CIS 1100

Recursion Cont.

Python
Fall 2024
University of Pennsylvania

Any Quetsions?

Any questions from last time?

Otherwise I plan to jump right in

Practice: (S7-S10)

```
def foo(N):
    if N <= 1:
        return 0
    return 1 + foo(N // 2) # Hint: // is integer division

def fizz(N):
    if N == 0:
        return 0
    return N % 10 + fizz(N // 10)
```

What do these print?

- (S7) `print(foo(3))`
- (S8) `print(foo(16))`
- (S9) `print(fizz(1100))`
- (S19) `print(fizz(8675309))`

Practice:

We want to write a function `sum_numbers` that gets the sum of all integers in a list recursively. Do this in (C12)

First think:

- What are the base case(s) ?
- What is the little bit of work done on each step?
- What do we tell the function to do recursively?

Hint: You may want to use list slicing

```
def sum_numbers(nums_list):
    # TODO
    # return a sum of all the numbers in the list
    # your soln doesn't have to be perfect, remember that this is practice
```

Practice: (L13)

We want to write the funcion `ping_pong(N)` which prints "ping" and then "pong" in alternating order for a total of N prints.

`ping_pong(3)` prints: `ping` then `pong` then `ping`
`ping_pong(2)` prings: `ping` then `pong`
`ping_pong(1)` prings: `ping`

(L13) Does this code work? Why or why not:

```
def ping_pong(N):
    if N <= 0:
        return
    if N % 2 == 1:
        print("ping")
    else
        print("pong")
    ping_pong(N - 1)
```

Helper functions

Sometimes to do recrusion we need to remember a bit more information than is provdided to the overall problem.

In this case, what other information do we need for `ping_pong(N)` to get a working recursive solution?

Why can't we just recursively call `ping_pong(N-1)`?

Practice (C14)

We want to write the funcion `ping_pong(N)` which prints "ping" and then "pong" in alternating order for a total of N prints.

`ping_pong(3)` prints: `ping` then `pong` then `ping`
`ping_pong(2)` prings: `ping` then `pong`
`ping_pong(1)` prings: `ping`

(C14) finish writing the fixed version:

```
def ping_pong_helper(N, extra):
    # TODO: do something here

def ping_pong(N):
    ping_pong_helper(N, ______)  # You probably want to pass in either 0 or a boolean here
```

Practice: (L15)

Consider we want to write the function `find_factors(N)` that returns a set containing all positive factors of the input integer `N`.

A number `x` is a factor of `N` if and only if `N % x == 0`

NOTE: This is a different problem than one you will see on the homework called `gcd`

Before writing any code (L15)

- What is the base case?
- What is the recursive case?
- What is the work done on each step?

Helper functions

Sometimes to do recrusion we need to remember a bit more information than is provdided to the overall problem.

In this case, what other information do we need for `find_factors(N)` to get a working recursive solution?

Why can't we just recursively call `find_factors(N-1)`?

Practice: (C16)

Finish writing this function:

```
def find_factors_helper(current, N):
    # TODO: What do you put here?
    # need to return a set of all factors of N

def find_factors(N):
    return find_factors_helper(1, N)
```