Update Functions

Quest

Sometimes, after storing a value in a variable, you want that value to change as your program runs. One example of this is the variable `i` in our for-loops! At a given point, `i` stores the value for which iteration of the loop is being run. After each iteration, `i` is increased by 1 to keep track of how many times the program has looped. This concept of setting and then updating a value is a common one, and we are going to learn about a neat conceptual way of writing code to make that process happen.

Let's play a game! We are going to write down a sequence of numbers starting with a positive integer that you pick. Go ahead, pick one now! The next number in your sequence depends on a few simple rules:

1. If the current number is 1, then the sequence ends
2. If the current number is *even*, then the next number is `n / 2`
3. If the current number is *odd*, then the next number is `(3 * n) + 1`

We apply these rules to each number in the sequence until the sequence finally ends on a 1. As an example, say we picked the number 5 to start:

- 5 is odd, so the next number is `(3 * 5) + 1 = 16`
- 16 is even, so the next number is `16 / 2 = 8`
- 8 is even, so the next number is `8 / 2 = 4`
- 4 is even, so the next number is `4 / 2 = 2`
- 2 is even, so the next number is `2 / 2 = 1`
- 1 is the last number of the sequence

So, the final sequence is `[5, 16, 8, 4, 2, 1]`.

Go ahead a try this out with the number you picked!

Could we write a program to do this for us? We know how to ask for user input, handle different cases with conditionals, and loop over code multiple times. We should be able to use our Python skills to get the job done!

First, we start with some pseudocode. This is always a great first step when figuring out how to structure your code at a high level:

PSEUDOCODE

```
number = some a positive integer
repeat until number = 1:
    write down that number
    number = the next number in the sequence
write down a 1
```

x = update(x)

The magic really happens in the line where we updated the value of `number`:

```
number = the next number in the sequence
```

We could accomplish this step with a helper function that takes in the current number and outputs the next number according to the rules we stated above! Let's call this function `next_number`:

```
number = next_number(number)
```

This instruction sets the value of `number` equal to the result of feeding `number` into `next_number`. We refer to this type of instruction as an **`x = update(x)`** instruction because `number` is both the argument passed into `next_number` and the variable where the result of `next_number` is stored. We decided to name functions like `next_number` **update functions** because we use them to update the value of a variable according to some rules.

How do we want `next_number` to work? Recall your skills in writing new functions and think about what this function needs to do...

Done thinking? Ok, let's look below:

```
def next_number(number)
    '''
    next_number takes in the current number of a sequence and determines what the next
    number in the sequence should be
    params:
        number (int): previous number in the current sequence
    return: (int): next number in the sequence
    '''
    # If number is 1 then return a zero to indicate the sequence has ended
    if number == 1:
        return 0
    elif number % 2 == 0:  # If number is even, the next number is number/2
        return number / 2
    else: # If number is odd, the next number is (3 * number) + 1
        return (3 * number) + 1
```

In this implementation, we chose to use a 0 to indicate to the program that the sequence should end, but how you go about doing that is entirely up to you!

Update functions are useful when the change you want to apply to a value is more complicated than what can be accomplished in a single line. In the case of `next_number`, the result depends on whether the number passed in is 1, even, or odd and requires multiple if-statements to handle each of these cases. This way, we decompose all that extra code into a singular, easy-to-read function.

Update functions aren't explicitly a feature of Python, but they are a concept built from tools Python

gives us, like variables, functions, and parameters. And they're super useful too!

Fun fact, these sequences come from a real mathematical concept called **Hailstone Sequences**. Mathematicians are still trying to figure out whether hailstone sequences always terminate at 1, or if there are some integers that result in an infinite hailstone sequence that never ends.

Curious about these peculiar sequences? Read more about this fascinating unsolved problem:
https://en.wikipedia.org/wiki/Collatz_conjecture