Discrete Dynamical Systems (DDS)

Any system (natural, engineered, social, etc.,) that changes its internal state in jumps rather than flowing from one to the next is called a discrete dynamical system. One might think that a functional program would not be able to model a DDS, which relies on the concept of a changing state. The next few labs will show that this isn't the case!

A DDS has four components:

- A current state from some state space, S
- An update function:

```
def update(currentState: S, cycle: Int) = the next state
```

Note: cycle = number of calls to update so far. For some DDSs the next and final states can depend on time as well as the current state.

- A halting function:

```
def halt(currentState: S, cycle: Int) =
   if (currentState is final?) true else false
```

- And a control loop that iterates update and increments cycle until a final state is reached (which may never happen):

```
def controlLoop[S](
   state: S,
   cycle: Int,
   halt: (S, Int)=> Boolean,
   update: (S, Int)=>S): S = the final state
```

1. Problem: Control Loop

Find a tail recursive implementation of controlLoop.

Note

A tail recursive control loop shows us that it's possible to model a state-changing system (such as a computer) without using variables!

2. Problem: Population Growth

A pond of amoebas reproduces asexually until the pond's carrying capacity is reached. More specifically, the initial population of one doubles every week until it exceeds 10^5. Use your controlLoop function to compute the size of the final population.

3. Problem: Finding Roots of Functions

Newton devised an algorithm for approximating the roots of an arbitrary differential function, f, by iterating:

```
guess = guess - f(guess)/f'(guess)
```

Recall that f'(x) is the limit as delta approaches zero of:

```
(f(x + delta) - f(x))/delta
```

Use Newton's method and your controlLoop to complete:

```
def solve(f: Double=>Double) = r where |f(r)| <= delta
```

4. Problem: Approximating Square Roots

Use your solve function to complete:

```
def squareRoot(x: Double) = solve(???)
```

5. Problem: Approximating Cube Roots

Use your solve function to complete:

```
def cubeRoot(x: Double) = solve(???)
```

6. Problem: Approximating Nth Roots

Use your solve function to complete:

```
def nthRoot(x: Double, n: Int) = r where |r^n - x | <= delta
```