PROGRAMMING PRACTICE FOR DAY 2

CS 121

1. **Clock arithmetic:** Suppose you keep time with a 24-hour clock. That is, hour 11 refers to 11am, hour 23 refers to 11pm, and hour 0 refers to midnight. If it is currently hour 9, and you set a timer to go off in 52 hours, then the clock will say hour 13 when the alarm goes off.

   Write a Python program that asks the user for the current hour and the number of hours on the timer. Your program should then print the hour on the clock when the alarm goes off.

2. **Wind chill:** If w represents the wind speed in miles per hour, and t represents the temperature in degrees Fahrenheit, then the wind chill is calculated according to the following formula:

$$35.74 + 0.6215t - 35.75w^{0.16} + 0.4275tw^{0.16}$$

   Write a program that asks the user for the wind speed and temperature, and then prints the wind chill.

   When you run your program, it should look something like this:

```
Enter the wind speed (in miles per hour): 20
Enter the temperature (in degrees Fahrenheit): 5
The wind chill is:  -15.435721635148337
```

   Here is another sample screenshot of the wind chill program:

```
Enter the wind speed (in miles per hour): 10
Enter the temperature (in degrees Fahrenheit): -5
The wind chill is:  -22.131599314136327
```

3. **Compound interest:** If a principle P is invested at interest rate r, compounded n times per year, then after t years the investment will have grown to amount A given by the following formula:

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

   Write a program that asks the user for the principle, interest rate, the number of compoundings per year, and number of years. Your program should then output the amount to which the investment will grow.

PRACTICE WITH TURTLE GRAPHICS

CS 121

1. Draw a square with a blue border and orange interior.
2. Draw a square with dots at its corners.
3. Ask the user for a side length and a color. Then draw a square of the user-specified size and color.
4. Ask the user for a number of sides and a side length. Then draw a regular n-gon according to the user-specified parameters.
5. Use Turtle graphics to create your own work of art.

PRACTICE WITH PYTHON MODULES

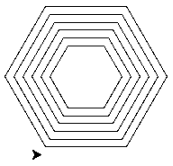
CS 121

1. Choose a random number between 20 and 200, and draw a hexagon with side lengths equal to that number of pixels.
2. Print three random floating-point numbers between 0 and 1, and also print the sum of the three numbers.
3. Write a program that asks the user for the radius of a circle, and then prints the circumference and area of the circle. Use math.pi.
4. Write a program that computes the length of the hypotenuse of a right triangle. Your program should first ask the user for the lengths of the two "legs" of the triangle (that is, the two sides that meet at a right angle). Your program should then calculate and print the length of the hypotenuse. Recall that if the lengths of the legs are a and b, and the length of the hypotenuse is c, then $a^2 + b^2 = c^2$.

PRACTICE WITH FUNCTIONS

CS 121

1. Write a fruitful function `areaOfRectangle(height, width)` that returns the area of a rectangle with dimensions `height` and `width`.
2. Write a function that uses the random module to simulate ten rolls of a standard six-sided die, and then returns the sum of the rolls.
3. Write a function `drawPolygon(anyTurtle, numSides, sideLength)` that makes a turtle draw a regular polygon with the specified number of sides and side length. Use your function to draw four different polygons on the screen.
4. Write a function `drawHexagon(anyTurtle, sideLength)` that calls your `drawPolygon` function from the previous question to have a turtle draw a regular hexagon. Use your function to draw six nested hexagons on the screen, like the image at right.



PRACTICE WITH SELECTION

CS 121

1. Write a function that prompts the user to enter a decimal number. If the number is between 0 and 1 (inclusive), then the program prints the number as a percent. Otherwise, the program prints the message "You entered a number that is not between 0 and 1."
2. Write a function that prompts the user to enter a number. If the user enters 1, then the program draws a square. Otherwise, the program draws a triangle.
3. Write a function that prompts the user to enter three numbers and prints the largest of the three numbers.
4. Write a function that asks the user to enter a day of the week, such as "Monday". If what the user enters is, in fact, the name of a weekday, then print "Thank you." However, if the user enters something that is not a day of the week, then print an error message.
5. Modify your previous function as follows. If the user enters a valid day name, then tell the user whether *today* is the day of the week that they entered. To do this, use the datetime module to get the current day of the week as follows:

```
import datetime
dayOfWeek = datetime.date.today().strftime("%A")
```

MORE PRACTICE WITH SELECTION

CS 121

1. Write a program that asks the user for a string of lowercase letters and uses the `is_vowel()` function from class to determine whether the string consists entirely of vowels.

   Hint: In Python, a `for` loop can iterate over the characters in a string.

   ```
   for i in mystring:
       #do something for each character
   ```

2. Write a function `is_rightangled` that accepts three lengths as parameters and determines whether the three lengths form a right triangle. The function should return `True` if the lengths form a right triangle, and `False` otherwise.

   Hint: It may be easiest to assume that the third argument to the function is the longest side. Then modify your function so that the side lengths can be given in any order.

   Also: Testing equality between floating-point numbers is not always accurate, so instead of `if(a**2 + b**2 == c**2)` use `if(math.isclose(a**2 + b**2, c**2))`.

3. Write a program that computes the day of the week for any date, past or present. First, use three separate input statements to ask the user for a year, a month, and a day. Let these values be stores in the variables `year`, `month`, and `day`. Then compute the following two values:

   ```
   century_item = 2*(3 - ((year // 100) % 4)) + 1
   year_item = (year % 100) + (year % 100) // 4
   ```

   Define another variable, called `month_item`, to have the following value determined by the month:

   `month_item` is 0 if the month is January or October
   `month_item` is 1 if the month is May
   `month_item` is 2 if the month is August
   `month_item` is 3 if the month is February, March, or November
   `month_item` is 4 if the month is June
   `month_item` is 5 if the month is September or December
   `month_item` is 6 if the month is April or July

   Let `total` be the sum `century_item + year_item + month_item + day`.

   If the date is in January or February of a leap year, then subtract 1 from `total`.

   Finally, `total % 7` indicates the day of week: 0 indicates Saturday, 1 indicates Sunday, 2 indicates Monday, and so on.

   Here are some dates to test your program:

   - January 30, 1984 was a Monday.
   - June 19, 2007 was a Tuesday.
   - November 4, 2054 will be a Wednesday.

PRACTICE WITH LOOPS

CS 121

1. Write a program that will figure out how many terms in the sum $1 + 2 + 3 + \cdots$ are necessary for the sum to exceed one million.
2. Write a program that prompts the user for an integer between 1 and 10. However, if the user enters something that is *not* an integer between 1 and 10, your program should print an error message and prompt the user to try again. Your program should repeat this as many times as necessary, until the user enters an integer between 1 and 10.
3. Use a loop to investigate the following sum:

   $$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots$$

   What happens when you add up many terms of this sum? How many terms are necessary to obtain a sum greater than 1.99? How many terms are necessary to exceed 1.999999? Do you think the sum will ever exceed 2?

4. Write a program that asks the user for an integer greater than 1. Your program should then find and print the largest factor of that number. (Recall that a *factor* divides the given number with remainder zero. For example, 6 is a factor of 24, but 7 is not a factor of 24.)

PRACTICE WITH LOOPS: IMAGE PROCESSING

CS 121

1. Write a program that converts an image to black and white. To do this, iterate over all pixels in the original image. For each pixel, if the sum of the red, green, and blue values is less than 383, then assign black (0, 0, 0) for to the corresponding pixel in the new image; otherwise assign white (255, 255, 255) for that pixel.
2. Write a program that resizes an image, creating a new image half as big (in height and width) as an original image. A simple way to do this is to read the colors of the pixels in every other row and column of the original image, and assign those colors to corresponding pixels in a new image.
3. A simple way to blur an image is to set the color values of each pixel to the average color values of the neighboring pixels. Write a program that does this.

PRACTICE WITH STRINGS

1. Write a program that asks the user to type in some text, and then reports the number of characters and also the number of vowels that the user entered.

2. Write a function that removes all non-alphabetic characters from a string. That is, your function should accept a string of text, and then return a string containing the same text but with all non-alphabetic characters removed.

3. As we will see in a few weeks, web pages are built with HTML strings such as "`<b>Yay</b>`", which prints **Yay** in bold text. The letter *b*, appearing between angle brackets, is called a *tag*. Tags come in pairs, and the second tag in the pair must have a slash before the tag text.

   Write a function called `makeTags` that accepts a tag word and some text, and returns an HTML string with the text surrounded by properly-formatted tags. Your function declaration should be:

   ```
   def makeTags(tag, text):
   ```

   A call to `makeTags("div", "some text")` should return the string "`<div>some text</div>`".

4. Now write a function that extracts text from inside of HTML tags. Your function declaration should be:

   ```
   def extractText(HTMLstring):
   ```

   A call to `extractText("<i>some text</i>")` should return "`some text`".

   Hint: consider the `find` and `rfind` Python string methods.

MORE PRACTICE WITH STRINGS

1. Write a function that counts how many times a substring occurs in a string. Your function header should be:

   ```
   def countOccurrences(bigstr, substr):
   ```

   Your function should return the number of times that string `substr` occurs in string `bigstr`, or `0` if `substr` is not found in `bigstr`. For example:

   `countOccurrences("Mississippi", "ss")` returns `2`

2. Write a function that removes the first occurrence of a substring from another string. Your function header should be:

   ```
   def removeFirst(bigstr, substr):
   ```

   Your function should return a string. For example:

   `removeFirst("Mississippi", "ss")` returns `"Miissippi"`

   `removeFirst("A bobcat is a big cat.", "cat")` returns `"A bob is a big cat."`

3. Write a function that removes *all* occurrences of a string from another string. Your function header should be:

   ```
   def removeAll(bigstr, substr):
   ```

   Your function should return a string. For example:

   `removeAll("Mississippi", "ss")` returns `"Miiippi"`

   `removeAll("A bobcat is a big cat.", "cat")` returns `"A bob is a big ."`

PRACTICE WITH LISTS

CS 121

1. Write a function to find the minimum of a list of numbers
2. Write a function that checks to see whether or not a list is sorted in increasing order. Your function header should be:

   ```
   def isSorted(list):
   ```

   The function should return `True` or `False`, depending on whether `list` is sorted or not.
3. Write a function that accepts three parameters: a turtle and two lists. The first list contains positive numbers and the second list contains color words. If the lists are not the same length, your function should print an error message. If the lists are the same length, then for each number in the first list, your function should cause the turtle to draw a square whose color is given by the corresponding item in the second list.
4. *Challenge*: Write a program that allows the user to play the "hangman" game. That is, the program should allow the user to guess the letters in some word. Use a list to keep track of which letters have been guessed.

MORE PRACTICE WITH LISTS

1. In the following code, a list comprehension is to be used to create `newlist`, which will contain only the positive numbers from the list `numbers`, but as integers instead of floats. Replace the question marks and complete the list comprehension.

   ```
   numbers = [12.5, -9.3, 29.1, 32.7, -5.4, 97.6]
   newlist = [  ???  ]
   print(newlist)
   ```
2. Write a function that accepts some text and converts every word to pig Latin. *Hint:* use the `split` and `join` methods, and your `toPigLatin` function from last week.
3. Say that a string is *complete* if it contains all of the characters from *a* to *z*. Write a function `isComplete` that determines whether or not a string is complete. For example:

   `isComplete("The quick brown fox jumps over the lazy dog.")` returns `True`

   `isComplete("some text")` returns `False`
4. An anagram is a word or a phrase made by transposing the letters of another word or phrase; for example, "parliament" is an anagram of "partial men," and "software" is an anagram of "swear oft." Write a program that figures out whether one string is an anagram of another string. The program should ignore white space and punctuation.

PRACTICE WITH FILES

1. The following file contains many lines containing numbers separated by tab characters:

   https://www.mlwright.org/teaching/cs121s17/numbers.txt

   Write a program that reads the file and prints the sum of numbers on each line.

2. Modify your program from #1 to save the sums to a new file.

3. Write a program that reads a text file, converts each word to upper case, and saves the result to a new file. The new file should be exactly like the old file, including the same line breaks, but with all words in uppercase. *Hint*: consider the Python string method `upper()`.

   Test your program with the following files:

   https://www.mlwright.org/teaching/cs121s17/milton.txt

   https://www.mlwright.org/teaching/cs121s17/pat_sat.txt

4. Modify your program from #3 so that, in addition to writing a file, it prints (to the console) the number of lines that it read from the input file and the number of words that it converted to uppercase.

PRACTICE WITH DICTIONARIES

1. Write a function `countLetters()` that counts the frequency of letters in a string (ignore non-alphabetic characters). Print the letters with counts in alphabetical order.

   Hint: recall `str.isalpha()` and `str.lower()`

2. Write a function `merge(d1, d2)` that merges two dictionaries and returns the result. Any key that appears in `d1` or `d2` should appear in the returned dictionary. If a key appears in only one of `d1` or `d2`, then its value in the returned dictionary should be the same as its value in `d1` or `d2`. However, if a key appears in both `d1` and `d2`, then its value in the returned dictionary should be the sum of its values in `d1` and `d2`.

   For example, if `d1 = {'a':5, 'b':2}` and `d2 = {'a':3, 'c':4}`, then the returned dictionary should be `{'a':8, 'b':2, 'c':4}`.

PRACTICE WITH EXCEPTIONS

1. The following function is supposed to print an element from a list.

   ```
   def printListElement(thelist, index):
   print(thelist[index])
   ```

   Add `try` and `except` statements to the function to handle a possible `IndexError`. If this exception occurs, print a helpful message for the user.

2. The following line of code might raise a `ValueError`:

   ```
   n = int(input("Enter an integer: "))
   ```

   Use `try` and `except` statements to display a helpful error message if the user does not enter an integer.

3. Modify your previous program so that, if the user does not enter an integer, the program asks the user for another integer. The program should continue asking until the user enters an integer.

4. Write a program that asks the user for the name of a file. The program should then attempt to open the file and print the contents of the file. However, if the file cannot be opened, the program should handle the resulting exception and print a helpful error message.

PRACTICE WITH TKINTER

1. Set up a Tkinter canvas and draw a face on it.

2. Program a button to draw a rectangle at a random position in a Tkinter canvas.

3. Program a button to update the text in a Tkinter label. (*Hint*: If you're not sure how to do this, Google will help!)

4. Program a button that changes the font or colors of text in a Tkinter label.

5. Create a button that removes all items from a Tkinter canvas. (*Hint*: Google)