Recursion

Outline

- Functions ↔ Python programs
- Function outputs as sequences
- Recursive definition of functions
- Recursive Python programs

Motivation

Why recursion?

- Some programming languages are built entirely around recursive structures
- Some functions, sets, or sequences are best represented via recursion
- Helpful representation for proving things about your functions

f(n) = n

```
1   def f(n: int) -> int:
2       return n
```

Sequence of outputs for n ≥ 0:

Recursive Definition of a Function

- Calling a function within itself, typically with a smaller input.
- Two components:
  - Base case(s)
    - Where recursion *ends*
    - Often smallest input(s)
    - Prevent infinite loops!
  - Recursive Rule
    - Definition to handle all inputs that aren't base case.
    - Expresses function in terms of smaller calls to the function.
      - (e.g. expressing f(n) in terms of f(n-1))

f(n) = n

Input	0	1	2	3	4	5	6	…	n
Output	0	1	2	3	4	5	6	…	f(n)

Recursive definition:

- Base case:

- Recursive rule:

In Python

```python
def f(n: int) -> int:
    if n == 0: # base case
        return 0
    else: # recursive rule
        return 1 + f(n-1)
```

In Python

```python
def f(n: int) -> int:
    if n == 0: # base case
        return 0
    else: # recursive rule
        return 1 + f(n-1)

f(2)
```

Summary

- Recursion is another way of defining functions
- Helpful to represent it as a sequence of inputs/outputs to get an idea of the recursive rule

More on Recursion

Goal

- Define the function f(n,b) = n + b, *recursively on n*
- Steps
  - Write out sequence of input/outputs
  - Use sequence to determine recursive definition
  - Translate recursive definition into Python program