Revature Week One quiz/QC study guide

What is a Full Stack Developer?

A full stack developer is someone who deals with both the back end of an application (what occurs on the server hidden from the average viewer) as well as the front end(what is seen by the viewer and what a viewer can interact with).

It may be fruitful to consider this in a different field; imagine a restaurant where there are wait staff and chefs. The wait staff and servers would provide a menu to the customers and then pass the orders to the chefs in the kitchen. In this example the backend not seen by the customers would be the chefs and the process of making the food. The front end would be the menu and wait staff the customer interacts with. The waitstaff does everything else in between these two sections of the business which is where a full stack developer would be. Constantly running back and forth between user interactivity and food creation.

What is .NET?

An open source developer platform, created by Microsoft, used for building different applications using C#. .
It is a collection of several programs that:

- Translate your code into computer readable instructions(machine code/assembly)
- A collection of libraries and classes used for building software
- Specify data types that aid your interaction with your code and the computer and that allow for certain information to be passed into your program and saved from.

What is C#

- A type-safe object oriented programming language
    - Type-safe: means that C# recognizes when data type conversions could lead to data loss and restricts those instances
    - Object Oriented: means that C# sees everything as an object or an instance of a class that describes the specific behavior and information of a particular object
- C# is versatile and can be processed on multiple platforms of Operating Systems. And while other programming languages have a specific purpose, C# was made to be general purpose and moldable to the programmer

Command Line

- pwd: Outputs the name of the current working directory

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ pwd
/c/Users/krish/Documents/Revature/Training/Training
```

- ls: Lists all files and directories in the working directory
  - -a: Lists the hidden files and directories (indicated with a . in front of the name) in the working directory

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ ls

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ ls -a
./  ../  .git/
```

- cd: Switches you into the directory you specify

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training
$ cd Kris

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$
```

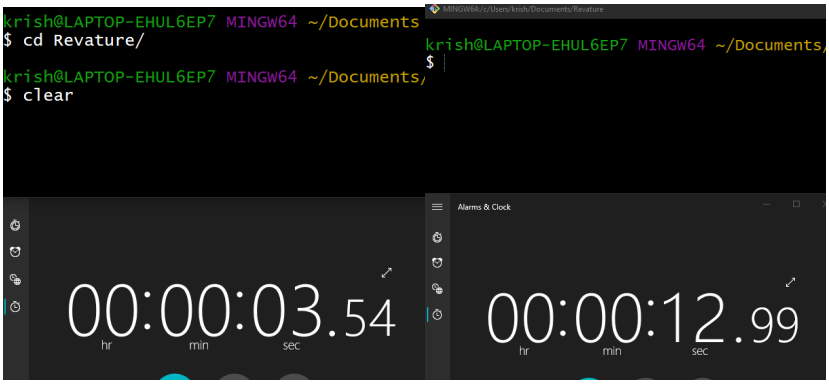
- mkdir: Creates a new directory in the working directory

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ mkdir NewFolder

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ ls
NewFolder/
```

- touch: Creates a new file inside the working directory

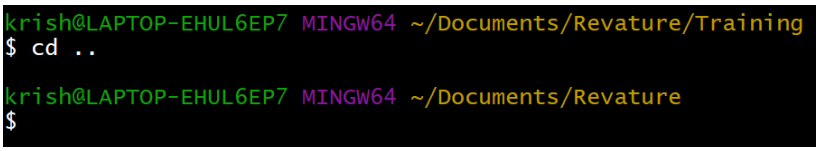
```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ touch NewFile.txt

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ ls
NewFile.txt
```

- clear: Clears the terminal

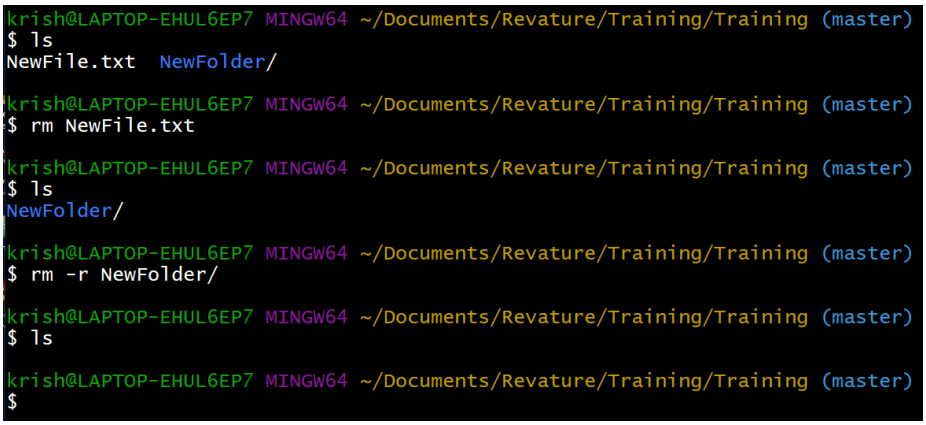


- Showing a timer to attempt to show that I did not just exit the application. I am just slow at taking a screenshot.

- tab: Auto completes the name of a file or directory
- cd ..: Moves up one directory



- rm: Removes the specified file or directory these files and directories will not appear in the recycle bin and are fully removed from the computer
  - -r: recursive flag that bypasses restrictions on removing hidden files



- Echo: Displays entered text on the console
  - >>: places text enter prior to a file specified after

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ echo "This is echoing"
This is echoing

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ echo "This is also echoing" >>Echo.txt

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ ls
Echo.txt

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$
```

- Cat: Displays the contents of one or more files to the terminal

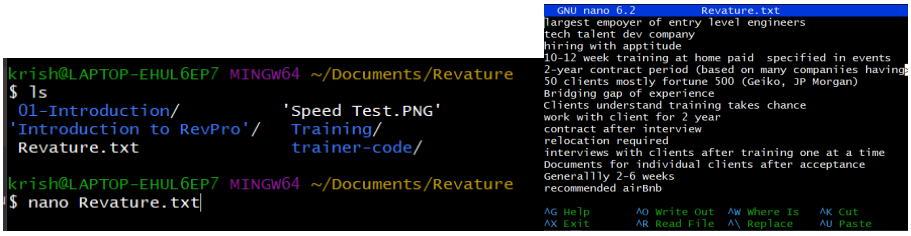
```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ ls
Echo.txt

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ cat Echo.txt
This is also echoing

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$
```

- grep: Searches files for lines that match a pattern and returns the results

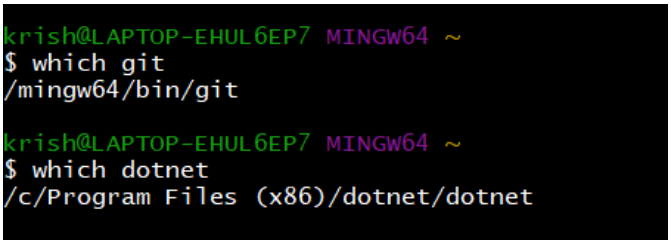
```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature
$ grep -r "welcome" .
Binary file ./trainer-code/.git/index matches
./trainer-code/welcomeToTraining.md:Welcome, I'm excited to welcome you to Revature's training. In the next
12 weeks, we will cover topics pertaining to developing fullstack web applications in .NET Platform. In this
 document, we'll discuss what to expect during the training.
Binary file ./Training/trainer-code/.git/index matches
./Training/trainer-code/welcomeToTraining.md:Welcome, I'm excited to welcome you to Revature's training. In
the next 12 weeks, we will cover topics pertaining to developing fullstack web applications in .NET Platform
. In this document, we'll discuss what to expect during the training.

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature
$ grep -rl "welcome" .
./trainer-code/.git/index
./trainer-code/welcomeToTraining.md
./Training/trainer-code/.git/index
./Training/trainer-code/welcomeToTraining.md

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature
$ grep -r --exclude-dir='.git' "welcome" .
./trainer-code/welcomeToTraining.md:Welcome, I'm excited to welcome you to Revature's training. In the next
12 weeks, we will cover topics pertaining to developing fullstack web applications in .NET Platform. In this
 document, we'll discuss what to expect during the training.
./Training/trainer-code/welcomeToTraining.md:Welcome, I'm excited to welcome you to Revature's training. In
the next 12 weeks, we will cover topics pertaining to developing fullstack web applications in .NET Platform
. In this document, we'll discuss what to expect during the training.
```

  - The above image shows three uses of the grep command. In each case the key word “welcome” is being searched
    - With the -r flag; grep can access all directories and return all directories and text that includes the word “welcome”
    - With the -rl flag; grep will only return the directories of the instances that contain “welcome” farther down my directory
      - By this I mean the folders inside Revature in the above image

    - With the -r –exlude-dir= ‘.git’ flag; grep will do the same task as the -r flag but will ignore all remote repositories.
- nano: Allows to create or edit content of a file
    - If you wanted to add text to a file without going to an application and opening up the folder this neat command can help
    - Luckily, command line has a neat little text editor just for you
    - If you specify a file you which to edit following the keyword nano a nice green, blue and black text-editor will pop up and you can type information in.
        - To save you will need to exit then press ‘y’ then the enter key



- Which: Displays the exact directory path of a specified application
    - This command is very useful in determining whether an application properly downloaded or not.
    - This command will display the EXACT directory path of a specified application.



- Find:
    - This command is very difficult to use but the basic gist is that the command will find any specified word within a specified directory path
        - The reason this is difficult to use is because the command returns every instance of that word in every file along that path
        - Would recommend looking at documentation for this one.https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/find

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~
$ find ./Documents/Revature/Training/Kris "HotCold"
./Documents/Revature/Training/Kris
./Documents/Revature/Training/Kris/.git
./Documents/Revature/Training/Kris/.git/COMMIT_EDITMSG
./Documents/Revature/Training/Kris/.git/config
./Documents/Revature/Training/Kris/.git/description
./Documents/Revature/Training/Kris/.git/FETCH_HEAD
./Documents/Revature/Training/Kris/.git/HEAD
./Documents/Revature/Training/Kris/.git/hooks
./Documents/Revature/Training/Kris/.git/hooks/applypatch-msg.sample
./Documents/Revature/Training/Kris/.git/hooks/commit-msg.sample
./Documents/Revature/Training/Kris/.git/hooks/fsmonitor-watchman.sample
./Documents/Revature/Training/Kris/.git/hooks/post-update.sample
./Documents/Revature/Training/Kris/.git/hooks/pre-applypatch.sample
./Documents/Revature/Training/Kris/.git/hooks/pre-commit.sample
./Documents/Revature/Training/Kris/.git/hooks/pre-merge-commit.sample
./Documents/Revature/Training/Kris/.git/hooks/pre-push.sample
./Documents/Revature/Training/Kris/.git/hooks/pre-rebase.sample
./Documents/Revature/Training/Kris/.git/hooks/pre-receive.sample
./Documents/Revature/Training/Kris/.git/hooks/prepare-commit-msg.sample
./Documents/Revature/Training/Kris/.git/hooks/push-to-checkout.sample
./Documents/Revature/Training/Kris/.git/hooks/update.sample
./Documents/Revature/Training/Kris/.git/index
./Documents/Revature/Training/Kris/.git/info
./Documents/Revature/Training/Kris/.git/info/exclude
./Documents/Revature/Training/Kris/.git/logs
./Documents/Revature/Training/Kris/.git/logs/HEAD
./Documents/Revature/Training/Kris/.git/logs/refs
```

Git

- Initializing a repository:
  - run the command `git init`
  - Initialized repositories are indication is a parenthesized blue text at the end of the directory on the terminal

```
~/Documents/Revature/Training


~/Documents/Revature/Training/Kris (main)
```

  - The above image indicates that the directory of
    - ~Documents/Revature/Training is not an initialized git repository
    - ~Documents/Revature/Training/Kris is an initialized git repository
- Creating a remote repository:
  - There is a longer method for creating a remote repository that requires the previous git command:
    - run the command `git remote add origin <github url of repository>

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training
$ git init
Initialized empty Git repository in C:/Users/krish/Documents/Revature/Training/Training/.git/

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$ git remote add origin https://github.com/220620NET/trainer-code.git

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Training (master)
$
```

        - The above image indicates that in order for a git repository to be made with the `git remote add ` set of commands you must first initialize the current directory to be a git repository
    - There is a shorter method if you simply do `git clone <github url of repository>` with the current directory being a non-initialized repository

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training
$ git clone https://github.com/220620NET/trainer-code.git
```

- Commit
    - When changes to a local git repository have been made it is best practice to add these changes to the github repository
    - It is best practice to write a message to your future self or others who may use the repository to understand what changes have been made

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git add .
warning: LF will be replaced by CRLF in Commands.md.
The file will have its original line endings in your working directory

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git commit -s -m "This file is new"
[main 608316d] This file is new
 1 file changed, 12 insertions(+)
 create mode 100644 Commands.md
```

        - The above image indicates that I added a file “Commands.md” and there were some changes made due to encoding problems with .md and .txt files
        - The next line indicates a message was added that states “This file is new”
            - There are two flags here:
                - -s: indicates a signed commit which takes time to set up and each person needs to figure out what works for their computer for the project
                    - Signing commits tells people viewing your remote repository that you made the changes and that the file is safe
                - -m: indicates a message is being added to the file, this is a required item for commit commands
- Pushing to a remote repository
    - After committing a message to a changed file the next step is to push that file to the remote repository connected to your current directory.
    - This can be done by using the command `git push -u origin` (This command only needs to happen once
        - Also if the local directory your changes are being pushed from are in a cloned directory this is not needed

Commented [1]: These pictures are super helpful!

Commented [2]: I plan on adding pictures for each command line and git statement because examples usually help me.

Commented [3]: I can't do some like tab but I will make a reasonable effort.

```
kris@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git push -u origin
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 510 bytes | 510.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/220620NET/Kris.git
   9b4e667..608316d  main -> main
branch 'main' set up to track 'origin/main'.
```

- Branch
    - Sometimes working with a remote repository and in a group can be difficult in file upkeep.
        - To resolve this we can create branching directories from the main/master branch that way multiple people can be working on different parts of a project at one time.
        - To create a new branch from your local repository use the command `git branch <NameOfBranch>` followed by `git checkout <NameOfBranch>`
        - While in this branch you can work on it all you like as if it were your main directory

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git branch BranchingTest

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git checkout BranchingTest
Switched to branch 'BranchingTest'

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (BranchingTest)
$ echo "I am in BranchingTest now">> BT-README.txt

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (BranchingTest)
$ git add .
warning: LF will be replaced by CRLF in BT-README.txt.
The file will have its original line endings in your working directory

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (BranchingTest)
$ git commit -s -m "Trust me"
[BranchingTest 16d8bb4] Trust me
 1 file changed, 1 insertion(+)
 create mode 100644 BT-README.txt

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (BranchingTest)
$ git push
fatal: The current branch BranchingTest has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin BranchingTest


krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (BranchingTest)
$ git push origin BranchingTest
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 320 bytes | 320.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'BranchingTest' on GitHub by visiting:
remote:      https://github.com/220620NET/Kris/pull/new/BranchingTest
remote:
To https://github.com/220620NET/Kris.git
 * [new branch]      BranchingTest -> BranchingTest
```

- Merge
    - Now that you have been working on a branch you will need to merge the branch with the main branch
        - This is because, while you were working with the new branch the main branch was stuck at a red light waiting its turn. However the main branch can't stay behind for too long so we need to merge the branches together to help the main branch catch up.
    - To do this run the command `git checkout main` followed by the command `git merge <NameOfBranchYouWereJustWorkingIn> main` and then finally push your changes to the remote repository

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (BranchingTest)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git merge BranchingTest main
Updating 5e8e45b..16d8bb4
Fast-forward
 BT-README.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 BT-README.txt

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git push
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/220620NET/Kris.git
   5e8e45b..16d8bb4  main -> main
```

- Push
    - If the local directory your changes are being pushed from are in a cloned directory or you have done the above command once you can instead use `git push`

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git push
Everything up-to-date
```

- Pull
    - Once changes have been made to the remote repository either from someone else working with the repository or if you made a new local repository you should “sync” up your local repository to the remote one.
    - This can be done by using the command `git pull origin` or `git pull`
        - If you want to pull a specific branch use `git pull origin <NameOfBranch>`

```
krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris (main)
$ git pull
Already up to date.
```

C#

- Managed Execution Process Overview
    - This is a process by which programs and code can be executed. There are 4 steps to this process:
        - Choosing a compiler:
            - The compiler in this training curriculum is .NET this compiler will take your code and translate it for the computer to understand.
        - Compiling your code to MSIL
            - .NET mixed with Visual Studio Code will compile your code to MSIL(Microsoft intermediate Language)
        - Compiling MSIL to native code
            - .NET also handles this step which translates the MSIL to verified natve code that the computer an understand and data is protected
        - Running code

- Using the command `dotnet run` while in the directory of your project will allow the .NET compiler to understand the code written and execute the program all in one go.

```
1   // See https://aka.ms/new-console-template for more information
2   Console.WriteLine("Hello, World!");
3   Console.WriteLine("What is your name?");
4   string name = Console.ReadLine();
5   // This is concating the string name to the middle of the Write Line
6   Console.WriteLine("Hello "+ name+ "!");
7   // This is using Lambda Notation for implementing the input in the Write Line.
8   Console.WriteLine($"Hello {name}!");
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  .NET INTERACTIVE  JUPYTER

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris/StudyGuide (main)
$ dotnet run
C:\Users\krish\Documents\Revature\Training\Kris\StudyGuide\Program.cs(6,39): error CS1002: ; exp
de.csproj]

The build failed. Fix the build errors and run again.

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris/StudyGuide (main)
$ dotnet run
C:\Users\krish\Documents\Revature\Training\Kris\StudyGuide\Program.cs(6,39): error CS1002: ; exp
de.csproj]

The build failed. Fix the build errors and run again.

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris/StudyGuide (main)
$ dotnet run
C:\Users\krish\Documents\Revature\Training\Kris\StudyGuide\Program.cs(4,15): warning CS8600: Co
s\krish\Documents\Revature\Training\Kris\StudyGuide\StudyGuide.csproj]
Hello, World!
What is your name?
Kris
Hello Kris!
Hello Kris!
```

- Expressions:
  - An expression in C# can be similar to an expression in math. It requires a variable type, a variable name, and an equivalency.
    - Sometimes the equivalency can be calling a method like in line 2 of the above code, “Hello, World” is an expression
- Statements:
  - Statements differ from expressions because the a statement is a full line of code so in the above image the whole of `Console.WriteLine(“Hello, World);` would be a statement while “Hello, World” would just be an expression

- Variables:
    - Think of variables like how they exist in math problems. Variables give a name to a certain value to shorten the length of statement. In the above code snippet line 4 creates a string variable called “name” and sets it equal to a user input. This allows the next two lines to use the user input.
- Common Built-in Data Types:
    - C# sharp comes with Built-in data types for the programmers convenience. Some common data types and their use are:
        - String
            - This data type allows for whole words or phrases to be passed into and through code. Must be surrounded by “”
        - Int
            - This data type allows for whole number (without decimals) to either be positive or negative
        - Char
            - This data type only allows for a single letter to be passed into and through code. Must be surrounded by ‘’
        - Double
            - This data type allows for numbers to have decimal places and is part and parcel to the notion of floating-point numbers in computer programming
            - Floating-Point Numbers simply mean a real number that can exist in a specified number of bits
        - Bool
            - This data type allows for the programmer to make a variable true or false. This True/False data type closely resembles binary methodology being implemented into the code and is best used in cases where a program can end or must continue.
- Basic Operators:
    - Arithmetic
        - +: addition
            - advance an int, char, or double variable by 1
            - can be used to add two or more int or double variables together
        - -: subtraction
            - Decrement an int, char, or double variable by 1
            - Can be used to subtract two or more int or double variables together
        - *: Multiplication
            - Multiply two or more int or double variables together
        - /: Division
            - Divide one int from another int variable
            - Divide one double from another double variable
            - Returns a whole number
        - %: Modulus

                - Returns the remainder of a division
        - Equality
            - =: Expression equivalency
                - Sets a variable to be a certain value
            - ==: Comparison
                - Compares a variable to another variable or expression
                    - If the variable matches to the second variable or expression returns true
            - Comparision:
                - >: Greater than
                - <: Less than
                - >=: Greater than or equal to
                - <=: Less than or equal to
- Input:
    - Sometimes when creating an application, it is desired for a variable to be adjusted by the user or for a simple game, user input is needed
    - To do this in a basic console program you can use `Console.ReadLine();`
        - This will take the next user input in the terminal as a string variable
- Output:
    - Sometime when creating an application, it is desirable to check if you have made the code properly and it can output the required information
    - This can be easily done by implementing `Console.WriteLine(<expression or variable>);` as done in the above snippet
- Comments:
    - When writing code, it is best practice to write describe what a particular line of code is doing or what the goal of a program or section is. There are two way to implement comments
        - //single line comment
        - */ Multiline comment /*

```
/*
    Hot Or Cold Game
    10 tries

    1. Generate a random number
    2. Ask the user for a number
    3. see if random number if above or below user
        a. use terms hot if near (within 5)
        b. use term warm (within 10)
        c. use term luke warm( within 15)
        d. use term cold  (within 20)
        e. use term freezing (within 25)
    4. add history for distance
        a. hotter if getting closer to
        b. colder if getting farther away
        c. use a hint system if they are getting colder twice in a row.
            i. above or below a certain number (only on the frst hint and if they have guessed in this range)
            ii. even or odd
            ii. if the two numbers that the user has put in recently share a modulo with the answer

*/
int[] leader = new int[5];

Beginning:
//Getting information from background commputer and user

bool play = true;
```

- Selection and Iteration:
    - When programming you will often come across an instance where code need to separate into different paths or continue to function while a given condition is true.
    - We can perform these tasks using the following selection statements and looping statements:
        - if statements
            - This is our key selection statement as you can pass into this statement after meeting a condition specified
            - The syntax for this statement is if(condition){<code>}
            - We can also create an else if for cases where we want to check multiple conditions and only do the first one that is met the syntax is as follows
                - if(condition 1){<code>}else if(condition 2){<code>}
            - We can finally create an else statement that will process when the previous if and else if statements have all not met the condition.
                - if(condition 1){<code>}else if(condition 2){<code>}else{<code>}

```
if(dist == 0){
    return "Correct";
}if(dist<5){
    return "Hot";
}else if(dist<10){
    return "Warm";
}else if(dist<15){
    return "Luke warm";
}else if(dist<25){
    return "Cold";
}else{
    return "Freezing";
}
```

- for statements
    - If we know that a certain loop need to last a specific number of times the for can help. The syntax for a for loop is as follows
        - for(<intialize>;<condition>;<increment/decrement>){<code>}
            - The initialize portion is where you would create an int or char and set it to a specified number or character
            - The condition would be specifying when want the loop to stop in relation to the initialized variable
            - The increment/decrement is exactly as it means. This will be how the initialized variable changes every loop

```
for(int i=0; i<5;i++){
    if(k[i]==0){
        k[i] = tries;
    }else if(k[i]>tries){
        k[i] =tries;
    }
}
```

- while statements
    - When we come to a point in the code that need loop an unspecified amount of times or until a value returns true or false a while loop can help
        - When the code comes to a specific while it will run the enclosed code while the condition returns false. If the condition is met when the code come to that condition the enclosed code will never be done
        - The syntax for this loop is
            - while(condition){<code>}

```
while (play){
    Console.Clear();
    Random num = new Random();
    int answer = num.Next(100);
    int tries = 12;
    int history= 0;
    int current =0;
    string movement ="_";
    string cu = "_";
    string p = "";
    if (leader[0] != 0){
        Console.WriteLine("The fastest game lasted "+ leader[0]+" turns. Can you beat that?");
    }

    while(tries>0 && cu != "Correct"){
    Round(tries);
    current = int.Parse(Console.ReadLine());
    // Need to check if number is near far or correct
    Console.WriteLine(HotCold(current,answer));
    cu = HotCold(current, answer);
    if (cu == "Correct"){
        Console.WriteLine("You won");

    }else{
        movement = Change(history,current,answer);
        Console.WriteLine(movement);
    }
    history = current;
    tries --;
    }
    Console.WriteLine("Would you like to continue playing?");
    p = Console.ReadLine();
    if(p =="y"){
        play = true;
        leader = LeaderBoard(tries);
        goto Beginning;
    }else{
        play = false;
    }
}
```

- do while statements
    - Similar to while except the enclosed code is guaranteed to process once before a condition is reached.
    - The syntax for this loop is
        - do{<code>}while(condition)
- switch statements
    - Switch statements are like a single block if statement where instead of having a condition you switch through all possible conditions of a variable and do a particular action
    - The syntax of the switch statement is switch(variable){Case A: <code> Case B: <Code> Default:<Code>}

```
switch(x){
    case 1:
        // code block
        Console.WriteLine("I implore you to either type 1 or 2 and not be adventurous");
        break;
    case 2:
        // code block
        Console.WriteLine("Ok... haha... funny... now please enter a 1 or 2.");
        break;
    case 3:
        Console.WriteLine("Second to last warning before you start losing tries. Type 1 or 2!");
        break;
    case 4:
        Console.WriteLine("Last warning before you start losing tries. Type 1 or 2!");
        break;
    default:
        Console.WriteLine("I warned you now you have "+ t-- +" tries.");
        break;
}
```

- Break and Continue:
    - Sometimes when a particular condition is met during a loop or selection you want to either exit the situation or skip over the next segment of code
        - Using the break; method will cause the loop or selection statement to be exited without further execution
        - Using the continue; method will cause the loop or selection statements code to be skipped and the overarching code to be continued upon.
- Conditional Operators:
    - &&: AND
        - Determines if two separate expressions are true if not returns false
    - ||: OR
        - Determines if one of two separate expressions are true if neither are true returns false otherwise return true
    - ^^: XOR
        - Determines if only one of two separate expressions are true, if both are true or false returns false otherwise returns true
    - ?: : ternary conditional operator
        - One line boolean expression that creates a unique expressional output for a condition being met or not
            - The syntax for this is `condition? Consequent:alternative`
- Basic Type Conversions:
    - When dealing with different data types, there sometimes comes a time where a data type conflict can happen.
    - To fix this C# has built in methods to convert certain data types into another. There are three methods for this
        - Implicit type casting:
            - This is done automatically when passing a smaller size type to a large one

- Explicit type casting
  - Must be done manually by surrounding the new type in parentheses before the variable
- Type conversion methods:
  - There is a built in method for almost every type to be converted using the Convert() class that is built in to C#.

```
10  int myInt = 8;
11  double myDouble = 8.1;
12  bool myBool = false;
13
14  //Implicit type casting
15  double x = myInt;
16  Console.WriteLine($"Implicit type casting an int into a double gives {x}");
17
18  // Explicit type casting
19  int k = (int) myDouble;
20  double y = (double) myInt;
21  Console.WriteLine($"Explicitly type casting {myDouble} into int gives {k}");
22  Console.WriteLine($"Explicitly type casting {myInt} into int gives {y}");
23  // Type Conversion
24  Console.WriteLine($"Using string conversion of an int gives {Convert.ToString(myInt)}");
25  Console.WriteLine($"Using double conversion of an int gives {Convert.ToDouble(myInt)}");
26  Console.WriteLine($"Using int conversion of a double gives {Convert.ToInt32(myDouble)}");
27  Console.WriteLine($"Using string conversion of a boolean gives {Convert.ToString(myBool)}");
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  .NET INTERACTIVE  JUPYTER

krish@LAPTOP-EHUL6EP7 MINGW64 ~/Documents/Revature/Training/Kris/StudyGuide (main)
$ dotnet run
Implicit type casting an int into a double gives 8
Explicitly type casting 8.1 into int gives 8
Explicitly type casting 8 into int gives 8
Using string conversion of an int gives 8
Using double conversion of an int gives 8
Using int conversion of a double gives 8
Using string conversion of a boolean gives False
```

- Static Methods:
  -
- Code Reuse:
- Objects:
- Classes:
- OOP:
- Instance Methods:
- Coding Conventions:
- Strings:
- Arrays:
- The Debugger:
- Building and Publishing Apps:
- Namespace:

- Using Directories:
- Fields:
- Public Access:
- Private Access:
- Projects:(I think this is just what is included in a project?)
- Properties:
- Other data type examples:
- Object Initializers:
- The Main Method:
- Constructors:
- Encapsulation:
- Value Types:
- Reference Types:
- The Stack:
- The Heap:
- Method Overloading:
- Exceptions: (And everything that goes with, throw catch, finally, custom exceptions)
- Interfaces
- Project References:
- Internal Access:
- Solutions:

Commented [4]: Was this covered because that is for next weeks agenda on Tuesday?

Commented [5]: She went over those on Thursday and Friday. I think they are going to be on there. I would study them as well.

Commented [6]: Thanks as well. This is so helpful