Grade 11 Assignment #2 Java Script

Dog Years

Let's practice the Java syntax you just learned.

Purina is always striving to find new and innovative ways to enrich the lives of pets and the people who love them. Recently, a New Idea that came across Purina's Think Tank's Team, to help pet owners understand the life cycle of their pets in order to better care for them. One way this can be achieved is to create a Pet Year (Converter) Calculator. An owner will be able to enter the age of their pet in Human Years into the converter which will return the actual age of their pet in Pet Years. You and your team has been hired to build a program that is geared towards the needs of Dogs, specifically focusing on the task of creating a Dog Year Converter.

There is a lot of information available online about important Dog characteristics like: Loyalty, Intelligence, Varieties, Breed, Weight, Obedience, Strength, Nutrition, Size, Life Cycles and even working Dog Year Calculators. With respect to a dog's age, dogs mature at a faster rate than human beings. We often say a dog's age can be calculated in "dog years" to account for their growth compared to a human of the same age. In some ways we could say, time moves quickly for dogs — 8 years in a human's life equates to 45 years in a dog's life. If this is true, how old would you be if you were a dog?

Here's how you convert your age from "human years" to "dog years":

- The first two years of a dog's life count as 10.5 dog years each.
- Each year following equates to 4 dog years.

Before you start doing the math in your head, breakdown this conversion and generate a formula to use, then let a computer take care of the rest!

This project requires an understanding of variables, data types, and arithmetic operators in order to do some math. Specifically, unit conversions using basic formulas on the output (or any integer you choose) from the thermometer.

The instructions provided are general guidelines. Upon completion of the project, feel free to explore the code on your own.

Tasks:

There are **22** Tasks to complete for this assignment:

← Part I: Setting Up the Project →

1. Set-Up Home Drive & Folder System

Before you start, create a system of folders to keep track of your projects on your Home Drive **H:\**.

- Project Folder called something like: Assignment2.
- Contained within a Master Folder called something like JavaScript, Grade11, Programming.

NOTE: Keep all the files associated with the program within the same folder "Assignment2".

2. Download (Files) Content

Download the starter files I created for you off the Assignment Tab on my website (NVSD Portal) and store them in the appropriate folder on your Home Drive H:\.

- JS_DYC.js
- Style_DYC.css
- Index_DYC.html

NOTE: In JavaScript, the index.html file generally represents the point of entry or starting point to launch a project and run all its files. Generally, this index.html file will call all the other files (.html, .jc, .css, .php, etc…) necessary for the program to run.

NOTE: You do not have to use my files, you can create your own from scratch. Just take a moment to look over my files, notice how each file type is set up.

3. Study (Look Over) My Starter Files

Using your IDE, **Open** my starter files. Take a moment to look them over, there are several (signatures) parts you should become familiar with.

NOTE: The only file you are responsible to complete for this assignment is the DogYearConverter.js file, the others are written for you. Any work you do to these other files will be added to your final mark (bonus marks).

← Part II: Project Code →

4. Project Premise

We Write Code → to Perform Tasks

Your Code → Dog Year (Converter) Calculator

This task is based on converting a Dog's Age in Human Years to Dog Years in order to better understand a Dog's Life Cycle

In the main part of the code, you are going to convert a person's age into doggy years with the use of an equation. The calculation for dog years is based on the idea that:

- The first two human years of a dog's life count as 10.5 dog years each.
- Each human year following counts as 4 dog years.

For example: Let's say your age today is approximately 16 years old, which means your Dog is around 77 years old (Dog Years refers to the age of something a wrt a Dog's Natural Life Cycle), which hinges on using a set of equations to perform those conversions. Determine the set equations used in this calculation based on the above information.

This program should be capable of changing **any** inputted number (an age given in Human Years) to a correct value (an age in Dog Years). This project will require a familiarity of variables, data types, and arithmetic operators at the least.

5. Top-Down (Incremental / Iterative) Design Philosophy

Once the project premise is understood, the next thing to do before you start coding, is to break the project up into smaller, more manageable parts, otherwise you can get overwhelmed and the project can spiral out of control.

Based on the above information, for this project there are 4 steps, the program at each step should be capable of:

a) Determining the set of equations required in the conversion
b) Performing a conversion and storing the result
c) Correctly converting a set of static test values (see the table below)
d) Converting any input provided by a user

This way we know the program is working properly at each stage of development, which reduces the number of bugs (time debugging) caused by specific tools like the actual thermometer and coding elements/ constructs.

6. Start (Project) Coding

Using your IDE, open your .js file to create the code needed for the program called Dog Years. Make sure to save this file in your ”Assignment2” folder.

NOTE: Must submit a separate JavaScript file, do not attempt to submit a combined file type.

7. Variable Declaration

To start, you need to create some variables to store and manipulate the data (AKA: the ages) used in your program. You will need at least 4 variables, one for each value in the conversion formula: myAge, firstYears, laterYearstogether, and myAgeInDogYears.

NOTE: Generally, there should be one variable for each calculation (intermediate value) performed and answer (final value) reported to the user

8. Instantiating a Variable (Declare and Assign)

myAge is the primary input for this program. It represents the variable used to store the age of a Dog in Human Years. In other words, how long have you had the dog for?

Initially, we are going to use a static value in order to setup the conversions and practice storing and retrieving the result.

Eventually, a user will be able to enter a different value for the myAge variable. It is important for a person to be able to change the value being inputted as different people will be using the program with each person having a different dog of a different age.

Therefore, set the value of your variable named myAge equal to your age as a number. Say 16.

9. First-Years Calculation (First of 2 Part Conversion)

Let's start with the first part of the age conversion. The firstYear variable should perform the calculation and store the intermediary result based on:

- Take the first 2 human years from myAge, and multiply it by 10.5.
- Then store this in a variable called firstYears.

This means that the first 2 years of a Dog's life ages differently than its later years. Moreover, it states that a dog ages much more quickly in its formidable years than a human. It also infers that if myAge is greater than 2, then both of those years can be used in this part of the calculation.

Conversion Formula:

```
firstYesrs  =  ??? * ??? + ??? / ??? - ???
```

NOTE: Never try to use a complicated formula, break it down into smaller parts, this makes it easier to track down any mistakes/ errors

10. Later-Years Calculation (Second of 2 Part Conversion)

The laterYears variable should perform the second part of the calculation and store its intermediary result. The laterYears calculation is based on:

- Each human year following the initial 2 years counts as 4 dog years.

Since we already accounted for the first two years in the first part of the conversion, we need to determine the remaining years from myAge. Thus, if myAge > 2, then take the myAge variable, and subtract 2 from it.

According to the description given above, each of these laterYears are affected differently. The development of a dog slows later on in its life as each year is now affected by a factor of 4. Therefore, we need to multiply these laterYears by 4.

For the conversion to laterYears, determine the appropriate equation. Remember to store this value in the variable named laterYears.

Conversion Formula:

```
laterYears  =  ??? * ??? + ??? / ??? - ???
```

NOTE: By storing values in variables we are able to access and use them later in other parts of the program, like when reporting results.

11. My Age in Dog Years Conversion

The total conversion of human years into dog years is just the sum of the first and later years equations.

Thus, Add firstYears and laterYears together, and store that final result in a variable named myAgeInDogYears. This value will later be reported out to the user.

12. Static Value Testing

As of now, when you run your program, it should work for the initial value of 16 that you gave to your myAge variable. If myAge = 16 → myAgeInDogYears = 77. Next, we need to make sure the program works for other possible (likely outcomes) values.

To do this, test out a few hypothetical Dog Age values (denoted by a number given in Human Years). Some values are given in the table below. Just change the value of your static variable myAge and run the program to see the results for each new static value.

A Dogs Age (Testing Values: Human Years to Dog Years)				
myAge (Human Years)	**firstYears**	**laterYears**	**myAgeInDogYears** (Dog Years = Dog Life Cycle)	**Reason**
2	**21**	**0**	**21**	How do you know what values to use in order to rigorously test the program Why did I choose these values??? What do I want you to notice???
6				
15				
16	**21**	**56**	**77**	
17				
18				
33	**21**	**124**	**145**	
50				
-3				
0	**0**	**0**	**0**	
1				
500				

NOTE: You need to create a set of values for each program you write. FYI: I have and use a set of test values when marking your programs.

NOTE: Need help checking your answers, see **Part III: Project Testing** below for advice.

13. Improve (Expand / Modify) Upon the Basic Program

Congratulations, now that you know the program (AKA: your code) works, you can move onto the next step → improve the program → adding features & functionality → adding complexity to the code.

a) Ideas 4 Basic Expansion (Improvement thru Coding Elements)

In this course, you need to get used to the idea of improving, expanding and modifying the basic code that is provided for you. You need to take some time to think about how to add complexity to your code. Basically, taking what you got and making it better!!!!!

As of now, the program is based upon the fixed /static values given to the variable myAge. Make it so that this primary input of your program (myAge Variable) is more dynamic (can be changed on demand). Here are some ideas:

- Allow a user to manually input data (change the value on the fly) using a prompt() or an alert().
- Personalize the program, allow the user to input some additional information about their dog, like their name. Make sure to store this information in a variable like myName. Also, you will need to include this information in the user interface (output).
- Try out some string methods. String methods are designed to improve the user interface like with .toLowerCase() or .toUpperCase(). The toLowerCase method returns a string with all lowercase letters.
- See if you can randomize an inputted value, by looking up math.random().
- See if you can enhance the output, make it more meaningful by rounding the value to fewer (more appropriate) decimal places. Do this by checking out math.round(), math.floor(), math.ceil().

b) Ideas 4 Brainstorming (Adding Features thru New Original Concepts)

Another way that you can improve, expand and modify the basic program provided for you is to think about adding different features and functionalities that do not exist yet. To do this you will need to take some time and Brainstorm some Original Ideas.

- **Current Client**
- One way to do this is to think about your client and their needs
- To start, lets think about your client, Purina. What could you add to this program to make them happier? What does Purina sell? What services does Purina offer its customers? What other features or functionalities could you add to your program? Look online to get some ideas!!!!!

For example, besides the age of an animal given in human years, animals come in different animal breeds, have different features (like size and weight), as well as there are other types of cycles (calculations or conversions not based on first and later) to consider.

Also, you could add enhance the information given to a customer, perhaps some facts about nutrition, climatization, breeding or country of origin just to list a few.

- **New/ Different Client**
- A second way to do this is to think about repurposing the program. How could you modify this program and sell it to another company/ client?
- Perhaps, one way to do this would be to add a different type of conversion that would be beneficial to someone in particular.

For example, veterinarians need to calculate the correct dose (amount of medication) to administer to an animal based on its age or size. These Age or Size readings can be converted into milligrams of medication for these veterinarians.

!!!!!Remember, Feel Free to Add or Modify Whatever you Like!!!!!

← Part III: Project Testing →

The Console or Terminal Window is a coders best friend, it represents the front line and is primarily used for debugging. The easiest way to test out a program is to log statements to the console. In JavaScript, this is done using console.log();.

14. Console.log();

Now you are ready to use some console.log() statements to print-out all the information associated with a Dog's Life Cycle (Human Years & Doggy Years) to the screen for the user to see (myAge, firstYears, laterYearstogether, and myAgeInDogYears).

Make sure to include some text so the user knows what is being printed and why. A bunch of numbers without context (AKA: an explanation) is meaningless. This is done in one of three ways: String Interpolation, String Concatenation, String Literals

Furthermore, make sure to format the text so it looks nice. You may need to add some blank spaces and/or lines between the different messages so it is easier to read.

NOTE: console.log(); should be used regularly to check out your progress, otherwise you will have a hard time tracking down the errors / figuring out how to fix it. You should use a console.log(); for every section of code written and/or whenever something is not working properly

NOTE: Comments can be used to omit sections of code, this helps to narrow down the root of the problem (AKA: target the error).

← Part IV: User Interface (Console: Terminal Window) →

There are 2 categories for User Interfaces (UI): Console/Terminal vs. GUI. The type of user interface chosen for a particular program depends on its application and users.

For example, programs written for Industry (like manufacturing) where the End-User is an employee (not a paying customer) is generally written using the Console or Terminal Windows. Basically, companies generally are not willing to spend the extra money on beauty modifications for practical applications. Console/Terminal UI's are Text Based.

In Contrary, programs written for Retail, where End-Users are paying customers, are generally written using a GUI (Graphical User Interface) developed for a Web-Browser (Internet Based) or an App (IOS, Android). In this course you will learn to create both types of User Interfaces.

Regardless of the type of interface that a client is paying for. The development of a User Interface is of utmost importance when creating a program and requires a completely different skillset than that needed to write the actual code that performs the task and makes the program work. Just remember, **a program (or answer) without context is meaningless**. For this section, your User Interface will be built with Console.log(x) & alert() and will need the following **TEXT Based** attributes:

15. Building an Interface

- Welcome MSG (including Program Title)
- Program Introduction, talk about the goal? what to do? how to use? what to expect?
- Reiterate User Input (Confirm the Values Entered by the User)
- Program Explanation (Code Transformation), explain a little about how the program works and why the answer is correct.
- Program Output (Display what the User wants)
- Thank-You (Good-Bye, Come Back Soon) MSG

16. Visuals

- Overall Presentation? What does the interface look like?
- Formatting? Is it easy to read / follow as opposed to cluttered and unorganized?
- Whitespace? Is it cramped?
- Dialogue? Are the words well chosen, concise and meaningful?

← Part V: Formatting →

17. Code Organization (Distinct Blocks of Code)

As you begin to look at more coding examples you will start to see some patterns emerge. WRT (With Respect To) organization, you are required to organize your code into distinct blocks

- You should not see similar lines of code all over the place. For example, all your variables should be declared in the same place. The same goes for your equations, function, console statements, etc....whatever!!!!
- Basically, a distinct block of code should only contain either:
  - 1) the same type of code (similar statements or constructs)
  - 2) the code necessary for a particular task to be completed.

18. Add Whitespace

Whitespace makes reading code easier

- Add 4-5 lines of whitespace between distinct blocks of code
- Add 1-2 lines of whitespace between similar code (contained within an block)

19. Add Comments

Comments are Required. It is important (helpful) to describe to other developers what this small JavaScript program does. Comments should be concise (short and to the point) and used to describe what **each** part of the program does. There are 2 types:

a. Use **multi-line** comments to (/* comment in the middle of */):
b. Use a **single line** comment to (//then comment):

a) Multi-Line Comments:

Generally, multi-line comments (/* comment in the middle of */) are used to identify a unique Signature (i.e: special part of the program). Some IDE's start you off with a shell for these comments, but they will need to be modified as follows:

- An overall Program Comment is located at the top of your file (first line of code), it requires the following:

```
/*
 * Title:   XXX-XXX-XXX
 * Summary:   XXX-XXX-XXX
 * Program Element List:   XXX-XXX-XXX

 */
```

- The End of the Program needs a comment for any Test Code or Notes (even if it is blank the frame is required).
    - NOTES: Anything deemed important that another programmer should know about, like weird bugs or irregular outcomes. Perhaps future upgrades or work in progress.
    - Test Code: The values of any input that create unexpected or weird results. Any values or things that need to be monitored

```
/*
 * NOTES:   XXX-XXX-XXX
 *
 *
 * Test Code:   XXX-XXX-XXX
 *
 *

 */
```

- Other signature comments belong to unique programming features (like above a function or method). These comments generally only need a quick summary, which usually involves the parameters / return values used.

  NOTE: The comment below is designed for a function.

  ```
  /*
   * Summary:   XXX-XXX-XXX
   * Parameters:   XXX-XXX-XXX
   * Return:   XXX-XXX-XXX
   */
  ```

b) Single Line Comments:

Use a single line comment (//then comment) for the rest:

- Place 1 comment above every Distinct Block of Code
- Every distinct block of code requires AT LEAST one comment on the side, describing the most important aspect of that block
- Some examples of commonly used single line comments are:
    i. Identify specific values of inputs, constants, variables, or original numbers of importance.
    ii. Identify the variable, primary input, primary output or calculation section (blocks) of code
    iii. Identify the end of or last line of code in a construct (anything that ends with a curly bracket }). Write "} End of BLAH-BLAH function" after every curly brace.

The following single line comments are required, I will be looking for these specifically in every program submitted and will be deducting marks for each one left out:

```
// Beginning of Program
// End of Program

// Beginning of Main
// End of Main

// Variable Declarations
// Function Declarations
// User Interface

// Primary Input
// Primary Output
// Primary Calculations
```

← Part VI: User Interface (HTML, CSS w/ JS) → OPTIONAL (BONUS Marks)

A GUI (Graphical User Interface) type interface is used when the End-User is a paying customer, when a company is worried about its Perception & Image (Branding). In this case a company is more willing to spend money on the overall User Interface (UI) and User Experience (UX) to ensure that their customers are satisfied. HTML and CSS represent the basic GUI used with JavaScript.

20. HTML & CSS

Now that you have an understanding of the assignment, create a Browser based UI for your project. A Browser based UI is one type of GUI (Graphical User Interface) that can be used with JavaScript.

1) To help you out, a shell .html & .css have been provided for you on the assignment tab of my website. These shell documents are working executable files.

2) Also, to help you out, some HTML templates have been provided for you on the Shared Documents Tab of the Portal (Class Website). Look at the templates, run them and see what kind of elements you would like to use in your Browser based GUI.

NOTE: Use these templates as you wish. Just copy and paste any elements you like. These templates represent a launching point for your project's GUI. They are intended to show you the necessary signatures, constructs (element types), syntax, keywords, formatting and layouts for creating a GUI w/ specific elements.

NOTE: You will be rewarded for ay effort made. Approximately 2.5% for every feature added.

← Part VII: Project Submission →

21. Test-it B4 Submitting-it

Once your program is complete, make sure to test it using the your IDE (do not submit a program that does not work).

- You will lose 30% off the top of your mark by submitting a project that does not compile & run

22. What-2-Submit

Once your program is complete, upload (drag and drop) your files to the portal. Look for your name under the Assignment 2 webpage.

Required:

- There are many files in a project folder. I ***at least*** need the .js files
- Submit the DogYearConverter-JohnVatougios.js file

Bonus Marks:

- Only submit the .html and/or .css files if you created an additional User Interface ***OR*** modified the started files I gave you).
- You will be awarded 2.5% for each feature added