Recursion

Announcements

Recursive Functions

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=def%20fact%28n%29%3A%0A%20%20%20%20if%20n%20%3D%3D%200%20or%20n%20%3D%3D%201%3A%0A%20%20%20%20%20%20return%201%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20return%20fact%28n-1%29%20*%20n%0A%0Afact%285%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Countdown

```
def countdown(n):
    if n == 0:
        print('Blastoff!')
    else:
        print(n)
        countdown(n-1)

countdown(10)
```

```
# What's different here?
def countdown(n):
    if n == 0:
        print('Blastoff!')
    else:
        countdown(n-1)
        print(n)

countdown(10)
```

Recursive Process

1: **Divide** – Break the problem down into smaller parts.

2: **Invoke** – Make the actual recursive call.

3. **Combine** – Use the result of the recursive call in your result.

```
def fact(n):
    """Compute n factorial.

    >>> fact(5)
    120
    >>> fact(0)
    1
    """
    if n == 0 or n == 1:
        return 1
    else:
        return fact(n-1) * n
```

Discussion Question: Factorial Two Ways

```
Rewrite fact(n) so that the result of fact(5) is computed using the following steps:
    5 (1 * 5)
   20 (1 * 5 * 4)
   60 (1 * 5 * 4 * 3)
  120 (1 * 5 * 4 * 3 * 2)
```

```
def fact(n):
    """Compute n factorial.

    >>> fact(5)
    120
    >>> fact(0)
    1
    """
    if n == 0 or n == 1:
        return 1
    else:
        return fact(n-1) * n
```

Recursion Visualizer

https://recursionvisualizer.com

View fact(10)

```
def fact(n):
    """Compute n factorial.

    >>> fact(5)
    120
    >>> fact(0)
    1
    """
    if n == 0 or n == 1:
        return 1
    else:
        return fact(n-1) * n
```

View fact(10)

Tracing Functions

```
from ucb import trace # download ucb.py
```

```
@trace
def fact(n):
   if n == 0 or n == 1:
        return 1
    else:
        return fact(n-1) * n

# or

fact = trace(fact)
```

(Optional)
Recursive Functions

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=def%20fact%28n%29%3A%0A%20%20%20%20if%20n%20%3D%3D%200%20or%20n%20%3D%3D%201%3A%0A%20%20%20%20%20%20%20%20return%201%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20return%20fact%28n-1%29%20*%20n%0A%0Afact%285%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Example: Add Up Some Numbers (Fall 2016 Midterm 1 Question 5)

Implement add_up, which takes a positive integer k. It returns a function that can be called repeatedly k times, one integer argument at a time, and returns the sum of these arguments after k repeated calls.

```
def add_up(k):
    """Add up k numbers after k repeated calls.

    >>> add_up(4)(10)(20)(30)(40)  # Add up 4 numbers: 10 + 20 + 30 + 40
    100
    """
    assert k > 0
    def f(n):
        if k == 1:
            return n
        else:
            return lambda t: add_up(k - 1) (n + t)
    return f
```

add_up(4)(10) returns a one-arg function & needs to remember 3 & 10

add_up(4) returns a one-arg function & needs to remember the 4

Evaluates to a one-arg function that adds k-2 more numbers to n + t

Discussion Question: Play Twenty-One

Rewrite play as a recursive function without a while statement.

- Do you need to define a new inner function? Why or why not? If so, what are its arguments?
- What is the base case and what is returned for the base case?

```
def play(strategy0, strategy1, goal=21):
    """Play twenty-one and return the winner.

    >>> play(two_strat, two_strat)
    1
    """
    n = 0
    who = 0  # Player 0 goes first
    while n < goal:
        if who == 0:
            n = n + strategy0(n)
            who = 1
        elif who == 1:
            n = n + strategy1(n)
            who = 0
    return who
```

```
def play(strategy0, strategy1, goal=21):
    """Play twenty-one and return the winner.

    >>> play(two_strat, two_strat)
    1
    """
    def f(n, who):
        if n >= goal:
            return who
        if who == 0:
            n = n + strategy0(n)
            who = 1
        elif who == 1:
            n = n + strategy1(n)
            who = 0
        return f(n, who)
    return f(0, 0)
```