not, goes with the “default” response. Break exits the loop. (return could be used too)  Can add multiple cases by  {  case “val1”:  case “val2”:  codeblock:  break;  } |
| Boolean Operators | | | |
| Contains | X.Contains(“Y”) | message.Contains(“Fox”) | Checks if the variable X contains the value Y somewhere. |
| Comparisons | X > Y X >= Y  X == Y |  | Various ways of comparing X and Y |
| Logical Negation | ! | string sent= "The fox is brown.";  Console.WriteLine(!sent.Contains("fox")); | Used to reverse logic of argument. Doesn’t have to be used with =.  Look at example for way of reversing boolean result |
| Conditional Operator | Conditional statement ? true value : false value | saleAmount > 1000 ? 100 : 50; | Determines whether the example is true or false and returns the value associated with the Boolean value |
| Arrays - [EXAMPLE](https://learn.microsoft.com/en-us/training/modules/csharp-arrays/5-solution) | | | |
| Array | datatype[] varNam = new datatype[# of elements]  datatype[] varNam = {elements}; | string[] fraudulentOrderIDs = new string[3];  string[] fraudulentOrderIDs = { "A123", "B456", "C789" }; | The [] creates an array. This is creating a variable named fraudulent order ids that is an array with the 3 possible values.  This creates an array with the length based on the amount of values entered. |
| Multi-dimensional Array (MATRIX) | datatype[,] varName = new datatype[#rows, #col] | string[,] ourAnimals = new string[maxPets, 6 | Creates a multi dimensional array and specifies the # of rows and columns. Rows is the max # of entries in the array. Columns is the # of data points per array entry.  The comma indicates it’s multidimensional |
| Character Array | char[] charArry = stringVar.ToCharArray(); |  | Creates an array from the string variable |
| Assigning Value | varName[position #] = “Value”; |  |  |
| Assigning value -  Matrix | varName[row #, col #] = “Value”; |  | Used to assign values to matrices |
| Pulling Value Matrix | varName[index#, |  |  |
| Length | varName.Length | fraudulentOrderIDs.Length | Returns number of elements in array |
| Sorting | Array.Sort(arryNam); | Array.Sort(pallets); | Sorts the array in () alphanumerically. |
| Reverse Sort | Array.Reverse(arryName) | Array.Reverse(pallets); | Sorts the array in the inverse of alphanumerical |
| Clear | Array.Clear(arryNam, StartInd#, EleAmt#) | Array.Clear(pallets, 0, 2); | Clears the value in the array starting at the starting index value, and goes for as many element amount # as specified. |
| Resize | Array.Resize(ref arryName, newSize) | Array.Resize(ref pallets, 6); | Resizes the array referenced to the new size. If you resize to shrink it, you lose any values at the end of the array. |
| Join | String.Join(",", arryNam); | string result = String.Join(",", valueArray); | Joins the values of the array with “ , “ in between. |
| For - [EXAMPLE](https://learn.microsoft.com/en-us/training/modules/csharp-for/4-solution) | | | |
| Foreach | datatype[] arrayNam = {elements};  foreach (datatype tempVarName in arrayrNam  {  BLOCK  } | string[] names = { "Rowena", "Robin", "Bao" };  foreach (string name in names)  {  Console.WriteLine(name);  } | Loops through and runs the code block for each element in the array starting from the first one. Holds the value for the current element being checked in the temp variable specific, in this case “Name” |
| For | for (initializer ; condition;iterator)  code block | for (int i = 0; i < 10; i++)  {  Console.WriteLine(i);  } | Initializer (int i=0) sets the variable going up at 0. Condition (i<10) determines when the loop will end. Iterator (i++) determines what happens after each loop, which in this case is go up by 1 (can be i+=3 or i-=2, etc).  Foreach also cannot change the value of the variable being used. Name in the example above. For can do this because it’s logic is based on the value of the initializer (i in this case) |
| For (reversing array) | Datatype[] varName = {val1,val2};  for (int i =varName.Length -1; i>=0;i--) | string[] names = { "Alex", "Eddie", "David", "Michael" };  for (int i = names.Length - 1; i >= 0; i--)  {  Console.WriteLine(names[i]);  } | This is a way to reverse an array which is not possible with a foreach script. It’s backward because you’re starting with the full length (back) of an array, then subtracting 1 per iteration. (in the example) |
| Break | Loop  code block  break | for (int i = 0; i < 10; i++)  {  Console.WriteLine(i);  if (i == 7) break;  } | Break tells a loop when to end. In the example, it’s telling a for loop to end at i=7 |
| While - [EXAMPLE](https://learn.microsoft.com/en-us/training/modules/csharp-do-while/4-solution) | | | |
| Do-while | do  {  code block  } while (condition) | do  {  // This code executes at least one time  } while (true); | Repeats the code block until the while statement is true. Because the condition is checked after the code block, the code will run for the instance when the condition becomes false as well. |
| While | while (condition)  {  code block } | while (current >= 3)  {  Console.WriteLine(current);  current = random.Next(1, 11);  } | Repeats the code block until the condition after while statement is false. Because condition is before code block, possible to not include instance when it becomes false. |
| Continue | do  {  code block  if (condition) continue;  } while (condition) | do  {  current = random.Next(1, 11);  if (current >= 8) continue;  Console.WriteLine(current);  } while (current != 7); | Continue will jump to the outside condition if the condition in () is met. |
| Methods | | | |
| Creating Method | returntype MethodName(parameters);  {  codeblock  } | void SayHello(); | Return type is the type of data returned by the method. In the case of void, it returns nothing. Method Name is whatever the method is called. Parameters are in (). Whatever is in the code block will be called when method is used.  Methods don’t need to come before the code calling it. Usually methods are at the end of a program.  After a method is complete it will return to the location of it’s caller. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Method comprised of following parts

* Structure is;
  + Return type method name ( parameters )