

# Introduction to computer programming with python

## Lecture 10

Fall Semester, 2023

# Contents

- Functions recap
- recursion

# Functions - recap

See selection sort using list mutability and global scope under  
'functionsEGs.py'

# Recursion

Recursion is the process of repeating items in a self-similar way  
i.e., Reducing a problem into a smaller version of the same problem



# Recursion

**Algorithmically:** Designing solution by 'divide and conquer' strategy  
By reducing a problem to a simpler version of the same problem

**Semantically:** a programming technique where a function calls itself

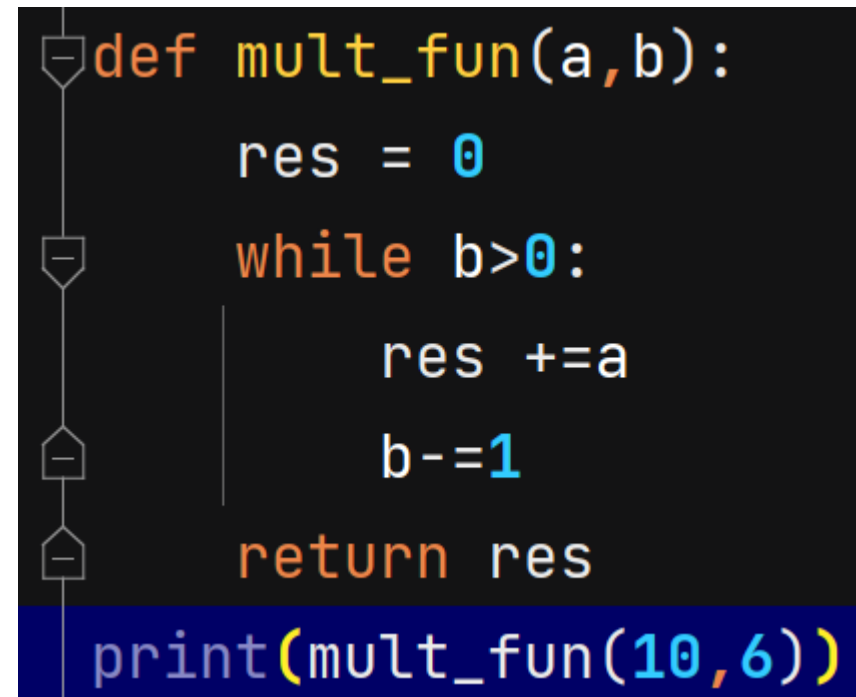
- The goal is to obtain final solution, and avoid infinite recursion
- The sub-problem must have 1 or more base cases that are easy to solve
- Solve the same problem on other input, to simplify the bigger problem

# Iterative Algorithms

**Looping constructs** (while/for loops ) are **iterative algorithms**

Iterative algorithms perform repeated computation on a set of state variables through loop operations

**Example:** calculate multiplicity  
 $a*b = a+a+a...+a$  (b times)



```
def mult_fun(a,b):  
    res = 0  
    while b>0:  
        res +=a  
        b-=1  
    return res  
print(mult_fun(10,6))
```

The image shows a code editor with a dark background. On the left side, there is a vertical line of four small, light-gray icons: a square with a minus sign, a diamond with a minus sign, a square with a minus sign, and a square with a minus sign. The code is written in a monospaced font with syntax highlighting: 'def' and 'return' are orange, 'while' is orange, 'print' is light blue, and function arguments and variable names are white. The code defines a function 'mult\_fun' that takes two arguments 'a' and 'b'. It initializes a variable 'res' to 0. It then enters a 'while' loop that continues as long as 'b' is greater than 0. Inside the loop, 'res' is incremented by 'a' ('res +=a') and 'b' is decremented by 1 ('b-=1'). After the loop, the function returns 'res'. Finally, the function is called with '10' and '6' as arguments, and the result is printed.

# Recursive Algorithms

**Example:** calculate multiplicity

$$\begin{aligned} a * b &= a + a + a \dots + a \text{ (b times)} \\ &= a + (a + a \dots + a) \text{ (b-1 times)} \\ &= a + a * (b-1) \end{aligned}$$

**Recursive step:**

Reduce the problem to a simpler/smaller version of the same problem

$$= a + \boxed{a * (b-1)} \leftarrow \text{Recursive reduction}$$

**Base case:**

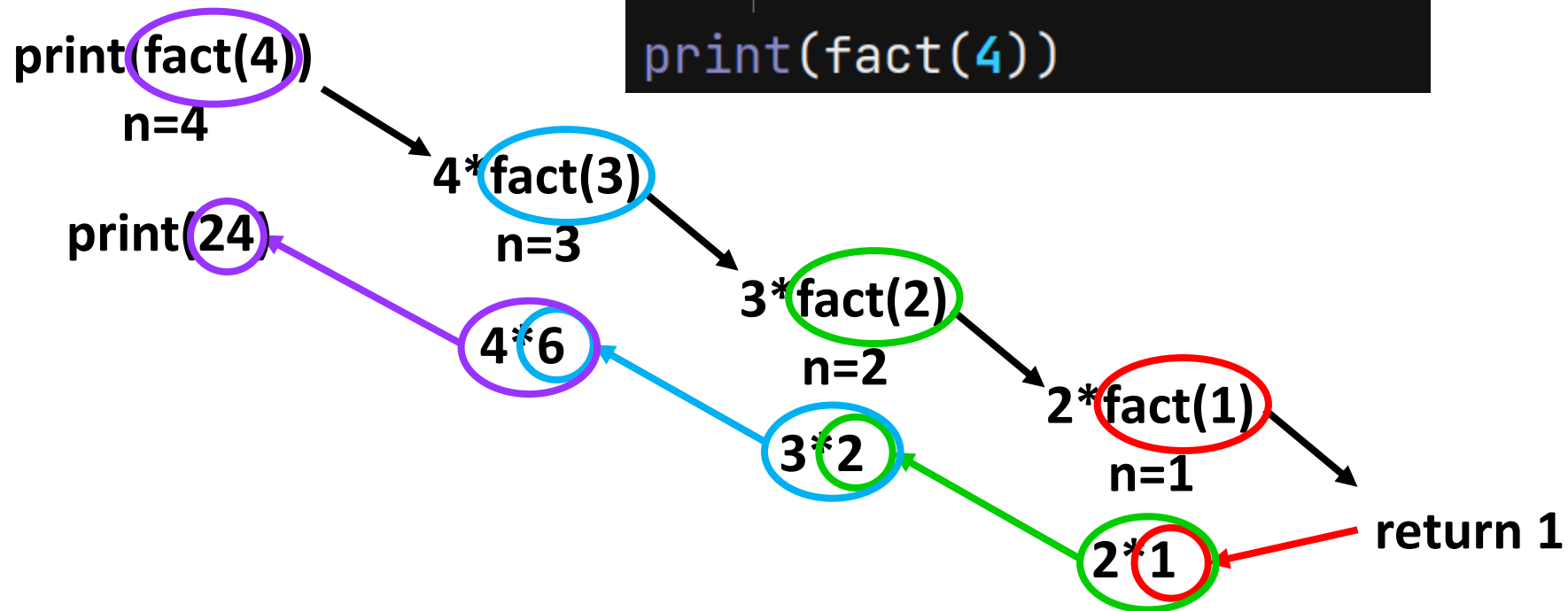
Keep reducing until reach a simple case that can be solved directly:

$$\boxed{\text{When } b=1, a * b = a} \leftarrow \text{Base case}$$

```
def mult_rec(a,b):  
    if b==1:  
        return a  
    else:  
        return a+mult_rec(a,b-1)  
print(mult_rec(10,6))
```

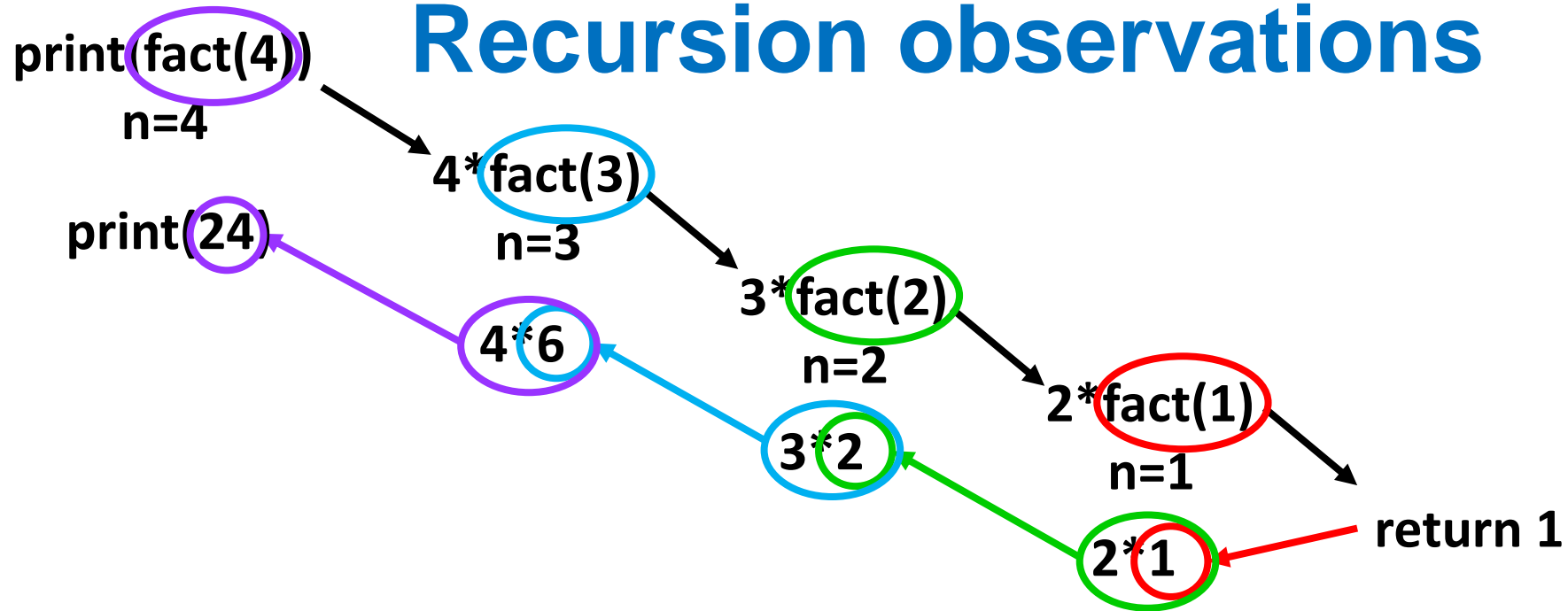
# Recursion execution demonstration

```
def fact(n):  
    if n==1:  
        return 1  
    else:  
        return n*fact(n-1)  
print(fact(4))
```





# Recursion observations



- Each recursive step creates its own scope/environment (use same variable names, but in separate scopes)
- Variable binding does not change by recursive call
- Flow control passes back to previous scope when function call returns a value

# Recursion vs. loop overview

```
def fact_it(n):  
    res=1  
    for i in range(1,n+1):  
        res*=i  
    return res  
print(fact_it(4))
```

```
def fact(n):  
    if n==1:  
        return 1  
    else:  
        return n*fact(n-1)  
print(fact(4))
```

# Fibonacci Series

## Recursion example with two base cases and two recursive calls

```
def fibonacci(x):  
    if x==0 or x==1:  
        return x  
    else:  
        return fibonacci(x-1)+fibonacci(x-2)
```

# Palindrome testing

## Recursion example with strings

```
def parseS(s):  
    return ''.join(s.lower().split(' '))  
  
def isPal(s):  
    if len(s) <= 1:  
        return True  
    else:  
        return s[0] == s[-1] and isPal(s[1:-1])  
  
print(isPal(parseS("Was it a car or a cat I saw")))
```

# Recursion exercise

## HW: Implement GCD with recursion

What is the greatest common divisor of two numbers, a and b?

Pseudo code:

$$a = q_0 b + r_0$$

$$b = q_1 r_0 + r_1$$

$$r_0 = q_2 r_1 + r_2$$

$$r_1 = q_3 r_2 + r_3$$

...

$$r_{k-2} = q_k r_{k-1} + r_k \Leftrightarrow r_k = r_{k-2} \% r_{k-1}$$

Until  $r_k = 0$  is reached.

The greatest common divisor is  $r_{k-1}$

For example,

For a = 1071 and b = 462

Step $k$	Equation	Quotient and remainder
0	$1071 = q_0 462 + r_0$	$q_0 = 2$ and $r_0 = 1071 \% 462 = 147$
1	$462 = q_1 147 + r_1$	$q_1 = 3$ and $r_1 = 462 \% 147 = 21$
2	$147 = q_2 21 + r_2$	$q_2 = 7$ and $r_2 = 147 \% 21 = 0$ ; algorithm ends