* *Programming Review Section 1 - Introuduction*
* The modulo operator - represented by the % symbol - returns the remainder of dividing two numbers.

|  |  |
| --- | --- |
| Code Snippit | Output |
| public class Modulo {  public static void main(String[] args) {  int myRemainder = 12 % 10;  System.out.println(myRemainder);  }  } | *2* |

* Relational Operators will always return a boolean value of true or false. A relational operator is placed between the two operands (the terms that you want to compare using the relational operator).
  + < : less than.
  + < = : less than or equal to.
  + > : greater than.
  + > = : greater than or equal to.

The equality operators are:

* + = = : equal to.
  + ! = : not equal to.

The boolean operators are:

* + & & : and (3rd)
  + | | : or (2nd)
  + ! : not (1st)

|  |  |
| --- | --- |
| Code Snippit | Output |
| public class RelationalOperators {  public static void main(String[] args) {  System.out.println( 4 > 5);  }  }  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  public class EqualityOperators {  public static void main(String[] args) {  System.out.println(5 == 5);  }  }  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  public class And {  public static void main(String[] args) {  System.out.println( 4 > 3 && 2 > 3);  }  }  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  public class OR {  public static void main(String[] args) {  System.out.println( 4 > 5 || 3 < 5);  }  }  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  public class Not {  public static void main(String[] args) {  System.out.println( !(2>=2) );  }  }  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  public class Precendence {  public static void main(String[] args) {  boolean riddle = !( 1 < 8 || (5 > 2 && 3 < 5));  System.out.println(riddle);  }  } | *false*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *true*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *false*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *true*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *false*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *false* |

* If statements: if is the first part of a conditional expression followed by a Boolean expression and then a block of code. If the Boolean expression evaluates to true, the block of code that follows will be run. Otherwise, if the Boolean expression evaluates to false, it will run the block of code after the else keyword.

|  |  |
| --- | --- |
| Code Snippit | Output |
| public class IfElse {  public static void main(String[] args) {  if (7 < 7) {  System.out.println("Try again...");  } else {  System.out.println("Success!");  }  } } | *Success!* |

* Ternary Conditions: means "composed of three parts".

These three parts are:

1. A Boolean expression
2. A single statement that gets executed if the Boolean expression is true
3. A single statement that gets executed if the Boolean expression is false.

|  |  |
| --- | --- |
| Code Snippit | Output |
| public class Ternary {  public static void main(String[] args) {  int fuelLevel = 3;  char canDrive= (fuelLevel > 0) ? 'Y' : 'N';  System.out.println(canDrive);  }  } | *Y* |

* Switch Statements

|  |  |
| --- | --- |
| Code Snippit | Output |
| public class Switch {  public static void main(String[] args) {  char penaltyKick = 'R';  switch (penaltyKick) {  case 'L':  System.out.println("First Case!"); break;  case 'R':  System.out.println("Second Case!”);  break;  case 'C': System.out.println("Third Case!”);  break;  default:  System.out.println();  }  }  } | *Second Case!* |

* Classes: a class is a set of instructions that describe how a data structure should behave. Create the starting state of our class by adding a class constructor to it. Modify by constructor by adding paramaters, which allow data types to be created with specified attributes.

|  |  |  |
| --- | --- | --- |
| Dog.java | Animal.java | Output |
| class Dog extends Animal { // class  int age;  public Dog(int dogsAge) { // contructor with int param  age=dogsAge;  }  public void bark(){ // public method  System.out.println("Woof!");  }  public void run(int feet){  System.out.println("Your dog ran " + feet + " feet!");  }  public static void main(String[] args) { // main method  Dog spike= new Dog(5); // initializes dog constructor  spike.bark(); // Calls the bark method on spike object  System.out.println("My dog is " + spike.age); //age  spike.run(50);  spike.checkStatus();  }  } | class Animal {  public void checkStatus() {  System.out.println("Your pet is healthy and happy!");  }  } | *Woof !*  *My Dog is 5*  *Your dog ran 50 feet !*  *Your pet is healthy and happy!* |

*Data Structures:*

* For Loop repeatedly runs a block of code until a specified condition is met.
* The statements within the parentheses of for loop compose the following parts:
* Initialization: the int variable named counter is initialized to the value of 0 before the loop is run.
* Test condition: the Boolean expression counter < 5 is a conditional statement that is evaluated before the code inside the control statement is run every loop. If the expression evaluates to true, the code in the block will run. Otherwise, if the expression evalutes to false, the for loop will stop running.
* Increment: Each time the loop completes, the increment statement is run. The statement counter++ increases the value of counter by 1 after each loop.

|  |  |
| --- | --- |
| Code Snippit | Output |
| public class For {  public static void main(String[] args) {  for (int waterLevel = 0; waterLevel < 7; waterLevel++) {  System.out.println("The pool's water level is at " + waterLevel + " feet.");  }  }  } | *The pool's water level is at 0 feet.*  *The pool's water level is at 1 feet.*  *The pool's water level is at 2 feet.*  *The pool's water level is at 3 feet.*  *The pool's water level is at 4 feet.*  *The pool's water level is at 5 feet.*  *The pool's water level is at 6 feet.* |

* ArrayList- stores a list of data of a specified type. To use first create an arraylist object (similar to constructor). Use the .add() function to insert new elements into the arraylist object. Use the .get() method to print data. We can access the elements of the arraylist object by using an element's index, or position, in the list. An element's index refers to its location within an ArrayList. ArrayLists are zero-indexed, which means that the first element in an ArrayList is at a position of 0.

|  |  |
| --- | --- |
| Code Snippit | Output |
| import java.util.ArrayList;  public class Temperatures {  public static void main(String[] args) {  ArrayList<Integer> weeklyTemperatures =  new ArrayList<Integer>();  weeklyTemperatures.add(78);  weeklyTemperatures.add(67);  weeklyTemperatures.add(89);  weeklyTemperatures.add(94);  weeklyTemperatures.add(2, 111); // Index 2 value 111 (pushes  next index down)  System.out.println( weeklyTemperatures.get(1) );  }  } | *67* |

* Iterating over an ArrayList-The statements within the parentheses of for loop compose the following parts:
* Initialization: the int variable *i* is set to 0 which is the first index of an arraylist
* Test condition: the code will run as long as *i* is less than the size of the object of the arraylist.
* Increment: I will be increemented by 1 with *i++* after each loop

|  |  |
| --- | --- |
| Code Snippit | Output |
| import java.util.ArrayList;  public class TemperaturesC {  public static void main(String[] args) {  ArrayList<Integer> weeklyTemperatures =  new ArrayList<Integer>();  weeklyTemperatures.add(78);  weeklyTemperatures.add(67);  weeklyTemperatures.add(89);  weeklyTemperatures.add(94);  weeklyTemperatures.add(2, 111);    for(int j=0; j<weeklyTemperatures.size(); j++)  {  System.out.println(weeklyTemperatures.get(j));  }  }  } | *78*  *67*  *111*  *89*  *94* |

* For Each loop - shortcut to reduce the amount of code required to write the loop called the for each loop. The colon ( : ) can be read as “in”. Removes the .get() function and prints value of element.

|  |  |
| --- | --- |
| Code Snippit | Output |
| import java.util.ArrayList;  public class TemperaturesC {  public static void main(String[] args) {  ArrayList<Integer> weeklyTemperatures =  new ArrayList<Integer>();  weeklyTemperatures.add(78);  weeklyTemperatures.add(67);  weeklyTemperatures.add(89);  weeklyTemperatures.add(94);    for( Integer temperature : weeklyTemperatures)  {  System.out.println(temperature);  }  }  } | *78*  *67*  *89*  *94* |

* Hashmaps - contains a set of keys and a value for each key. If we look up a word in a dictionary, we can get the definition. If you provide a HashMap with a key that exists, you can retrieve the value associated with the key. (like a double data type constructor). Use the .put() method to add keys and values to the hashmap. Again use .get() to retrive values of the key. (Use for each loop)

|  |  |
| --- | --- |
| Code Snippit | Output |
| import java.util.HashMap;  public class Restaurant {  public static void main(String[] args) {  HashMap<String, Integer> restaurantMenu= new HashMap<String, Integer>();  restaurantMenu.put("Turkey Burger", 13);  restaurantMenu.put("Naan Pizza" , 11);  restaurantMenu.put("Cranberry Kale Salad", 10);    System.out.println( restaurantMenu.get("Naan Pizza") );  }  } | *11* |

* Iterating over a hashmap. Access properties of a HashMap, such as the number of entries or the contents of the HashMap.

|  |  |
| --- | --- |
| Code Snippit | Output |
| import java.util.HashMap;  public class Restaurant {  public static void main(String[] args) {  HashMap<String, Integer> restaurantMenu= new HashMap<String, Integer>();  restaurantMenu.put("Turkey Burger", 13);  restaurantMenu.put("Naan Pizza" , 11);  restaurantMenu.put("Cranberry Kale Salad", 10);    System.out.println(restaurantMenu.size());  for (String item : restaurantMenu.keySet()) {  System.out.println("A " + item + " costs " + restaurantMenu.get(item) + " dollars.");  }  }  } | *3*  *A Turkey Burger costs 13 dollars.*  *A Cranberry Kale Salad costs 10 dollars.*  *A Naan Pizza costs 11 dollars.* |

Data Structures Summary:

* For Loops: used to repeatedly run a block of code.
* For Each Loops: a concise version of a for loop.
* ArrayList: stores a list of data
* HashMap: stores keys and associated values like a dictionary

|  |  |
| --- | --- |
| Code Snippit | Output |
| import java.util.\*;  public class GeneralizationsD {  public static void main(String[] args) {  ArrayList<String> sports = new ArrayList<String>();  sports.add("Football");  sports.add("Boxing");  for(String sport : sports) {  System.out.println(sport);  }  HashMap<String, Integer> majorCities = new HashMap<String, Integer>();  majorCities.put("New York", 1624);  majorCities.put("London", 43);  majorCities.put("Mexico City", 1521);  majorCities.put("Sao Paulo", 1554);  for ( String city : majorCities.keySet() ) {  System.out.println(city + " was founded in " + majorCities.get(city));  }  } } | *Football*  *Boxing*  *New York was founded in 1624*  *London was founded in 43*  *Sao Paulo was founded in 1554*  *Mexico City was founded in 1521* |

*Javascript Review Section 2 - Introduction*

* Four essential data types in JavaScript include strings, numbers, booleans, and null.
* Data is printed, or logged, to the console with console.log().
* Four built-in mathematical operators include +, -, \*, and /.
* JavaScript associates certain properties with different data types.
* JavaScript has built-in methods for different data types.
* Libraries are collections of methods that can be called without an instance. Common math methods.
* Math.floor
* Math.ceil
* Number.isInteger()
* Math.random()

*C# Review Section 3 - Introduction*

* The Main Method : By default, C# will try and execute a method called Main as the starting point of the application. This is known as the application Entry Point.

|  |
| --- |
| The Main Method |
| using System;  public class Program  {  public static void Main( )  {  }  } |

* Variables : A variable by definition is an identifier pointing to a storage location in memory, which contains its value.
* C# supports type inference - which means that you don't always have to explicitly specify a type - you can let the compiler try and understand the type of variable automatically. However, once the type of variable has been determined, it cannot be assigned a different type.

|  |
| --- |
| Declaring Variables |
| using System;    public class Program  {  public static void Main()  {  string name = "Matt";  int age = 31;  DateTime today = DateTime.Now;  float myFloat = 1f; //f is required in float variables  bool myBoolean = true;  string myName = "John";  char myChar = 'a';  double myDouble = 1.75;  *var x =1; // cant be assigned to different type.*  }  } |

* Classes – Every object is made from a class.

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  //Write Vehicle class here  class Vehicle{    public string Type;  public int NumTires;  public int Year;  public bool Runs;  public Vehicle(string type,int numTires, int year, bool runs){  Type=type;  NumTires=numTires;  Year=year;  Runs=runs;  }  }  class MainClass{  public static void Main(){    //Create vehicles here  Vehicle car= new Vehicle("Car",4,2000,true);  Vehicle oldcar= new Vehicle("Old Car",4,2000,true);  Vehicle bike= new Vehicle("Bike",4,2000,true);    // Test code  Console.WriteLine(car.Type);  Console.WriteLine(oldcar.Runs);  Console.WriteLine(bike.NumTires);  }  } | *Car*  *True*  *4* |

* Arrays: are defined using the brackets operator [], and they are initialized using a list defined with curly braces. For example: int[] nums = { 1, 2, 3, 4, 5 }; or an empty array: int[] nums = new int[10];
* To get the size of the array use the Length method: Console.WriteLine(nums.length);
* Zero based index : int firstNumber = nums[0]; nums[2]=10;

|  |  |  |
| --- | --- | --- |
| For Each Loop to print arrays | Equivalent For Loop | Output |
| int[] arr = {2, 0, 1, 3};  for (int el : arr) {  Console.WriteLine(el);  } | int[] arr = {1, 9, 9, 5};  for (int i = 0; i < arr.Length; i++) {  int el = arr[i];  Console.WriteLine(el);  } | *2 1*  *0 9*  *1 9*  *3 5* |

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  using System.Collections.Generic;  public class Arrays  {  public static void Main()  {  int[] nums = new int[10];  nums[0] = 8;  nums[2] = 10;  nums[5] = 12;  nums[9] = 4;    string error ="End of line";  int counter=1;  foreach(int i in nums){  Console.WriteLine("{0}. Array value is {1}. {2}",  counter,i,error);  counter++;  }  }  } | *1. Array value is 8. End of line*  *2. Array value is 0. End of line*  *3. Array value is 10. End of line*  *4. Array value is 0. End of line*  *5. Array value is 0. End of line*  *6. Array value is 12. End of line*  *7. Array value is 0. End of line*  *8. Array value is 0. End of line*  *9. Array value is 0. End of line*  *10. Array value is 4. End of line* |

* MultiDimensional Arrays

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  using System.Collections.Generic;  public class Matrix  {  public static void Main()  {  int[,] matrix = new int[2,2];    matrix[0,0] = 1;  matrix[0,1] = 2;  matrix[1,0] = 3;  matrix[1,1] = 4;    int[,] predefinedMatrix = new int[2,2] { { 5, 4 }, { 0, 2} };    foreach(int i in matrix){  Console.WriteLine("Matrix 1: {0}",i);  }  Console.WriteLine("");  foreach(int j in predefinedMatrix){  Console.WriteLine("Matrix 2: {0}",j);  }  }  } | *Matrix 1: 1*  *Matrix 1: 2*  *Matrix 1: 3*  *Matrix 1: 4*  *Matrix 2: 5*  *Matrix 2: 4*  *Matrix 2: 0*  *Matrix 2: 2* |

* Lists (Similar to Arraylist in java) : A list is an object which holds variables in a specific order. The type of variable that the list can store is defined using the generic syntax. List<int> numbers= new List<int>(); Use .Add() to insert item into the list. Add an array to the list using the AddRange ()function for concatenating lists. Use .Remove() to remove item from list or .RemoveAt() to specifiy an index of an item to remove.

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  using System.Collections.Generic;  public class Lists  {  public static void Main()  {  List<int> numbers = new List<int>();  int[] array = new int[] { 1, 5, 3, 6 };  numbers.AddRange(array);  Console.WriteLine("There are " + numbers.Count + " items in  the List" );  foreach(int a in numbers){    Console.WriteLine(a);  }  }  } | *There are 4 items in the list.*  *1*  *5*  *3*  *6* |

* Dictionary (Similar to Hashmap in Java) : special lists, whereas every value in the list has a key which is also a variable. To define a dictionary, provide a generic definition with two types for the key and value. Add values by using brackets operator or Add method. To check whether a dictionary ahs a certain key in it, use the ContainsKey() method.

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  using System.Collections.Generic;  public class Dictionary  {  public static void Main()  {  Dictionary<string, int> phonebook = new Dictionary<string,  int>();  phonebook.Add("Alex", 415434543);  phonebook["Jessica"] = 415984588;    if (phonebook.ContainsKey("Jessica"))  {  Console.WriteLine("Jessica's number is " + phonebook["Jessica"]);  }  }  }  } | *Jessica's number is 415984588* |

* Strings:
* ToString(): To cast a non-string object into a string, use the built in 'object.ToString()' function.
* String.Format() : To format a string, use the String.Format function. Each additional argument to the function can be referred to in the string using the brackets operator with the index number.
* Substring() : The Substring string method returns a part of the string, beginning from the index specified as the argument. Substring also accepts a maximum length for the substring.
* Search and Replace: The Replace string method replaces an occurrence of a string with another string.
* Index of: The IndexOf string method finds the first occurrence of a string in a larger string. If the string is not found, -1 is returned.
* Parsing Strings/ Delimiters - A string can be parsed using the String.Split() method. As input, Split takes an array of characters that indicate which characters separate interesting sub strings of the target string. The function returns an array of the sub strings.
* The following example uses spaces, commas, periods, colons, and tabs, all passed in an array containing these separating characters. Each word in the target string's sentence displays separately from the resulting array of strings.
* writer.WriteLine(“{0} {1} {2}”, counter, words, error ); // string error= ‘Sentence has been modified.’ // Says print the first index being i in the loop, the text split, and error.

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  class TestStringSplit  {  static void Main()  {  char[] delimiterChars = { ' ', ',', '.', ':', '\t' };  // space, comma, period, colon, and tab.  string text = "one\ttwo three:four,five six seven"; // one (tab) two (space) three (colon) four (comma) five(space) six (space) seven(space)  Console.WriteLine("Original text: '{0}'", text);  string[] words = text.Split(delimiterChars, StringSplitOptions.RemoveEmptyEntries); // Array of strings to  split the characters in the *text* string using delimiterChars array, and remove the empty strings in the output (optional).  string error= ‘Sentence has been modified.’  Console.WriteLine("{0} words in text:", words.Length); // Prints the words in the text  int counter = 1;  foreach (string s in words) // for every string s in the words array write the s to the console  {  Console.WriteLine(“{0}. {1} {2}”, counter, words, error);  counter ++;  }  }} | *Original text: 'one two three:four,five six seven'*  *7 words in text:*  *1. one Sentence has been modified*  *2. two Sentence has been modified*  *3. three Sentence has been modified*  *4. four Sentence has been modified*  *5. five Sentence has been modified 6. six Sentence has been modified*  *7. seven Sentence has been modified* |

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  using System.Collections.Generic;  public class Strings  {  public static void Main()  {  //ToString() Method  int integer = 5;  string ourString = "Something to be replaced by an int";  ourString = integer.ToString();  Console.WriteLine(ourString);  //String.Format() Method  int x = 1, y = 2;  int sum = x + y;  string sumCalculation = String.Format("{0} + {1} = {2}", x, y,  sum);  Console.WriteLine(sumCalculation);  string firstName = "John";  string lastName = "Doe";  int age = 27;  string sentence=String.Format("{0} {1} is {2} years old ",firstName,lastName, age);  Console.WriteLine(sentence);  // Substring() Method  string fullString = "The cat and the hat";  string partOfString = fullString.Substring(4);  string shorterPart = fullString.Substring(4, 3);  string shortestPart = fullString.Substring(4, 7);  Console.WriteLine(fullString);  Console.WriteLine(partOfString);  Console.WriteLine(shorterPart);  Console.WriteLine(shortestPart);  // Replace() Method  string name = "Erin Bell";  string newName = name.Replace("Bell", "Jones");  Console.WriteLine("Before: " + name);  Console.WriteLine("After: " + newName);  string married= newName.Replace("Erin", "-");  Console.WriteLine("Married Name: " + name +  married.Replace("- ","-"));  // IndexOf() Method  string fruit = "apple,orange,banana";  Console.WriteLine(“Found orange in position: " + fruit.IndexOf("orange"));  //Console.WriteLine("Found lemon in position: " + fruit.IndexOf("lemon"));  String input = Console.ReadLine();  if(fruit.IndexOf("banana")>= 0){  Console.WriteLine("Item Found in this position: " + (fruit.IndexOf("banana")));  }  else{  Console.WriteLine("Item was not found");  }  }  }  }  } | *5*  *1 + 2 = 3*  *John Doe is 27 years old.*  *The cat and the hat*  *cat and the hat*  *cat*  *cat and*  *Before: Erin Bell*  *After: Erin Jones*  *Married Name: Erin Bell-Jones*  *Found orange in position: 6*  *Item Found in this position: 13* |

* For Loops: For loops are used to allow you to repeat sections of code a fixed, or variable amount of times. This allows you to make your code more compact and clean. Use *break* to exit a loop without finishing. Use *continue* to skip straight to next iteration. This example prints only the even numbers by skipping iterations where *i* is odd. Note: i % 2 == 0 represents even numbers; ==1 for odd numbers
* While Loops: allows you to continously repeat a section of code while a condition is satisfied

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  using System.Collections.Generic;  public class For  {  public static void Main()  {  for(int i=0; i < 16; i ++)  {  if( i % 2 ==1) {  continue;  }  }  }  }  public class Functions  {  public static void Main()  {  string x = "Hi";  int n = 10;  //write while loop here  int counter=0;  while(counter<n){  Console.WriteLine("{0}. {1}",counter,x);  counter++;  }  }  } | *0*  *2*  *4*  *6*  *8*  *10*  *12*  *14*  *0. Hi*  *1. Hi*  *2. Hi*  *3. Hi*  *4. Hi*  *5. Hi*  *6. Hi*  *7. Hi*  *8. Hi*  *9. Hi* |

* Console.WriteLine : To be able to use the Console class in our program, we needed to add a using *directives* at the top of our program file. Applications built on the .NET Framework organise their classes into containers called Namespaces. In our example, the Console class belongs to the System namespace. There are many namespaces provided by the .NET Framework:
* System.Collections - Contains classes for creating lists and dictionaries for storing data.
* System.IO - Contains classes used to read and write files, and other disk operations.
* System.XML - Contains classes for using XML.

|  |  |
| --- | --- |
| Console.WriteLine() | Without Directive |
| using System;  public class Program  {  public static void Main( )  {  Console.WriteLine("Hello World");  }  } | public class Program  {  public static void Main( )  {  System.Console.WriteLine("Hello World");  }  } |

* Console.ReadLine : The Console API provides various methods for reading input from the user.
* Read - Reads the next character in the string and returns it as an integer character code.
* ReadLine - Reads until it finds a new line, carriage return, or the end of your input string, and returns all that it reads as a new string.
* ReadKey - Reads the next character in the string and returns it as an instance of ConsoleKeyInfo which provides additional information about they key that was pressed, and the state of any modifier keys (like Shift, Alt, etc.)

|  |  |
| --- | --- |
| Console.WriteLine() | Output |
| using System;  public class Program  {  public static void Main()  {  Console.WriteLine("Hi, what is your name?");  string name = "";    name=Console.ReadLine();  Console.WriteLine("Hi, " + name);  }  } | Hi, what is your name?   * Erin   Hi, Erin |

* Console.ReadLine() vs Console.Read() : the ReadLine method reads an entire line at a time and returns you the input text. Another common operation is the Read method. Unlike ReadLine, the read method returns one character at a time until it reaches the end of the input. This includes, letters, numbers, line breaks, and all other characters.

One of the big differences between Read and ReadLine is that Read doesn't actually return a string, it returns an integer, moreover it returns the ASCII representation of the character you entered. To obtain the actual character entered, we need to convert this integer into a character. The simplest, easiest approach is to perform a cast

|  |  |
| --- | --- |
| Console.read() | Casting |
| using System;  public class Program  {  public static void Main()  {  Console.WriteLine("Say anything!");  int value = Console.Read();  Console.Write("You typed: " + (char)value);  }  } | int value = 65; // The ASCII code for 'A'  char letter = (char)value; |

* StreamReader and StreamWriter
* Found in the System.IO namespace, both classes are useful when you want to read and write character based data.
* StreamReader derives from the Abstract class "TextReader" and StreamWriter derives from "TextWriter".

|  |  |
| --- | --- |
| StreamWriter | StreamReader |
| Close()- Closes the current StreamWriter object and the underlying stream.  Write()- This method is used to write data to a text stream without a newline.  WriteLine()-This method is used to write data to a text stream with a newline. | Close()-Closes the current StreamReader object and the underlying stream.  Read()-Reads the next character from the input stream.  ReadLine()-Reads a line of characters from the current stream and returns the data as a string.  ReadToEnd()-Reads the stream from the current position to the end of the stream. |

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  class Program  {  static void Main( )  {  WriteToFile();  ReadFromFile();  }  public static void ReadFromFile()  {  using (StreamReader sr =  File.OpenText(@"E:\Programming  Practice\CSharp\Console\table.tbl"))  {  string tables = null;  while ((tables = sr.ReadLine()) != null)  {  Console.WriteLine("{0}",tables);  }  Console.WriteLine("Table Printed.");  }  }    public static void WriteToFile()  {  using (StreamWriter sw =  File.CreateText(@"E:\Programming  Practice\CSharp\Console\table.tbl"))  {  sw.WriteLine("Please find the below generated  table of 1 to 10");  sw.WriteLine("");  for (int i = 1; i <= 10; i++)  {  for (int j = 1; j <= 10; j++)  {  sw.WriteLine("{0}x{1}= {2}",i,j,(i\*j));  }  sw.WriteLine("==============");  }  Console.WriteLine("Table successfully written on  file."); }}} | *Table successfully written on file*  *Please find the below generated table of 1 to 10*  *1 x 1 = 1*  *1 x 2 = 2*  *1 x 3 = 3*  *1 x 4 = 4*  *1 x 5 = 5*  *1 x 6 = 6*  *1 x 7 = 7*  *1 x 8 = 8*  *1 x 9 = 9*  *1 x 10 = 10*  *===========*  *2 x 1 = 2*  *2 x 2 =4*  *2 x 3 =6*  *2 x 4 = 8*  *2 x 5 = 10*  *2 x 6 = 12*  *2 x 7 = 14*  *2 x 8 = 16*  *2 x 9 = 18*  *2 x 10 = 20*  *============* |

* Methods: portions of a larger program that perform specific tasks and for reuse in previous code.

[Modifiers (E.G public or static)] [Type of output] [Name] ( [parameter 1],[parameter 2] ...)

{

}

|  |  |
| --- | --- |
| Code Snippit | Output |
| using System;  using System.Collections.Generic;  public class Methods  {  public static void Main() {  Console.WriteLine(Multiply(5, 3));  int x = 45;  int y = 15;  int a = Foo(x,y);  Console.WriteLine(a);  }  public static int Multiply(int a, int b) {  return a \* b;  }  public static int Foo(int x, int y) {  return x / y;  }    } | *15*  *3* |