Intro to Coding with Python– Recursion Pt 2

Dr. Ab Mosca (they/them)

Slides based off slides courtesy of Jordan Crouser (https://jcrouser.github.io/)

Plan for Today

- Tough problems, simple solutions
- More Recursion & Recursive Functions
  - Finding the Largest in a List
  - Finding the Smallest in a List
  - Traversing a Maze
  - Fractal Trees

Basic structure of a recursive algorithm

- **A base case**: what to do in the simplest possible case (i.e. when you have a single disk)
- **A recursive step**: break the original problem into one or more smaller problems, and solve that (saving the intermediate result)

Recursion themes

- "Looping **without a loop**"
- "A function that **calls itself** as part of its definition"
- "Solving a problem by solving **smaller instances**"
- Key components of all three:
  - a recursive step (i.e. knowing when to split)
  - a "base case" (i.e. knowing when to stop)

Recap: recursive functions (Hanoi)

*hanoi.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noona...

```
def moveTower(nDisks, s, e, h):
    if height >= 1:
        moveTower(nDisks-1, s, h, e)
        moveDisk(s, e)
        moveTower(nDisks-1, h, e, s)

def moveDisk(s, e):
    print("moving disk from", s, "to", e)



moveTower(3,"A","B","C")
```

Ln: 11 Col: 24

Recap: recursive functions (Hanoi)

*hanoi.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noona...

```
def moveTower(nDisks, s, e, h):
    if height >= 1:
        moveTower(nDisks-1, s, h, e)
        moveDisk(s, e)
        moveTower(nDisks-1, h, e, s)

def moveDisk(s, e):
    print("moving disk from", s, "to", e)



moveTower(3,"A","B","C")
```

Ln: 11 Col: 24

Discussion

What actually happens **in memory** when you **call a function**?

the "stack"

the program

```
def f1(a):
      y = f2(a+1)
      return y

def f2(b):
      z = b
      if (z > 2):
            z = z/2
      return b^2

f1(3)
```

in memory

the “stack”

the program

```
def f1(a):
      y = f2(a+1)
      return y

def f2(b):
      z = b
      if (z > 2):
            z = z/2
      return b^2

f1(3)
```

in memory

the "stack"

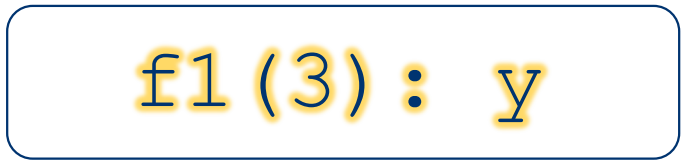
the program

```
def f1(a):
        y = f2(a+1)
        return y

def f2(b):
        z = b
        if (z > 2):
                z = z/2
        return b^2

f1(3)
```

in memory



the “stack”

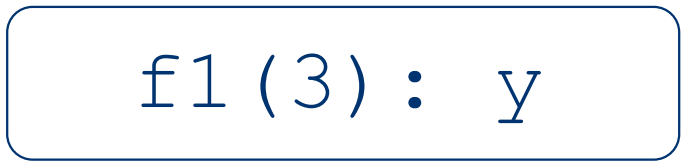
the program

```
def f1(a):
        y = f2(a+1)
        return y

def f2(b):
        z = b
        if (z > 2):
                z = z/2
        return b^2

f1(3)
```

in memory



the "stack"

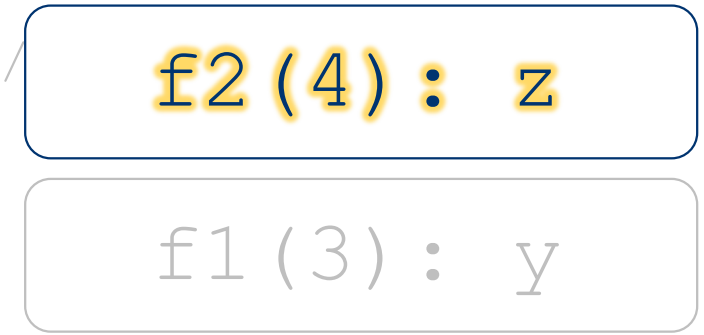
the program

```
def f1(a):
        y = f2(a+1)
        return y

def f2(b):
        z = b
        if (z > 2):
                z = z/
        return b^2

f1(3)
```

in memory



the “stack”

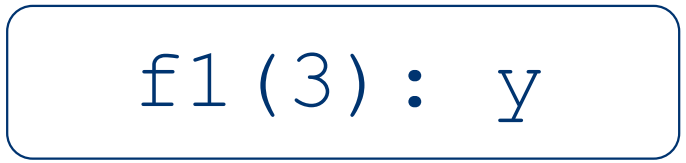
the program

```
def f1(a):
        y = f2(a+1)
        return y

def f2(b):
        z = b
        if (z > 2):
                z = z/2
        return b^2

f1(3)
```

in memory



the "stack"

the program

```
def f1(a):
        y = f2(a+1)
        return y

def f2(b):
        z = b
        if (z > 2):
                z = z/2
        return b^2

f1(3)
```

in memory

…whatever's next!

Discussion

What actually happens **in memory** when you **call a function recursively**?

the "stack"

the program

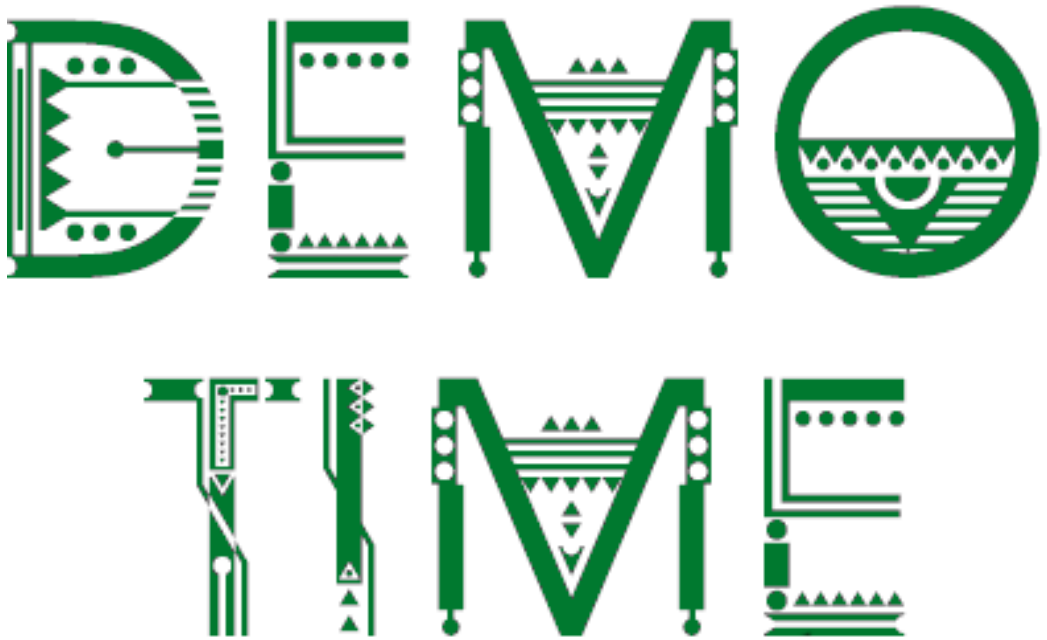
```
def f3(a):
    if (a == 1):
        return 1
    else:
        x = f3(a-1)+1

f3(100)
```

in memory

…but isn't there **limited space**?

Demo:
recursive
addition



Recursive vs. iterative addition

```
def recursiveSum(lst):
    if len(lst) == 2:
        return lst[0]+lst[1]
    else:
        return lst[0]+recursiveSum(lst[1:])

def regularSum(lst):
    sum = 0
    for num in lst:
        sum += num
    return sum
```

Discussion

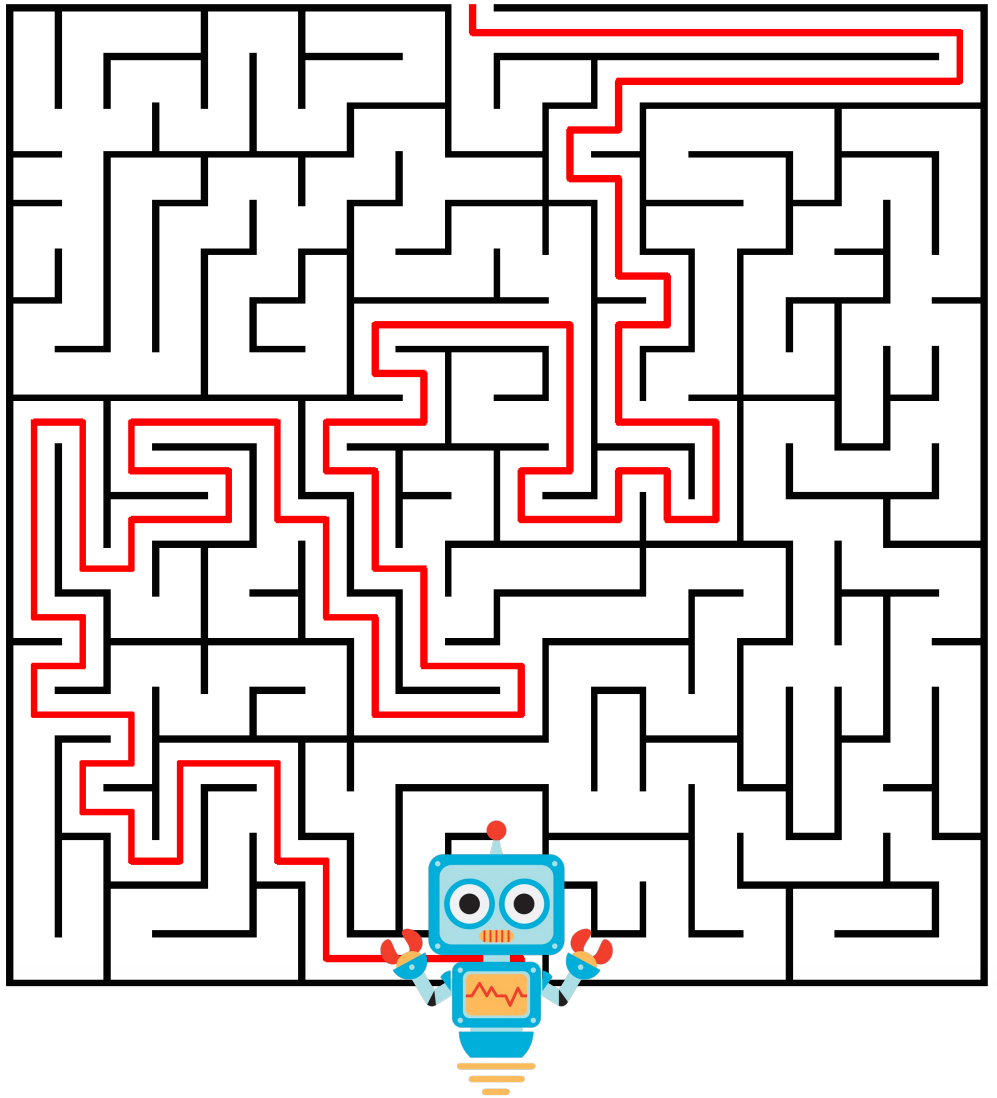
How would you solve Tower of Hanoi **iteratively**?

```
def moveTower(nDisks, s, e, h):
    if height >= 1:
        moveTower(nDisks-1, s, h, e)
        moveDisk(s, e)
        moveTower(nDisks-1, h, e, s)
```

More problems with recursive solutions

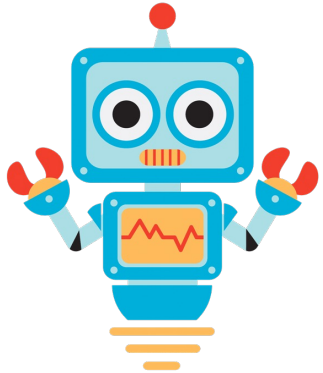
15 minute (non-programming) Challenge:

How would you program a **robot** to **solve a maze**?



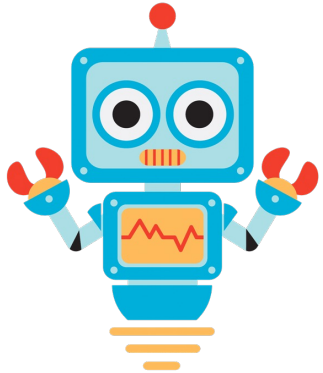
A recursive solution

1. Mark your current location as **visited**
2. If you're at the **end**, you're done!
3. If not:
   a. If unmarked, go NORTH, solve maze. If not solved, go back and:
   b. If unmarked, go SOUTH, solve maze. If not solved, go back and:
   c. If unmarked, go EAST, solve maze. If not solved, go back and:
   d. If unmarked, go WEST, solve maze. If not solved, NO SOLUTION



Clever recursion allows backtracking!

1. Mark your current location as **visited**
2. If you're at the **end**, you're done!
3. If not:
   a. If unmarked, go NORTH, solve maze. If not solved, go back and:
   b. If unmarked, go SOUTH, solve maze. If not solved, go back and:
   c. If unmarked, go EAST, solve maze. If not solved, go back and:
   d. If unmarked, go WEST, solve maze. If not solved, NO SOLUTION



Discussion

What is the
most confusing thing
about recursion?