CS 61A Discussion 2

Environment Diagrams and Recursion

Raymond Chan
Discussion 121
UC Berkeley

Announcements

- Project 1 Hog due tonight
  - tinyurl.com/61a-unstuck
- Guerrilla Section on Recursion 2/7 10am-noon
- CSM small group tutoring sections sign ups
  - csmscheduler.herokuapp.com

Environment Diagrams

- Environment diagrams allow us to keep track of variables that have been defined and the values they are bound to.
- Assignment Statements
- Def Statements
- Function Calls
- Lambda Expressions

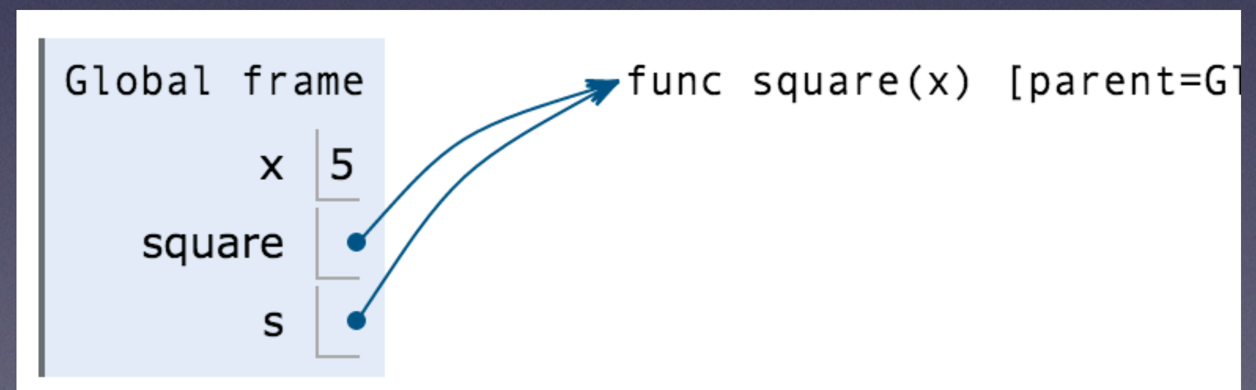
Assignment Statements

- Evaluate the expression on the right hand side of the = sign.
  - Look up variable names in the current frame. If it does not exist, look up in the parent frame.
  - Evaluate primitive expressions and operations
  - Evaluate call expressions

Assignment Statements

- Write the variable name and the expression value in the current frame.
- If the expression is a function, use an arrow.

```
1  x = 5
2  def square(x):
3      return x**2
4
5  s = square
```



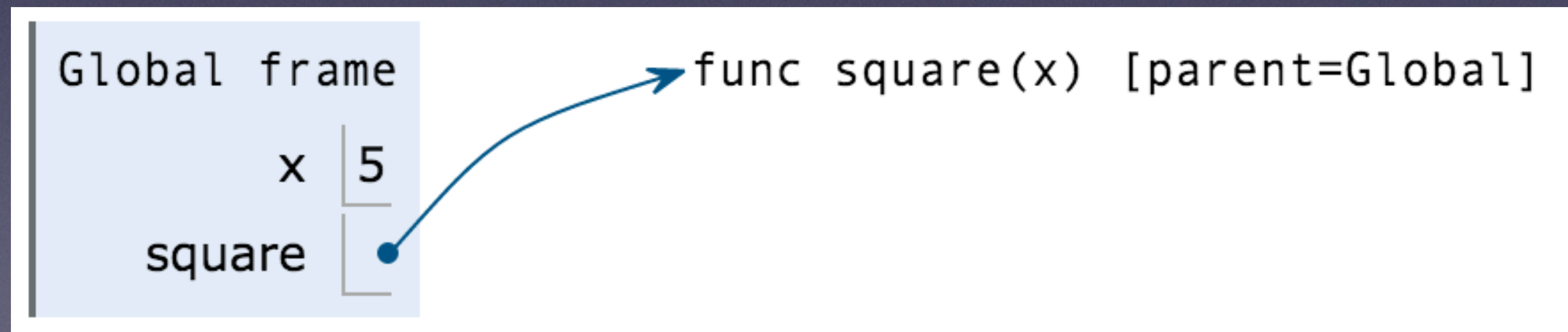
Def Statements

- Write the function name in the frame and point it to the function object.
- Function object contains the function signature and the parent frame.
- The parent frame is the frame in which the frame is defined.
- **Do not** evaluate the body of the function at this time.

Def Statements

- Function signature contains the function's intrinsic name and the formal parameters.

```
1  x = 5
2  def square(x):
3      return x**2
4
```



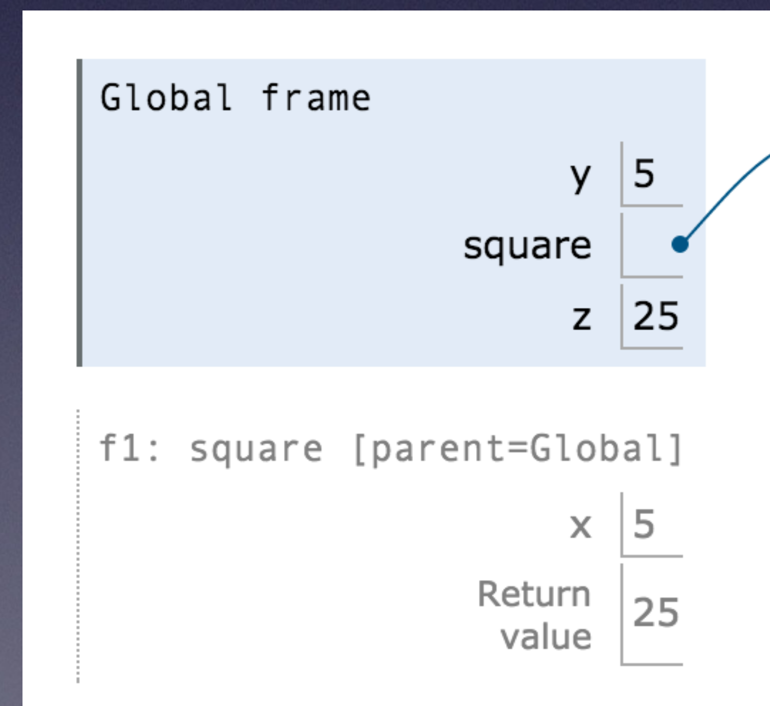
Call Expressions

- After evaluating operator and all the operands, we apply the arguments to the function to make a function call.
- Draw a new frame with a unique frame index, the function's intrinsic name, and the parent frame.
- Bind the formal parameters to the argument(s) passed in.
- Evaluate the body of the function

Call Expressions

- Remember to denote the return value. If a function does not return anything, the return value is by default **None**.
- If we are assigning a variable to a call expression, assign the return value to the variable in the frame of the call expression.

```
1  y = 5
2  def square(x):
3      return x**2
4
5  z = square(y)
```



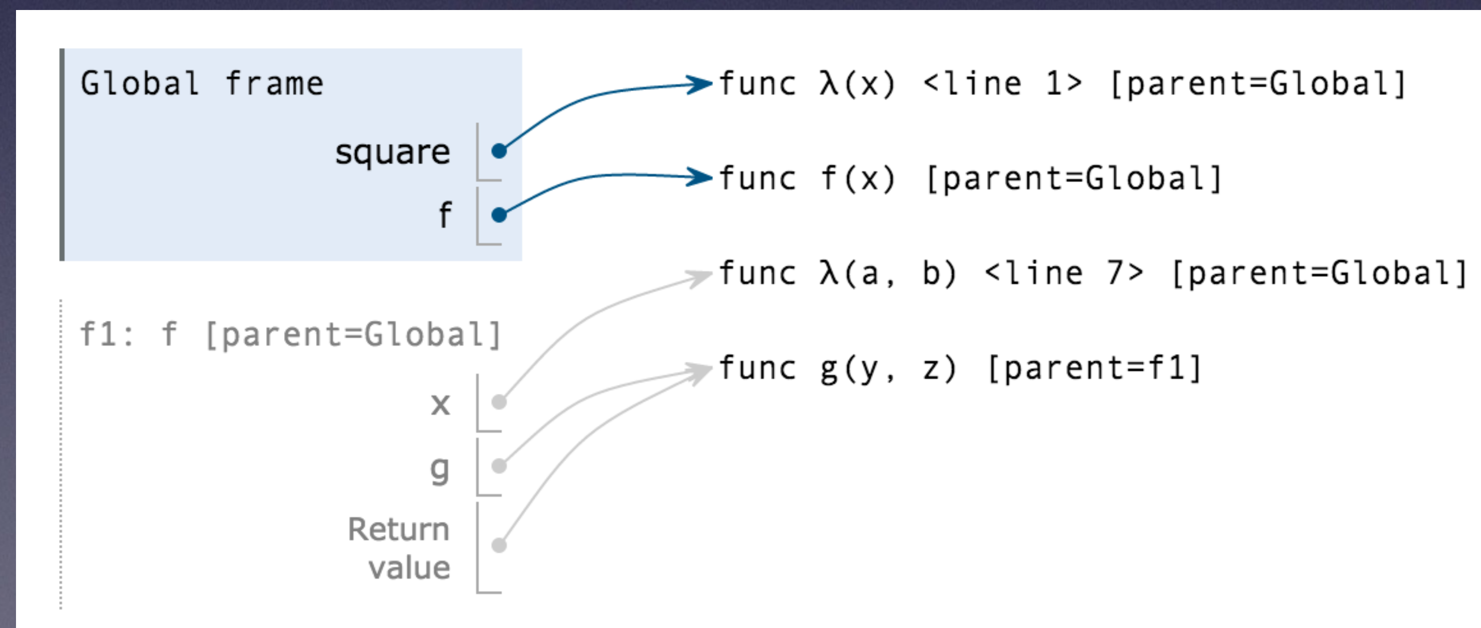
Lambda Functions

- lambda <parameters>: <body>
- There can be multiple parameters delimited by commas.
    - lambda x, y, z: <body>
- Lambda functions create function objects with the function name as λ.
- Create the function object in the environment diagram even if it is not assigned to a variable.

Lambda Functions

- Lambda functions cannot be accessed if it is not assigned to variables either by
  - using an explicit assignment statement or
  - passing the lambda function into another function’s argument.

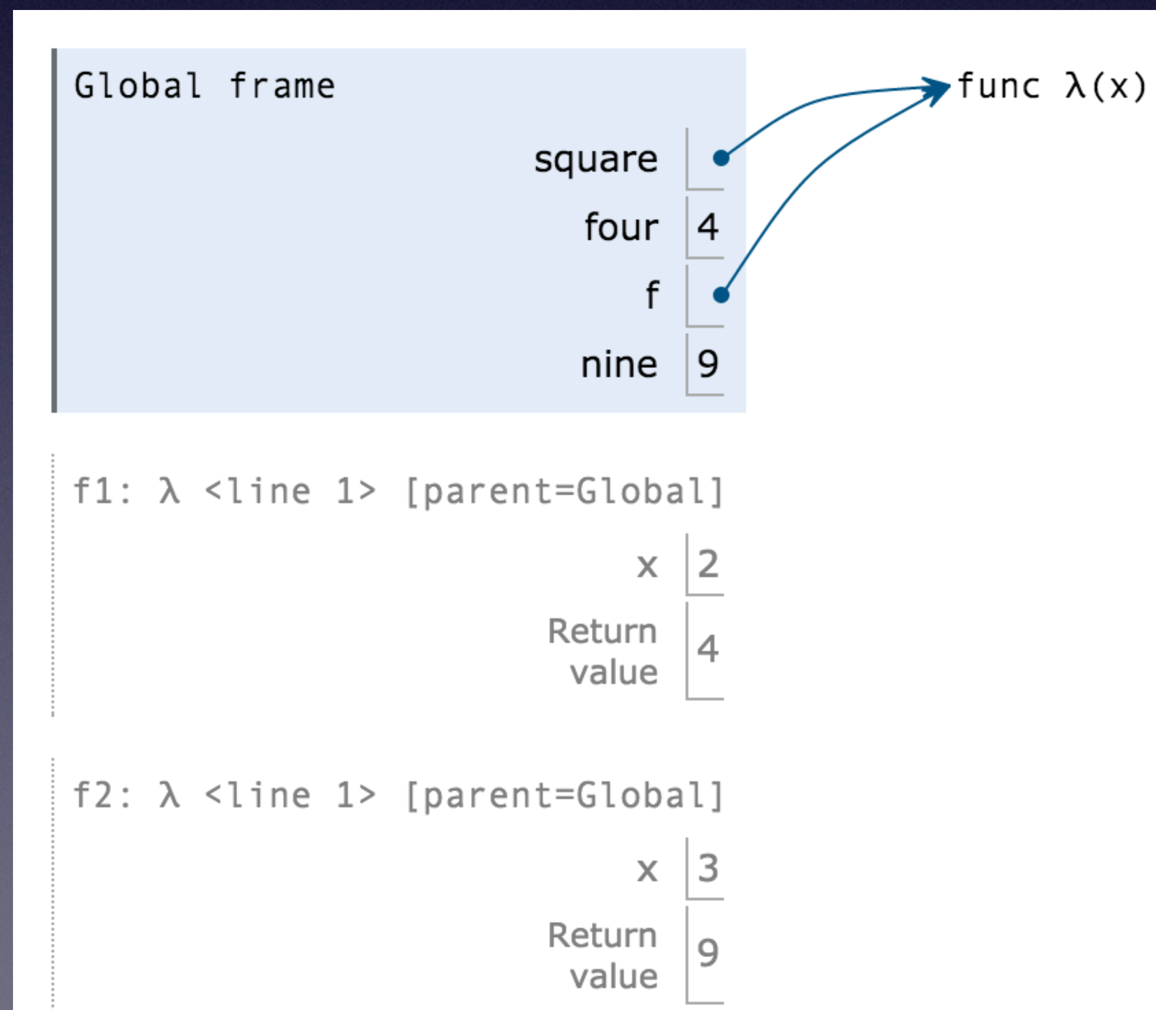
```
1  square = lambda x: x * x
2  def f(x):
3      def g(y, z):
4          return x(y, z)
5      return g
6
7  f(lambda a, b: a + b)
```



Function Call vs. Function

- Variables can be assigned to the return value of a function call or the function object itself.
- Remember that variables are assigned to whatever the result of evaluating the right hand side is pointing at.

```
1  square = lambda x: x * x
2  four = square(2)
3  f = square
4  nine = f(3)
```



Recursion

- A recursive function is a function that calls itself.
- Three common steps
  - Figure our your base case(s)
  - Make the problem smaller and make a recursive call with that simpler argument
  - Use your recursive call to solve the full problem

Recursion

- Base cases are there to stop the recursion.
- No base case —> continue making recursive calls forever

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Recursion

- Find a smaller problem for the recursive call.
- Make sure the problem is getting smaller **toward** the base case.
- Call the recursive function with this smaller argument.

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Recursion

- Take the ***leap of faith*** and trust that your recursive function is correct on the smaller argument.
- Knowing that the recursive call returns what you want, how can you solve the bigger problem?

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Recursion

factorial(5)

Recursion

factorial(5)

5 * factorial(4)

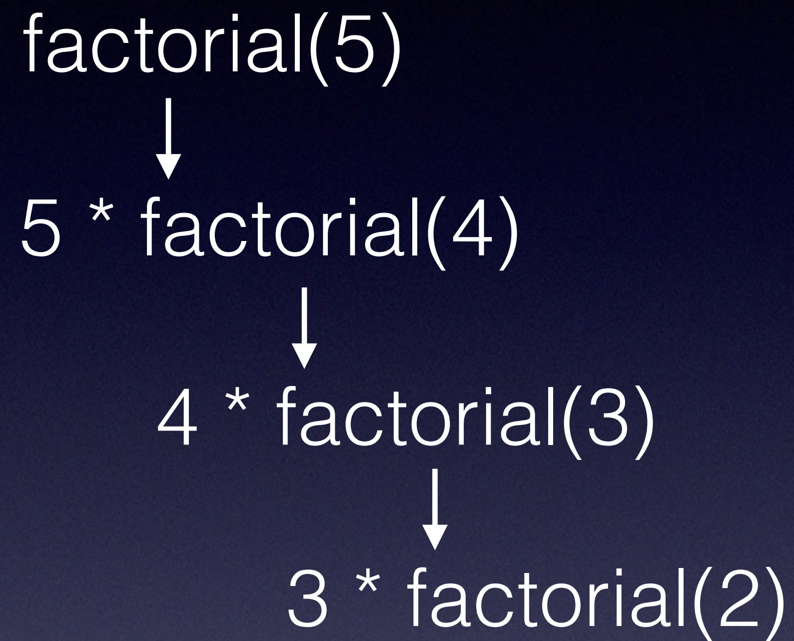
Recursion

factorial(5)

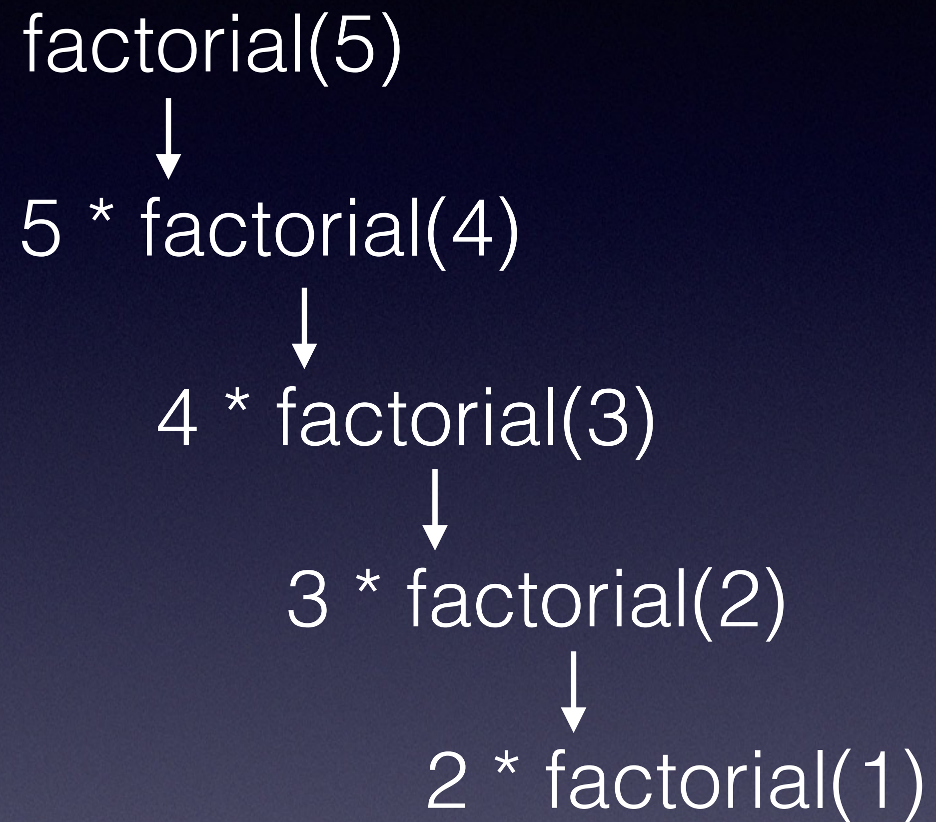
5 * factorial(4)

4 * factorial(3)

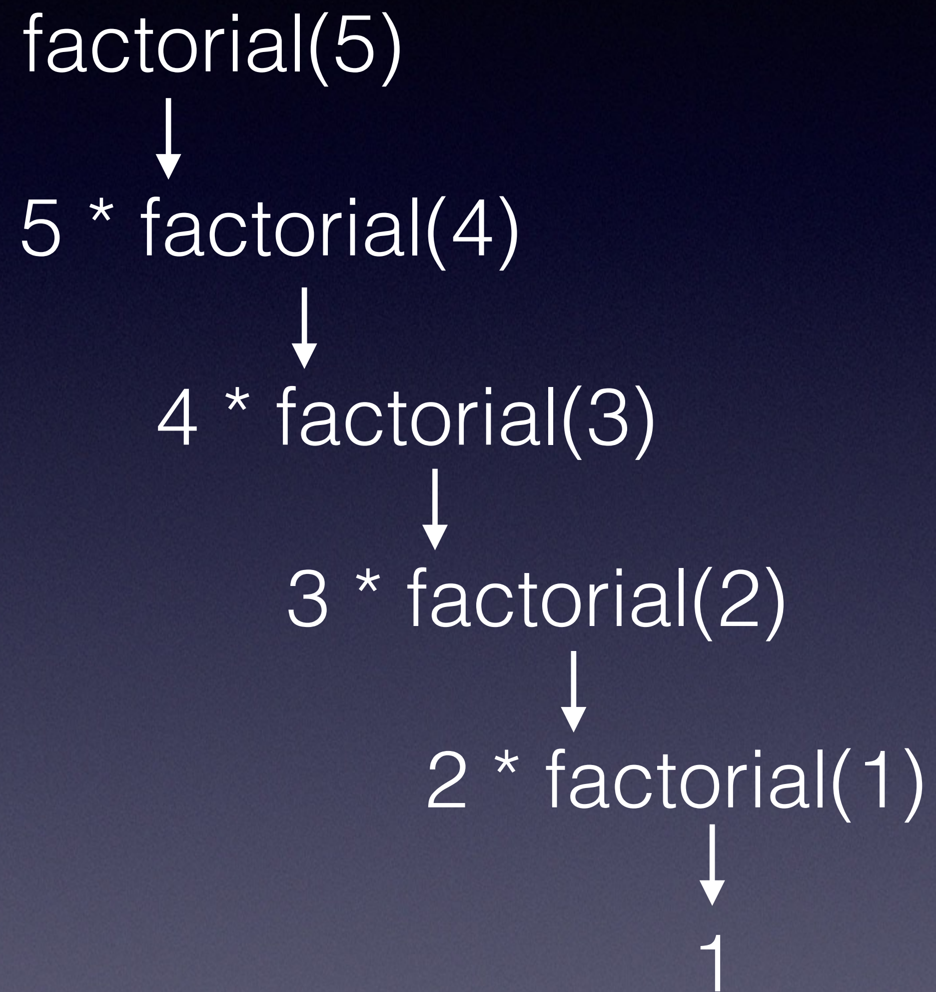
Recursion



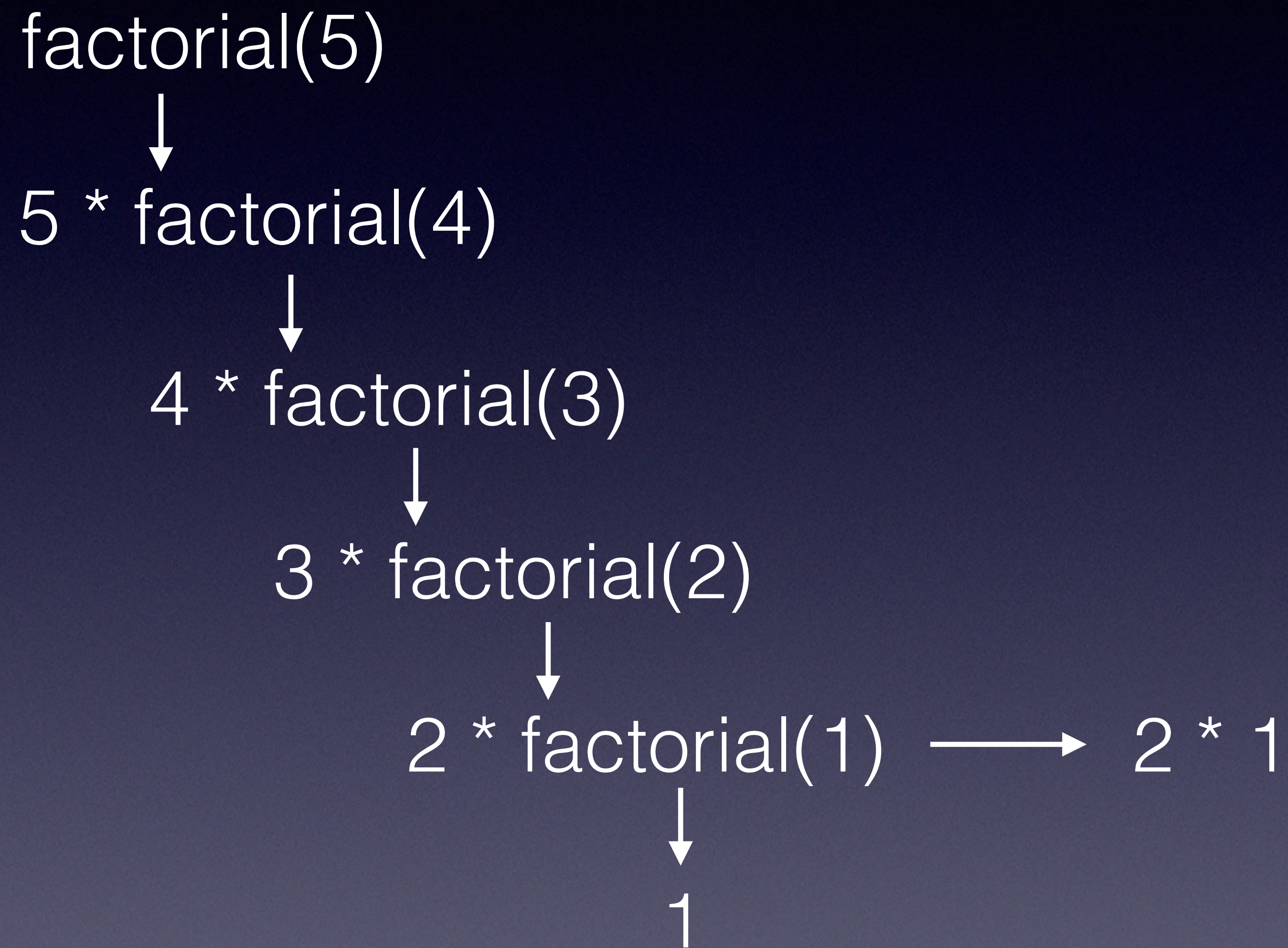
Recursion



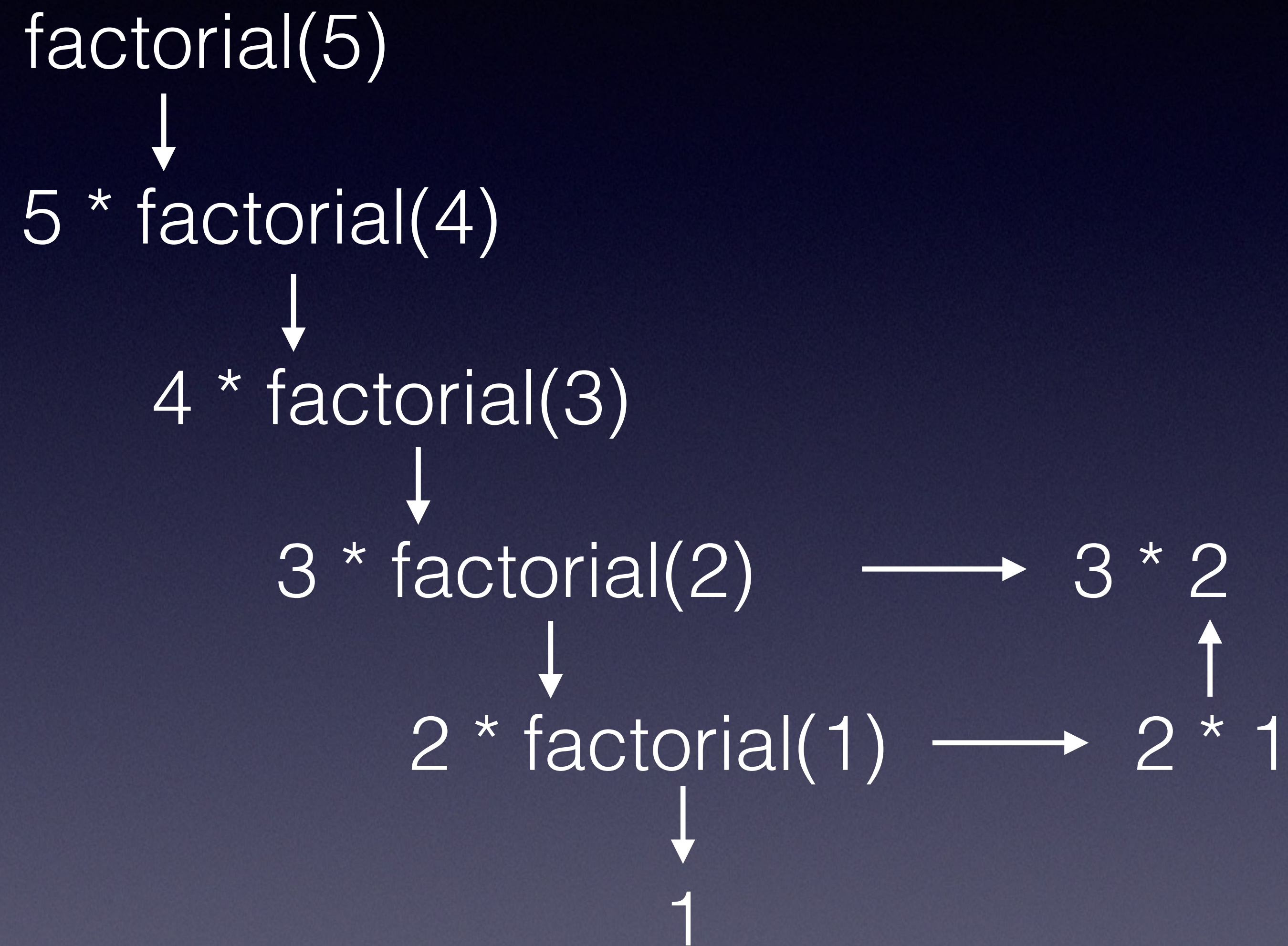
Recursion



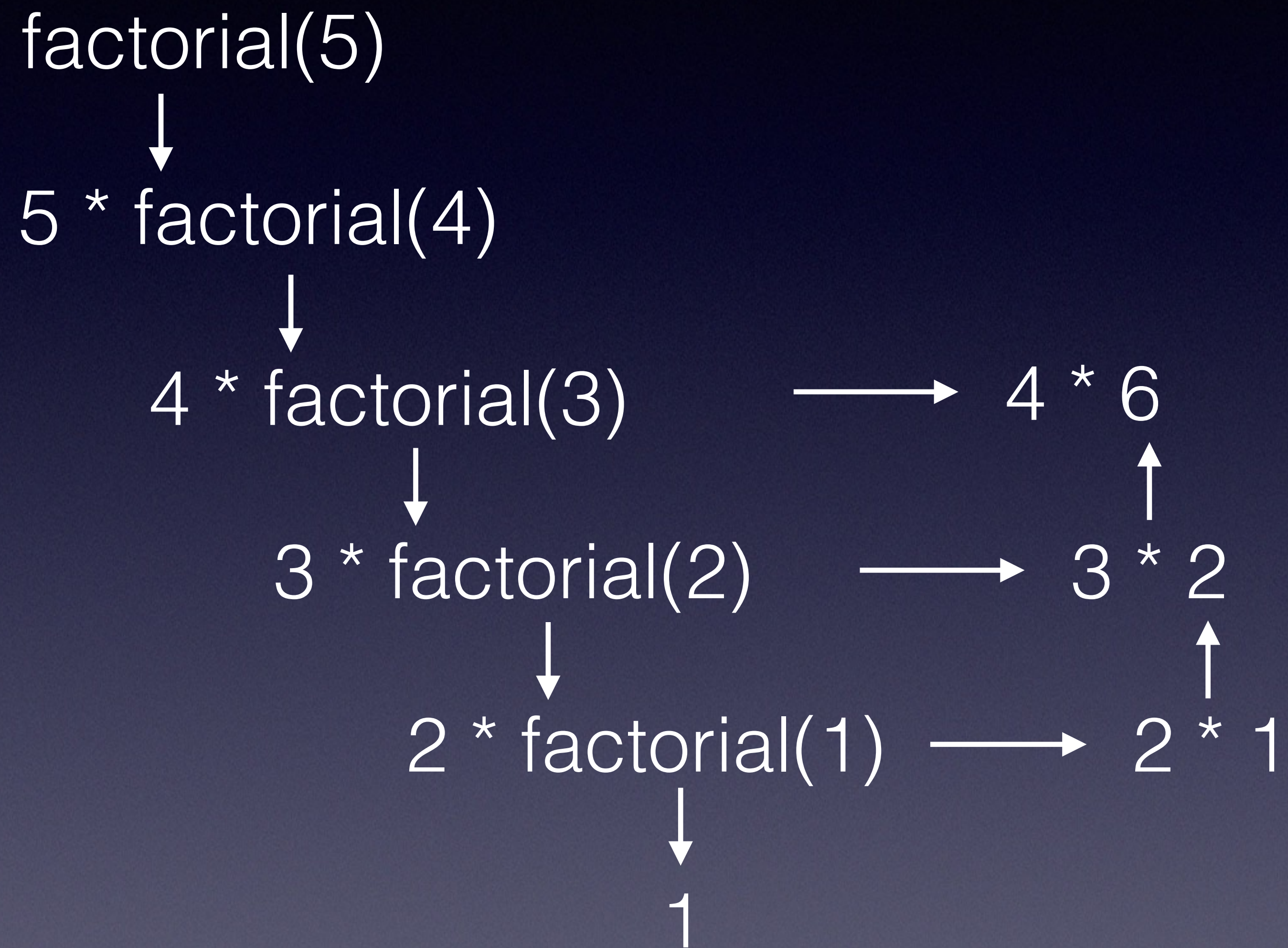
Recursion



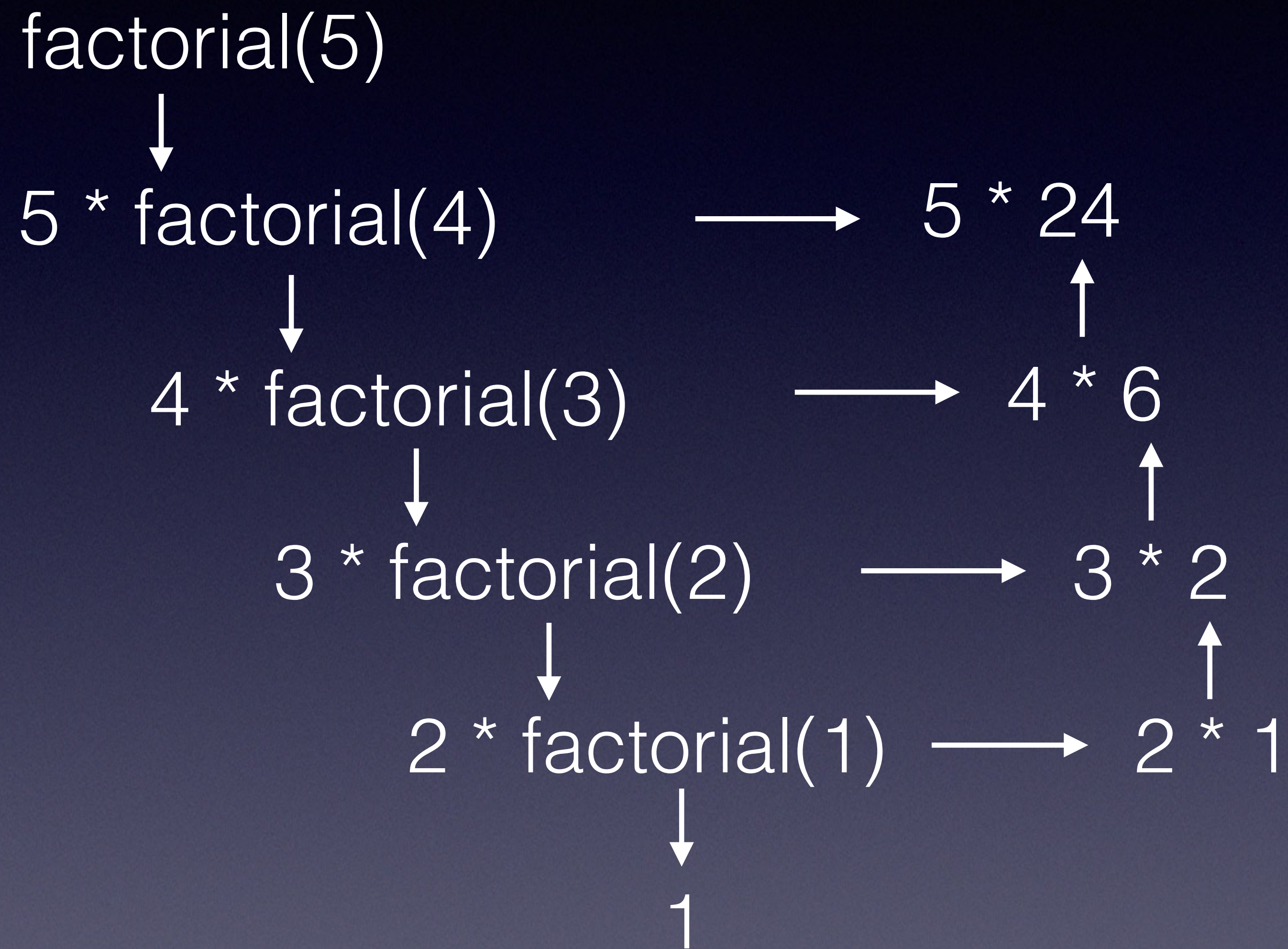
Recursion



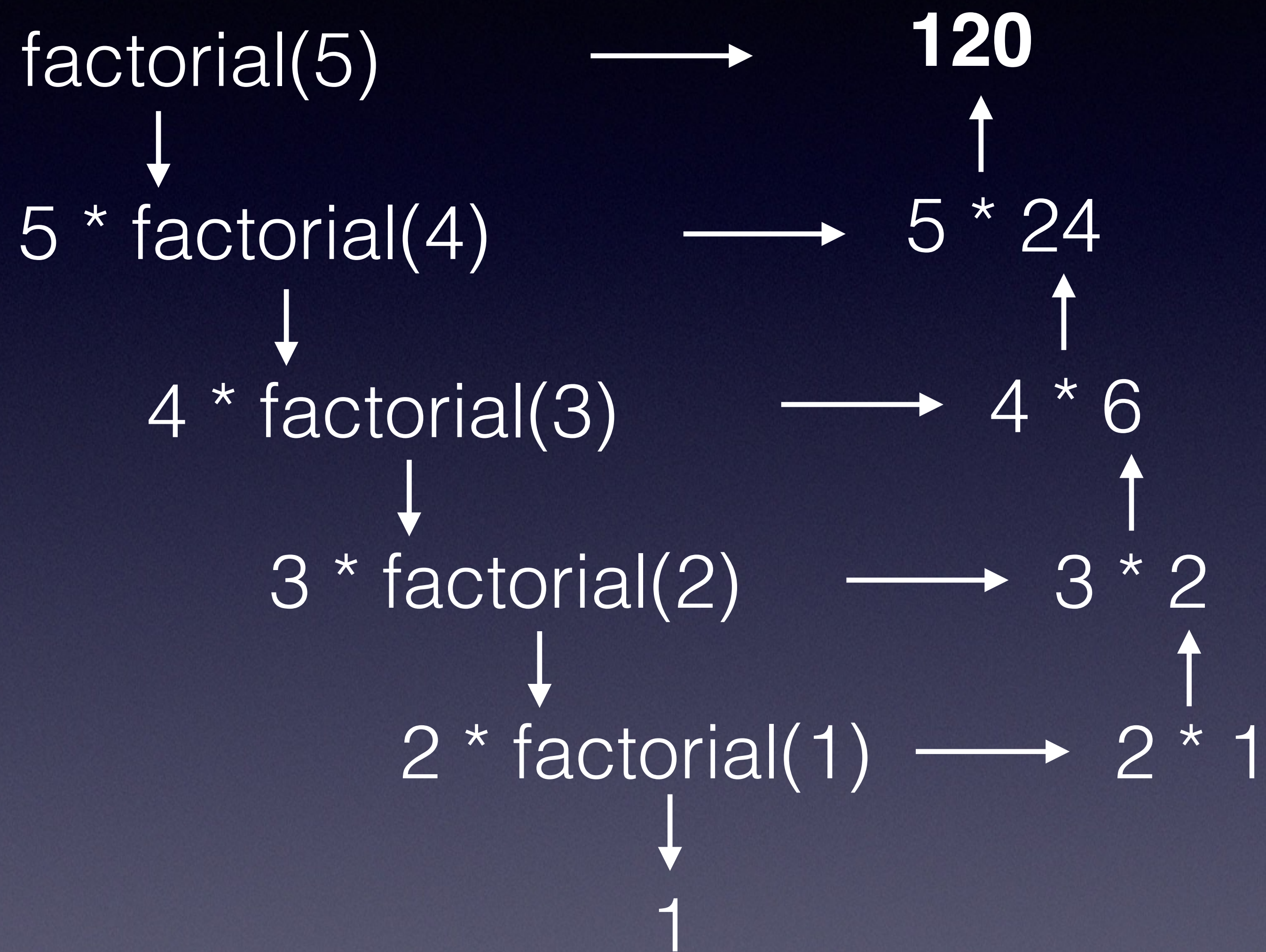
Recursion



Recursion



Recursion



Tree Recursion

- Recursive functions that make more than one recursive call in its recursive case.

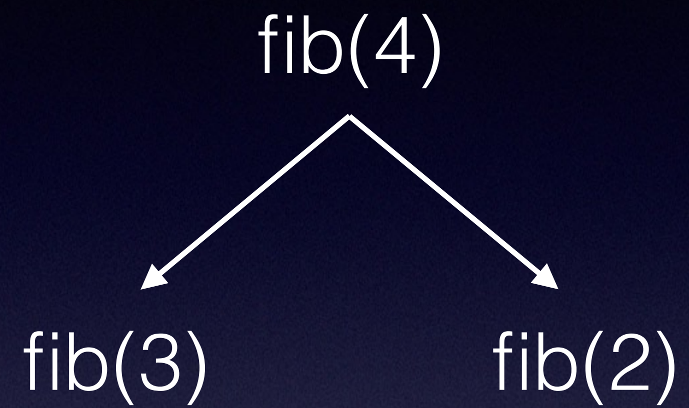
- Example: fibonacci sequence

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

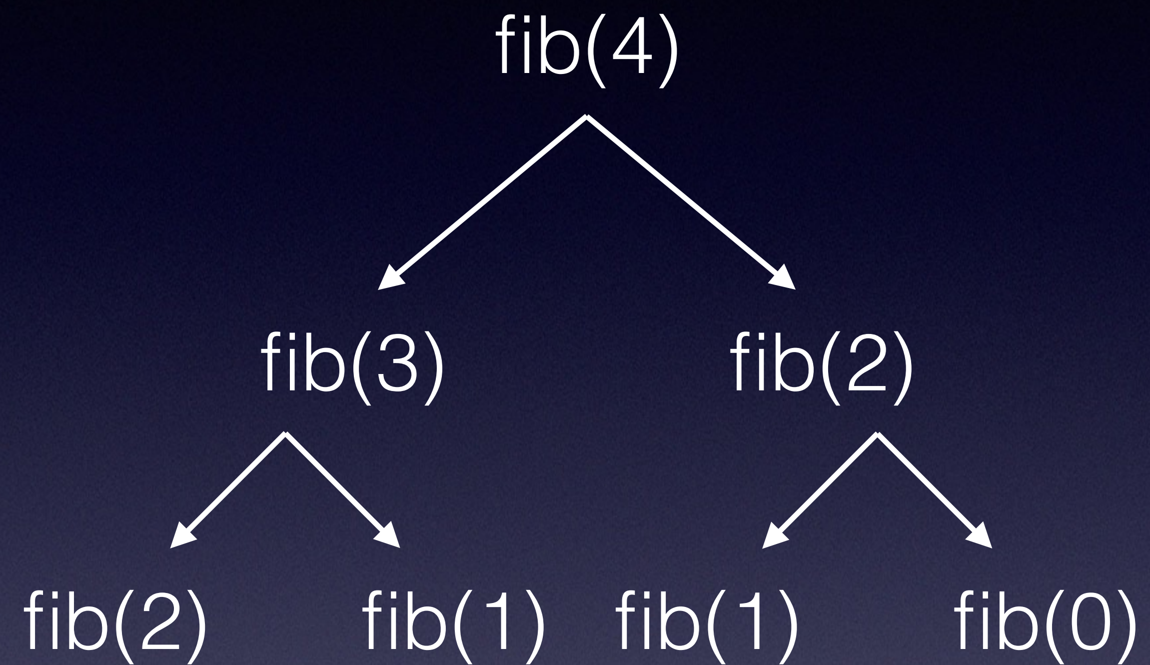
Tree Recursion

fib(4)

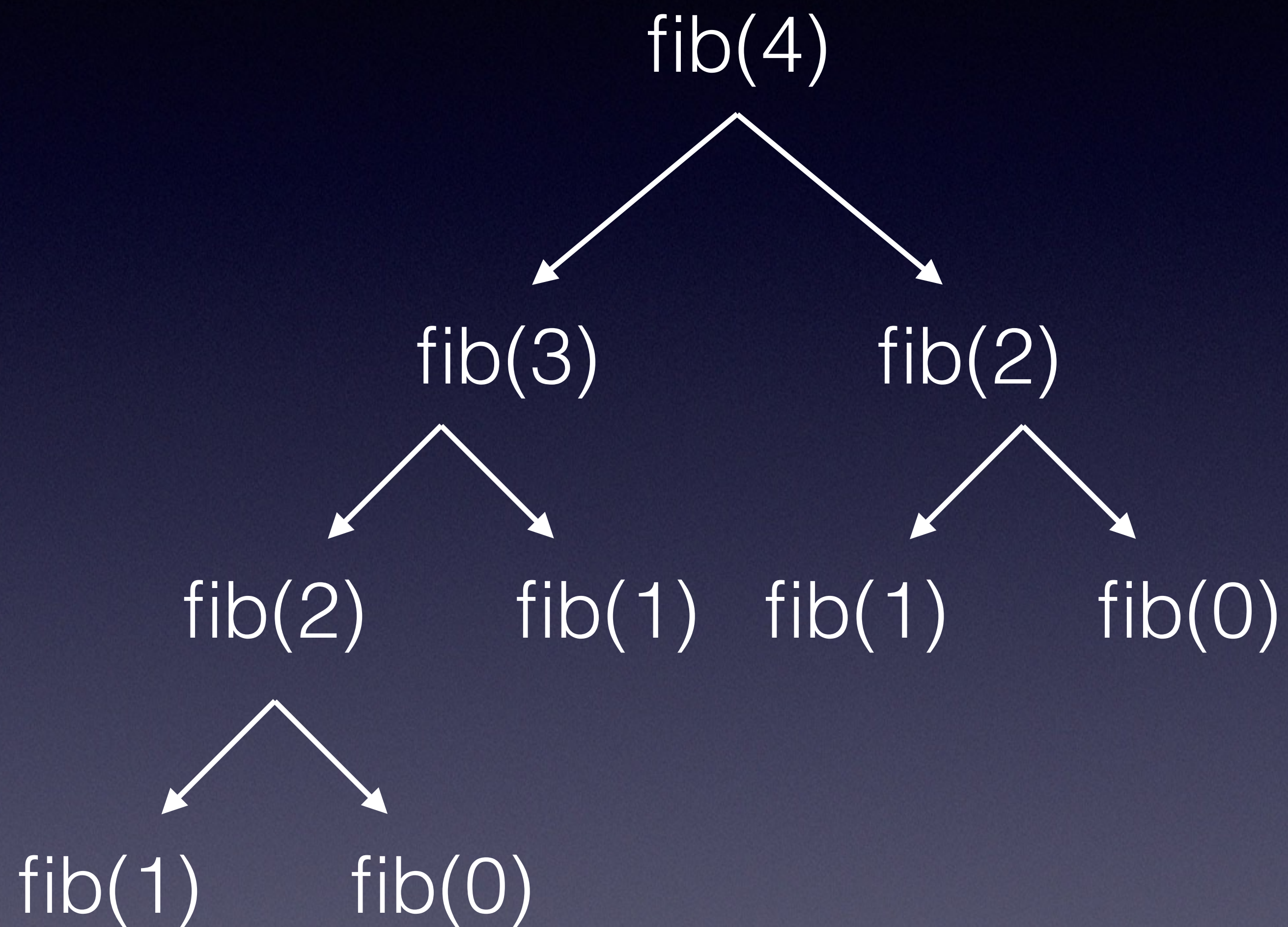
Tree Recursion



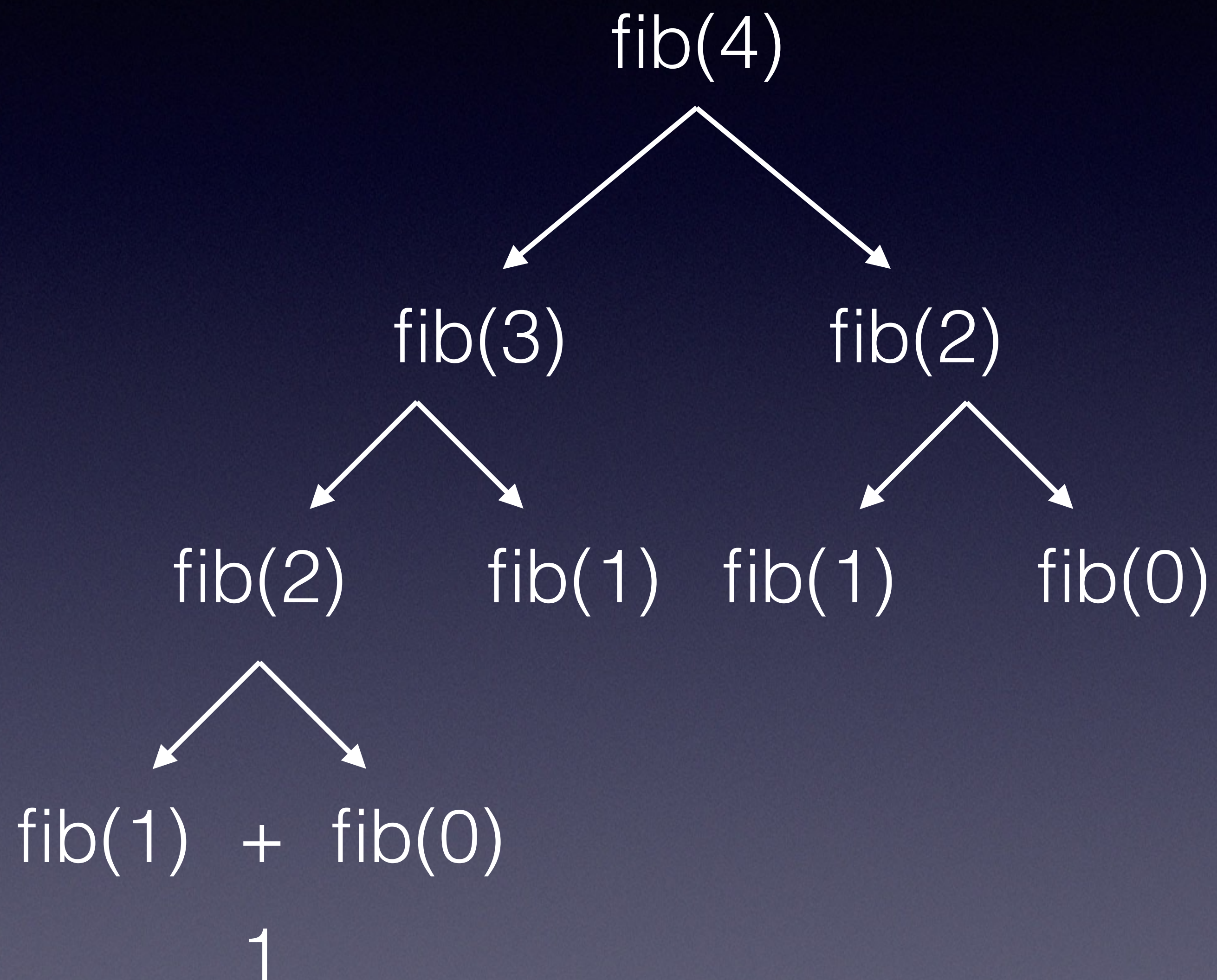
Tree Recursion



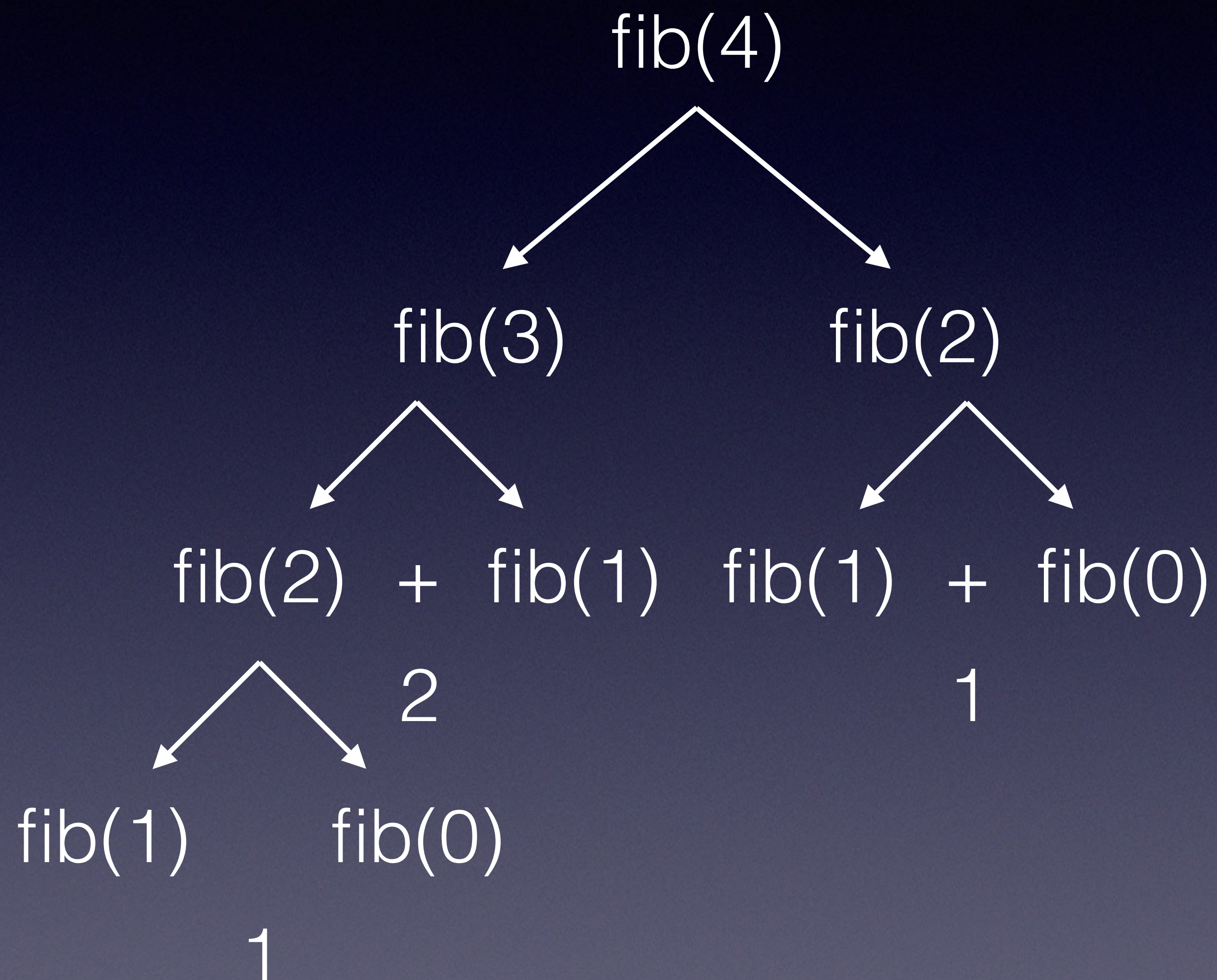
Tree Recursion



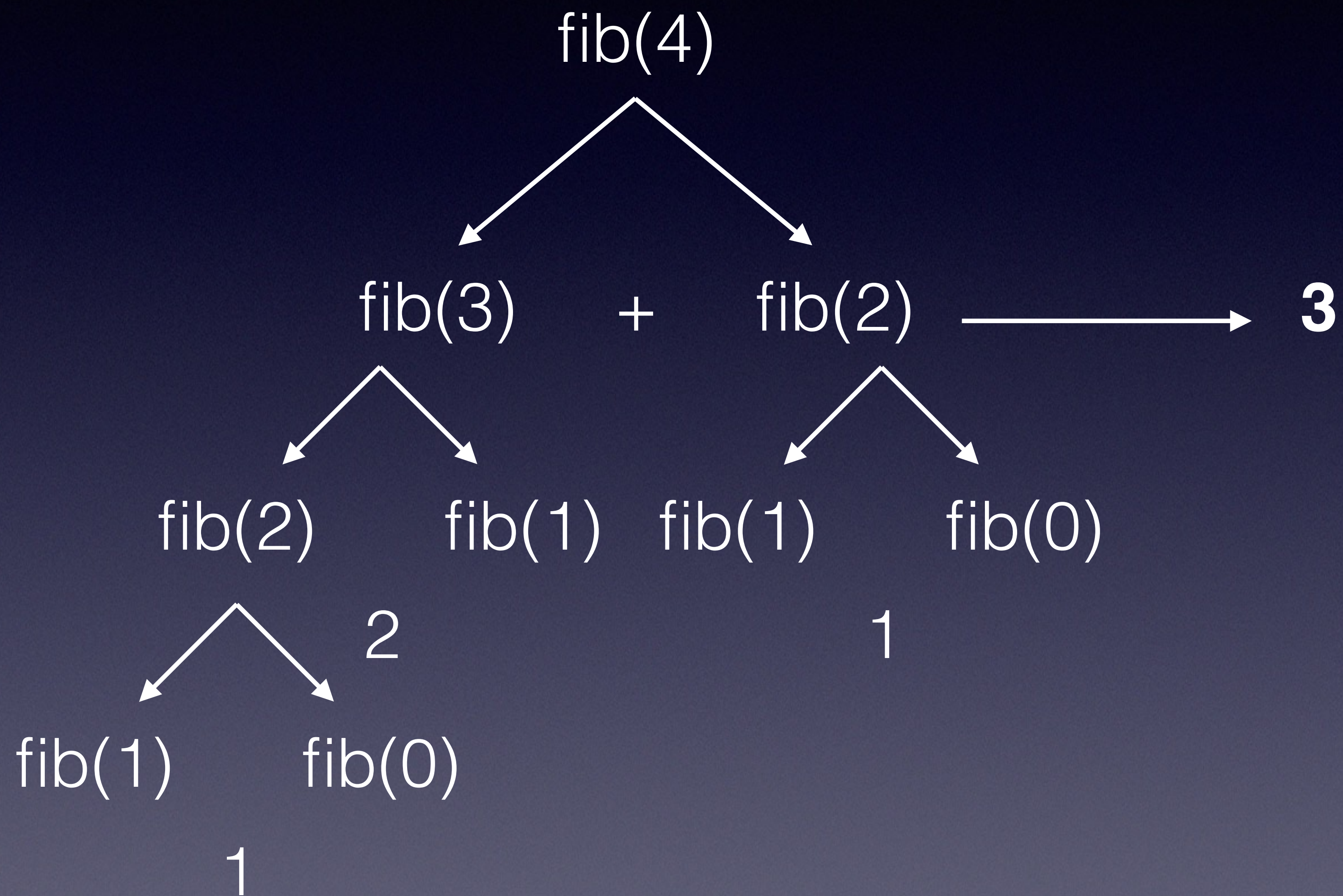
Tree Recursion



Tree Recursion



Tree Recursion



Worksheet

- 2.1 Cool recursion questions!
  - q1 - q2
- Tree recursion
  - q1

Recap

- Environment diagrams allow us to keep track of a variables and their values.
- Recursion functions call themselves.
- Tree recursive functions call themselves multiple times from one frame.