Recursion

Recursive Algorithms

A recursive function is one that calls itself (either directly or indirectly).

A recursive algorithm is made up of two parts:

- **Base case:** the simplest version of the problem where recursion ends.
- **Recursive case:** simplifies the problem and uses recursion to solve the rest.

Advantages and Disadvantages of Recursion

Any problem that can be solved recursively can always be solved using iteration.

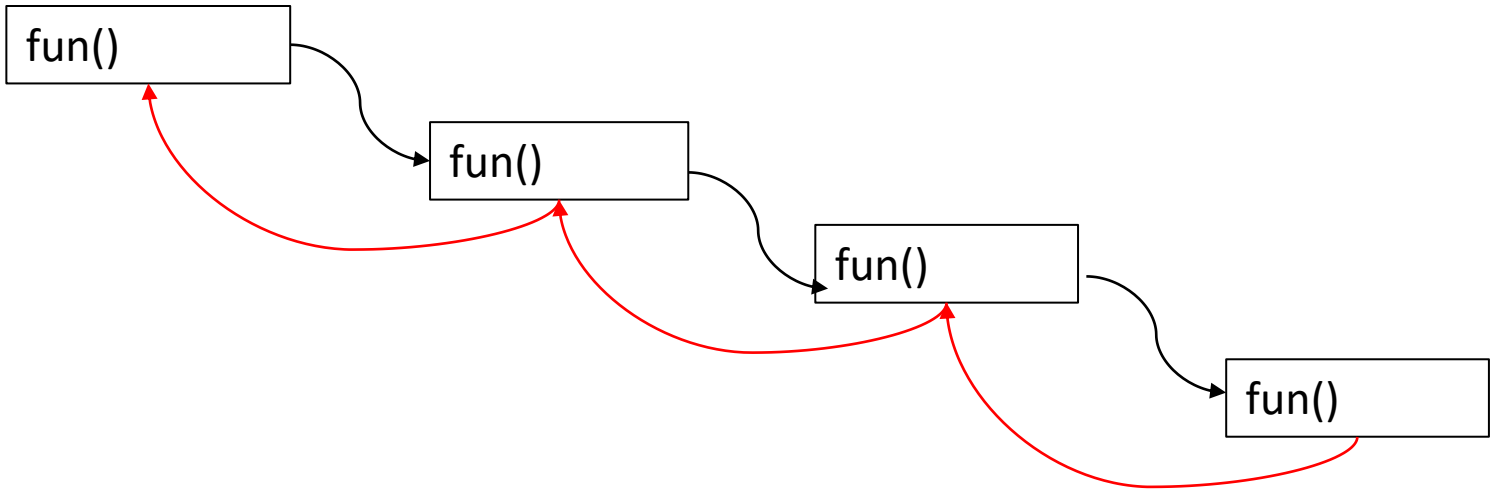
Recursion is generally slower and uses more memory, so why would we choose a recursive solution?

There are some problems for which the recursive solution is more obvious and understandable.

Recursive Function Calls

Each recursive call to a function creates its own scope/memory space.

When a recursive method returns a value, it returns it to its caller.



Finding the Factorial using Recursion

When finding the factorial of a number, the following holds true:

base case -> `1! = 1`

`0! = 1`

recursive case -> `n! = n * (n - 1)!`

base case

```
if n == 1 or n == 0
        fact = 1                    ->solved, no further steps necessary
```

recursive case

```
if n != 1
        fact = n * (n-1)!   -> use recursion to solve for (n-1)!
```

Factorial with Recursion

```
def factorial(n):
    if(n == 1 or n == 0):
        return 1
    else:
        return n * factorial(n - 1)
val = 5
fact = factorial(5)
print('The factorial of',str(val),'is',fact)
```

See: 08_factorialRecursion.py

Maximum List Example

Write a recursive function that finds and returns the longest string in a list of strings.

See: `08_listRecursion.py`

Palindrome Example

Write a recursive function that takes a word or sentence as a parameter and returns true if it is a palindrome, false if not. Case is not important.

See: `08_palindromeRecursion.py`

Terms of Use

- This presentation was adapted from lecture materials provided in MIT Introduction to Computer Science and Programming in Python.
- Licenced under terms of Creative Commons License.