Recursion

Rahul Narain

COL100: Introduction to Computer Science
I Semester, 2021-22

Announcements

- Labs will continue in same meeting links from last week
- Assignment 1 will (hopefully) be posted today

Review: Control flow in functions and conditionals

```
def mean(a, b):
    print('before return')
    return (a + b)/2
    print('after return')

four = mean(3, 5)
```

```
a = False
b = True
c = True

if a:
    print('a is true')
elif b:
    print('b is true')
elif c:
    print('c is true')
else:
    print('none are true')
```

By the way, it's a good idea to document your code using *comments*:

```
def mean(a, b):
    # The mean of two numbers.
    #
    # Parameters:
    # a: float -- the first number
    # b: float -- the second number
    #
    # Returns: float -- the mean of a and b

    print('before return')
    return (a + b)/2       # This is the actual calculation
    # Nothing after this point will run
    print('after return')
```

Conditionals again

The code in the branches can include anything, including other `ifs`:

```
if condition1:
    if condition2:
        print('case A')
    else:
        print('case B')
else:
    if condition3:
        print('case C')
    else:
        print('case D')
```

Can visualize all these possibilities as a *decision tree*

```
include math

def solveQuadratic(a, b, c):
    # The roots (x1, x2) of the quadratic ax^2 + bx + c = 0,
    # or None if the roots are complex.
    d = b**2 - 4*a*c
    if d < 0:
        # Roots are complex
        return None
    else:
        # Roots are real
        rootd = math.sqrt(d)
        x1 = (-b + rootd)/(2*a)
        x2 = (-b - rootd)/(2*a)
        return (x1, x2)
```

User input

You can prompt the user to enter some text using the `input` function.

```
name = input('What\'s your name? ')
print('Nice to meet you,' name)
```

The input is always a `string`. To interpret it as a number, use the `int` or `float` functions.

```
in1 = input('Enter a number: ')
in2 = input('Enter another number: ')
a = int(in1)
b = int(in2)
print('Their sum is', a + b, 'and their product is', a*b)
```

Pop quiz

So far all our computations can only take a limited number of steps…

```
n = int(input('Enter a positive integer: '))
if n < 10:
    print('Your number has 1 digit.')
elif n < 100:
    print('Your number has 2 digits.')
elif n < 1000:
    print('Your number has 3 digits.')
else:
    # Tired of typing :(
    print('Your number has more than 3 digits.')
```

How to define a computation that can take as many steps as needed?

Suppose you are inventing mathematics from the ground up. Let's assume that so far you have defined the natural numbers $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, equality, addition, and subtraction.

How to define multiplication and division in terms of these operations?

In particular, given two natural numbers a and b, what is $a \times b$?

Thinking recursively

"A journey of a thousand miles begins with a single step."
—Tao Te Ching

To figure out how to do arbitrarily many steps, break the problem down into (i) one step, and (ii) the rest of the steps.

$$
\begin{aligned}
a \times b &= \underbrace{a + a + a + \cdots + a}_{b \text{ times}} \\
&= a + \underbrace{a + a + \cdots + a}_{b-1 \text{ times}} \\
&= a + a \times (b - 1).
\end{aligned}
$$

Think of this not as a *property*, but as a candidate *definition* of multiplication. Can we calculate 4×3 using only this?

$$a \times b \stackrel{?}{=} a + a \times (b - 1).$$

$$\begin{aligned}
4 \times 3 &= 4 + 4 \times 2 \\
&= 4 + (4 + 4 \times 1) \\
&= 4 + (4 + (4 + 4 \times 0)) \\
&= 4 + (4 + (4 + (4 + 4 \times \underbrace{(0 - 1)}_{???})))
\end{aligned}$$

$$a \times b \stackrel{?}{=} a + a \times (b-1).$$

We also need a *base case*: When $b = 0$, we can stop because we know $a \times 0 = 0$ always. (Anyway $b - 1$ is not even in $\mathbb{N}$ when $b = 0$.)

$$a \times b = \begin{cases} 0 & \text{if } b = 0, \\ a + a \times (b-1) & \text{otherwise.} \end{cases}$$

I have now defined multiplication in terms of more basic operations... and multiplication itself! This is called *recursion*.

$$a \times b = \begin{cases} 0 & \text{if } b = 0, \\ a + a \times (b - 1) & \text{otherwise.} \end{cases}$$

Can implement directly in Python:

```
def mult(a, b):
    # The product of two natural numbers a and b,
    # computed via repeated addition.
    if b == 0:
        # Base case
        return 0
    else:
        # Recursive case
        return a + mult(a, b - 1)
```

What happens on the stack when calling `mult(4,3)`?

The factorial function

$$n! = 1 \times 2 \times \cdots \times (n-1) \times n$$

How to define this recursively?

One way:

$$\begin{aligned} n! &= \big(1 \times 2 \times \cdots \times (n-1)\big) \times n \\ &= (n-1)! \times n \end{aligned}$$

Is this enough?

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{otherwise.} \end{cases}$$

```
def fact(n):
    # The factorial of a natural number n.
    if n == 0:
        return 1
    else:
        return fact(n - 1) * n
```

There isn't always just one way to do it!

$$\begin{aligned}
n! &= 1 \times \big(2 \times 3 \cdots \times (n-1) \times n\big) \\
&= 1 \times \Big(2 \times \big(3 \times \ldots \cdots \times (n-1) \times n\big)\Big) \\
&\vdots
\end{aligned}$$

Let's introduce a function $p(a, b) = a \times (a+1) \times \cdots \times (b-1) \times b$. Then

$$\begin{aligned}
n! &= p(1, n), \\
p(a, b) &\stackrel{?}{=} a \times p(a+1, b).
\end{aligned}$$

Exercises: Figure out a good base case for the definition of p.

Does your definition result in $0! = 1$? If not, fix it.

The Fibonacci sequence

Consider the following sequence of numbers:

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ \ldots$$

After the first two numbers, every number is the sum of the previous two numbers.

This is also a recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F_{n-1} + F_{n-2} & \text{otherwise.} \end{cases}$$

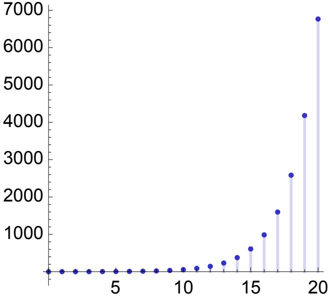
Exercise: Write a program to compute the 10th, 20th, and 30th Fibonacci numbers.

```
def fib(n):
    # The nth Fibonacci number.
    # [Write this code yourself :)]

print(fib(10))
print(fib(20))
print(fib(30))
```

You should get 55, 6765, and 832040.

These numbers are growing exponentially!



Let's move away from Python for a bit, and consider a real-world example.

Dictionaries contain words in alphabetical order. How do you look up a word in a dictionary? Can you describe it as an algorithm?

Bisection

What's the key idea?

- We know that the solution we are looking for is in a particular range.
- We can check a point inside that range to determine whether the solution is to the left or right of it.
- So, we can make our range smaller, and repeat the process...
  recursively!

Practice exercises

- Give a recursive definition of x^n in terms of multiplication, where x is any real number and n is any natural number. Implement and test it as a function `power(x,n)` in Python.