IF STATEMENTS

int age = 37;

//see all the results-make multiple if statements

if (age >= 21)

{

Console.WriteLine("Can Drink");

}

else if (age >= 18)

{

Console.WriteLine("Can Vote");

}

else if (age >= 16)

{

Console.WriteLine("Can Drive");

}

//the last one must always be an else

else

{

Console.WriteLine("Too Young");

}

SWITCH CASE = a menu of choices

Console.WriteLine("Please enter a menu choice 1-3");

int menuChoice = int.Parse(Console.ReadLine());

switch (menuChoice)

{

case 1: Console.WriteLine("Here are all books");

break;

case 2: Console.WriteLine("Check out a book");

break;

case 3: Console.WriteLine("Return book");

break;

default: Console.WriteLine("Invalid choice");

break;

}

//ternary operator

int age2 = 14;

string status = age2 >= 18 ? "Adult" : (age2>=13 ? "Teen" : "Child");

Console.WriteLine(status);

//-----------------------------------------------------

//LOOPS

//For Loop

//if you know exactly how many times you want to loop

// 1 create variable that holds loop interations (loop count)

// 2 determine boolean expression (true = run, false = stop) for how long it runs

// 3 counter increment/decrement

for (int i = 0; i <= 10; i++)

{

Console.WriteLine(i);

}

//starts at 10 and will count down to 0

for (int i = 10; i >= 0; i--)

{

Console.WriteLine(i);

}

//While loop

// 1 make a variable to use as a condition

// 2 create while loop-inside while () has to be a bool expression

bool runProgram = true;

while (runProgram)

{

//Main code here

//ask to continue

Console.WriteLine("Would you like to continue? y/n");

string choice = Console.ReadLine();

if (choice == "n")

{

runProgram = false;

}

}

//do while

// 1 create bool variable

// 2 create "do" loop, then "while" loop

bool loop = false;

do

{

Console.WriteLine("do while");

} while(loop);

//another example

int menu = -1;

do

{

Console.WriteLine("Menu choice 1-3 (0 to leave)");

menu = int.Parse(Console.ReadLine());

if (menu == 1)

{

Console.WriteLine("See all books");

}

else if (menu == 2)

{

Console.WriteLine("Check out");

}

else if (menu == 3)

{

Console.WriteLine("Return");

}

} while(menu != 0);

//Foreach loop

//square brackets indicate an array

string[] staff = { "Justin", "Ethan", "Billy" };

foreach (string s /\*for staff\*/ in staff)

{

Console.WriteLine(s);

}

//break

//this is a validation loop

int positive = -1;

while(true)

{

Console.WriteLine("Please enter a number. Must be positive.");

positive = int.Parse(Console.ReadLine());

if (positive >= 1)

// break gets out of any loops

{ break;

}

else

{

Console.WriteLine("Invalid. Try again");

}

}

Console.WriteLine(positive);

//Continue

//this loop only shows the even numbers

for (int i = 0; i <=10; i++)

{if (i % 2 ==1)

{

//skip to next loop

continue;

}

Console.WriteLine(i);

}

Random randomNumber = new Random();

//fight

Console.WriteLine("Welcome to the fight arena.");

for (int round = 1; round <= 3; round++)

{

//fist number after next is the first number, and not including 6

int bot1 = randomNumber.Next(1, maxAttack1 + 1);

int bot2 = randomNumber.Next(1, maxAttack2 + 1);

Console.WriteLine($"Bot 1: {bot1}");

Console.WriteLine($"Bot 2: {bot2}");

if (bot1 == bot2)

{

Console.WriteLine("It's a tie");

}

else if (bot1 > bot2)

{

bot1Score++;

Console.WriteLine($"Bot 1 wins round {round}");

}

else if (bot1 < bot2)

{

bot2Score++;

Console.WriteLine($"Bot 2 wins round {round}");

}

}

Console.WriteLine("FINISHED");

Console.WriteLine();

Console.WriteLine($"Bot 1's final score is: {bot1Score}");

Console.WriteLine($"Bot 2's final score is: {bot2Score}");

if (bot1Score == bot2Score)

{

Console.WriteLine($"Its a tie!");

}

else if (bot1Score > bot2Score)

{

Console.WriteLine($"Bot 1 won the battle!");

}

else

{

Console.WriteLine($"Bot 2 won the battle!");

}

//use a method for cubing and for squaring

// Console.WriteLine(int.MaxValue);

//the cube root of the max value: 2147483647 is:1290

//prevent users from entering a number higher than 1290

Console.WriteLine("Welcome to the Powers Table!");

Validation Loop

bool runProgram = true;

while (runProgram)

{

//prompt the user to enter an integer

Console.WriteLine("Enter an integer between 1-1290.");

int answer = int.Parse(Console.ReadLine());

//provide validation by rejecting 0 or negative numbers and limit the user to the max number whose cube will fit in an int

if (answer > 0 && answer < 1290)

{

//right-align the numbers

//display a table of squares and cubes from 1 to the "answer" (value entered)

Console.WriteLine(String.Format("{0,15} | {1,15} | {2,15}", "Number", "Squared", "Cubed"));

Console.WriteLine(String.Format("{0,15} | {1,15} | {2,15}", "------", "------", "------"));

for (int counter = 1; counter <= answer; counter++)

{

//calling the methods that are at the bottom

Console.WriteLine(String.Format("{0,15} | {1,15} | {2,15}", counter, GetSquare(counter), GetCube(counter)));

}

bool keepGoing = true;

while (keepGoing)

{

//the application should prompt the user to continue and continue only if the user agrees to

Console.Write("Do you want to continue? Y/N");

string end = Console.ReadLine().ToLower().Trim();

if (end == "n")

{

//use a break when there is a valid input

Console.WriteLine("Bye!");

keepGoing = false;

runProgram = false;

break;

}

else if (end == "y")

{

//use a break when there is a valid input

break;

}

else

{

//do not use a break when there is no valid input

Console.WriteLine("That is not a valid response.");

}

}

}

}

Console.WriteLine("Thank you for using the Exponents Powers Table!");

//Methods

//use a different value from the "answer" above

static int GetCube(int x)

{

return x \* x \* x;

}

static int GetSquare(int x)

{

return x \* x;

}

METHODS

//CALLING METHOD 1

string greeting = GetHelloUser("Maria");

Console.WriteLine(greeting);

//CALLING METHOD 1

string name = "Alice";

string greeting2 = GetHelloUser(name);

Console.WriteLine(greeting2);

//CALLING METHOD 1

Console.WriteLine(GetHelloUser("Bob"));

//METHOD 1

//string = the return type

static string GetHelloUser(string user)

{

string result = $"Hello, {user}!";

//return matches the return type

return result;

}

//CALLING METHOD 2

string userName = GetCleanInput();

Console.WriteLine(userName);

//METHOD 2

//not all methods take an existing value, no perameters

static string GetCleanInput()

{

Console.WriteLine("Please enter a name.");

return Console.ReadLine().ToLower().Trim();

}

//CALLING METHOD 3

//method that takes in multiple perameters

double sum = AddThreeNumbers(9, 5, 3);

Console.WriteLine(sum);

//METHOD 3

static double AddThreeNumbers(double x, double y, double z)

{

return (x + y + z);

}

//CALLING METHOD 4

//not every method needs to return something, can be used to just display something.

//void means no return

DisplayDate();

//METHOD 4

static void DisplayDate()

{

//specific and current date/time

DateTime date = DateTime.Now;

Console.WriteLine(date);

}

//Excercises

//1. Create a method called convertMinuteToSeconds. It should take in an int called minutes.

//It should convert and return it as seconds.

Console.WriteLine(ConvertMinuteToSeconds(50));

static int ConvertMinuteToSeconds(int minutes)

{

return minutes \* 60;

}

//2. Create a method that is called areaTriangle. It should take in a double height

//and a double length. Return the area as a double.

Console.WriteLine(AreaTriangle(10, 20));

static double AreaTriangle(double height, double length)

{

return (height \* length) / 2;

}

//3. Create a method called convertNegative(). It should take in an int and return it as a

// negative version of that int.

// If it is already negative, don’t do anything with it. 0 should stay 0.

Console.WriteLine(ConvertNegative(5));

static int ConvertNegative(int x)

{

if (x <= 0)

{

return x;

}

else

{

return -x;

}

}

//4. Create a method called convertPostive(). It should take in an int and return it as a positive version of that int.

//If it is already positive, don’t do anything with it. 0 should stay 0.

Console.WriteLine(ConvertPositive(-5));

static int ConvertPositive(int y)

{

if (y >= 0)

{

return y;

}

else

{

return y \* -1;

}

}

//5. Create a method called greetUser(). It will have no parameter and no return.

//Get the name from the user and display “Hello “ and then their name.

GreetUser();

static void GreetUser()

{

Console.WriteLine("What is your name?");

string name5 = Console.ReadLine();

Console.WriteLine($"Hello, {name5}");

}

//another way to do number 5

GreetUser2();

static void GreetUser2()

{

string name = GetCleanInput();

Console.WriteLine(GetHelloUser(name));

}

Console.WriteLine(GetHelloUser(GetCleanInput()));

//1. Write a program in C# Sharp to create a user defined function.

//Expected Output :

//Welcome Friends!

//Have a nice day!

function1();

static void function1()

{

Console.WriteLine("Welcome friends! \nHave a nice day!");

}

//2. Write a program in C# Sharp to create a user defined function with parameters.

//Test Data :

//Please input a name : John

//Expected Output :

//Welcome friend John !

//Have a nice day!

function2();

static void function2()

{

Console.WriteLine("Please input a name:");

string funName2 = Console.ReadLine();

function1();

}

//3.Write a program in C# Sharp to create a function for the sum of two numbers.

//Test Data :

//Enter a number: 15

//Enter another number: 16

//Expected Output :

//The sum of two numbers is : 31

Console.WriteLine(function3(num1, num2));

static int function3(int num1, int num2)

{

Console.WriteLine("Enter a number: ");

int num1 = int.Parse(Console.ReadLine());

Console.WriteLine("Enter another number: ");

int num2 = int.Parse(Console.ReadLine());

Console.WriteLine("\nThe sum of two numbers is : {0} \n", function3(num1, num2));

}

// called IsEnoughMoney, take in 2 doubles: cost and payment. It will return a bool.

// if there is enough payment retrun a true, else return a false

//using System;

//using System.ComponentModel;

Console.WriteLine(IsEnoughMoney(20, 30.1));

static bool IsEnoughMoney(double cost, double payment)

{

bool IsEnoughMoney = true;

if (payment >= cost)

{

return true;

}

else

{

return false;

}

}

////method called GetHouse. Take in double called salary and will return a string telling you

////60k or les, get apartment

////80k or less, you can get a 1 br house

////100k or less, it will be 2 br house

////anything greater will be 3 br house

Console.WriteLine(GetHouse(52000.10));

static string GetHouse(double salary)

{

if (salary <= 60000)

{

return "Apartment";

}

else if (salary <= 80000)

{

return "1 Bedroom House";

}

else if (salary <= 100000)

{

return "2 Bedroom House";

}

else

{

return "3 Bedroom House";

}

}

////method called CombineWOrds

////take in 3 strings and return back a single string, combine all 3 into 1 string separated like commas.

Console.WriteLine(CombineWords("apple", "banana", "cherries"));

static string CombineWords(string x, string y, string z)

{

return $"{x}, {y}, {z}";

}

//---------------------------------------------------------------------

//New method topics - Pass by reference

//pass by value, the value of whatever we pass in goes into the parameter/variable

//ref creates a link

//pass by reference, you want the entire variable to pull through

int num1 = 5;

int num2 = 10;

FlipValues(ref num1, ref num2);

Console.WriteLine(num1);

Console.WriteLine(num2);

static void FlipValues(ref int x, ref int y)

// pass in 2 values and have them switch the values of the variables. Create temp variable

// that holds the values

{

int temp = x;

x = y;

y = temp;

}

// going to be using this one a lot on validation

// out creates a replacement when the method is done running

// Out Perameter - remembers a value, can return multiple values

// the remainder will = the change and return true

double item = 80;

double money = 100;

double remainder = 0;

if(CanPurchase(item, money, out remainder))

{

Console.WriteLine($"Item purchased {remainder} in change");

}

else

{

{

Console.WriteLine("Could not purchase.");

}

}

static bool CanPurchase(double cost, double wallet, out double change)

{

if (wallet >= cost)

{

change = wallet - cost;

return true;

}

else

{

change = wallet;

return false;

}

}

//another Out Perameter

double remain = 0;

double result = Divide(5,2, out remain);

Console.WriteLine($"Result = {result}");

Console.WriteLine($"Remainder = {remain}");

static double Divide(double x, double y, out double remainder)

{

remainder = x % y;

return x / y;

}

DAY 4 STRING LECTURE

// See https://aka.ms/new-console-template for more information

//strings are a group of things, a string is an array of data

string myString = "Maria";

string word = "Hello!";

//character data type is single quotes, only allowed to put one letter, #, or symble

//the square brackets are for the array

char[] letters = { 'E', 't', 'h', 'a', 'n' };

string myString2 = new string(letters);

Console.WriteLine(myString);

Console.WriteLine(myString2);

//add 2 strings together

string myString3 = myString + myString2;

Console.WriteLine(myString3);

myString = myString + " whatever";

myString2 = string.Concat(myString2, " whatever");

Console.WriteLine(myString);

Console.WriteLine(myString2);

//compare strings

string val1 = "hi";

string val2 = "Hi";

string sentence = "I am a GC student";

//pretend these are if statements

Console.WriteLine(val1 == val2); //returns false

Console.WriteLine(val2.Contains("i")); //case sensitive partial search - good for search that contains a keyword

Console.WriteLine(sentence.Contains("student"));// <--Justin's favorite

Console.WriteLine(String.Compare(val1, val2, true)); //not case sensitive. returns a value -1== does not exist, 0= do exist (same value)

//1 the value is found inside, "true" ignores the case

Console.WriteLine(val1.ToLower() == val2.ToLower()); // <--Justin's favorite

// not case sensitive, returns true which is easier than the random number above

// useful built in methods

//IndexOf finds the first instance

string value = "session";

Console.WriteLine(value.IndexOf("s"));//returns 0 because "s" is in the first spot

Console.WriteLine(value.IndexOf("ion"));//returns 4 because "ion" starts in the 4th spot

//LastIndexOf

Console.WriteLine(value.LastIndexOf("s")); //returns 3 because the last time "s" is shown is in the 3rd position

//EndsWith

string value2 = "I entered bootcamp!";

Console.WriteLine(value2.EndsWith("!")); //returns true because the sentence ends with "!"

//String Replace

string value3 = "where\_the\_sidewalk\_ends";

string result3 = value3.Replace('\_', ' ');

Console.WriteLine(result3); //replaces the \_ with spaces

//String[] Split (one of two ways you can do the PigLatin lab)

//Splits a sentence and make an array out of it

string value4 = "Dungeon and Dragons";

string[] splitLetters = value4.Split(" "); //what is inside the quotes is what you split it by.

//gets out every single word

//easier example

foreach (string words in splitLetters) //goes between each value in the array

{

Console.WriteLine(words);

}

//use this one for PigLatin lab

//Substring

// first way

string value5 = "chicken";

string firstLetter = value5.Substring(0, 2);//0 is the position of the letter you want to get so 0 is the first letter,

// the second number is how many letters you want to get

// this returns "ch"

Console.WriteLine(firstLetter);

//length

// second way

Console.WriteLine(value5.Length);

string lastLetter = value5.Substring(value5.Length - 3); //the -3 grabs everything from the end -3 so it prints "orm"

Console.WriteLine(lastLetter);

//To.Upper() or To.Lower()

string value6 = "This WILL all beCOME a diffERENT CASE";

Console.WriteLine(value6.ToLower());

Console.WriteLine(value6.ToUpper());

//Trim

string value7 = " Hi Maria "; //trims off of the spaces around the value on the beginning and the end

Console.WriteLine(value7);

Console.WriteLine(value7.Trim());

//Example Uses

//take a string, uppercase the first letter and then make everything else a lower case

Console.WriteLine(CorrectCasing("maRIa"));

string FullName = "marIA RAgone";

string result = "";

foreach (string n in FullName.Split(" "))

{

result += CorrectCasing(n) + " ";

}

Console.WriteLine(result);

static string CorrectCasing(string dirty)

{

string cleaned = dirty.Substring(0, 1).ToUpper() + dirty.Substring(1).ToLower();

return cleaned;

}

//Loop through letters

//Part of PigLatin Lab

string input = "Cocktails";

for (int i = 0; i < input.Length; i++)

{

Console.WriteLine(input[i]);

if (input[i] == 'a' || input[i] == 'e')

{

Console.WriteLine("Vowel spotted");

}

}

DAY 6 ARRAY LECTURE

//CREATING ARRAY

using System.Xml.Linq;

using System;

int[] grades = new int[9];//empty array (default values)

string[] names = { "Justin", "Ethan", "Maria", "Timothy", "Zach" }; //max size is 5 in this case

//create starting values

//ACCESSING VALUES

//use index

Console.WriteLine(names); //prints out the object type System.String[]

Console.WriteLine(names[1]); //prints "Ethan"

int chosen = 2; //plug in variable names

Console.WriteLine(names[chosen]); //what is within the [] "chosen" has to be an int

//prints "Maria"

//WRITE FULL ARRAY

foreach (string n in names)

{

Console.WriteLine(n); //displays all the names

}

//CHANGING VALUES

names[2] = "Kim"; //change "Maria" to "Kim" can use any variable, or other elements from the array

Console.WriteLine(names[2]);

//LOOPING THROUGH ARRAYS

//\*\*\*\*PRACTICE THESE

double[] prices =

{

40,

1000,

150,

5,

200

};

//FOR LOOP

// i == index to access the values

//Can edit array mid-loop

for (int i = 0; i < prices.Length; i++)

{

if (prices[i] >= 100)

{

prices[i] = prices[i] \* .85;

}

Console.WriteLine(prices[i]);

}

//FIND AVERAGE AFTER DISCOUNT

double total = 0; //holder variable gives running total

for (int i = 0; i < prices.Length; i++) //loops thru each value in array

{

total += prices[i];

}

double avg = total / prices.Length;

Console.WriteLine($"Average price with discounts: {avg}");

Console.WriteLine(prices.Average()); //another way to average, but use the loop to practice!

//FOR EACH (use 70% of the time)

//lets you process without needing indexes

//simplifies the code because it goes thru all values

//can't change value as you loop

//don't have access to index (knowing the location of a variable)

double[] prices2 =

{

8,

10,

13,

10,

25,

8

};

//CREATE HOLDER VARIABLE

//can do either one below

//double largest = prices[2];

double largest = double.MinValue;

foreach (double p in prices2) //p is variable to hold, in the prices2 array

{

Console.WriteLine(p);//prints each of the values

if (p > largest)

{

largest = p;

}

}

Console.WriteLine($"Largest is: {largest}");//prints the largest #

//FIND THE LOWEST VALUE IN THE ARRAY

double smallest = double.MaxValue;

foreach (double p in prices2)

{

if (p < smallest)

{

smallest = p;

}

}

Console.WriteLine($"Using a foreach loop, the smallest is: {smallest}");//prints the smallest #

Console.WriteLine($"This is the smallest using 1 line of code: {prices2.Min()}");//another way to print the smallest #

//2 DIMENSIONAL ARRAY

//RECTANGLE ARRAY

//has the same amount of rows per column

int[,] rectangle = new int[3, 5];

rectangle[2, 3] = 100;

Console.WriteLine(rectangle[2, 3]); //use for minesweeper game OR bingo card

for (int i = 0; i < rectangle.Length; i++)//pull each number in the rectangle array

{

for (int j = 0; j < rectangle.Length; j++)

{

}

}

//JAGGED ARRAY

//not used very often - not used in the lab!

//has different amount of rows per column

//number of classrooms in each class type

int[][] jagged = new int[3][];

//define each column

jagged[0] = new int[5];

jagged[1] = new int[3];

jagged[2] = new int[4];

//call the value in column 0 row 2

jagged[0][2] = 9;

//METHODS

//pass arrays in and out of methods

int[] values = { 1, 2, 3, 4 };

int[] result = AddOne(values);

Console.WriteLine("Values");

foreach (int v in values)

{

Console.WriteLine(v);

}

Console.WriteLine("Results");

foreach (int r in result)

{

Console.WriteLine(r);

}

static int[] AddOne(int[] numArr)

{

//any changes to arrays in methods effect the original array

//the method itself will effect the original array

//\*\*pass by reference

int[] temp = new int[numArr.Length];

for (int i = 0; i < numArr.Length; i++)

{

temp[i] = numArr[i] + 1;

}

return temp;

}

//EXCERCISES

//-----------------------------------------------

//1.USEFUL FOR THE LAB!

//make an array of values

// ask user to choose an index (your location in your array, 0 - (length - 1))

// use the index and display the value in console

int[] excercise1 = { 40, 1000, 150, 5, 200 };

Console.WriteLine($"Choose an index: 0 - {excercise1.Length - 1}");

int val1Input = int.Parse(Console.ReadLine());

Console.WriteLine(excercise1[val1Input]); //what is within the [] "chosen" has to be an int

//2. create another array of values (make sure there are some duplicates)

// ask user to choose any number (choose 6)

// loop through the array and only display index location when the value matches

int[] exercise2 = { 3, 6, 3, 2, 6, 7 };

Console.WriteLine($"Enter any number: ");

int val2Input = int.Parse(Console.ReadLine());

for (int i = 0; i < exercise2.Length; i++) //loops thru each value in array

{

if (exercise2[i] == val2Input)

{

Console.WriteLine(i); //only displays the index location (prints "1" and "4")

Console.WriteLine(exercise2[i]);

}

}

string[] names2 = {

"Justin Jones",

"Zach Buth",

"Omar Abdulla",

"Ethan Thomas",

"Joe Heath",

"Forrest Verellen",

"Doug Chu",

"Maya Araquil",

"Shane Chastain",

"Timothy Montague",

"Maria Ragone" };

Console.WriteLine($"Enter a name: ");

string searchName = (Console.ReadLine());

for (int n = 0; n < names.Length; n++) //loops thru each value in array

{

names[n] = names[n].ToLower();

if (names[n] == searchName)

{

Console.WriteLine(n); //only displays the index location (prints "1" and "4")

Console.WriteLine(names[n]);

}

}

DAY 6 SHOPPING LIST LAB

decimal total = 0;

Console.WriteLine("Welcome to Maria's Humorous Haberdashery for Humans.");

// Dictionary - store list of items and prices

Dictionary<string, decimal> store = new Dictionary<string, decimal>

{

// Start with some items and prices

// store.Add("Ravioli", 17.99) //can also add

{ "Zany Zoink Zapper", 119.99m },

{ "Shuffle and Sniff Bacon Slippers", 54.99m },

{ "Nostalgia Scented Air Freshener", 2.49m },

{ "Soup-er Hero Supper & Snack Ladle", 5.29m },

{ "Snooze Master Reverse Alarm Clock", 153.00m },

{ "Flame-Defiant Toaster", 72.99m },

{ "Chatter Leaf Talking Houseplant", 10.99m },

{ "Unicorn Glitter Perfume", 45.00m },

{ "Booty Bliss Buzzing Butt-erfly Chair", 749.99m }

};

// View the store list

Console.WriteLine(String.Format("{0,5} {1,-35} {2,10}", "Number", "Item", "Price"));

Console.WriteLine(String.Format("{0,5} {1,-35} {2,10}", "------", "----", "-----"));

// create a way to show a number next to the menu item

int number = 1;

//create the list

foreach (KeyValuePair<string, decimal> kvp in store)

{

Console.WriteLine(String.Format("{0,5}. {1,-35} {2,10}", number, kvp.Key, kvp.Value));

number++; // Add index every time

}

//create empty list to add into later

List<string> cart = new List<string>();

// get user input

// Ask the user to enter an item number

while (true)

{

Console.WriteLine("\nEnter an item number to add to your cart.");

string input = Console.ReadLine().Trim();

//if(store.ContainsKey(input)

//Console.WriteLine(store.ElementAt(int.Parse(numberChoice)).Key);

//tryparse is a method that tries the input without an exception. if it can then = true

if (int.TryParse(input, out int numberChoice) && numberChoice >= 1 && numberChoice <= store.Count)

{

string item = store.ElementAt(numberChoice - 1).Key;

Console.WriteLine($"You added {item} for {store[item]} to your cart");

cart.Add(item);

}

else //if user choice is invalid

{

Console.WriteLine("Invalid item number. Please try again.");

}

// Ask user if they want to add something else

Console.WriteLine("Do you want to add something else? y/n");

string answer = Console.ReadLine().ToLower().Trim();

if (answer == "n")

{

// Display the cart

Console.Clear();

Console.WriteLine("Here is your cart");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(String.Format("{0,-35} {1,10}", "Item", "Price"));

Console.WriteLine(String.Format("{0,-35} {1,10}", "------", "------"));

foreach (string c in cart)

//foreach (string c in cart.OrderByDescending(c => store[c])) // Display the items in the cart

{

total += store[c];

Console.WriteLine(String.Format("{0,-35} {1,10}", c, store[c]));

}

//display the total

Console.WriteLine($"\nPlease pay: {total}");

List<string> sortedCart = cart.OrderBy(c => store[c]).ToList();

// Display the most and least expensive items

string mostExpensive = sortedCart.Last();

string leastExpensive = sortedCart.First();

Console.WriteLine($"Most expensive item: {mostExpensive} @ {store[mostExpensive]}");

Console.WriteLine($"Least expensive item: {leastExpensive} @ {store[leastExpensive]}");

break; // Stop asking if they want to add something to the cart

}

else if (answer == "y")

{

}

else

{

Console.WriteLine("Invalid response.");

break;

}

}

DAY 7 LINQ LECTURE

using System.Diagnostics.Metrics;

//LINQ

//the name of the package we are using

List<int> ages = new List<int>

{

23,

21,

19,

30,

25

};

//COUNT METHOD

//returns how many match the condition

int adultCount = ages.Count(p => p >= 21 ); //this is a linq that can double as a method

// first value is the variable (p is for person

// p is the name of the variable to check against

Console.WriteLine($"There are {adultCount} adults");

//THE LINQ DOES THE SAME AS FOREACH

//ANY METHOD

//return a true or false if any elements match

bool childCheck = ages.Any(p => p < 21); //looks thru the list and checks if any of the people are under or = to 21

Console.WriteLine($"Is there a child here? {childCheck}");

//Write a senior check to see if anyone is 65 or older

bool seniorCheck = ages.Any(p => p >= 65); //looks thru the list and checks if any of the people are under or = to 21

Console.WriteLine($"Is there a senior here? {seniorCheck}");

//ALL METHOD

//all of them must match

//returns true if all match condition, returns false if any don't

bool allAdultCheck = ages.All(p => p >= 18); //returns a true if all are >= 18

Console.WriteLine($"Is everyone 18 or older? {allAdultCheck}");

//create a variable AllCanDrink

//check if all ages are 21 or older

bool allCanDrink = ages.All(p => p >= 21);

Console.WriteLine($"Can everyone drink? {allCanDrink}");

//USE WITH LINQ 70% OF TIME

//WHERE - filters the list and gives a brand new list

//create a new collection based on condition

//filtering method

//gives you all that matches

List<double> clothes = new List<double>()

{

40.00,

50.00,

200.99,

500.00,

10.00,

100.02

};

List<double> expensive = clothes.Where(c => c >= 100).ToList(); // c is variable name for clothes

// returns iEnumerable - .ToList() converts it to a list

Console.WriteLine("These are the expensive items:");

foreach (double c in expensive)

{

Console.WriteLine(c);

}

//Create a new list.

//Use where to filter it where the value == 100

//display it (NOTE: you won't get any back)

List<double> closet = new List<double>()

{

40.00,

50.00,

200.99,

500.00,

10.00,

100.02

};

List<double> hundred = closet.Where(c => c == 100).ToList();

Console.WriteLine("These are the $100 items:");

foreach (double h in hundred)

{

Console.WriteLine(h);

}

//another example with string

List<string> store = new List<string>()

{

"pants",

"shorts",

"shirt",

"scarf",

"jacket",

"dress",

"short",

"dresses"

};

Console.WriteLine("Search for a clothing");

string input = Console.ReadLine();

List<string> inventory = store.Where(i => i == input).ToList();

Console.WriteLine("These are the inventory items:");

foreach (string i in inventory)

{

Console.WriteLine(i);

}

DAY 8 EXCEPTIONS LECTURE

//RUNTIME ERROR

//Most common use case is parsing a number

//Code that can break goes in try

//if you want acces to x you have to create it before

using System.Text.RegularExpressions;

int x = 0;

try

{

//throws exception

x = int.Parse("9");

Console.WriteLine(5 / x);

}

catch (FormatException e) //child exception

{

Console.WriteLine("Value was not a number"); //developer handles the message

}

catch (Exception e) //this is the parent exception. anywhere the parent can go then the

//child can go

// this is the catch all

// the catches have to be in order

{

Console.WriteLine(e.Message); //c# handles the message

}

// Create a try catch

// ask the user for a number

// parse the number to an int

// display an error message

// display the parsed number outside of the try catch

// gives exact control over your code

int number = 0;

while (true)

{

try

{

Console.WriteLine("Type in a number.");

number = int.Parse(Console.ReadLine());

if (number <= 0)

{

// This can just be a cw

throw new Exception("Input too low."); // if you are writing code for someone else to use

// then this is handy

}

else

{

break;

}

}

//catch (FormatException n)

//{

// Console.WriteLine("Value was not a number");

//}

catch (Exception n)

{

Console.WriteLine(n.Message);

}

Console.WriteLine(number);

}

// TRYPARSE

// there is a way to do all logic in a few lines

// method that returns a bool if the parse is succesful

// uses out perameter to return back the value

Console.WriteLine("TryParse: Enter another number.");

string input = Console.ReadLine();

int value = -1; //random temp number to hold

if (int.TryParse(input, out value)) // when successful it returns a true

// input is the value you are trying to parse, the value is

// the parced value

{

Console.WriteLine(value);

}

else

{

Console.WriteLine("Not valid");

}

// VALIDATION LOOP with TRYPARSE

// not used with strings

// good for validating numbers

// loop constatnly asks user until they put something valid

Console.WriteLine("Validation loop with TryParse: Please enter a number.");

int result = -1;

while (result <0 || result > 10) // add a range validation

{

while (int.TryParse(Console.ReadLine(), out result) == false)

// only runs infinately if the user types something that fails

{

Console.WriteLine("Invalid response. Try again"); //error message

}

if (result < 0 || result > 10) // add a range validation

{

Console.WriteLine("Number out of range. Please try again");

}

}

// VALIDATION LOOP with TRYPARSE without the extra while loop

Console.WriteLine("Validation with TryParse without the extra While loop: Please enter a number.");

int result2 = -1;

while (int.TryParse(Console.ReadLine(), out result2) == false || result2 < 0 || result2 > 10)

// only runs infinately if the user types something that fails

{

Console.WriteLine("Invalid response. Try again"); //error message

}

Console.WriteLine("Great you entered a valid number!");

// REGEX

// Google and grab patterns from the internet

Regex regex = new Regex(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,4})+)$"); //pattern for email

if (regex.IsMatch("maria@ragone.net")) // if this value matches the pattern then it returns true

{

Console.WriteLine("Valid Email");

}

else

{

Console.WriteLine("Not a valid email");

}

DAY 9 WEEK 2 REVIEW

// 1 ARRAY

// Create an int array

// Fill it with some numbers

//loop thru the numbers and display all to console

int[] numbers = { 1, 10, 8, 4, 20 };

for (int i = 0; i < numbers.Length + 1; i++)

{

Console.WriteLine(i);

}

//or

foreach (int n in numbers)

{

Console.WriteLine(n);

}

// 2

// Ask the user for a sentence. Save it in a string

// Split the sentence into an array of words (strings)

// Display each word in console

Console.WriteLine("Write a sentence.");

string sentence = Console.ReadLine().ToLower().Trim();

string[] words = sentence.Split(" ");

foreach (string w in words)

{

Console.WriteLine(w);

}

// 3

// Make a method called AvgArray

// take in a double array

// retrun a double

// find the average of the values in the array and return it

double[] nums = { 1, 5, 8, 9, 22 };

Console.WriteLine($"The average of the numbers in the list is: {AvgArray(nums)}");

static double AvgArray(double [] values)

{

//return values.Average();

double sum = 0;

double count = 0;

foreach (double v in values)

{

count++;

sum += v;

}

return sum / count;

}

// 4

// create a method called FindIndex

// take in a string list and a string

// return an int

// look thru the string list and return the index of the string perameter

// if not found return -1

List<string> breakfast = new List<string> { "eggs", "bacon", "oatmeal", "cereal", "sausage", "toast"};

Console.WriteLine($"The word 'bacon' is found at index {FindIndex(breakfast, "bacon")}");

static int FindIndex(List<string> values, string word)

{

// return strings.FindIndex(s => s == input);

int index = -1;

for (int i = 0; i < values.Count; i++) //.count is for lists

{

if (values[i] == word)

{

index = i;

}

}

return index;

}

//.count is for lists

//.length is for arrays and strings

DAY 10 OOP/OBJECT ORIENTED PROGRAMMING LECTURE

using Day\_10\_OOP\_Lecture;

Pen myPen = new Pen("Blue", "Pilot", "Gel", 1.50m, 5);

//myPen.Color = "Blue";

//myPen.Brand = "Pilot";

//myPen.InkType = "Gel";

//myPen.Price = 1.50m;

//myPen.Length = 5;

Pen myPen2 = new Pen("Red", "Bic", "Gel", 0.75m, 5);

//myPen2.Color = "Red";

//myPen2.Brand = "Bic";

//myPen2.InkType = "Gel";

//myPen2.Price = 0.75m;

//myPen2.Length = 5;

Pen myPen3 = new Pen("Black", "PaperMate", "Gel", 2.50m, 6.5f);

//without values it isn't a pen

//need to give defaults

Console.WriteLine(myPen.Color);

Console.WriteLine(myPen2.Color);

myPen.Draw();

myPen2.Draw();

// ARRAY/LISTS

// a list of the pen object

List<Pen> CupOfPens = new List<Pen>

{ myPen, // use existing objects

myPen2,

myPen3,

new Pen ("Pink", "Sharpie", "Ball Point", 5m, 6.3f) //make a new object

};

foreach (Pen singlePen in CupOfPens)

{

DisplayPenDetails(singlePen);

singlePen.Draw();

}

// METHODS

static void DisplayPenDetails(Pen inputPen)

{

Console.WriteLine($"{inputPen.Brand} {inputPen.Color} {inputPen.InkType} {inputPen.Length}in ${inputPen.Price}");

}

//-----------------------------------------

// BANK SECTION

Bank1 myBank = new Bank1(5);

myBank.Deposit(10);

Console.WriteLine(myBank.GetBalance());

Console.WriteLine(myBank.Withdraw(1000));

//-----------------------------------------

// DOG SECTION

Dog sillyDog = new Dog("red", "lab", "huge", 5);

sillyDog.Bark();

DAY 11 OOP CONTINUED

// build a class that you can turn into objects to use in your code

using Day\_11\_OOP\_Lecture\_2;

Player p = new Player(); //p is the single object created using the Player class

// will default to overloaded empty constructor

Console.WriteLine(p.GetDetails());

// Player is now a datatype that can be used in an array

List<Player> MyClan = new List<Player>()

{

new Player("Jeo", "Support", 100),

new Player("Kat", "Trucker", 50),

new Player("Bob","Builder",30),

new Player("Bob","Builder",30),

new Player("Bob","Builder",30),

new Player("Eval", "Tank", 105),

new Player("Jo", "Tank", 95),

new Player("Shaggy", "Offender", 80),

new Player("Jo", "Tank", 95),

new Player("Hunter", "Shooter", 23),

};

Console.WriteLine("My Clan:");

foreach (Player pl in MyClan)

{

Console.WriteLine((pl.GetDetails()));

}

// LINQ sort collections, get true/false, find counts

// use for the lab to filter thru movie genres

// make a group to get anyone level 80 or over

List<Player> group = MyClan.Where(c => c.Level >= 80).ToList();

Console.WriteLine($"Able to fight the boss");

foreach (Player pl in group)

{

Console.WriteLine(pl.GetDetails());

}

DAY 11 INHERATINCE LECTURE

//BANKS

using Day\_11\_Inheritance\_Lecture;

BankAccount bank = new BankAccount(1, 1000, "Maria", "Farmington Hills");

Console.WriteLine();

Console.WriteLine(bank.Balance);

bank.Deposit(100);

Console.WriteLine(bank.Balance);

bank.WithDrawl(1500);

Console.WriteLine(bank.Balance); //uses the basic "BankAccount" class

//SAVINGS ACCOUNT

SavingsAccount saving = new SavingsAccount(2, 500, "Maria", "Farmington Hills", .05m);

Console.WriteLine();

Console.WriteLine(saving.Balance);

saving.Deposit(100);

Console.WriteLine(saving.Balance);

saving.WithDrawl(1500);

Console.WriteLine(saving.Balance);//savings account doesn't allow us to withdraw more than the balance

//CHECKING ACCOUNT

CheckingAccount checking = new CheckingAccount(3, 400, "Maria", "Farmington Hills", 50);

Console.WriteLine();

Console.WriteLine(checking.Balance);

checking.Deposit(100);

Console.WriteLine(checking.Balance);

checking.WithDrawl(1500);

Console.WriteLine(checking.Balance);

//POLYMORPHISM

// take all children by typing in the parent object

Console.WriteLine($"Account number: {bank.AccountNumber}. Need to Audit? {Audit(bank)}");

Console.WriteLine($"Account number: {saving.AccountNumber}. Need to Audit? {Audit(saving)}");

Console.WriteLine($"Account number: {checking.AccountNumber}. Need to Audit? {Audit(checking)}");

//create a function

static bool Audit(BankAccount acc)

{

if (acc.Balance < 0 || acc.AccountNumber <=0)

{

return true;

}

else

{

return false;

}

}

MOVIE DATABASE LAB

MOVIE DATABASE LAB

//Console.WriteLine("enter a number");

//int x = Validator.GetPositiveInputInt();

LIST

Console.WriteLine("Welcome to the Movie Database!");

List<Movie> MoviesList = new List<Movie>()

{

new Movie("Forest Gump", "Romance", 142, 1994, 8.8),

new Movie("The Shawshank Redemption", "Drama", 142, 1994, 9.3),

new Movie("The Godfather", "Drama", 175, 1972, 9.2),

new Movie("Jurassic Park", "Sci-Fi", 127, 1993, 8.2),

new Movie("The Avengers", "Action", 143, 2012, 8),

new Movie("The Lion King", "Romance", 87, 1997, 9.3),

new Movie("The Little Mermaid", "Romance", 83, 1989, 9.2),

new Movie("Avatar", "Action", 262, 2009, 7.9),

new Movie("Don't Look Up", "Sci-Fi", 245, 2021, 7.2),

new Movie("Good Will Hunting", "Drama", 126, 1997, 8.3),

new Movie("Once upon a time in the West", "Western")

};

//search for the movie by category

bool runProgram = true;

while (runProgram)

{   
while (true)

{

//ANOTHER WAY

//-----------------------------------------------------------

//List<string> DistinctCategories = MoviesList.Select(c => c.Category).Distinct().ToList();

////List<Movie> DistinctCategories = MoviesList.GroupBy(d => d.Category.ToLower().Trim()).Select(m => m.First()).ToList();

//for(int i = 0; i<DistinctCategories.Count; i++)

//{

// Console.WriteLine($"{i +1}. {DistinctCategories[i].Category}");

//}

//int menuChoice = -1;

//while (menuChoice <0 || menuChoice >= DistinctCategories.Count)

//{

// menuChoice = Validator.GetPositiveInputInt();

// if(menuChoice <= 0)

// {

// Console.WriteLine("Out of range.");

// }

//}

//string category = DistinctCategories[menuChoice].Category;

//ANOTHER WAY

//-----------------------------------------------------------

//program takes user input

Console.WriteLine($"Select a category: 1. Drama, 2. Romance, 3. Action, 4. Sci-Fi, 5. Western");

int categoryChoice = 0;

//int categoryChoice = Validator.GetPositiveInputInt();

while (int.TryParse(Console.ReadLine(), out categoryChoice) == false || categoryChoice < 1 || categoryChoice > 5)

{

{

Console.WriteLine("Invalid response. Try again"); //error message

}

//display the movies

if (categoryChoice == 1)

{

//put in alphabetical order (.OrderBy(m => m.Title)

List<Movie> displayCateogryChoice = MoviesList.Where(d => d.Category == "Drama").OrderBy(m => m.Title).ToList();

foreach (Movie m in displayCateogryChoice)

{

Console.WriteLine(m.GetDetails());

}

break;

}

else if (categoryChoice == 2)

{

List<Movie> displayCateogryChoice = MoviesList.Where(d => d.Category == "Romance").OrderBy(m => m.Title).ToList();

foreach (Movie m in displayCateogryChoice)

{

Console.WriteLine(m.GetDetails());

}

break;

}

else if (categoryChoice == 3)

{

List<Movie> displayCateogryChoice = MoviesList.Where(d => d.Category == "Action").OrderBy(m => m.Title).ToList();

foreach (Movie m in displayCateogryChoice)

{

Console.WriteLine(m.GetDetails());

}

break;

}

else if (categoryChoice == 4)

{

List<Movie> displayCateogryChoice = MoviesList.Where(d => d.Category == "Sci-Fi").OrderBy(m => m.Title).ToList();

foreach (Movie m in displayCateogryChoice)

{

Console.WriteLine(m.GetDetails());

}

break;

}

else if (categoryChoice == 5)

{

List<Movie> displayCateogryChoice = MoviesList.Where(d => d.Category == "Western").OrderBy(m => m.Title).ToList();

foreach (Movie m in displayCateogryChoice)

{

Console.WriteLine(m.GetDetails());

}

break;

}

else

{

Console.WriteLine("That is not a valid choice. Please try again.");

continue;

}

}

//if ("drama".Contains(categoryChoice))

//{

// List<Movie> displayCategoryChoice = MoviesList.Where(d => d.Category.ToLower().Trim() == categoryChoice).ToList();

// foreach (Movie m in displayCategoryChoice)

// {

// Console.WriteLine(m.GetDetails());

// }

// break;

//else

// {

// Console.WriteLine("That is not a valid choice. Please try again.");

// continue;

// }

//}

runProgram = Validator.GetContinue("Do you want to view another category?");

//while (true)

//{

// Console.WriteLine($"Do you want to view another category? y/n");

// string choice = Console.ReadLine().Trim().ToLower();

// if (choice == "y")

// {

// runProgram = true;

// break;

// }

// else if (choice == "n")

// {

// Console.WriteLine("Bye!");

// runProgram = false;

// break;

// }

// else

// {

// Console.WriteLine("Invalid input. Try again.");

// continue;

// }

//}

}

}

.CS FILE

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Reflection.Emit;

using System.Text;

using System.Threading.Tasks;

namespace Day\_11\_MOVIE\_DATABASE\_Lab

{

internal class Movie

{

//PROPERTIES

public string Title { get; set; }

public string Category { get; set; }

public int RunTime { get; set; }

public int YearReleased { get; set; }

public double Rating { get; set; }

//CONSTRUCTOR

public Movie(string \_title, string \_category)

{

Title = \_title;

Category = \_category;

RunTime = 0;

YearReleased = 0;

Rating = 0;

}

//OVERLOADING

public Movie(string \_title, string \_category, int \_runTime, int \_yearReleased, double \_rating)

{

Title = \_title;

Category = \_category;

RunTime = \_runTime;

YearReleased = \_yearReleased;

Rating = \_rating;

}

//METHODS

public string GetDetails()

{

if(RunTime == 0)

{

return $"{Title}, Genre: {Category}";

}

else

{

return $"{Title}, Genre: {Category}, Run time: {RunTime} minutes, Year Released {YearReleased}, Rating: {Rating}/10";

}

}

}

}

DAY 12 ABSTRACT INTERFACE

.CS PAGE

{

internal class Car

{

//PROPERTIES

public string Type { get; set; }

public int MaxSpeed { get; set; }

public iTurbo Turbo { get; set; }

//CONSTRUCTOR

public Car(string \_type, int \_maxSpeed)

{

Type = \_type;

MaxSpeed = \_maxSpeed;

}

//METHOD

public void Drive()

{

Console.WriteLine($"{Type} drives at {MaxSpeed} miles per hour");

if (Turbo != null ) //when turbo is not assigned

{

Turbo.UseTurbo(MaxSpeed);

}

// METHODS

//any method not overwritten can be accessed

//we are only making a change to the deposit method

public override decimal WithDrawl(decimal amount)

{

//return base.WithDrawl(amount);

if (amount > Balance)

{

Balance -= amount + overDraftFee;

return amount;

}

else

{

return amount;

}

}

}

}

}

using Day\_12\_Abstract\_Interface;

Cat c = new Cat("Gee", 500, 9, "grey", true);

Console.WriteLine(c.GetDetails());

c.Eat();

c.Move();

c.Speak();

Hedgehog h = new Hedgehog("Miss McTickles", 10, 4, "brown", true, 3);

Console.WriteLine(c.GetDetails());

c.Eat();

c.Move();

c.Speak();

//POLYMORPHISM

List<Animal> Zoo = new List<Animal>()

{

new Cat("Boo", 8, 2, "Black", false),

new Hedgehog("Spiky Pricklpants", 2, 8, "black", false, 3),

new Cow("Mooney", 1000, 3, "white", false, true),

new Elephant("Trunkzilla", 4000, 3, "big", "short", 15)

};

Console.WriteLine("My Zoo");

foreach (Animal a in Zoo)

{

Console.WriteLine(a.GetDetails());

}

Console.WriteLine();

//INTERFACES

Car car1 = new Car("Mustang", 200);

car1.Drive();

Car car2 = new Car("Mustang v2", 220);

car2.Turbo = new BasicTurbo();

car2.Drive();

car2.Turbo = new MaxTurbo();

car2.Drive();

USED CAR LOT LAB

Console.WriteLine("Welcome to Grand circus Car Lot.");

List<Car> Allcar = new List<Car>()

{

new Car ("Chevrolet", "Corvette", 2023, 90000),

new Car("Chevrolet", "Blazer", 2023, 50000),

new Car("Ford", "F150", 2023, 80000),

new UsedCar("Ford", "Expedition",2014, 10500.99m,180000),

new UsedCar("Kia", "Soul", 2008, 800,2000),

new UsedCar("Ford", "Mustang", 2009, 8000,50000),

new Car($"chevrolet", "corvette", 2020, 59995m),

new Car($"audi", "A4 premium", 2017, 26000m),

new Car($"lamborghini", "Huracan", 2023, 212090m),

new UsedCar($"Buick", "Regal", 2003, 850m, 150000),

new UsedCar($"Ford","Explorer", 2010, 8017m, 125000),

new UsedCar($"Honda", "accord", 2008, 2200m, 160500)

};

//looping

bool runProgram = true;

while (runProgram)

{

DisplayMenu(Allcar);

int menuChoice = -1;

while (menuChoice <= -1 || menuChoice >= Allcar.Count + 3)

{

Console.WriteLine($"Choose a menu option. 1-{Allcar.Count + 2}");

while (int.TryParse(Console.ReadLine(), out menuChoice) == false)

{

Console.WriteLine("incorrect format");

}

}

//LOGIC for buying Car

if (menuChoice < Allcar.Count+1)

{

Car carChoice = Allcar[menuChoice - 1];

if (BuyCar(carChoice)==true)

{

Allcar.Remove(carChoice);

Console.WriteLine("Our finance team will reach out to you shortly.");

}

else

{

Console.WriteLine("Feel free to keep browsing.");

}

}

//LOGIC for adding Car

else if (menuChoice == Allcar.Count +1)

{

Car newCar = AddCar();

Allcar.Add(newCar);

}

//LOGIC for quitting

else if (menuChoice == Allcar.Count + 2)

{

Console.WriteLine("Bye!");

runProgram = false;

break;

}

}

//METHODS

static bool BuyCar(Car c)

{

Console.WriteLine("Would you like to buy this car? y/n");

Console.WriteLine(c);

string buyChoice = Console.ReadLine();

if (buyChoice == "y")

{

return true;

}

else

{

return false;

}

}

static Car AddCar ()

{

Console.WriteLine("What is the car's make?");

//pretent these are all validated

string make = Console.ReadLine();

Console.WriteLine("What is the car's model?");

string model = Console.ReadLine();

Console.WriteLine("What is the car's year?");

int year = int.Parse(Console.ReadLine());

Console.WriteLine("What is the car's price?");

decimal price = decimal.Parse(Console.ReadLine());

Console.WriteLine("Is this car used? y/n");

string used = Console.ReadLine();

if (used == "y")

{

Console.WriteLine("What is the used car's mileage?");

int mileage = int.Parse(Console.ReadLine());

return new UsedCar(make, model, year, price, mileage);

}

else

{

return new Car(make,model,year, price);

}

}

static void DisplayMenu(List<Car>fakeList)

{

for (int i = 0; i < fakeList.Count; i++)

{

Console.WriteLine($"{i+1}. {fakeList[i]}"); //you need the override ToString for this

}

Console.WriteLine($"{fakeList.Count+ 1}. Add Car");

Console.WriteLine($"{fakeList.Count + 2}. Quit");

}

DAY 13 JULY 26,

FILE IO - INPUT / OUTPUT

// 3 different things required to make a file

//file path

//where file is located

using Day\_13\_File\_IO\_Exercise;

string filePath = "../../../Students.txt";

//check to see if the file doesn't exist

if (File.Exists(filePath) == false) //can make this into a method

{

StreamWriter tempWriter = new StreamWriter(filePath);

tempWriter.WriteLine("Justin Jones|26|10");

tempWriter.WriteLine("Ethan Tomas|21|12");

tempWriter.WriteLine("Maria Ragone|30|11");

tempWriter.Close();

}

//Read the file and fill the list

StreamReader reader = new StreamReader(filePath);

List<Student> allStudents = new List<Student>();

while (true)

{

string line = reader.ReadLine();

if (line == null)

{

break;

}

else

{

//split by: name|age|grade

string[] parts = line.Split('|');

//parts[0] = name

//parts[1} = age

//parts[2] = grade

Student s = new Student(parts[0], int.Parse(parts[1]), int.Parse(parts[2]));

allStudents.Add(s);

}

}

reader.Close();

//display all students, main code

foreach(Student s in allStudents)

{

Console.WriteLine(String.Format("{0,30}: {1,15}, {2,5}", s.Name, s.Grade, s.Age));

}

//add another student (pretend these are validated)

Console.WriteLine("Enter new student's name");

string name = Console.ReadLine();

Console.WriteLine("Enter age");

int age = int.Parse(Console.ReadLine());

Console.WriteLine("Enter grade");

int grade = int.Parse(Console.ReadLine());

Student newStu = new Student(name, age, grade);

allStudents.Add(newStu);

//Update the text file with list updates

StreamWriter writer = new StreamWriter(filePath);//this overwrites

foreach(Student stu in allStudents)

{

writer.WriteLine($"{stu.Name}|{stu.Age}|{stu.Grade}");

}

writer.Close();

July 28 Softskills

The why

Get super specific and intentional

* Differentiating factors\
  + Work well with people
  + Communication
  + Fast learner
  + Experience of working with overseas vendors and teams outside of the company
  + Good at taking constraints and designing a product within those constraints
* Intentional actions -
* Directional awareness – helps you stay on track
  + Long term goals and career planning
* Specific goals

Understand your VIPs values interests Personality and skills

Are you doing something that you love?

That the world needs?

That you are good at?

And that you can be paid for?

Values – I want to make more money, work remote, have a better schedule, flexibility of family, do something that makes me feel happy and fulfilled. Choose 3

Interets – unique ways to combine my background with what I’m learning

Personality – ideal roles in target with your inherant personality

Make sure the work culture fits with your personality (introvert/extrovert)

Skills – need to have the skillset, don’t let the imposter monster thwart success

What is your why.

Go really far back to make an authentic story.

When I was in college I was introduced to a coding class that peaked my interest

I worked on a website project

Be specific – give examples

**These are too overused**

I enjoy problem solving

I’m a creative person

I want to help people

Effective goal setting

1. Thinking and dreaming about what you see yourself doing
   1. I want to make more money
   2. Do something that makes me happy
   3. Have a flexible schedule
   4. Work from home
2. Putting thoughts to paper. Creating S.M.A.R.T. goals
   1. Specific – I want to have a fully remote job by the end of 2023 with a company in the United Stats, that has a flexible schedule, pays at least 95k, with a company that is making something I’m passionate about (insert specific) that spends at least 50% of the time coding.
   2. Measureable - I need to apply to specific number of jobs each week in order to get a job by the end of the year.
   3. Achievable - Yes, I need to work hard to apply to jobs, research companies, and network.
   4. Relevant - It will allow me to be happy with what I spend most of my day on, be better for my family balance.
   5. Time - Jan 1. 2024
3. Taking action

Resume

Education technology

Technical resumes are different

* Focus on all things technical
  + Deemphasize previous work experience that isn’t technical
    - One bullet instead of 5 on all things not technical
  + Focus on the audience of your resume
* Tech skills are more common than core compentencies
* Projects are huge to showcase your tech skills
* Focused on transferabble skill when necessary
* Keyword loaded
* Go only 10 years back
* Easy to read font 10-12pt
* Clear organized content, not flashy
* One page resume

A screenshot of a screen

Description automatically generated

Summary

* Opening line that summarizes your career history so far
* Compelling reason for the career change
* Key technical skills/projects/initiatives you are passionate about
* How you will leverage your skills for the reader
* Write a summary for a career pivoter who just graduated from a codoing bootcamp for this job.

A close-up of a document

Description automatically generated

Technical skills

2-3 columns

C# core

.NET core

SQL Server

Git/GitHub

EF Core

HTML/CSS

OOPTDD

Angular

Bootstrap

DOM

Javascript

A white sheet with black text

Description automatically generated

EDUCATION

Grand circus listed first with certificate title listed first, then the company name

A close up of a certificate

Description automatically generated

PROJECTS – stand alone section! Hyperlink the title

* Project title bolded
* Then skills

A close-up of a project

Description automatically generated

EXPERIENCE

Focus on accomplishments

Think about your prior roles with a different perspective

Metrics and numbers stand out

Xyz formula

Accomplished x as measured by y, by doing z

A screenshot of a web page

Description automatically generated

A close up of a document

Description automatically generated

ADDITIONAL SECTIONS

Volunteering

Certifications

Hobbies/Interests

Due aug 21

Final due sept 12

Notes about the interview:

C# is Strongly typed, regularly updated, good errors, c# is Object oriented progamming

Overriding when you have a method wht a parent and you replace it with your own code.

Overloading 2 methods that have the same name that have different paremeters (2 different constructors)