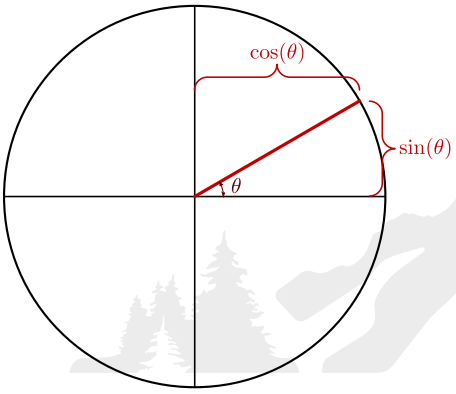
# HW 8: Memoizing the Trig Functions

You must implement a class of static functions which memoizes the sine, cosine, and tangent functions. The class should be called `TrigCalculator`.




Trig functions are *very* important for 3D simulations, including games. However, they are also very slow. Storing these results in tables can help speed a program up, when it must do hundreds of these operations every second.

As you should have learned in high school, sines and cosines represent the relative sizes of a rotated object. This is demonstrated in the figure on the left. If a radius of the unit circle is rotated by an angle $\theta$, its height is the sine of the angle, and its width is the cosine. The tangent is just the ratio of the sine to the cosine.

You will need to implement the following `public static` functions:

- `double sin(double angle)`, which calculates the sine of the angle.
- `double cos(double angle)`, which calculates the cosine of the angle.
- `double tan(double angle)`, which calculates the tangent of the angle.

In addition you need a `main()` function to test them, as is described below. Fortunately, once `sin()` is implemented, `cos()` and `tan()` can each be done in one line. Use the identity

$$\cos(\theta) = \sin(\pi/2 - \theta)$$

to implement `cos()`, and calculate the ratio of `sin()` and `cos()` to do `tan()`.

Let us assume for the moment that the angle $\theta$ is in the first quarter, so $0 \leq \theta \leq \pi/2$. You will need to have a pre-built double array of sine values, which we will call `sinTable`. The length of `sinTable` will be a constant called `TABLE_SIZE`, and it should hold `101`. The array will be initialized in the static block, so that element $i$ contains $\sin(i/\texttt{TABLE\_SIZE}-1 \times \pi/2)$. If the angle is indeed in the first quarter, calculating the sine is just a matter of linearly interpolating the array.

What if $\theta$ is not in the first quarter? There are several mathematical identities that we can use to treat any angle as if it's in the first quarter. These are:

$$\sin(\theta) = -\sin(-\theta) \tag{1}$$

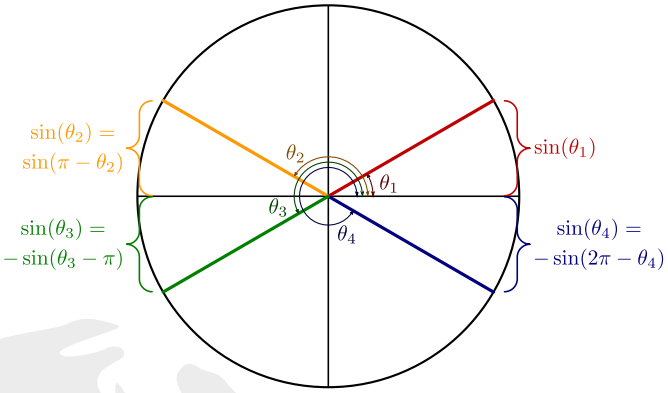
$$\sin(\theta) = \sin(\theta - 2\pi) \tag{2}$$

$$\sin(\theta) = \sin(\pi - \theta) \tag{3}$$

$$\sin(\theta) = -\sin(\theta - \pi) \tag{4}$$

$$\sin(\theta) = -\sin(2\pi - \theta) \tag{5}$$

The first identity says that if the angle is negative, you can flip the sign (but remember to negate your final answer). The second identity says that if the angle is greater than or equal to $2\pi$, you can just mod it by $2\pi$ in order to calculate the sine of an equivalent angle. So no matter what $\theta$ is, we can always reduce the problem to calculating the sine of some $\theta$ where $0 \leq \theta < 2\pi$.




The remaining identities let you find an equiavlant angle such that $0 \leq \theta \leq {}^{\pi}/_{2}$:

- If ${}^{\pi}/_{2} < \theta \leq \pi$, calculate and return $\sin(\pi - \theta)$.
- If $\pi < \theta \leq {}^{3\pi}/_{2}$, calculate and return $-\sin(\theta - \pi)$.
- If ${}^{3\pi}/_{2} < \theta < 2\pi$, calculate and return $-\sin(2\pi - \theta)$.

Since we are linearly interpolating the array, the results of these functions will not be exact. Your `main()` method must calculate the errors for each of sine, cosine, and tangent, for values ranging from 0 radians to 10 radians. The error is defined as:

$$error = \frac{calculated\ value - true\ value}{true\ value}$$

The sine errors are included here, so that you may test your work.

```
Testing sine(theta):
  0: NaN
  1: -2.7575447244476327E-5
  2: -2.6996038967544347E-5
  3: -1.7722406960652687E-6
  4: -2.8105708088786508E-5
  5: -2.6367785167844438E-5
  6: -3.4307624255069626E-6
  7: -2.8586317183413856E-5
  8: -2.5690895834280773E-5
  9: -5.039830367713609E-6
  10: -2.901630254706844E-5
```

These errors are probably small enough for any game. However, if you needed to decrease the error further, you could always increase `TABLE_SIZE`.