```
1)  let x = 5 - 3 in (x * 2)
    -> let x = 2 in (x * 2)
    -> 2 * 2
    -> 4

2)  (\x -> x * 2) (5 - 3)
    -> (\x -> x * 2) 2
    -> 2 * 2
    -> 4

3)  let y = 3 in (let x = 4 in (y + (3 * x)))
    -> let x = 4 in (3 + (3 * x))
    -> 3 + (3 * 4)
    -> 3 + 12
    -> 15

4)  let y = 3 in (let x = 4 + y in (y + (3 * x)))
    -> let x = 4 + 3 in (3 + (3 * x))
    -> let x = 7 in (3 + (3 * x))
    -> 3 + (3 * 7)
    -> 3 + 21
    -> 24

5)  ((\y -> (\x -> y + (2 * x))) 3) 5
    -> (\x -> 3 + (2 * x)) 5
    -> 3 + (2 * 5)
    -> 3 + 10
    -> 13

6)  ((\y -> (\x -> x + (2 * x))) 3) 5
    -> (\x -> x + (2 * x)) 5
    -> 5 + (2 * 5)
    -> 5 + 10
    -> 15

7)  (\y -> y * (let y = 4 - 1 in (y + 2)) + y) 4
    -> (\y -> y * (let y = 3 in (y + 2)) + y) 4
    -> (\y -> y * (3 + 2) + y) 4
    -> (\y -> y * 5 + y) 4
    -> 4 * 5 + 4
    -> 20 + 4
    -> 24

8)  g ((let x = 2 in (\y -> x + y)) 5)
    -> g ((\y -> 2 + y) 5)
    -> g (2 + 5)
    -> g 7
    -> (\z -> z + 3) 7
    -> 7 + 3
    -> 10
```

```
9)  let y = 5 in (\z -> y * z)
    -> (\z -> 5 * z)
    -> error: z not defined within scope

10) f ((\fn -> fn Paper) (\z -> whatItBeats z))
    -> f ((\z -> whatItBeats z) Paper)
    -> f (whatItBeats Paper)
    -> f ((\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Paper)
    -> f (case Paper of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper})
    -> f Rock
    -> (\s -> case s of {Rock -> 50; Paper -> 100; Scissors -> 10}) Rock
    -> case Rock of {Rock -> 50; Paper -> 100; Scissors -> 10}
    -> 50

11) ((\f -> (\z -> f (f z))) whatItBeats) Scissors
    -> (\z -> whatItBeats (whatItBeats z)) Scissors
    -> whatItBeats (whatItBeats Scissors)
    -> whatItBeats ((\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Scissors)
    -> whatItBeats (case Scissors of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper})
    -> whatItBeats Paper
    -> (\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Paper
    -> case Paper of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}
    -> Rock

12) case (Win (whatItBeats Rock)) of {Draw -> m; Win z -> (m + f z)}
    -> case (Win ((\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Rock)) of {Draw -> m; Win z -> (m + f z)}
    -> case (Win (case Rock of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper})) of {Draw -> m; Win z -> (m + f z)}
    -> case (Win Scissors) of {Draw -> m; Win z -> (m + f z)}
    -> m + f z
    -> 5 + f z
    -> 5 + (\s -> case s of {Rock -> 50; Paper -> 100; Scissors -> 10}) z
    -> 5 + case z of {Rock -> 50; Paper -> 100; Scissors -> 10}
    -> error: z not defined within scope

13) e = \s -> (\n -> case s of {Rock -> n * n; Paper -> 2 * n; Scissors -> n})

    [Rock/s][4/n]e
    -> e s n
    -> e Rock 4
    -> (\s -> (\n -> case s of {Rock -> n * n; Paper -> 2 * n; Scissors -> n})) Rock 4
    -> (\n -> case Rock of {Rock -> n * n; Paper -> 2 * n; Scissors -> n}) 4
    -> case Rock of {Rock -> 4 * 4; Paper -> 2 * 4; Scissors -> 4}
    -> 4 * 4
    -> 16

    [Paper/s][4/n]e
    -> e s n
    -> e Paper 4
    -> (\s -> (\n -> case s of {Rock -> n * n; Paper -> 2 * n; Scissors -> n})) Paper 4
```

```
-> (\n -> case Paper of {Rock -> n * n; Paper -> 2 * n; Scissors -> n}) 4
-> case Paper of {Rock -> 4 * 4; Paper -> 2 * 4; Scissors -> 4}
-> 2 * 4
-> 8

[Scissors/s][4/n]e
-> e s n
-> e Scissors 4
-> (\s -> (\n -> case s of {Rock -> n * n; Paper -> 2 * n; Scissors -> n})) Scissors 4
-> (\n -> case Scissors of {Rock -> n * n; Paper -> 2 * n; Scissors -> n}) 4
-> case Scissors of {Rock -> 4 * 4; Paper -> 2 * 4; Scissors -> 4}
-> 4
```

14) Given the sun is used as a compass and that ants have an internal pedometer that they use for path integration, the system would need to support float addition, multiplication, and trigonometric functions. We can utilize the Pythagorean identity (let theta be the angle of the sun such that facing it directly equals 0 radians) to provide a normalized direction vector, upon which we can multiply the magnitude of distance provided by the ant's internal pedometer (steps * const dist_per_step). Set the nesting site to vector (0, 0, 0). Every time a change in travel direction occurs, calculate a directional vector and multiply it by the distance traveled since the last time a directional change occurred. Add this value to the current position vector. Note that traveling in the x or y directions doesn't necessarily correlate with the cardenal directions on a map where +x is east, +y is north, etc. since theta is equal to 0 when facing the sun (which is usually somewhere between east and west).

15) The friend makes a valid point in saying that the computer model doesn't have any validity in describing ant cognition. In terms of Marr's levels of analysis, I did not consider any computational constraints when creating a computer representation and algorithms for ant path integration. When the friend states that ants don't have the capacity to make such calculations, she is implicitly operating on the computational theory level of analysis by arguing that the logic behind the algorithm is bad. That being said, I could make a counter-argument as to the cogency of my implementation for the analysis of ant cognition. Although the exact math is likely nowhere near the level at which ant cognition occurs, note that theta is determined relative to facing the sun, not in terms of a compass and bearings (such that north is 0 degrees). Relative positioning is far less computationally taxing than absolute positioning (even humans tend to describe directions in terms of in front, behind, left, and right), and tracking distance traveled by multiplying a constant value for leg length makes sense on a computational theory level of analysis because ants overshot their nesting sites when their functional leg length changed via stilts, suggesting ants assume leg length is constant.