

CSCI 127: Joy and Beauty of Data

Lecture ~~12~~ 9: Recursion

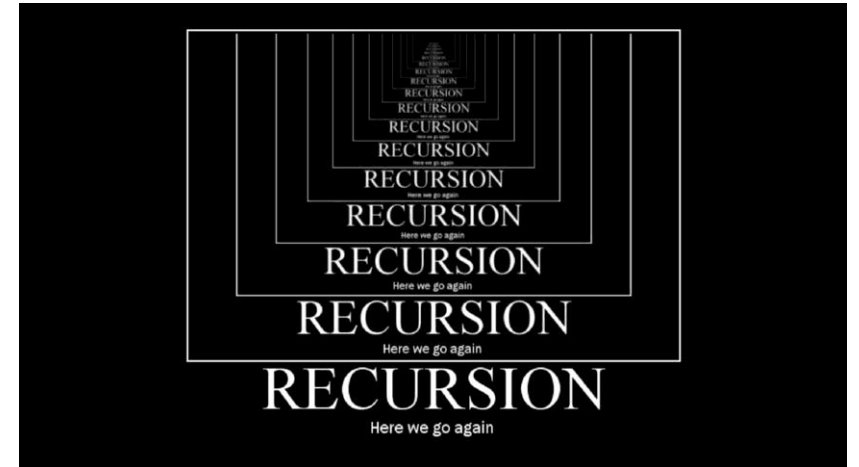
Reese Pearsall
Summer 2021

<https://reese.github.io/classes/summer2021/127/main.html>

Recursion

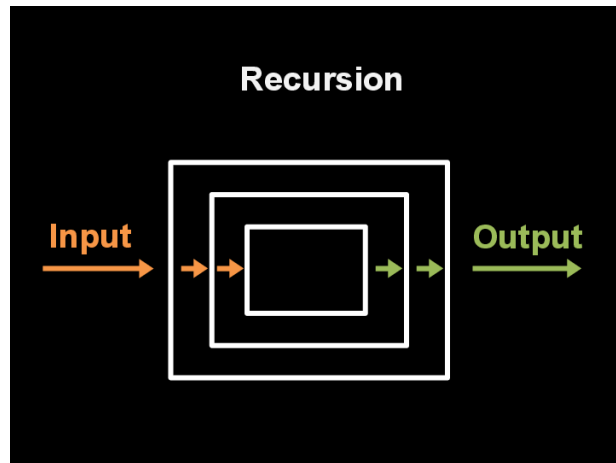
Recursion is a method of solving a problem that involves solving smaller instances of the same problem using the same function

Can be very challenging.....



Recursion Requirements

1. Base Case
 - The “stopping point” for your recursive calls
2. Recursive Case
 - Call the function again and solve a smaller problem



Factorial

```
factorial(5)
```

Factorial

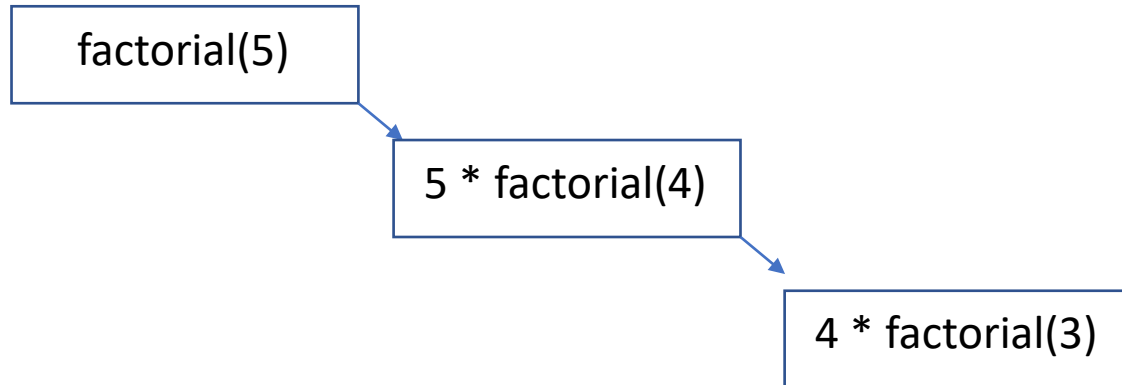
factorial(5)

5 * factorial(4)

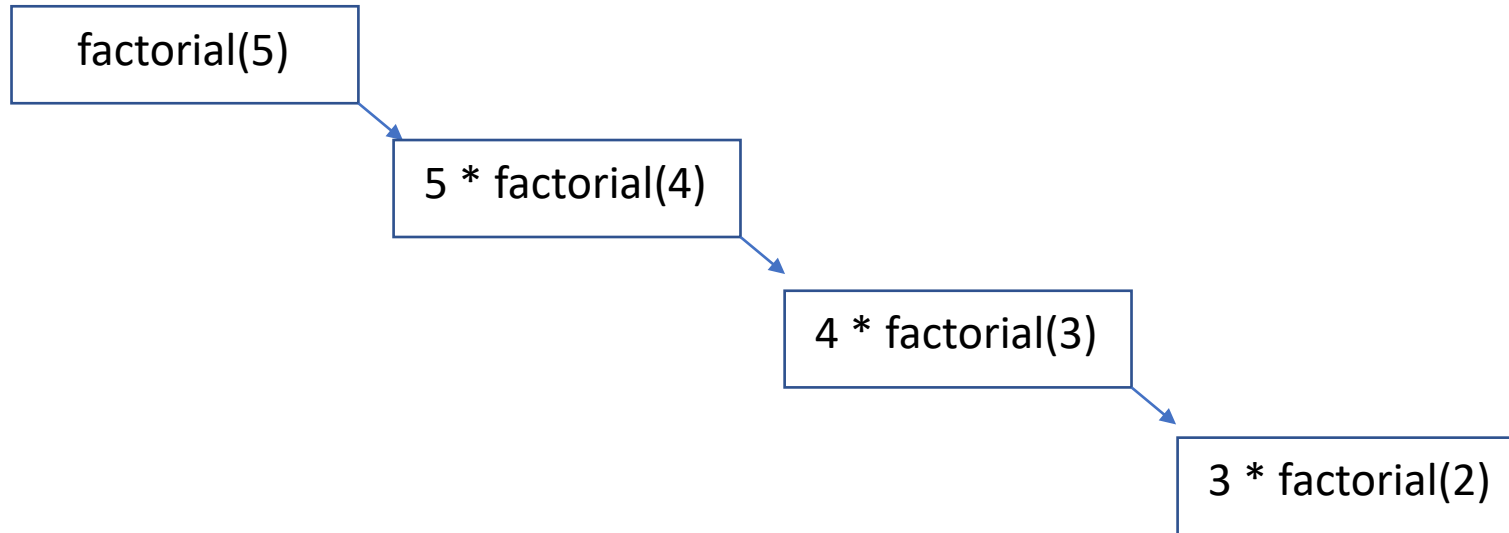


```
graph LR; A[factorial(5)] --> B[5 * factorial(4)]
```

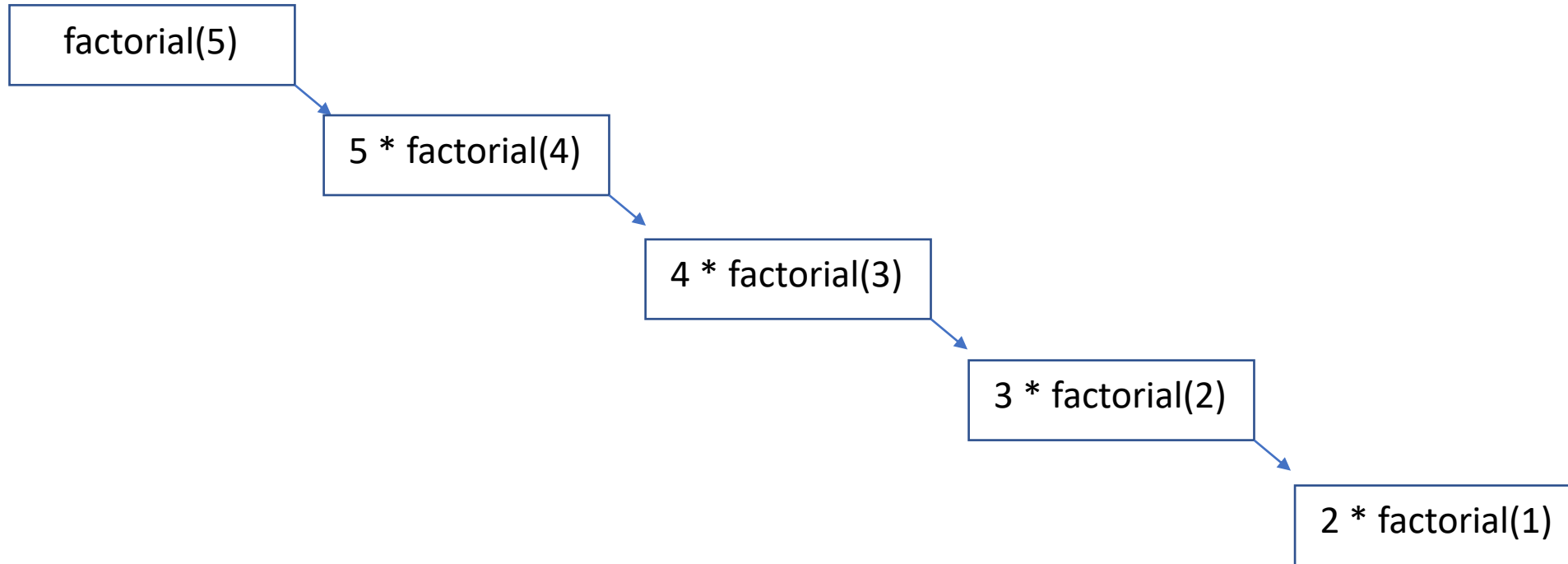
Factorial



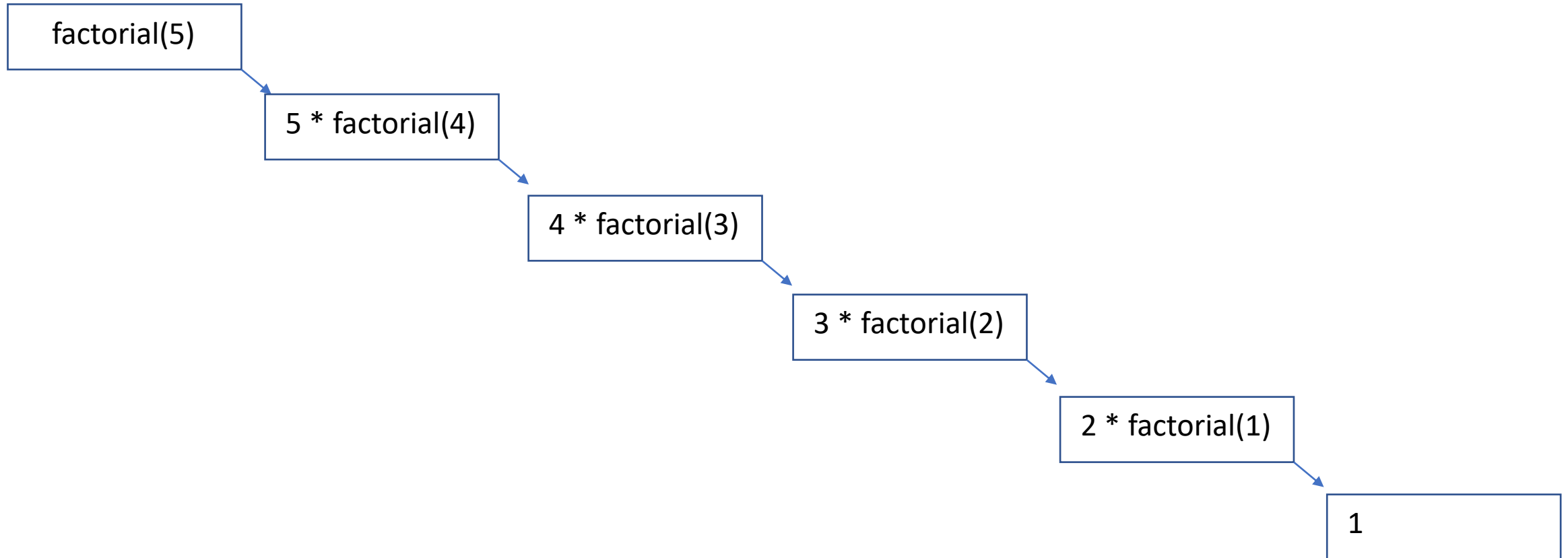
Factorial



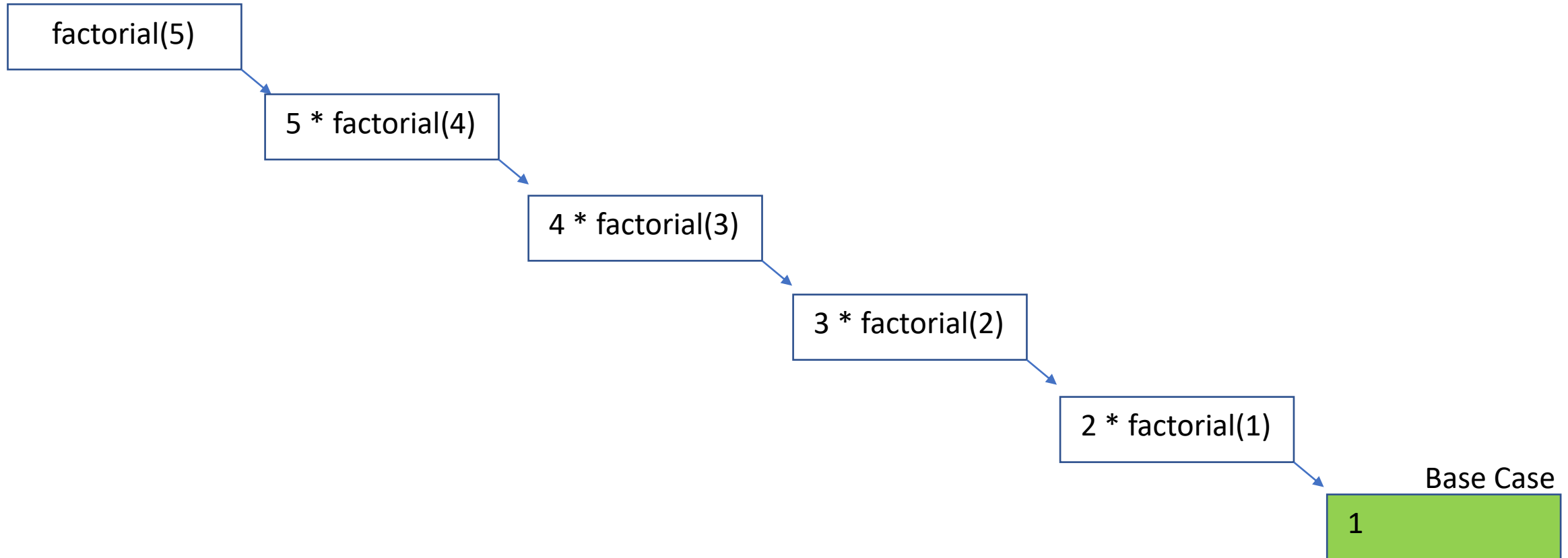
Factorial



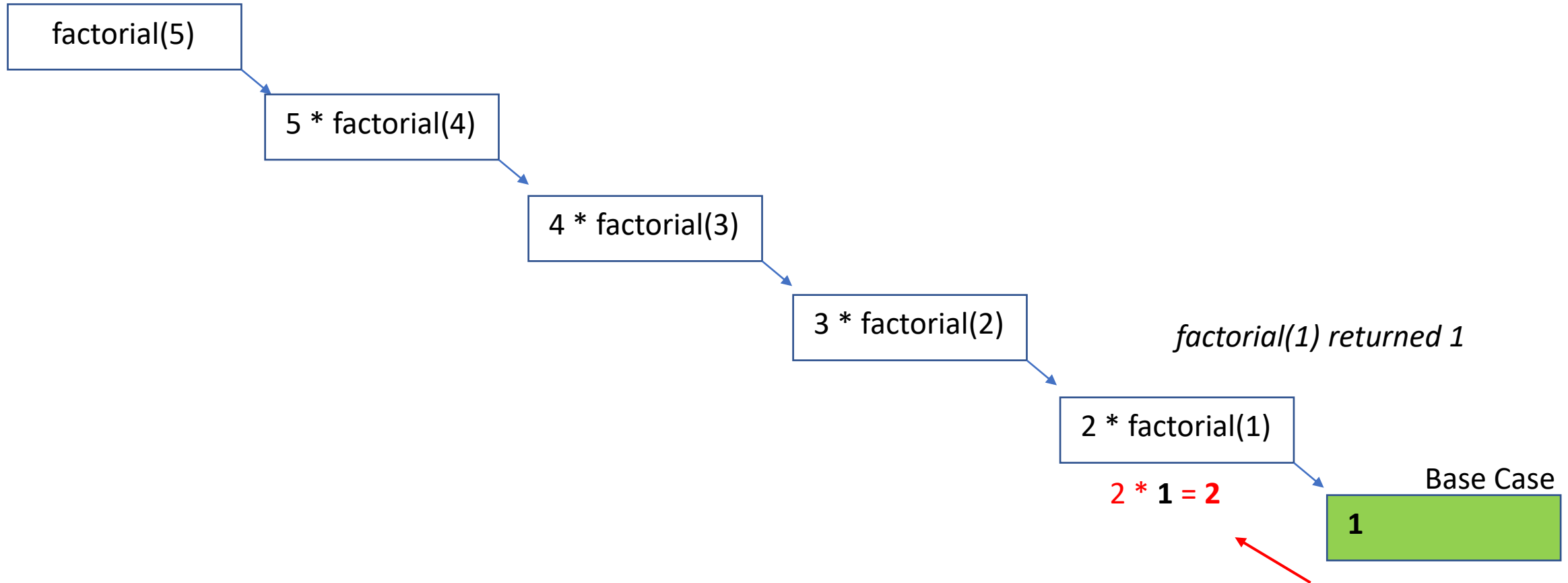
Factorial



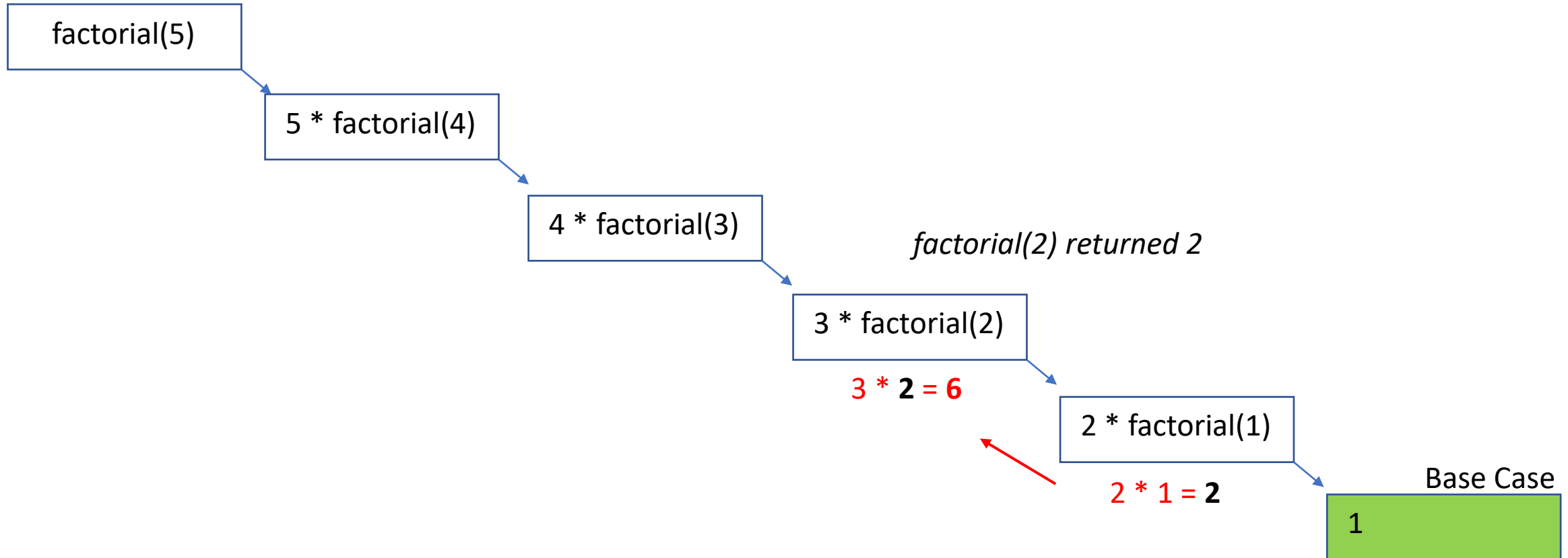
Factorial



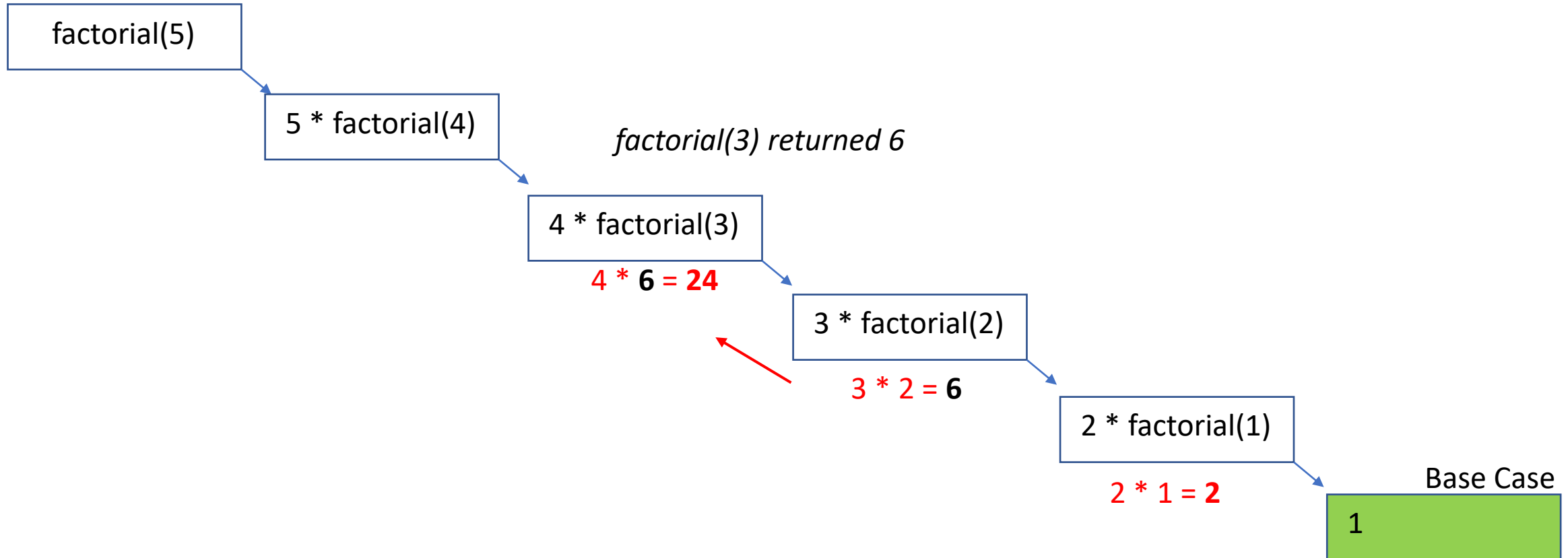
Factorial



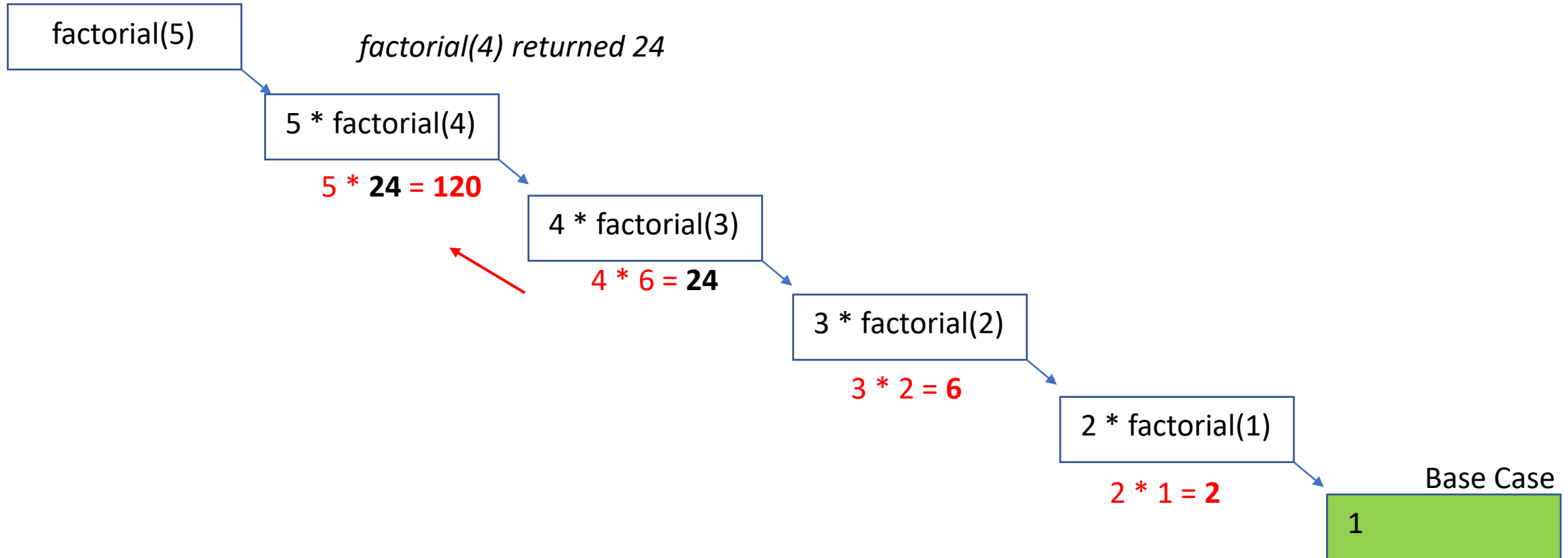
Factorial



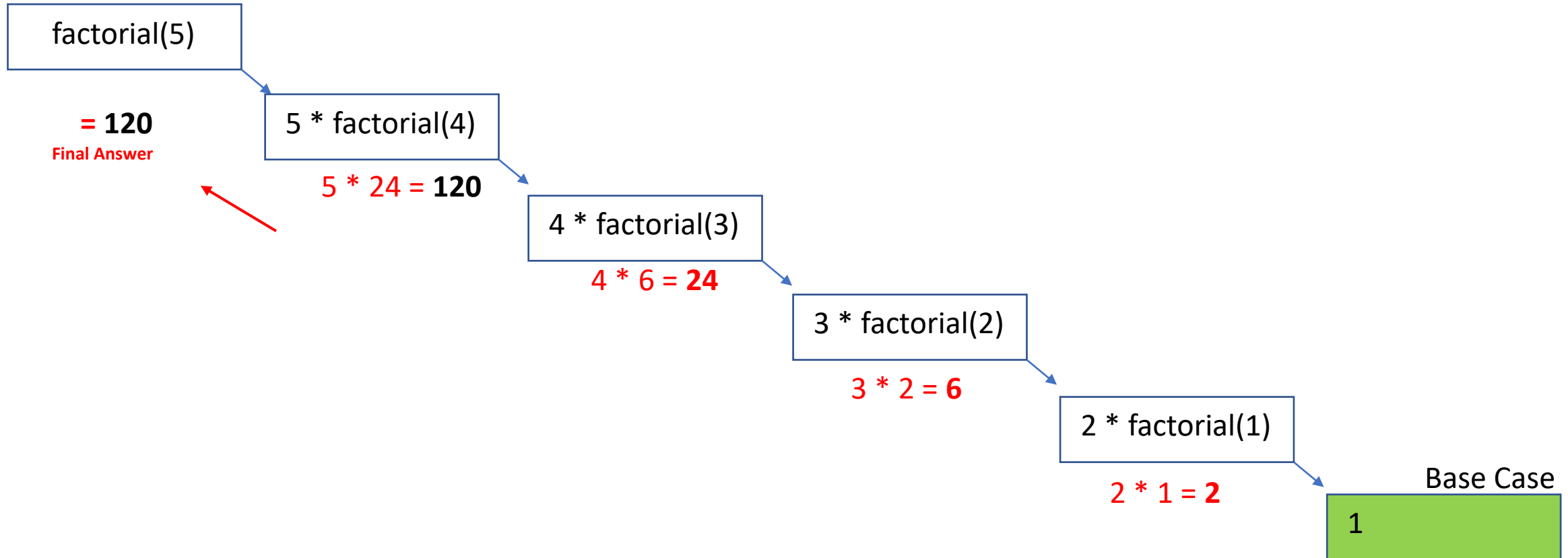
Factorial



Factorial



Factorial



Recursion Example

Write a recursive program that will count the number of **P**'s in a string

Write a recursive function that will return True if a string is a palindrome

Recursive Palindrome Strategy

What is the smallest sub-problem that where will *always* have an answer?

Recursive Palindrome Strategy

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Recursive Palindrome Strategy

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome

Recursive Palindrome Strategy

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Recursive Palindrome Strategy

What is the smallest sub-problem that where will *always* have an answer?

Fact:
If a string has a size of 1, it is a palindrome

Fact:
If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

How can we recursively make this problem smaller to get to our base case?

Recursive Palindrome Strategy

What is the smallest sub-problem that where will *always* have an answer?

Fact:
If a string has a size of 1, it is a palindrome

Fact:
If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Let's check the first and last character of the string.

Recursive Palindrome Strategy

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Let's check the first and last character of the string.

If the are not equal, then we do not have a palindrome



```
elif(word[0] != word[-1]):  
    return False
```

Recursive Palindrome Strategy

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Let's check the first and last character of the string.

If they are not equal, then we do not have a palindrome



```
elif(word[0] != word[-1]):  
    return False
```

Otherwise, let's remove the first and last character of string and pass the new string into our function (recursion)



```
else:  
    return is_palindrome(word[1:-1])
```


Recursive Palindrome Strategy

“aabccbaa”

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Let's check the first and last character of the string.

If the are not equal, then we do not have a palindrome



```
elif(word[0] != word[-1]):  
    return False
```

Otherwise, lets remove the first and last character of string and pass the new string into our function (recursion)



```
else:  
    return is_palindrome(word[1:-1])
```

Recursive Palindrome Strategy

“aabccbaa”
“abccba”

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Let's check the first and last character of the string.

If the are not equal, then we do not have a palindrome



```
elif(word[0] != word[-1]):  
    return False
```

Otherwise, lets remove the first and last character of string and pass the new string into our function (recursion)



```
else:  
    return is_palindrome(word[1:-1])
```

Recursive Palindrome Strategy

“aabccbaa”
“abccba”
“bccb”

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Let's check the first and last character of the string.

If the are not equal, then we do not have a palindrome



```
elif(word[0] != word[-1]):  
    return False
```

Otherwise, lets remove the first and last character of string and pass the new string into our function (recursion)



```
else:  
    return is_palindrome(word[1:-1])
```

Recursive Palindrome Strategy

“aabccbaa”

“abccba”

“bccb”

“cc”

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Let's check the first and last character of the string.

If the are not equal, then we do not have a palindrome



```
elif(word[0] != word[-1]):  
    return False
```

Otherwise, lets remove the first and last character of string and pass the new string into our function (recursion)



```
else:  
    return is_palindrome(word[1:-1])
```

Recursive Palindrome Strategy

"aabccbaa"

"abccba"

"bccb"

"cc"

""

← Base case reached!

What is the smallest sub-problem that where will *always* have an answer?

Fact:

If a string has a size of 1, it is a palindrome

Fact:

If a string has a size of 0, it is a palindrome



```
if(len(word) == 1 or len(word) == 0):  
    return True
```

Let's check the first and last character of the string.



```
elif(word[0] != word[-1]):  
    return False
```

If the are not equal, then we do not have a palindrome

Otherwise, lets remove the first and last character of string and pass the new string into our function (recursion)



```
else:  
    return is_palindrome(word[1:-1])
```


Recursion Example

`is_palindrome("racecar")`

```
def is_palindrome(word):  
  
    if len(word) == 1 or len(word) == 0:  
        return True  
    elif word[0] != word[-1]:  
        return False  
    else:  
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")

```
def is_palindrome(word):  
     if len(word) == 1 or len(word) == 0:  
        return True  
    elif word[0] != word[-1]:  
        return False  
    else:  
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")

```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```



```
    elif word[0] != word[-1]:
```

```
        return False
```

```
    else:
```

```
        return is_palindrome(word[1:-1])
```


Recursion Example

is_palindrome("racecar")

```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```

```
    elif word[0] != word[-1]:
```

```
        return False
```



```
    else:
```

```
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```

```
    elif word[0] != word[-1]:
```

```
        return False
```

```
    else:
```

```
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



```
def is_palindrome(word):  
    if len(word) == 1 or len(word) == 0:  
        return True  
    elif word[0] != word[-1]:  
        return False  
    else:  
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



```
def is_palindrome(word):  
    if len(word) == 1 or len(word) == 0:  
        return True  
    elif word[0] != word[-1]:  
        return False  
    else:  
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



```
def is_palindrome(word):  
    if len(word) == 1 or len(word) == 0:  
        return True  
    elif word[0] != word[-1]:  
        return False  
    else:  
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



is_palindrome("cec")



```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```

```
    elif word[0] != word[-1]:
```

```
        return False
```

```
    else:
```

```
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



is_palindrome("cec")



```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```

```
    elif word[0] != word[-1]:
```

```
        return False
```

```
    else:
```

```
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



is_palindrome("cec")

```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```

```
    elif word[0] != word[-1]:
```

```
        return False
```

```
    else:
```

```
        return is_palindrome(word[1:-1])
```



Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



is_palindrome("cec")

```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```

```
    elif word[0] != word[-1]:
```

```
        return False
```

```
    else:
```

```
        return is_palindrome(word[1:-1])
```



Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



is_palindrome("cec")



is_palindrome("e")



```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```

```
    elif word[0] != word[-1]:
```

```
        return False
```

```
    else:
```

```
        return is_palindrome(word[1:-1])
```

Recursion Example

is_palindrome("racecar")



is_palindrome("aceca")



is_palindrome("cec")



is_palindrome("e")

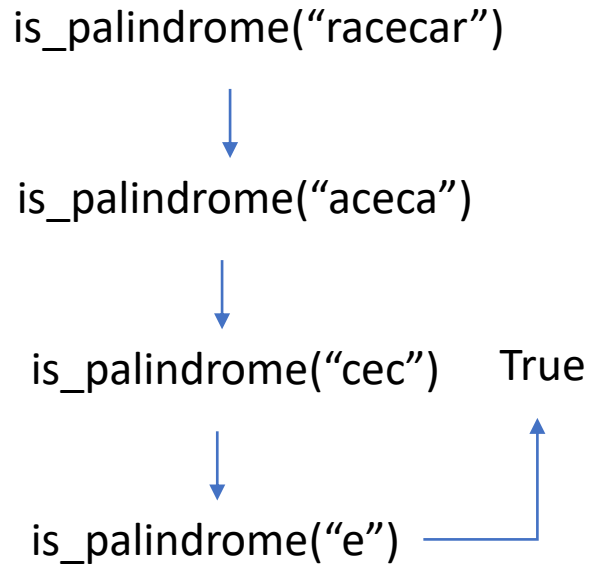
```
def is_palindrome(word):
```



```
    if len(word) == 1 or len(word) == 0:  
        return True  
    elif word[0] != word[-1]:  
        return False  
    else:  
        return is_palindrome(word[1:-1])
```

Recursion Example

Base case reached!



def is_palindrome(word):

if len(word) == 1 or len(word) == 0:
 return True
elif word[0] != word[-1]:
 return False
else:
 return is_palindrome(word[1:-1])



Recursion Example

Base case reached!

is_palindrome("racecar")



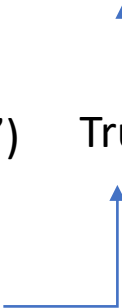
is_palindrome("aceca") True



is_palindrome("cec") True



is_palindrome("e")



```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
```

```
        return True
```

```
    elif word[0] != word[-1]:
```

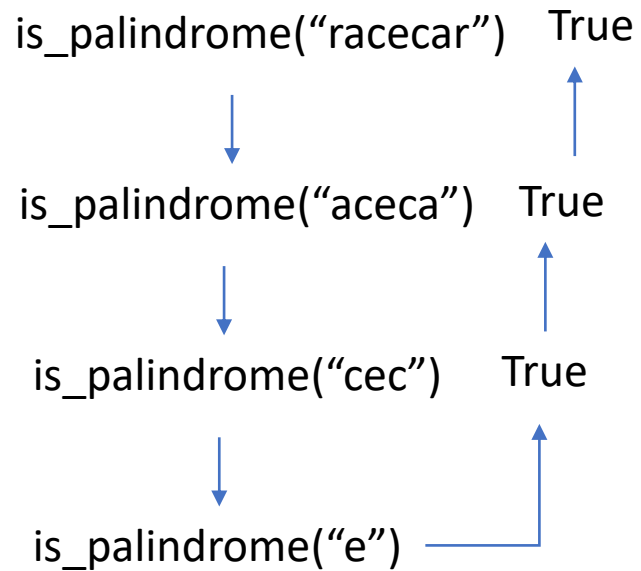
```
        return False
```

```
    else:
```

```
        return is_palindrome(word[1:-1])
```

Recursion Example

Base case reached!

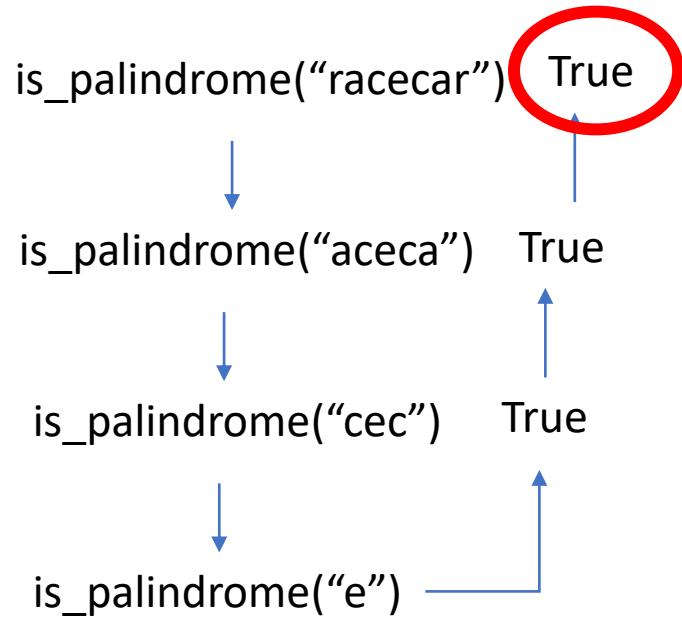


```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
        return True
    elif word[0] != word[-1]:
        return False
    else:
        return is_palindrome(word[1:-1])
```

Recursion Example

Base case reached!

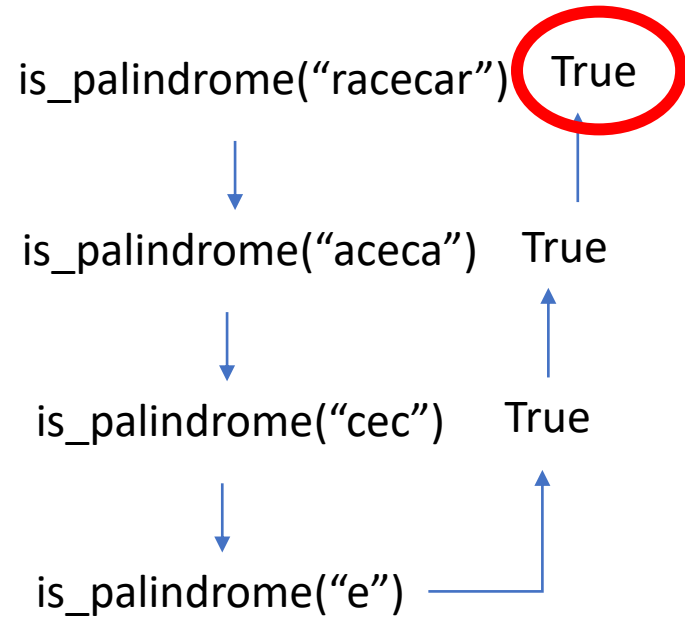


```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
        return True
    elif word[0] != word[-1]:
        return False
    else:
        return is_palindrome(word[1:-1])
```

Recursion Example

Base case reached!



```
def is_palindrome(word):
```

```
    if len(word) == 1 or len(word) == 0:
        return True
    elif word[0] != word[-1]:
        return False
    else:
        return is_palindrome(word[1:-1])
```