# Haskell Programming Assignment Specification

## Learning Abstract

This is the first assignment of Haskell. It used ghci to access the Haskell and load the Haskell code to use for solving questions below.

### Task 1 - Mindfully Mimicking the Demo

```
PS C:\Users\pogoz\Desktop\CSC344\Haskell\haskell> ghci
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> length [2,3,5,7]
4
>>> words "need more coffee"
["need","more","coffee"]
>>> unwords ["need", "more", "coffee"]
"need more coffee"
>>> reverse "need more coffee"
"eeffoc erom deen"
>>> reverse ["need",

<interactive>:6:18: error:
    parse error (possibly incorrect indentation or mismatched brackets)
>>> reverse ["need", "more", "coffee"]
["coffee","more","need"]
>>> "head ["need", "more", "coffee"]

<interactive>:8:33: error:
    lexical error in string/character literal at end of input
>>> head ["need", "more", "coffee"]
"need"
>>> tail ["need", "more", "coffee"]
["more","coffee"]
>>> last ["need", "more", "coffee"]
"coffee"
>>> init ["need", "more", "coffee"]
["need","more"]
>>> take 7 "need more coffee"
"need mo"
>>> drop 7 "need more coffee"
"re coffee"
>>> ( \x -> length x > 5 ) "Friday"
True
>>> ( \x -> length x > 2 ) "uhoh"
True
>>> ( \x -> length x > 5 ) "uhoh"
False
>>> ( \x -> x /= ' ' ) 'Q'
True
>>> ( \x -> x /= ' ' ) ' '
False
>>> filter ( \x -> x /= ' ' ) "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
>>>
Leaving GHCi.
PS C:\Users\pogoz\Desktop\CSC344\Haskell\haskell>
```

## Task 2 - Numeric Function Definitions

### Function specifications

```
squareArea s = s ^ 2

circleArea c = pi * c ^ 2

blueAreaOfCube b = ( 6 * ( (squareArea b) - ( circleArea b/16)))
paintedCube1 p =  if ( p > 2) then ( 6 * (( p - 2 ) ^ 2 ) ) else 0
paintedCube2 p =  if ( p > 2) then ( 6 * ( 2 * ( p - 2 ) ) ) else 0
```

### The given demo that you are to recreate

```
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load hpa.hs
[1 of 1] Compiling Main             ( hpa.hs, interpreted )
Ok, one module loaded.
>>> squareArea 10
100
>>> squareArea 12
144
>>> circleArea 10
314.1592653589793
>>> circleArea 12
452.3893421169302
>>> blueAreaOfCube 10
482.19027549038276
>>> blueAreaOfCube 22
2333.8009333734526
>>> blueAreaOfCube 1
4.821902754903828
>>> map blueAreaOfCube [1..3]
[4.821902754903828,19.287611019615312,43.39712479413445]
>>> paintedCube1 1
0
>>> paintedCube1 2
0
>>> paintedCube1 3
6
>>> map paintedCube 1 [1..10]

<interactive>:14:5: error:
    ? Variable not in scope: paintedCube :: a1 -> b0
    ? Perhaps you meant one of these:
        ‘paintedCube1’ (line 6), ‘paintedCube2’ (line 7)
>>> map paintedCube1 [1..10]
[0,0,6,24,54,96,150,216,294,384]
>>> paintedCube2 1
0
>>> paintedCube2 2
0
>>> paintedCube2 3
12
>>> map paintedCube2 [1..10]
[0,0,12,24,36,48,60,72,84,96]
>>>
```

---

## Task 3 - Puzzlers

---

This task requires that you write 2 function definitions, and that you then demo them by creating a “proper” demo. For this task, please add to your presentation document (1) a text containing the 2 function definitions, and (2) a text containing the demo that you are asked to create. What is a proper demo with respect to this task? Run each of the 2

functions with the applications that I provide in my sample demo, and then, for each of the 2 functions, add 2 applications of your own invention. Thus, a proper demo will have 4 applications for each of the 2 functions.

## Function specifications

```
squareArea s = s ^ 2

circleArea c = pi * c ^ 2

blueAreaOfCube b = ( 6 * ( (squareArea b) - ( circleArea b/16)))
paintedCube1 p =  if ( p > 2) then ( 6 * (( p - 2 ) ^ 2 ) ) else 0
paintedCube2 p =  if ( p > 2) then ( 6 * ( 2 * ( p - 2 ) ) ) else 0

reverseWords w = unwords ( reverse ( words w ))

averageWordLength s = fromIntegral ( length $ filter ( \x -> x /= ' ') s ) / fromIntegral ( length $ words s)
```

## The given demo that you are to augment

```
PS C:\Users\pogoz\Desktop\CSC344\Haskell\haskell> ghci
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> :load hpa
[1 of 1] Compiling Main             ( hpa.hs, interpreted )
Ok, one module loaded.
ghci>
Leaving GHCi.
PS C:\Users\pogoz\Desktop\CSC344\Haskell\haskell> ghci
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load hpa
[1 of 1] Compiling Main             ( hpa.hs, interpreted )
Ok, one module loaded.
>>> reverseWords "appa and baby yoda are the best"
"best the are yoda baby and appa"
>>> reverseWords "want me some coffee"
"coffee some me want"
>>> reverseWords "desde el alma"
"alma el desde"
>>> reverseWords "esta noche de luna"
"luna de noche esta"
>>> averageWordLength "appa and baby yoda are the best"
3.5714285714285716
>>> "averageWordLength "want me some coffee"

<interactive>:8:41: error:
    lexical error in string/character literal at end of input
>>> "averageWordLength "want me some coffee"

<interactive>:9:41: error:
    lexical error in string/character literal at end of input
>>> averageWordLength "want me some coffee"
4.0
>>> averageWordLength "desde el alma"
3.6666666666666665
>>> averageWordLength "esta noche de luna"
3.75
>>>
Leaving GHCi.
PS C:\Users\pogoz\Desktop\CSC344\Haskell\haskell>
```

---

## Task 4 - Recursive List Processors

---

This task requires that you write 3 recursive function definitions, and that you then demo them by recreating a given demo. For this task, please add to your presentation document (1) a text containing the 3 function definitions, and (2) a text containing a recreation of the demo that I have provided

---

## Function specifications

```
list2set [] = []
list2set ( x: xs ) = if ( elem x xs ) then ( list2set xs ) else( x : list2set xs )

isPalindrome [] = True
isPalindrome ( x : xs ) = if ( length ( x : xs ) ) == 1 then True else (if ( x == last xs ) then ( isPalindrome ( head xs : drop 1 (init xs)) ) else False )

collatz 1 = [1]
collatz n = if ( even n ) then n : collatz x else n : collatz y
    where x = div n 2
          y = 3 * n + 1
```

## The given demo that you are to recreate

```
PS C:\Users\pogoz\Desktop\CSC344\Haskell\haskell> ghci
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load hpa
[1 of 1] Compiling Main             ( hpa.hs, interpreted )
Ok, one module loaded.
>>> list2set [1,2,3,2,3,4,3,4,5]
[1,2,3,4,5]
>>> list2set "need more coffee"
"ndmr cofe"
>>> isPalendrome ["coffee", "latte","coffee"]

<interactive>:5:1: error:
    ? Variable not in scope: isPalendrome :: [String] -> t
    ? Perhaps you meant ‘isPalindrome’ (line 16)
>>> isPalindrome ["coffee", "latte", "coffee"]
True
>>> isPalindrome "racecar"
True
>>> isPalindrome "racecars"
False
>>> collatz 10
[10,5,16,8,4,2,1]
>>> collatz 11
[11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
>>> collatz 100
[100,50,25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
>>>
```

## Task 5 - List Comprehensions

This task requires that you write 2 function definitions by using list comprehensions, and that you then demo them by creating a “proper” demo. For this task, please add to your presentation document (1) a text containing the 2 function definitions, and (2) a text containing the demo that you are asked to create. What is a proper demo with respect to this task? Run each of the 2 functions with the 2 applications that I provide in my sample demo, then add 2 applications of your own invention for each of the functions. Thus, your demo will have 4 applications for each of the 2 functions.

## Function specifications

```
count obj list = length [ x | x <- list, obj == x]

freqTable list = [(x,count x list) | x <- list2set list]
```

## The given demo that you are to augment

```
>>> count 'e' "need more coffee"
5
>>> count 4 [1,2,3,2,3,4,3,4,5,4,5,6]
3
>>> count 'i' "like a bird on a wire"
3
>>> count True [False,True,True,False,True]
3
>>> freqTable "need more coffee"
[('n',1),('d',1),('m',1),('r',1),(' ',2),('c',1),('o',2),('f',2),('e',5)]
>>> freqTable [1,2,3,2,3,4,3,4,5,4,5,6]
[(1,1),(2,2),(3,3),(4,3),(5,2),(6,1)]
>>> freqTable ["one", "one", "two", "one", "two", "three"]
[("one",3),("two",2),("three",1)]
>>> freqTable [ odd x | x <- collatz 496 ]
[(False,71),(True,40)]
>>>
```

## Task 6 - Higher Order Functions

This task requires that you write 4 function definitions that feature higher order programming, and that you then demo them by creating a “proper” demo. For this task, please add to your presentation document (1) a text containing the 4 function definitions, and (2) a text containing the demo that you are expected to create. What is a proper demo with respect to this task? Run each of the 4 functions with the applications that I provide in my sample demo, and then add 2 applications of your own invention. Thus, your proper demo will have 4 applications for each of the 4 functions.

## Function specifications

```
tgl n = foldl (+) 0 [1..n]

triangleSequence n = map tgl [1..n]

vowelCount n = length ( filter (\x -> x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u') n )

lcsim f p e = map ( f ) ( filter (p) e)
```

## The given demo that you are to augment

```
>>> tgl 5
15
>>> tgl 10
55
>>> tgl 20
210
>>> tgl 55
1540
>>> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
>>> triangleSequence 20
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210]
>>> triangleSequence 26
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210,231,253,276,300,325,351]
>>> triangleSequence 4
[1,3,6,10]
>>> vowelCount "cat"
1
>>> vowelCount "mouse"
3
>>> vowelCount "penguin"
3
>>> vowelCount "treacherous"
5
```

```
>>> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
>>> animals = ["elephant", "lion", "tiger", "orangatan", "jaguar"]
>>> lcsim length ( \w -> elem ( head w ) "aeiou") animals
[8,9]
>>> lcsim ( \x -> x*x*x ) odd $ triangleSequence 20
[1,27,3375,9261,91125,166375,753571,1157625,3581577,5000211]
>>> lcsim ( \x -> head x ) ( \x -> length x > 5 ) animals
"eoj"
>>>
```

---

## Task 7 - An Interesting Statistic: nPVI

---

This tasks invites you to implement the “normalized pairwise variability index” (nPVI) by making good use of zip and map. This statistic has been used extensively in the field of lingustics, and is also used to significant effect in the field of music cognition. In case you find yourself with a bit of time, and the inclination to see an impressive application of nPVI, you might like to spend some time with the following paper:

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1063.772&rep=rep1&type=pdf

What is the nPVI? As the name implies, it is a measure of the pairwise variability of terms in a sequence of numeric terms. In mathematical notation, the nPVI is defined by the following expression:

$$nPVI = \left(\frac{100}{m-1}\right)\sum_{k=1}^{m-1}\left|\frac{d_k - d_{k+1}}{(d_k + d_{k+1})/2}\right|$$

Should that seem like a lot to unpack, no worries, the plan is for you to reconstruct the nPVI expression in Haskell by writing a sequence of functions that are consistent with the obvious deconstruction of the expression, the last of which actually computes the nPVI for a sequence of integral values.

Please be aware of the fact that, for this little exercise in Haskell programming, the type of a function will habitually be expressed prior to function definition, and, moreover, the type for each function will be very narrowly construed.

---

## Task 7a - Test data

---

```
a = [2,5,1,3]

b = [1,3,6,2,5]

c = [4,4,2,1,1,2,2,4,4,8]

u = [2,2,2,2,2,2,2,2,2,2]

x = [1,9,2,8,3,7,2,8,1,9]
```

```
>>> a
[2,5,1,3]
>>> c
[4,4,2,1,1,2,2,4,4,8]
>>> u
[2,2,2,2,2,2,2,2,2,2]
>>> x
[1,9,2,8,3,7,2,8,1,9]
>>>
```

---

## Task 7b - The pairwiseValues function

---

```
pairwiseValues ( x : xs ) = zip ( x : xs ) xs
```

>>> pairwiseValues a

```
>>> pairwiseValues a
[(2,5),(5,1),(1,3)]
>>> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
>>> pairwiseValues c
[(4,4),(4,2),(2,1),(1,1),(1,2),(2,2),(2,4),(4,4),(4,8)]
>>> pairwiseValues u
[(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2)]
>>> pairwiseValues x
[(1,9),(9,2),(2,8),(8,3),(3,7),(7,2),(2,8),(8,1),(1,9)]
>>>
```

---

**Task 7c - The** pairwiseDifferences **function**

---

```
pairwiseDifferences v = map ( \(x,y) -> x - y)(pairwiseValues v)
```

```
>>> pairwiseDifferences a
[-3,4,-2]
>>> pairwiseDifferences b
[-2,-3,4,-3]
>>> pairwiseDifferences c
[0,2,1,0,-1,0,-2,0,-4]
>>> pairwiseDifferences u
[0,0,0,0,0,0,0,0,0]
>>> pairwiseDifferences x
[-8,7,-6,5,-4,5,-6,7,-8]
>>>
```

---

**Task 7d - The** pairwiseSums **function**

```
pairwiseSums v = map ( \(x,y) -> x + y)(pairwiseValues v)
```

```
>>> pairwiseSums a
[7,6,4]
>>> pairwiseSums b
[4,9,8,7]
>>> pairwiseSums c
[8,6,3,2,3,4,6,8,12]
>>> pairwiseSums u
[4,4,4,4,4,4,4,4,4]
>>> pairwiseSums x
[10,11,10,11,10,9,10,9,10]
>>>
```

---

**Task 7e - The** pairwiseHalves **function**

---

```
half number = ( fromIntegral number ) / 2

pairwiseHalves v = map half v
```

```
>>> pairwiseHalves [1..10]
[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
>>> pairwiseHalves u
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
>>> pairwiseHalves x
[0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
```

---

**Task 7f - The** pairwiseHalfSums **function**

---

```
pairwiseHalfSums v = pairwiseHalves ( pairwiseSums v)
```

```
>>> pairwiseHalfSums a
[3.5,3.0,2.0]
>>> pairwiseHalfSums b
[2.0,4.5,4.0,3.5]
>>> pairwiseHalfSums c
[4.0,3.0,1.5,1.0,1.5,2.0,3.0,4.0,6.0]
>>> pairwiseHalfSums u
[2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0]
>>> pairwiseHalfSums x
[5.0,5.5,5.0,5.5,5.0,4.5,5.0,4.5,5.0]
```

---

**Task 7g - The** pairwiseTermPairs **function**

---

```
pairwiseTermPairs v = zip ( pairwiseDifferences v) ( pairwiseHalfSums v)
```

```
>>> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
>>> pairwiseTermPairs b
[(-2,2.0),(-3,4.5),(4,4.0),(-3,3.5)]
>>> pairwiseTermPairs c
[(0,4.0),(2,3.0),(1,1.5),(0,1.0),(-1,1.5),(0,2.0),(-2,3.0),(0,4.0),(-4,6.0)]
>>> pairwiseTermPairs u
[(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0)]
>>> pairwiseTermPairs x
[(-8,5.0),(7,5.5),(-6,5.0),(5,5.5),(-4,5.0),(5,4.5),(-6,5.0),(7,4.5),(-8,5.0)]
```

## Task 7h - The pairwiseTerms function

```
term ndPair = abs ( fromIntegral ( fst ndPair ) / ( snd ndPair ) )

pairwiseTerms v = map term ( pairwiseTermPairs v )
```

```
>>> pairwiseTerms a
[0.8571428571428571,1.3333333333333333,1.0]
>>> pairwiseTerms b
[1.0,0.6666666666666666,1.0,0.8571428571428571]
>>> pairwiseTerms c
[0.0,0.6666666666666666,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666]
>>> pairwiseTerms u
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
>>> pairwiseTerms x
[1.6,1.2727272727272727,1.2,0.9090909090909091,0.8,1.1111111111111112,1.2,1.5555555555555556,1.6]
>>>
```

## Task 7i - The nPVI function

```
nPVI xs = normalizer xs * sum ( pairwiseTerms xs ) where normalizer xs = 100 / fromIntegral ( ( length xs ) - 1 )
```

```
>>> nPVI a
106.34920634920636
>>> nPVI b
88.09523809523809
>>> nPVI c
37.03703703703703
>>> nPVI u
0.0
>>> nPVI x
124.98316498316497
```

## Task 8 - Historic Code: The Dit Dah Code

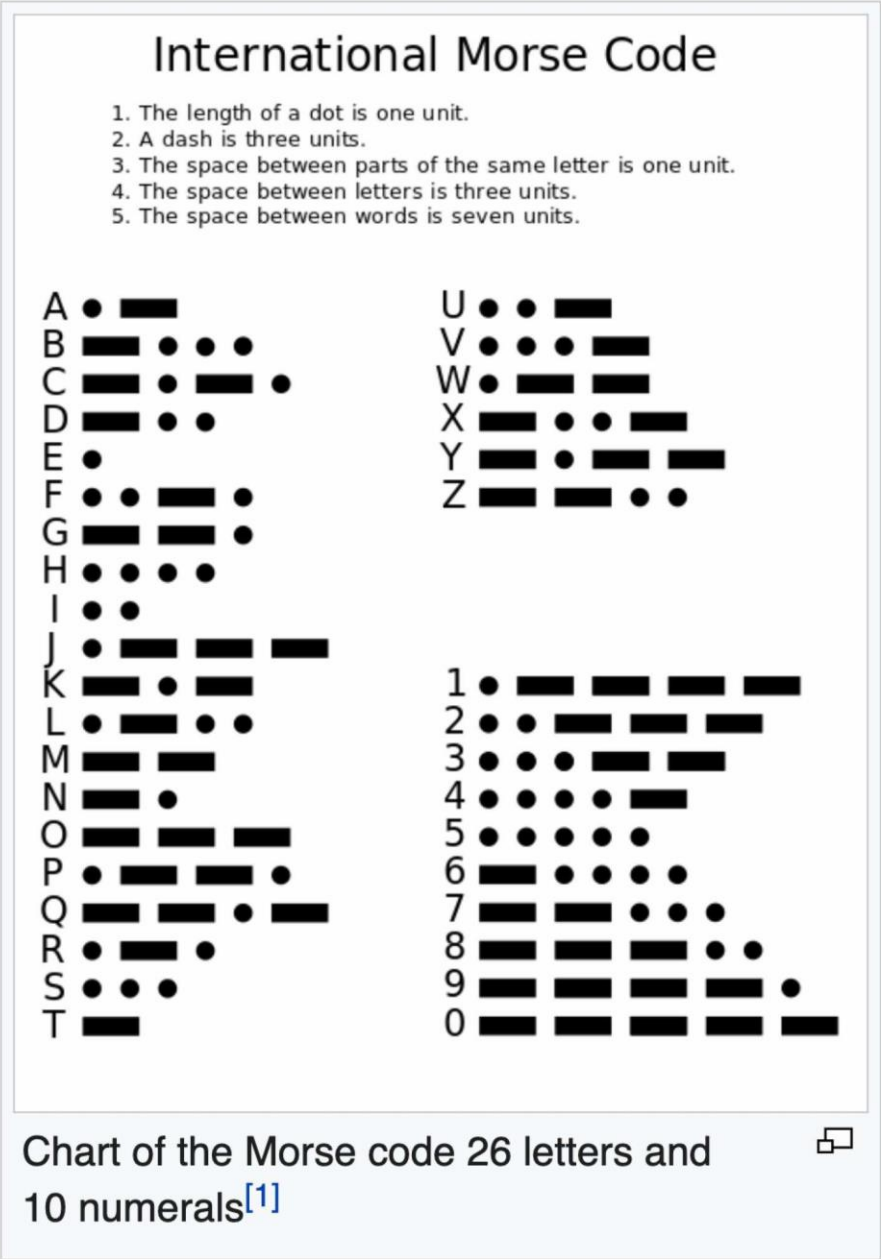


Chart of the Morse code 26 letters and 10 numerals[1]

This task does not require that you write function definitions. Rather, it asks you to read some code, display some variable bindings, and write expressions to illuminate the behavior of a collection of functions.

Haskell programmers seem to enjoy playing with famous codes when showcasing the language. No matter that the Caesar cipher is mostly thought of as a cognitive toy of some historical interest. It still appears as a programming example in a number of texts devoted to learning to program in Haskell. The present task honors another historically significant code, one that once served as a very useful technology, Morse code.

The idea is for you to download a file called ditdah.hs, study it, load it into a Haskell process, and perform the following subtasks. By doing so, perhaps you will learn a little something more about Haskell programming.

Please incorporate your successfull interactions into just one complete demo, and include the demo in your presentation document.

## Subtask 8a


## Subtask 8b

```
>>> assoc 'a' symbols
('a',"- ---")
>>> assoc 'g' symbols
('g',"--- --- -")
>>> find 'e'
"-"
>>> find 'g'
"--- --- -"
```

## Subtask 8c

```
>>> addletter "CSC" "344"
"CSC   344"
>>> addword "Thank" "you so much"
"Thank       you so much"
>>> droplast3 "semester over"
"semester o"
>>> "droplast7 "this is the last assignment"
```

```
>>> droplast7 "lastassignment"
"lastass"
```

## Subtask 8d

```
>>> encodeletter 'm'
"--- ---"
>>> encodeletter 'n'
"--- -"
>>> encodeletter 'o'
"--- --- ---"
>>> encodeletter 'yay'
```

```
>>> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
>>> encodeword "word"
"- --- ---   --- --- ---   - --- -   --- - -"
>>> encodeword "decode"
"--- - -   -   --- - --- -   --- --- ---   --- - -   -"
>>> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -       --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
```

```
>>> encodemessage "semster over"
"- - -   -   --- ---   - - -   ---   -   - --- -       --- --- ---   - - - ---   -   - --- -"
>>> encodemessage "this is my last"
"---   - - - -   - -   - - -       - -   - - -       --- ---   --- - --- ---       - --- - -   - ---   - - -   ---"
>>>
```

---

## Due Date

---

Please complete your work on this assignment, and post your work to your web work site, by sometime on Friday, December 10, 2021.