Introduction to Recursion

What is Recursion?
Divide-conquer-glue...
...where the sub-problems involve the problem itself!

With recursion, the solution to a problem depends on solutions to smaller instances of the same problem

A recursive function is a function that invokes ____________.
Review: functions calling other functions
(in anticipation of writing functions that call themselves)

Which would work? Why/why not?

```python
def print2(s):
    print s
    print s

def print4(s):
    print2(s)
    print2(s)

print4('okay')
```

Our first recursive function:

**countDown**

Let's write a function that prints the integers from \( n \) down to 1 (without using loops):

```python
def countDown(n):
    '''Prints integers from \( n \) down to 1'''
    if n < 1:
        pass  # Do nothing
    else:
        print n
        countDown(n-1)
```

**countDown**: Base Case

The base case. When is the problem so simple that we can solve it trivially and we needn't decompose it into subproblems.

```python
def countDown(n):
    '''Prints integers from \( n \) down to 1'''
    if n < 1:
        pass  # Do nothing
```

**countDown**: Recursive Case

The recursive case. For all instances of the problem not covered by the base case, we'll decompose the problem into subproblems, at least one of which is a smaller instance of the countDown problem and can be solved by invoking the countDown function.

```python
def countDown(n):
    '''Prints integers from \( n \) down to 1'''
    if n < 1:
        pass  # Do nothing
    else:
        print n
        countDown(n-1)
```

```python
def countDownImplicitBase(n):
    '''Prints integers from \( n \) down to 1'''
    if n > 0:
        print n
        countDownImplicitBase(n-1)
```
Structure of Recursion

All recursive functions must have two types of cases:

- **BASE case**: a simple case where the result is so simple, it can just be returned. In this case the function does not invoke itself, since there is no need to decompose the problem into subproblems.

- **RECURSIVE case**: a case where the problem is decomposed into subproblems at least one of the subproblems is solved by invoking the function being defined, i.e., the function is invoked in its own body. You should assume the recursive function works correctly for the smaller subproblems (this is known as “wishful thinking”)

Review: function call frames

Recursion isn’t magic. It works because of the frame model for functions we introduced back in Lecture 03.

Invocation of `countDown(3)`

Anatomy of function call frames

```python
def countDown(n):
    '''Prints integers from n down to 1'''
    if n>0:
        print(n)
        countDown(n-1)
```

12-9

Review: function call frames

Recursion isn’t magic. It works because of the frame model for functions we introduced back in Lecture 03.

Invocation of `countDown(3)`

Anatomy of function call frames

```python
def countDown(n):
    '''Prints integers from n down to 1'''
    if n>0:
        print(n)
        countDown(n-1)
```

12-10

Invocation of `countDown(3)`

Anatomy of function call frames

```python
def countDown(n):
    '''Prints integers from n down to 1'''
    if n>0:
        print(n)
        countDown(n-1)
```

12-11

Invocation of `countDown(3)`

Anatomy of function call frames

```python
def countDown(n):
    '''Prints integers from n down to 1'''
    if n>0:
        print(n)
        countDown(n-1)
```

12-12
Recursion

**GOTCHA! #1:**

The problem that you are solving recursively must get smaller each time you recur, i.e., you must get closer to the base case. Otherwise, the recursion will not terminate -- a so-called infinite recursion.

```python
def countDown(n):
    if n < 1:  # Base case
        pass  # Do nothing
    else:  # Recursive case
        print n
countDown(n)
```

Recursion

**GOTCHA! #2:**

The recursion must eventually reach a base case in order to end. If it doesn't, that's another way to get an infinite recursion.

```
def countDown(n):
    print n
countDown(n-1)
```

"Maximum recursion depth exceeded"

In practice, the infinite recursion examples will terminate when Python runs out of resources for creating function call frames, leading to a maximum recursion depth exceeded error message:

```
In [2]: countDown(3)
  3
  2
  1
  0
-1
-2
-3
...
RuntimeError: maximum recursion depth exceeded while calling a Python object foo
```

**tower**

Write a function that prints a tower using the characters of the input string `name`. The width of the tower should start with `len(name)` characters down to the last character.

```python
def tower(name):
    '''Prints a tower based on the string name from all len(name) characters down to the last character'''
    In [6]: tower('Wellesley')
    Wellesley
    ellesley
    lesley
    esley
    sley
    ley
    ey
    y
```

What does this function do?

```python
def mystery(n):
    if n < 1:
        pass
    else:
        mystery(n - 1)
        print n
```

What does `mystery(3)` print?
**Invocation of mystery(3)**

```python
def mystery(n):
    '''Prints integers from 1 to n'''
    if n>0:
        mystery(n-1)
    print(n)
```

---

**countDownUp**

Let's write a function that prints the integers from \( n \) down to 1 and then from 1 up to \( n \):

```python
def countDownUp(n):
    '''Prints integers from \( n \) down to 1 and then from 1 up to \( n \)'''
    if n < 1:
        pass  #Do nothing
    else:
        mystery(n-1)
        print(n)
```

---

**Target Practice (concentric circles)**

Let's draw PS04 `cslgraphics` “targets” using recursion:

```python
def drawDisc(canvas, x, y, r, color):
    '''Draw a circle centered at point (x,y) on the canvas with given radius r and color'''
    c = Circle(radius,Point(x,y))
    c.setFillColor(color)
    canvas.add(c)

def drawDiscInvocations:
    pass
```

```python
paper = Canvas(400, 300, 'darkolivegreen4', 'drawDisc')
drawDisc(paper, 100, 175, 75, 'blue')
drawDisc(paper, 250, 75, 50, 'red')
drawDisc(paper, 275, 225, 35, 'yellow')
paper.show()
```
**drawTarget: base case?**

```python
def drawTarget(canvas, x, y, radius, thickness, color1, color2):
    '''On the specified canvas, draws a bullseye target with the given radius, centered at (x,y) with alternating colors, color1 and color2, where color1 is the outermost color; thickness is the width of each "band" in the ring; thickness is also the minimum radius of a drawn circle.''
```

**drawTarget: recursive case?**

Hint: how can we decompose the problem into two subproblems such that one of the subproblems involves drawing a target?

**Turtles: a mini crash-course**

(we’ll make fun drawings with them)

Python has a built-in module named turtle.

```python
from turtle import *
setup(400,400)
shape('turtle')
pencolor('blue')
pensize(2)
fd(150)
lt(90)
fd(100)
rt(30)
bk(150)
exitonclick()
```

You’ll experiment with turtles in lab this week.

---

**Define drawNestedCircles**

```python
def drawNestedCircles(canvas, x, y, radius, minRadius, color1, color2):
    '''On the specified canvas, draws a nested circle pattern. The largest circle is centered at (x,y) with the given radius and is filled with color1. There are two nested subpatterns, each with half the radius, internally tangent along the main horizontal diameter, with colors alternating between color2 and color1. No circle whose radius is less than minRadius is drawn.''
```

---

Python has a built-in module named turtle.