

Lecture 33: Recursion

Recursion: When you are inside a function, calling that same function

```
def fun():  
    fun()
```

- Can be useful to formulate solutions to problems that would otherwise require lots of code.

When to use recursion?

- ★ When a solution to a problem can be expressed in terms of solutions to simpler versions of the same problem

Example: Computing the factorial of a number

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

n needs to be positive int ≥ 1 .

Define $0! = 1$

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

$$3! = 3 \cdot 2 \cdot 1 = 6 \quad \begin{array}{l} \text{4. factorial(3)} \\ \text{3!} \end{array}$$

factorial(4)

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 4 \cdot (3 \cdot 2 \cdot 1)$$

factorial(3)

$$3! = 3 \cdot 2 \cdot 1 = 3 \cdot (2 \cdot 1) \quad \left. \begin{array}{l} \text{2!} \\ \text{3. factorial(2)} \end{array} \right\}$$

factorial(2)

$$2! = 2 \cdot 1 = 2 \cdot (1) \quad \left. \begin{array}{l} \text{1!} \\ \text{2. factorial(1)} \end{array} \right\}$$

$$1! = 1 \cdot 1 = 1 \cdot (1) \quad \left. \begin{array}{l} \text{0!} \\ \text{1. factorial(0)} \end{array} \right\}$$

$0!$ defined as 1

"base case"
problem boils down to
a fixed number: 1

factorial(1)

Let's code up the function

Factorial Symbol table

print(factorial(3))

6

return 6

Mem

Name	Value
factorial	<fun>

factorial

Name	Value
n	3

$n \cdot \text{factorial}(n-1)$

$3 \cdot 2 = 6$

$3 \cdot \text{factorial}(2)$

2

factorial

Name	Value
n	2

$n \cdot \text{factorial}(n-1)$

$2 \cdot 1 = 2$

$2 \cdot \text{factorial}(1)$

1

factorial

Name	Value
n	1

$n \cdot \text{factorial}(n-1)$

$1 \cdot 1 = 1$

$1 \cdot \text{factorial}(0)$

1

After return
Symbol
table goes
away

factorial

Name	Value
n	0

return 1

then Symbol
table goes
away

X