**C# INTRODUCTION**

GETTING STARTED

**Your first C# Program**

\* define a variable called name

\* set the variable to your own name

\* print that variable to the console

**Doing Math**

\* Use two variables, 10 and 3

\* Perform each of the arithmetic operations on them (+ - \* / %)

\* You might be surprised by the quotient (if you’re not getting the expected answer, try declaring them as ‘double’s instead of ‘int’s.

OPERATORS AND STRINGS

**User Input**

\* ask the user for their name (ie: print the question to the console)

\* save the answer to a variable

\* greet the user by name (ie: Hello …)

DECISION MAKING

**The Else If statement**

\* Ask the user who they are (think of 3 possibilities of who the user might be)

\* Ask a question to the user depending on who the user is

\* Print something if you don’t recognise the user (ie: who is not one of those 3 possibilities)

**Logical Operators**

\* You are a bouncer, and will only admit people who -

\* Look over 25, OR

\* Have id AND are over 18.

There are 8 combinations, which you can check by asking the user for and providing for the following information. (This could also be achieved using nested loops, but don’t worry if this seems difficult at this stage)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **HAS ID** | | Looks over 25 | |  | **HAS NO ID** | | Looks over 25 | |
| TRUE | FALSE | TRUE | FALSE |
| Over 18 | TRUE | (1) | (2) | Over 18 | TRUE | (5) | (6) |
| FALSE | (3) | (4) | FALSE | (7) | (8) |

**The switch statement**

\* Do the same as in the “The If Else statement”, but using a switch statement

**The Ternary Operator**

\* Someone is asking where the toilets are. Use the user input to determine their gender.

\* Use the conditional operator to set a Direction variable according to their answer, (left if male and right if female).

\* print the direction they need to go.

LOOPS

**The while loop**

\* Research how to generate a random integer

\* Generate a random integer between 1 and 5

\* Ask the user to guess the number

\* for each wrong guess, prompt the user to try again

\* On a correct guess, congratulate the user.

**Break and Continue (Break)**

\* Create an ATM:

\* Generate a random 4 digit code

\* Give the user 3 attempts to provide the correct code

\* Provide the relevant service if correct within 3 attempts

\* swallow the card if wrong on the third attempt.

**Break and Continue (Continue)**

\* You are admitting children onto a ride with an age restriction

\* iterate through ages 1 – 10 (ie: **for (var i = 1; i <= 10; i++) …** )

\* if heights are less than 7, state that the child is too young, otherwise, continue through to the next child

**C# INTERMEDIATE**

CLASSES AND OBJECTS

In your “ExerciseWork” folder -

create a new class called “Tyranid” (this class will not be static)

also create a new static class called “ClassesAndObjects”, (as you have for previous exercises)

In your Tyranid class add and implement the following members -

private fields

\* \_isAlive

Properties -  
\* Name { get; } // get only properties can still be set by the constructor

\* Health { get; private set; } // private set means this can only be set by methods within the object

\* Strength { get; }

Constructor -

\* Set Name and Strength with constructor parameters

\* Set Health to 100

\* Set \_isAlive to true

Methods -

\* Attack () // attack should be based on the tyranid’s strength. A dead tyranid cannot attack

\* ReceiveDamage (damage) // this should decrease the tyranids health by the damage. If health drops to zero or below, the tryanid dies

\* Sleep (hours) // increases the health by the number of hours slept, capped at 100, unless dead.

\* Status () // this should log the health status of the tyranid to the console using its name

In your ClassesAndObjects class -

Instantiate two tyranids

Have one tyranid attack another tyranid

Use Status() to check that the other tyranid’s health decreases by the expected amount

Allow the injured tyranid to sleep for a number of hours

Use Status() to check that the tyranid’s health has recovered by the expected amount

Have the injured tyranid kill the other tyranid, (you could use a ‘for’ loop to attack)

Use Status() to check that the tyranid is dead.

Try to make the dead tyranid attack().

Try to revive the dead tyranid using Sleep()

Use Status() to confirm that Health is still at zero.

(we can have the tyranids attack each other as part of the next array module)

EXTRA?

Indexing strings

<https://www.w3schools.com/cs/cs_strings_access.php>

String methods (just have a look at the method names and definitions to know what there is).

<https://www.programiz.com/csharp-programming/library/string>

Arrays (there’s more on arrays later)

<https://www.w3schools.com/cs/cs_arrays.php>

Lists (similar to arrays)

<https://www.programiz.com/csharp-programming/list>

**LINQ methods**

Tutorial:

<https://canro91.github.io/2021/01/18/LinqGuide/>

Exercises: (1, 3, 4, and 11 should do it. You’re welcome to do more if you like).

<https://www.w3resource.com/csharp-exercises/linq/index.php>

**Dictionaries – this is mentioned later, but could be useful here.**

Tutorial:

<https://www.programiz.com/csharp-programming/dictionary>

Exercise:

As an employee of the RSPB, your boss asked you do count the following birds over the period of an hour.

\* robin

\* starling

\* blackbird

\* sparrow

1. Create an array of these birds called ‘birdTypes’

2. Create an empty List called ‘birdsSeen’

3. Create a Dictionary called ‘birdCount’, whose key will be the bird type and value will be the count for that bird type, (What would be an appropriate starting value might be for each bird?)

4. Think of a way that you can populate the birdsSeen List with at least 20 birds, randomly using the birdTypes array.

5. Use a Linq method to change each of the birds in birdsSeen to start with a capital letter

6. Iterate over the birdsSeen List using a **foreach** loop to populate your birdCount Dictionary

7. Create a report to show how many of each bird there are, (you’ll need to iterate through the birds)

8. New data has come in from another report with 4 more blackbirds. Add this to your current count

9. Check that blackbirds have been updated as expected

10. Make a final report with your new data

EXERCISM

You may want to call your method multiple times with different parameters to check all cases.

1. Leap

2. Two fer

3. Space age

4. Grains

5. Perfect Numbers

6. Collatz Conjecture

7. Difference of Squares

8. Armstrong numbers (possible, but hard at this point)

9. Darts (you probably already know that Pythagoras’ theorem will come in useful here)

10. Triangle

11. Raindrops

12. Beer song

13. Sieve