Assignment Guidelines:

- Submission details:
  - Solutions to these questions must be placed in files `a1q1.rkt`, `a1q2.rkt`, and `a1q3.rkt`, respectively, and must be completed in Racket.
  - All solutions must be submitted through MarkUs. Solutions will not be accepted through email.
  - Verify your basic test results using MarkUs to ensure that your files were submitted properly and are readable on MarkUs. Note, however, that passing the basic tests does not guarantee that you will pass all our correctness tests.

- For this assignment only, you are not required to use the design recipe when writing functions. In each case, you are only required to include the function header and body. You are not required to include the purpose, contract, examples, or tests. Your grade on this assignment only is based entirely on correctness and following the stated requirements of the questions.

- Download the interface file from the course Web page to ensure that all function names are spelled correctly, and each function has the correct number and order of parameters.

- Restrictions:
  - You may only use the built-in functions and special forms introduced in the lecture slides in Module 01. A list of these functions can be found on Learn.
  - Read each question carefully to see if any additional restrictions apply.
  - Test data for correctness tests will always meet the stated assumptions for consumed values.

- The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

Plagiarism: The following applies to all assignments in CS115.

All work in CS 115 is to be done individually. The penalty for plagiarism on assignments (first offense) is a mark of 0 on the affected question and a 5% reduction of the final grade, consistent with School of Computer Science policy. In addition, a letter detailing the offense is sent to the Associate Dean of Undergraduate Studies, meaning that subsequent offenses will carry more severe penalties, up to suspension or expulsion.

To avoid inadvertently incurring this penalty, you should discuss assignment issues with other students only in a very broad and high-level fashion. Do not take notes during such discussions, and avoid looking at anyone else’s code, on screen or on paper. If you find yourself stuck, contact the ISA or instructor for help, instead of getting the solution from someone else. Do not consult other books, library materials, Internet sources, or solutions (yours or other people’s) from other courses or other terms.

Be sure to read the Plagiarism section at https://www.student.cs.uwaterloo.ca/~cs115/#assignments
There are three parts to this assignment. Each part has three sections: you are required to read and submit solutions for each Question section, and you may refer to the Examples section as a guide to test your functions. You are not required to read the Interesting but unnecessary trivia sections, although they provide an opportunity to explore popular topics in computer science such as video game design, computer graphics and geometry (you will not be evaluated on them now or later). The questions are self-contained, so previous experience with these trivia topics is neither required nor advantageous.

The design recipe is not required for this assignment, but you should attempt to test your functions in the Interaction panel by applying them to several input values before submitting the required assignment code. If the function produces any unexpected values, then there is a bug that should be fixed before resubmitting the code!

1. Distance between circles.

Interesting but unnecessary trivia.
A common task in video game design is checking whether two objects are touching. For example, this allows us to check whether the bullet has hit the target, or whether someone is standing too close to the fire! One of the easiest ways to check this is to put invisible circles or spheres around the objects, and then calculate whether the circles are touching (or similarly, how far apart they are).

Question.
Suppose we are given a circle centered at position \((x_1, y_1)\) with radius \(r_1\), and another circle centered at \((x_2, y_2)\) with radius \(r_2\) illustrated below.

![Diagram of two circles with exterior distance labeled]
The total distance between the centers of the circles is:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The radii ($r_1$ and $r_2$) are a part of this total distance, so the exterior distance between the circles (not including the radii) is:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - r_1 - r_2$$

Complete the function body for `(circle-distance x1 y1 r1 x2 y2 r2)` which consumes the circle positions $x_1$, $y_1$ and $x_2$, $y_2$ and the radii $r_1$, $r_2$, and produces the exterior distance between the two circles. You don’t have to handle the case where the circles overlap or contain each other, and you can assume that the radii are positive.

**Examples.**

Consider a circle centered at (0,0) with radius 1 and a circle centered at (3,4) with radius 2. Then the exterior distance between the circles is `(circle-distance 0 0 1 3 4 2)` => 2 because

\[
\begin{align*}
\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - r_1 - r_2 \\
= \sqrt{(0 - 3)^2 + (0 - 4)^2} - 1 - 2 \\
= 5 - 1 - 2 = 2
\end{align*}
\]

Similarly, consider a circle centered at (1.5, 1.5) with radius 0.5 and a circle centered at (-1.5,-2.5) with radius 3.5. Then the exterior distance between the circles is `(circle-distance 1.5 1.5 0.5 -1.5 -2.5 3.5)` => 1 because

\[
\begin{align*}
\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - r_1 - r_2 \\
= \sqrt{(1.5 - (-1.5))^2 + (1.5 - (-2.5))^2} - 0.5 - 3.5 \\
= \sqrt{(1.5 + 1.5)^2 + (1.5 + 2.5)^2} - 0.5 - 3.5 \\
= 5 - 0.5 - 3.5 = 1
\end{align*}
\]

2. **Weighted average of numbers.**

**Interesting but unnecessary trivia.**

Computer display screens are a grid of small square cells called pixels, and each pixel displays its own colour. Early video game systems were only powerful enough to calculate the colours for a small number of undesirably large pixels, so the graphics appeared blocky and jagged (left picture below).
The picture on the right shows a screenshot of the popular 90’s game *Starfox*. Note that almost all lines and edges appear jagged and distorted. These jagged edges are especially distracting to humans because the human visual cortex relies heavily on edges to understand what it’s seeing.

The impact of these jagged edges can be reduced by *anti-aliasing* which softens and smooths the edge (left picture below). Instead of displaying the original pixels, the computer calculates an anti-aliased colour for each pixel based on the colours of the original neighbouring pixels.

Assume for simplicity that every pixel has eight neighbouring pixels. The rightmost picture below shows a black pixel (value 255) with eight neighbouring white pixels (value 0). Anti-aliasing the black pixel produces a new greyscale pixel (value 127) if we use a weighted average function that favours the original pixel colour without placing too much emphasis on the neighbouring pixels.
Question.
The weighted average function

\[ \text{wavg}(x, a, b, c, d, e, f, g, h) = [0.5x + 0.0625(a + b + c + d + e + f + g + h)] \]

calculates the average of nine values, \( x \) and \( a \) thru \( h \), but the weighting allows us to emphasize some of the nine values over others. The function puts extra weight on the value labeled \( x \), whereas the other values \( a \) thru \( h \) have less influence on the average.

Complete the function body of \( \text{wavg}(x, a, b, c, d, e, f, g, h) \) which consumes nine natural numbers, \( x \) and \( a \) thru \( h \), and produces the value of the provided formula. Use the \text{floor} function to round the sum down to the nearest natural number.

Examples.
(\text{wavg } 255 0 0 0 0 0 0 0 0) => 127
since \( \text{wavg}(255, 0,0,0,0,0,0,0,0) = [0.5(255) + 0.0625(0)] = [127.5] = 127 \)
(\text{wavg } 255 0 0 0 0 0 0 255 255) => 143
(\text{wavg } 255 255 255 255 255 255 255 255 255) => 255
(\text{wavg } 0 0 0 0 0 0 0 0 0) => 0

3. Affine transformations.

\textit{Interesting but unnecessary trivia.}
Modern graphics in movies and video games rely on detailed geometric models as seen below. These geometric models are a collection of connected triangles. We can move and reshape the model by manipulating individual triangles, and in turn, we manipulate each triangle by transforming each of its three vertices. For simplicity assume that a vertex is a point \((x, y) \in \mathbb{R}^2\).
An affine transformation can be represented by a matrix $A$ that allows us to transform a vertex $(x, y)$ to a new location $(x', y')$ using matrix multiplication:

$$
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix}
= A
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
$$

Representing the transformation using matrix multiplication is convenient because it allows us to apply a sequence of transforms by multiplying a sequence of matrices (below). For example, we may want to translate the vertex to a new position using matrix $A$, and then rotate that new vertex position about the origin using matrix $B$ (imagine that something similar could be used to animate a rotating tire on a car that is driving down the road).

$$
BA
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
= B
\left(A
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}\right)
= B
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix}
= \begin{bmatrix}
  x'' \\
  y'' \\
  1
\end{bmatrix}
$$

Let’s change our focus back to the matrix $A$, which we can expand to derive the simple non-matrix equations for the affine transformation (i.e., the equations used in Question).

$$
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix}
= \begin{bmatrix}
  a & b & u \\
  c & d & v \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
$$

**Question.**
Affine transformations allow us to transform a point $(x, y) \in \mathbb{R}^2$ to a new coordinate location $(x', y')$ using the parameters $a, b, c, d, u$ and $v$, and the equations:

$$
\begin{align*}
  x' &= ax + by + u \\
  y' &= cx + dy + v
\end{align*}
$$

The parameters allow us to specify whether the transformation is a rotation, reflection or translation (or some combination!). The *Examples* table lists some common transformations and the matching parameters.

The functions `affine-x` and `affine-y` produce the numbers $x'$ and $y'$ using the given equations. Complete the function bodies for `(affine-x x y a b u)` and `(affine-y x y c d v)` which consume the numbers $x$ and $y$ (i.e., the original point), as well as the numbers $a, b$ and $u; \text{ and } c, d \text{ and } v, \text{ respectively (i.e., the transform parameters).}
Examples.
The table below lists the parameters for common and useful affine transformations:

<table>
<thead>
<tr>
<th>Affine Transformation</th>
<th>Parameters</th>
<th>Example</th>
</tr>
</thead>
</table>
| **Identity** – do nothing, i.e., \((x', y') = (x, y)\) | \(a = 1, b = 0,\)  
\(c = 0, d = 1,\)  
\(u = 0, v = 0\) | \((3,4) \rightarrow (3,4)\)  
(affine-x 3 4 1 0 0) => 3  
(affine-y 3 4 0 1 0) => 4 |
| **Reflection** – reflect the point about the y axis, i.e., \((x', y') = (-x, y)\) | \(a = -1, b = 0,\)  
\(c = 0, d = 1,\)  
\(u = 0, v = 0\) | \((3,4) \rightarrow (-3,4)\)  
(affine-x 3 4 -1 0 0) => -3  
(affine-y 3 4 0 1 0) => 4 |
| **Rotation** – rotate the point about the origin by \(\theta\) radians (clockwise is positive) | \(a = \cos \theta, b = \sin \theta,\)  
\(c = -\sin \theta, d = \cos \theta,\)  
\(u = 0, v = 0\) | \(\theta = \frac{\pi}{2}\) (e.g., 90°)  
(3,4) \rightarrow (4,-3)  
(affine-x 3 4 0 1 0) => 4  
(affine-y 3 4 -1 0 0) => -3 |
| **Translation** – move the x coordinate by \(m\) and the y coordinate by \(n\), i.e., \((x', y') = (x+m, y+n)\) | \(a = 1, b = 0,\)  
\(c = 0, d = 1,\)  
\(u = m, v = n\) | \((m,n) = (1,-1)\)  
(3,4) \rightarrow (4,3)  
(affine-x 3 4 1 0 1) => 4  
(affine-y 3 4 0 1 -1) => 3 |