Control Flow

fxquah

3rd December 2019

1 Problem statement: Collatz conjecture

Consider the following operation on a **postive integer**:

- If the number is even, divide by two.
- If the number is odd, multiply by three and add one.

Form a sequence by repeating this operation, using the result of each step as the input of the next.

Here's an example with the number 6. It reaches one after.

The Collatz conjecture states that: *This process will eventually reach one, regardless of the initial positive integer.*

Eccentric Hungarian mathematician Paul Erdős once said about the conjecture: "Mathematics may not be ready for such problems", and offered US$500 for its solution. As of 2017, the conjecture has been checked by computers to hold true for starting values up to 87×2^{60}.

2 Introduction to control flow

Despite the challenges to obtain a mathematical proof, the Collatz conjecture is a useful problem to learn basic control flow in Julia. **Control flow** concerns the order in which individual lines of code within a program are executed.

2.1 Conditional evaluation

For this problem, we want our program to perform different operations depending on whether the integer is positive or negative. We can achieve this behaviour using the `if-else` conditional syntax:

```
x = 6
if x % 2 == 0
        x / 2   # even
else
        3x + 1  # odd
end
```

```
3.0
```

This can be written as a one-liner using the **ternary operator**.

```
x%2==0 ? x/2 : 3x+1
```

```
3.0
```

2.2 Repeated evaluation: while

We want the above operations to be repeatedly evaluated until we reach one, so we put them within a `while` loop. As long as the number does not equal one (`x != 1`), the body of the loop keeps evaluating. To tidy things up, we wrap everything inside a function that also prints the current number per iteration.

```
using Printf
function collatzConjecture(x)
        @printf "starting number is %.0f\n\n" x
        while x != 1
                @printf "%.0f\n" x
                x = x%2==0 ? x/2 : 3x+1
        end
        @printf "1\n"
end

collatzConjecture(6)
```

```
starting number is 6

6
3
10
5
16
8
4
2
1
```

We are also interested to know the stopping time, so let's add a `counter` to track the number of iterations the loop has been run.

We initialise its value as `1` right before the `while` loop, update it by one (`counter += 1`) per iteration, and print its final value when the loop exits.

```
function collatzConjectureWithCounter(x)
        @printf "starting number is %.0f\n\n" x
        counter = 1
        while x != 1
                @printf "term %.0f: %.0f\n" counter x
                x = x%2==0 ? x/2 : 3x+1
                counter += 1
        end
        @printf "\nfinal term of 1 is reached, stopping time is %.0f\n" counter
end

collatzConjectureWithCounter(6)
```

```
starting number is 6

term 1: 6
term 2: 3
term 3: 10
term 4: 5
term 5: 16
term 6: 8
term 7: 4
term 8: 2

final term of 1 is reached, stopping time is 9
```

Technical note. The `counter` variable is first defined in the **parent scope** of the `while` loop (i.e. the `collatzConjectureWithTerm` function). Notice how it is possible for the loop to modify the value of `counter`. On a more general note, `while`, `for` and `try` can modify variables in their parent scope. It is, however, worth bearing in mind that the converse is not true: any variables defined within the `while` loop would instead be within a **local scope**, and thus cannot be accessed by the parent.

2.3 Repeated evaluation: for

Let's rewrite the function so that only the stopping time (i.e. the final value of `counter`) is returned.

```
function collatzConjectureFinalCount(x)
        @assert x > 0 "x should be a positive integer!"
        counter = 1
        while x != 1
                x = x%2==0 ? x/2 : 3x+1
                counter += 1
        end
        return counter
end

collatzConjectureFinalCount(6)
```

```
9
```

Let's use a `for` loop to obtain the stopping time for the first twenty numbers.

```
for i in 1:20
        @printf "%.0f, t = %.0f\n" i collatzConjectureFinalCount(i)
end
```

```
1, t = 1
2, t = 2
3, t = 8
4, t = 3
5, t = 6
6, t = 9
7, t = 17
8, t = 4
9, t = 20
10, t = 7
11, t = 15
12, t = 10
```

```
13, t = 10
14, t = 18
15, t = 18
16, t = 5
17, t = 13
18, t = 21
19, t = 21
20, t = 8
```

A neat one-liner to gather all these values in an Array is by a **list comprehension**:

```
counts = [collatzConjectureFinalCount(i) for i in 1:20];
```

The maximum stopping time for the first twenty numbers is `maximum(counts)`= 21, which occurs at initial values of 18 and 19, as found by `findall(counts .== maximum(counts))`.

In a later section, you will learn how to visualise the distribution of stopping times using plots.

2.4 Exercises

1. What is the stopping time for an initial value of 1?
2. If we were to allow the sequence to proceed even after reaching one, what would happen?
3. Suppose we modify the operations such that when we reach an odd number, we do $3x - 1$ (instead of plus one). What happens? Try this for initial values 1, 3 and 9. You can modify the code above (hint: use Ctrl+C to halt a program if necessary).

3 More advanced control flow

3.1 the break statement

From Question 3, one noticed that if the rule is modified such that we do $3x - 1$ (instead of plus one) when reaching an odd number, there are certain initial values which do not reach one and get trapped in a cycle. An example is 9:

$ 7,20,10,5,14,7,... $

If we simply modify the `3x+1` to `3x-1` in the `collatzConjecture` function and run it, the loop will run forever for these initial values because it never reaches one. A never halting program is not the coolest thing to have, so let's rewrite the function so that loop stops after a fixed number of iterations (set as 25 here). Here we use the `break` keyword in an `if` statement to terminate the `while` loop, so that it does not run forever.

```
function collatzConjectureWithCounterModified(x)
        @printf "starting number is %.0f\n\n" x
        counter = 1
        while x != 1
                if counter > 25
                        @printf "\nthe number 1 is not reached after 25 iterations,
final term is %.0f" x
                        break
```

```
            end
            @printf "term %.0f: %.0f\n" counter x
            x = x%2==0 ? x/2 : 3x-1
            counter += 1
        end
end

collatzConjectureWithCounterModified(9)
```

```
starting number is 9

term 1: 9
term 2: 26
term 3: 13
term 4: 38
term 5: 19
term 6: 56
term 7: 28
term 8: 14
term 9: 7
term 10: 20
term 11: 10
term 12: 5
term 13: 14
term 14: 7
term 15: 20
term 16: 10
term 17: 5
term 18: 14
term 19: 7
term 20: 20
term 21: 10
term 22: 5
term 23: 14
term 24: 7
term 25: 20

the number 1 is not reached after 25 iterations, final term is 10
```

3.1.1 Exercise

How many numbers from 1 to 20 manage to reach one using the modified rule? Tip: use a conditional statement to return True/False near the end of the function.

3.2 Dealing with exceptions with @assert

Those who are eagle-eyed would also notice that the original `collatzConjecture` would loop forever if given zero or a negative number. Ideally, we wouldn't want this function to run if the input were a negative integer. t should throw an **exception** and print an error message. We achieve this with the `@assert` macro.

```
function collatzConjectureWithAssert(x)
        @assert x > 0 "please give a positive integer"
        @printf "starting number is %.0f\n\n" x
        while x != 1
```