Recursive Code

Chapter 4 Notes

- Real World Examples help you understand

factorials review:

$\boxed{n! = n\cdot(n-1)\cdot(n-2)\cdots 3\cdot 2\cdot 1}$ $\leftrightarrow$ $n! = n\cdot(n-1)!$

$4! = 4\cdot 3\cdot 2\cdot 1 = 24$

$2! = 2\cdot 1 = 2$

$1! = 1$

$0! = 1$

$(n-1)! = (n-1)\cdot(n-2)\cdots 3\cdot 2\cdot 1$

$n! = n\cdot(n-1)!$

$2! = 2\cdot(2-1)! = 2\cdot 1! = 2\cdot 1 = \underline{2}$

$1! = 1\cdot(1-1)! = 1\cdot 0! = 1\cdot 1 = \underline{1}$

$0! = 0(0-1)! = 0\cdot(-1)!$

NOT Defined

$$n! = \begin{cases} n\cdot(n-1)! & \text{if } n \geq 1 \\ 1 & \text{otherwise (if } n=0) \end{cases}$$

- Two seperate Cases for n!

How would one solve 3!

$3! = 3\cdot 2!$ 6 ~ ≥ 1

~ Plug back in here

$\underline{2!} = 2\cdot 1!$ 2 ~ ≥ 1

~ Plug back in here

$1! = 1\cdot \underline{0!}$

We know $0!$ is $\underline{1}$ by definition.

1

This Kind of thinking Comes back.

$$n! = \begin{cases} n\cdot(n-1)! & \text{if } n \geq 1 \\ 1 & \text{otherwise (if } n=0) \end{cases}$$

Python:

```python
def fact(n):
    """Assume that n is a positive integer or 0"""

    if n >= 1:
        return n * fact(n - 1)
    else:
        return 1
```

← recursive b/c this function calls itself

← Base case

← Value of 0!

Examples:

`print("0! =", fact(0))` → 1

What if we called fact(4)?

Return Value

$\text{fact}(0) \rightarrow \underbrace{1}$

$\text{fact}(1) \rightarrow \underbrace{1 \cdot f(0)} \approx 1\cdot 1 = 1$ ~ Now we know this value

$\text{fact}(2) \rightarrow \underbrace{2 * f(1)} \approx 2\cdot 1 = 2$

$\text{fact}(3) \rightarrow \underbrace{3 * f(2)} \approx 3\cdot 2 = 6$

fact(4) $\qquad \text{fact}(4) \rightarrow \underbrace{4 * \underbrace{\text{fact}(3)}_{\text{We need to call } f(3) \text{ to find the return value}}}_{\text{return value}} = 4 \cdot 6 = 24$

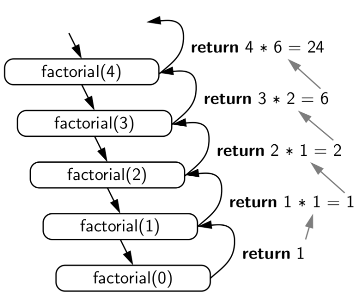

Figure 4.1: A recursion trace for the call factorial(5).

Another Example:

Fibonacci Sequence

Two Ones at the begining → 1 · 1 · 2 · 3 · 5 · 8

- Then the #'s after that are generated after that by adding up the previous two #s.

Problem:

1 · 1 · 2 · 3 · 5 · 8

$F_n = F_{n-1} + F_{n-2}$

$F_5 = F_4 + F_3$

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n \geq 3 \\ 1 & \text{Otherwise (if } n=1 \text{ or } 2) \end{cases}$$

```
def  fib(n):
        "Assuming n is a positive integer"
        if n >= 3:
            return fib(n-1) + fib(n-2)     ← Recursive (Calls itself)
        else:
            return 1                       ← Base case
```

```
def fib(n):
    "Assuming n is a positive integer"
    if n >= 3:
        return fib(n-1) + fib(n-2)
    else:
        return 1
```

Example:

fib(2) → 1   fib(1) → 1

fib(3) → fib(2) + fib(1) = 1+1

Return 2

~ Since 2 fails are base condition

fib(2) → 1   fib(1) → 1

fib(3) → fib(2) + fib(1)

fib(4) → fib(3) + fib(2)   fib(2) → 1

3   2 + 1

Actual Python Code:

```
def fact(n):
    # assuming that n is a positive integer or 0
    if n >= 1:
        return n * fact(n - 1)
    else:
        return 1

print("0! =", fact(0))
print("1! =", fact(1))
print("2! =", fact(2))
print("3! =", fact(3))
print("4! =", fact(4))


def fib(n):
    # assuming that n is a positive integer
    if n >= 3:
        return fib(n-1) + fib(n-2)
    else:
        return 1

print("fib(1) =", fib(1))
print("fib(2) =", fib(2))
print("fib(3) =", fib(3))
print("fib(4) =", fib(4))
print("fib(5) =", fib(5))



0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
```