Lecture #8: More on Functions

Announcements

- We strongly suggest that you watch discussion orientations before attending tutorials.
- The first set of grades has been released on howamidoing.cs61a.org. Regrade requests can be submitted on links.cs61a.org/okpy-regrades. howamidoing will be updated with new scores once or twice a week, usually on Fridays.
- Ask questions on the Piazza thread for today's lecture (@676).

The Towers of Hanoi

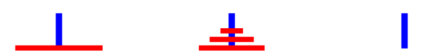
- The Towers of Hanoi is a familiar puzzle.
- There are three pegs holding piles of flat disks of different sizes.
- Initially, all disks are on the first peg, piled in decreasing order of size.

- The goal is to move all disks to the third peg.

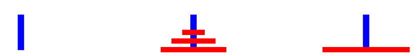
- Only the top disk of one pile may be moved at a time.
- It must be moved to an empty peg, or to a peg whose top disk is larger.

Strategy for Solving Towers of Hanoi

- Moving a tower consisting of a single disk is, of course, immediate, and forms the base case.
- The crucial insight is that to move the top N disks from a starting peg to a goal peg, we can first move the top $N-1$ from the first peg to the remaining (spare) peg



- Then move the remaining (largest) disk to the goal



- And finally move the disks on the spare peg to the goal:

- This all works as long as we are careful to arrange that on each move, the spare peg contains only disks larger than the ones we're moving.

Specification and Strategy

- First, what exactly are we trying to do?

```
def move_tower(n, start_peg, end_peg):
    """Perform moves that transfer an ordered tower of N>0 disks in the
    Towers of Hanoi puzzle from peg START_PEG to peg END_PEG, where
    1 <= START_PEG, END_PEG <= 3, and START_PEG != END_PEG. Assumes
    the disks to be moved are all smaller than those on the other pegs."""
```

- Our strategy is:

  0. If $N = 1$, just move the one disk. Otherwise,
  1. First move $N - 1$ disks off the start peg to the spare peg.
  2. Second, move the now-uncovered N^{th} disk to the end peg.
  3. Finally, move $N - 1$ disks from the spare peg to the end peg.

- To do the actual moving (step 0), let's assume the existence of a `move_disk(p0, p1)` function that moves the top disk from peg `p0` to peg `p1`.
- Our strategy translates almost directly to a recursive function.

The Program

0. If $N = 1$, just move the one disk. Otherwise,
1. First move $N - 1$ disks off the start peg to the spare peg.
2. Second, move the now-uncovered N^{th} disk to the end peg.
3. Finally, move $N - 1$ disks from the spare peg to the end peg.

```
def move_tower(n, start_peg, end_peg):
    """Perform moves that transfer an ordered tower of N>0 disks in the
    Towers of Hanoi puzzle from peg START_PEG to peg END_PEG, where
    1 <= START_PEG, END_PEG <= 3, and START_PEG != END_PEG. Assumes
    the disks to be moved are all smaller than those on the other pegs."""

    if n == 1:
        ??
    else:
        ??
```


The Program

0. If $N = 1$, just move the one disk. Otherwise,
1. First move $N - 1$ disks off the start peg to the spare peg.
2. Second, move the now-uncovered N^{th} disk to the end peg.
3. Finally, move $N - 1$ disks from the spare peg to the end peg.

```
def move_tower(n, start_peg, end_peg):
    """Perform moves that transfer an ordered tower of N>0 disks in the
    Towers of Hanoi puzzle from peg START_PEG to peg END_PEG, where
    1 <= START_PEG, END_PEG <= 3, and START_PEG != END_PEG. Assumes
    the disks to be moved are all smaller than those on the other pegs."""

    if n == 1:
        move_disk(start_peg, end_peg)
    else:
        ??
```


The Program

0. If $N = 1$, just move the one disk. Otherwise,
1. First move $N - 1$ disks off the start peg to the spare peg.
2. Second, move the now-uncovered N^{th} disk to the end peg.
3. Finally, move $N - 1$ disks from the spare peg to the end peg.

```
def move_tower(n, start_peg, end_peg):
    """Perform moves that transfer an ordered tower of N>0 disks in the
    Towers of Hanoi puzzle from peg START_PEG to peg END_PEG, where
    1 <= START_PEG, END_PEG <= 3, and START_PEG != END_PEG. Assumes
    the disks to be moved are all smaller than those on the other pegs."""

    if n == 1:
        move_disk(start_peg, end_peg)
    else:
        spare_peg = ??
        ??
```

The Program

0. If $N = 1$, just move the one disk. Otherwise,
1. First move $N - 1$ disks off the start peg to the spare peg.
2. Second, move the now-uncovered N^{th} disk to the end peg.
3. Finally, move $N - 1$ disks from the spare peg to the end peg.

```
def move_tower(n, start_peg, end_peg):
    """Perform moves that transfer an ordered tower of N>0 disks in the
    Towers of Hanoi puzzle from peg START_PEG to peg END_PEG, where
    1 <= START_PEG, END_PEG <= 3, and START_PEG != END_PEG. Assumes
    the disks to be moved are all smaller than those on the other pegs."""

    if n == 1:
        move_disk(start_peg, end_peg)
    else:
        spare_peg = 6 - start_peg - end_peg  # Why does this work?
        ??
```


The Program

0. If $N = 1$, just move the one disk. Otherwise,
1. First move $N - 1$ disks off the start peg to the spare peg.
2. Second, move the now-uncovered N^{th} disk to the end peg.
3. Finally, move $N - 1$ disks from the spare peg to the end peg.

```
def move_tower(n, start_peg, end_peg):
    """Perform moves that transfer an ordered tower of N>0 disks in the
    Towers of Hanoi puzzle from peg START_PEG to peg END_PEG, where
    1 <= START_PEG, END_PEG <= 3, and START_PEG != END_PEG. Assumes
    the disks to be moved are all smaller than those on the other pegs."""

    if n == 1:
        move_disk(start_peg, end_peg)
    else:
        spare_peg = 6 - start_peg - end_peg
        move_tower(n - 1, start_peg, spare_peg)
        move_disk(start_peg, end_peg)
        move_tower(n - 1, spare_peg, end_peg)
```


Semi-Philosophical Interlude on Preconditions

- Many of our comments contain precondtions, such as

```
    """Perform moves that transfer an ordered tower of N>0 disks in the
    Towers of Hanoi puzzle from peg START_PEG to peg END_PEG, where
    1 <= START_PEG, END_PEG <= 3, and START_PEG != END_PEG. Assumes the
disks to be moved are all smaller than those on the other pegs."""
```

- Here, the red portions indicate preconditions: conditions the caller (the "client") must meet before the function is guaranteed to work.
- So what's supposed to happen if they aren't met?
- Clearly, the function might just not work.
- But if that's all we say, then `move_tower` would technically correct if it deleted all the client's files when $N \leq 0$.
- It would be nice, if feasible, for the implementer to do something more useful and informative.

Exceptions

- A pretty standard language feature to help with this sort of problem is the *exception*.
- An exception is a value that indicates that something "exceptional" has happened.
- Certainly errors, such as arguments not in accord with preconditions, at least *should* be exceptional!
- Python has other uses for its exceptions, but that's another topic for another lecture.
- Operations on exceptions include control statements that abruptly terminate a computation, and allow the programmer to take corrective action.

Raise

- To indicate an exception, a program *raises an exception*, which in Python means creating an exception value and applying the **raise** statement to it. For example,

```
if N <= 0:
    raise ValueError("Number of disks must be positive")
```

- The expression after **raise** creates a kind of exception value (the `ValueError` type is conventially used to indicate an improper value.)
- Many built-in Python expressions and statements do this internally to indicate, among other things:
  - Division by 0.
  - Infinite recursions,
  - Attempts to add numbers to things that aren't.

[Demo]

Try

- When you anticipate an exception might occur, and have a more useful response than blowing up, you can *catch* a raised exception using a **try** statement.
- For example:

```
try:
    input = open(myfile).read()
except FileNotFoundError:  # Another standard exception
    print("Warning: could not open", myfile)
    input = ""
```

- This tries to read the contents of an input file into the variable `input`. If that file does not exist, it substitutes the empty string.

[Demo]

Exercise: Removing Digits

- Problem: I'd like to define a function that removes all instances of a particular digit (0–9) from a given number.
- For example, I'd like to have

```
remove_digit(3141592653589793, 5) == 3141926389793
```

- A few useful tips for fiddling with non-negative integers:
  - The last digit of `N` is `N % 10`.
  - All but the last digit of `N` is `N // 10`, if `N > 9`.