University of Strathclyde
Science

Lecture 16: Functions and Demons

Dr John Levine

CS103 Machines, Languages and Computation
November 26th 2015

The Second Half of the Class

- Assignment 6: Recursion and Languages: how to use recursion to allow for potentially unbounded sentences.
- Assignment 7: Recursion and Strings: how to specify an infinite set of strings using a finite set of rules.
- Assignment 8: Recursion and Logical Definitions: what does it mean to be someone’s ancestor?
- Assignment 9: The λ-calculus: how can we create and use computable functions using only symbols?
- Assignment 10: Recursion and Function Definitions: how can we use recursion to create functions?

The Second Half of the Class

- Assignment 6: Recursion and Languages: how to use recursion to allow for potentially unbounded sentences.
- Assignment 7: Recursion and Strings: how to specify an infinite set of strings using a finite set of rules.
- Assignment 8: Recursion and Logical Definitions: what does it mean to be someone's ancestor?
- Assignment 9: The λ-calculus: how can we create and use computable functions using only symbols?
- Assignment 10: Recursion and Function Definitions: how can we use recursion to create functions?

Writing Functions in Python

- To write a function in Python, we have to specify what output is to be returned for a given input:

```
def f1(x):
    return 3*x + 1
```

Writing Functions in Python

- To write a function in Python, we have to specify what output is to be returned for a given input:

```
def f1(x):
    return 3*x + 1
```

- We need to include the `return` statement for all possible outputs:

```
def f2(x):
    if x % 2 == 0:
        return x // 2
    else:
        return 3*x + 1
```

Computing using Lazy Demons

- Imagine an infinite line of lazy demons:

- Each demon will do one (and only one!) calculation for you – after that, they get tired and fall asleep
- How can you make the demons do proper calculations for you, like computing the factorial function?

Recursive Functions

- Recall the factorial function:

  factorial(7) = 7 x 6 x 5 x 4 x 3 x 2 x 1 = 5040

- To get a line of lazy demons to calculate the factorial function, we can do it like this:

  Rule 1: if I give you 0, then return 1.

  Rule 2: if I give you an integer, n, then ask the demon one down the line to calculate the factorial of n-1, and then multiply that value by n.

Factorial Function in Python

- In Python, we can code the two rules for the factorial function like this:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

- We say that this Python function is recursive because a call to the function occurs within the function itself
- Recursive code is *code which calls itself*

Factorial Function in Python

- In Python, we can code the two rules for the factorial function like this:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

- We say that this Python function is recursive because a call to the function occurs within the function itself
- Recursive code is *code which calls itself*

Another Recursive Function

- In Python, lists are used to hold an arbitrary number of objects of any type – e.g. ['spam', 42, 'fish', True]
- How can we get our demons to calculate the length of a list of items?

Rule 1: if I give you the empty list, return 0.

Rule 2: if I give you a non-empty list, then discard the first element, ask the demon one down the line to give you the length of the resulting list, then add 1 to that.

Length Function in Python

- In Python, we can code the two rules for the length function like this:

```
def length(l):
    if l == []:
        return 0
    else:
        return 1 + length(l[1:])
```

- The code `l[1:]` will return the list `l` with the first item removed – so if `l` is `[1,2,3]`, `l[1:]` will return `[2,3]`

Length Function in Python

- In Python, we can code the two rules for the length function like this:

```
def length(l):
    if l == []:
        return 0
    else:
        return 1 + length(l[1:])
```

- The code `l[1:]` will return the list `l` with the first item removed – so if `l` is `[1,2,3]`, `l[1:]` will return `[2,3]`

Calling Diagram for Length

- We use a *calling diagram* to show how `length` works:

Calling Diagram for Length

- We use a *calling diagram* to show how `length` works:

```
length([1, 2, 3, 4])
```

Calling Diagram for Length

- We use a *calling diagram* to show how `length` works:

```
length([1, 2, 3, 4])
=> 1 + length([2, 3, 4])
```

Calling Diagram for Length

- We use a *calling diagram* to show how `length` works:

```
length([1, 2, 3, 4])
=> 1 + length([2, 3, 4])
=> 1 + 1 + length([3, 4])
```

Calling Diagram for Length

- We use a *calling diagram* to show how `length` works:

```
length([1, 2, 3, 4])
=> 1 + length([2, 3, 4])
=> 1 + 1 + length([3, 4])
=> 1 + 1 + 1 + length([4])
```

Calling Diagram for Length

- We use a *calling diagram* to show how `length` works:

```
length([1, 2, 3, 4])
=> 1 + length([2, 3, 4])
=> 1 + 1 + length([3, 4])
=> 1 + 1 + 1 + length([4])
=> 1 + 1 + 1 + 1 + length([])
```

Calling Diagram for Length

- We use a *calling diagram* to show how `length` works:

```
length([1, 2, 3, 4])
=> 1 + length([2, 3, 4])
=> 1 + 1 + length([3, 4])
=> 1 + 1 + 1 + length([4])
=> 1 + 1 + 1 + 1 + length([])
=> 1 + 1 + 1 + 1 + 0
```

Calling Diagram for Length

- We use a *calling diagram* to show how `length` works:

```
length([1, 2, 3, 4])
=> 1 + length([2, 3, 4])
=> 1 + 1 + length([3, 4])
=> 1 + 1 + 1 + length([4])
=> 1 + 1 + 1 + 1 + length([])
=> 1 + 1 + 1 + 1 + 0
=> 4
```

Another Example

- I want to count all the occurrences of 'fish' in a list: `count_fish(['spam', 'fish', 3, 'fish']) => 2`

- How can we get our demons to do this?

Rule 1: if I give you the empty list, return 0.

Rule 2: if I give you a list which starts with 'fish', discard the first item, ask the demon one down the line to count the fish in resulting list, then add 1 to that.

Rule 3: if I give you any other list, discard the first item, ask the demon one down the line to count the fish in the resulting list, then return that number.

Python exercises to try

- Write the Python code for `count_fish(l)`
- Write a *recursive* version of `collatz(n)`

✓ Reminder: no lecture tomorrow!

✓ Next lecture on Monday 30th November

✓ Tutorials on Thursday 3rd December

✓ Class Test on Friday 4th December