COMP 110

Recursion

First… review

```
def divide(num1: int, num2: int) -> float:

divide(num1 = 11, num2 = 3)
```

These are called keyword arguments, since you are assigning values based on the parameter names.

Keyword arguments

```
def divide(num1: int, num2: int) -> float:

divide(num1 = 11, num2 = 3)
```

Benefit of keyword arguments: order of arguments doesn't matter.

Keyword arguments

```python
def divide(num1: int, num2: int) -> float:

divide(num1 = 11, num2 = 3)
divide(num2 = 3, num1 = 11)
```

Benefit of keyword arguments: order of arguments doesn’t matter.

Positional Arguments

```
def divide(num1: int, num2: int) -> float:


divide(11, 3)
```

For positional arguments, values are assigned based on the order (*position*) of the arguments.

Variables

Declaration of a variable

```
<name>: <type> = <value>
students: int = 300
message: str = "Howdy!"
```

Update a variable

```
<name> = <new value>
students = 325
message = "See ya!"
```

COMP 110

Recursion

Outline

- 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

$f(n) = n$

```python
def f(n: int) -> int:
    return n
```

Sequence of outputs for $n \geq 0$:

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:
        return 0 # base case
    else:
        return 1 + f(n=n-1) # recursive rule
```

In Python

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

f(n=2)
```

Summary

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

Define a Function Recursively

- 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

Recap

From standard definition to recursive definition.

Steps:

- Standard function
- Sequence representation
- Recursive definition
- Recursive Python function

Recap

We started with standard function definition and followed steps to define it as a recursive python program.

Steps:

- Standard function
- Sequence representation
- Recursive definition
- Recursive Python function

$$f(n) = n$$

Recap

We started with standard function definition and followed steps to define it as a recursive python program.

Steps:

- Standard function
- Sequence representation
- Recursive definition
- Recursive Python function

Input n	0	1	2	3	…	n
Output f(n)	0	1	2	3	…	n

Recap

We started with standard function definition and followed steps to define it as a recursive python program.

Steps:

- Standard function
- Sequence representation
- Recursive definition
- Recursive Python function

- Base case:
  - for $n = 0$: $f(n) = 0$
- Recursive rule:
  - for $n > 0$, $f(n) = f(n-1) + 1$

Recap

We started with standard function definition and followed steps to define it as a recursive python program.

Steps:

- Standard function
- Sequence representation
- Recursive definition
- Recursive Python function

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

In the Other Direction…

- Start with a recursive python program and find out the standard function representation that it is describing.

Do it in Reverse

- Start with recursive Python function
- From that, get the recursive definition
- From that, get the sequence representation
- From that, get the standard definition

Do it in Reverse

- Start with recursive Python function
- From that, get the recursive definition
- From that, get the sequence representation
- From that, get the standard definition

```python
def mystery(n: int) -> int:
    if n == 0:
        return 1
    else:
        return 2 * mystery(n-1)
```

```python
def mystery(n: int) -> int:
    if n == 0:
        return 1
    else:
        return 2 * mystery(n-1)
```