In[2]:=
```
(* Coin flip, problem 3 *)
```

In[3]:=
```
(* Assumption: Coin radius is 1, height is 1 *
```

In[3]:=

In[4]:=
```
(* Time to go back to initial height *)
```

In[5]:=
```
timeFunction[v_] := 2 v / 9.8
timeFunction[4.5]
```

Out[6]=
```
0.918367
```

In[7]:=
```
(* Angular velocity is in radians *)
(* converts angular velocity to degrees per second *)
angularToDegrees[a_] := a * (180 / Pi)
```

In[8]:=
```
(* Function to find where the coin is roated once it falls back into initial height *)
finalDegrees[timetofunction_, degrees_] := timetofunction * degrees

(* Reduces the finalDegrees into a value from 0 to 360 degrees *)
reducetobounds[finaldegrees_] := finaldegrees - (360 * Floor[finaldegrees / 360])

(* Takes a reduced degree and finds if it will land heads or tails. 1 = heads,
0 = tails *)
headsortails[reducedDegree_] := If[(reducedDegree > 270 || reducedDegree < 90), 0, 1]
```

In[11]:=
```
(* This is where I am going to introduce error *)

(* To land on the side, the coin needs to rotate exactly 90 or 270 degrees *)
(* However this will never happen as the
 precision of my calculation always has a decimal *)
(* Instead of rounding to the nearest n-th degree, I am going to take a ratio *)
(* If reducedDegree/90 or reducedDegree/270 is between .999 and 1.001,
it lands on its side *)
(* 1 = lands on side, 0 = no *)
```

In[13]:=
```
side[reducedDegree_] := If[((reducedDegree / 90 ≥ .999 && reducedDegree / 90 ≤ 1.001) ||
      (reducedDegree / 270 ≥ .999 && reducedDegree / 270 ≤ 1.001)), 1, 0];
```