

Functions

Announcements

Announcements

Section 115 is now open.

Everyone(?) on the WL was added to 115.

You **must be enrolled in a lab** to be enrolled in the course.

Please contact ds-enrollments@berkeley.edu if you need help.

My “Tea Hours”: Weds 3:30–4:30pm 784 Soda.

Tea Hours → advising / general topics, more casual. Encouraged to bring your (legal) beverage of choice.

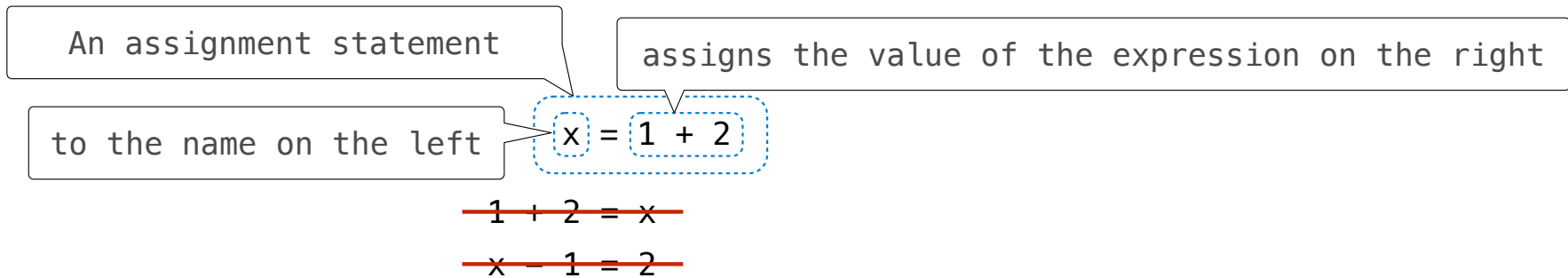
This Week:

Lab 0 and Lab 1 due Friday. Normally just 1 lab will be due.

Homework: Released on Tuesday, due the following Wednesday (8 days total)

Assignment Statements

Assignment Statements



The expression (right) is evaluated, and its value is assigned to the name (left).

```
>>> x = 2
>>> y = x + 1
>>> y
3
>>> x = 5
>>> y
3
```

(Demo)

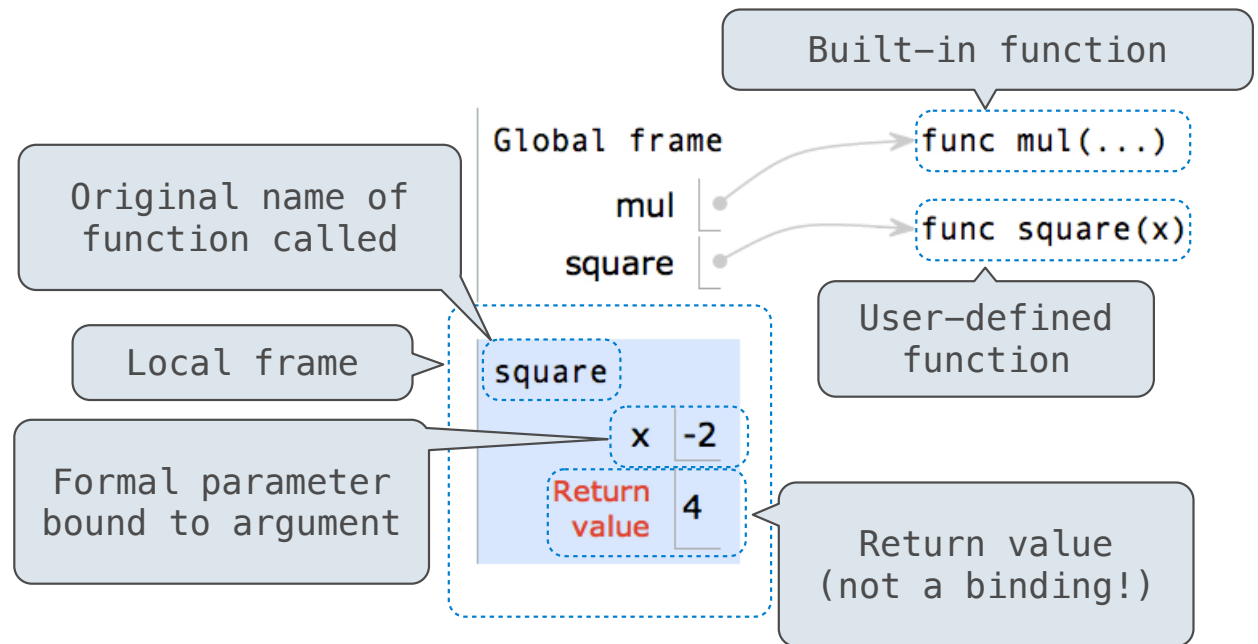
Environment Diagrams

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(-2)
```



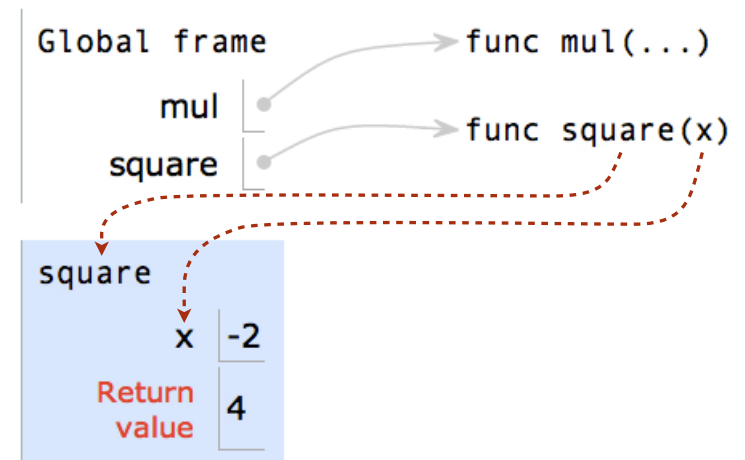
Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(-2)
```

A function's signature has all the information needed to create a local frame



Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

An environment is a sequence of frames.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

E.g., to look up some name in the body of the square function:

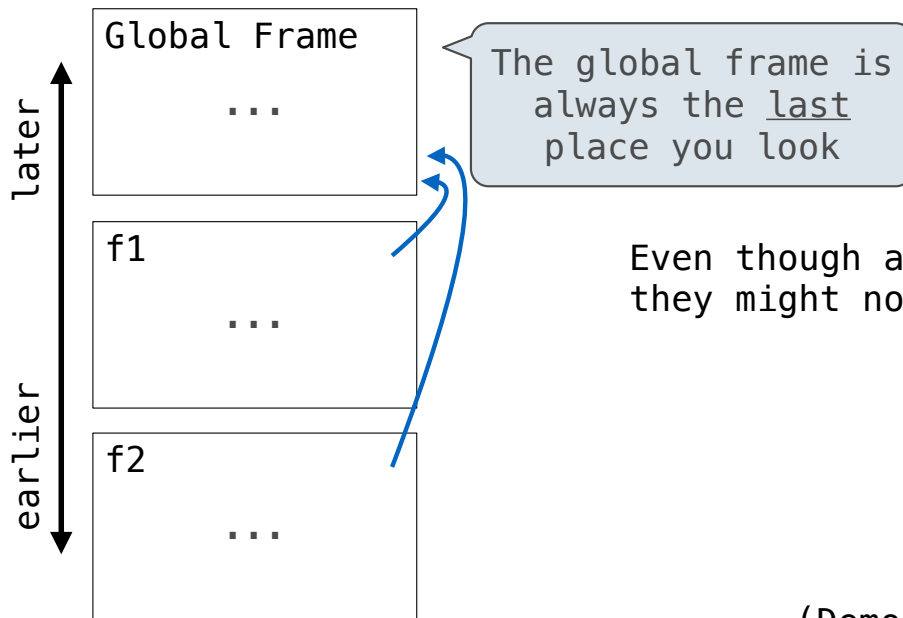
- Look for that name in the local frame.
- If not found, look for it in the global frame.
(Built-in names like “max” are in the global frame too, but we don't draw them in environment diagrams.)

A Sequence of Frames

An environment is a sequence of frames.

A sequence is a first frame and then the rest of the sequence

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.



Even though all three frames are in the same **diagram**, they might not be in the same **environment**

(Demo)

Multiple Assignment

Print and None

(Demo)

(Optional)
Small Expressions

Problem Definition

From Discussion 0:

Imagine you can call only the following three functions:

- $f(x)$: Subtracts one from an integer x
- $g(x)$: Doubles an integer x
- $h(x, y)$: Concatenates the digits of two different positive integers x and y . For example, $h(789, 12)$ evaluates to 78912 and $h(12, 789)$ evaluates to 12789.

Definition: A *small expression* is a call expression that contains only f , g , h , the number 5, and parentheses. All of these can be repeated. For example, $h(g(5), f(f(5)))$ is a small expression that evaluates to 103.

What's the **shortest** *small expression* you can find that evaluates to 2024?

Fewest calls?
Shortest length when written?

A Simple Restatement:

You start with 5. You can:

- Subtract 1 from a number
- Double a number
- Glue two numbers together

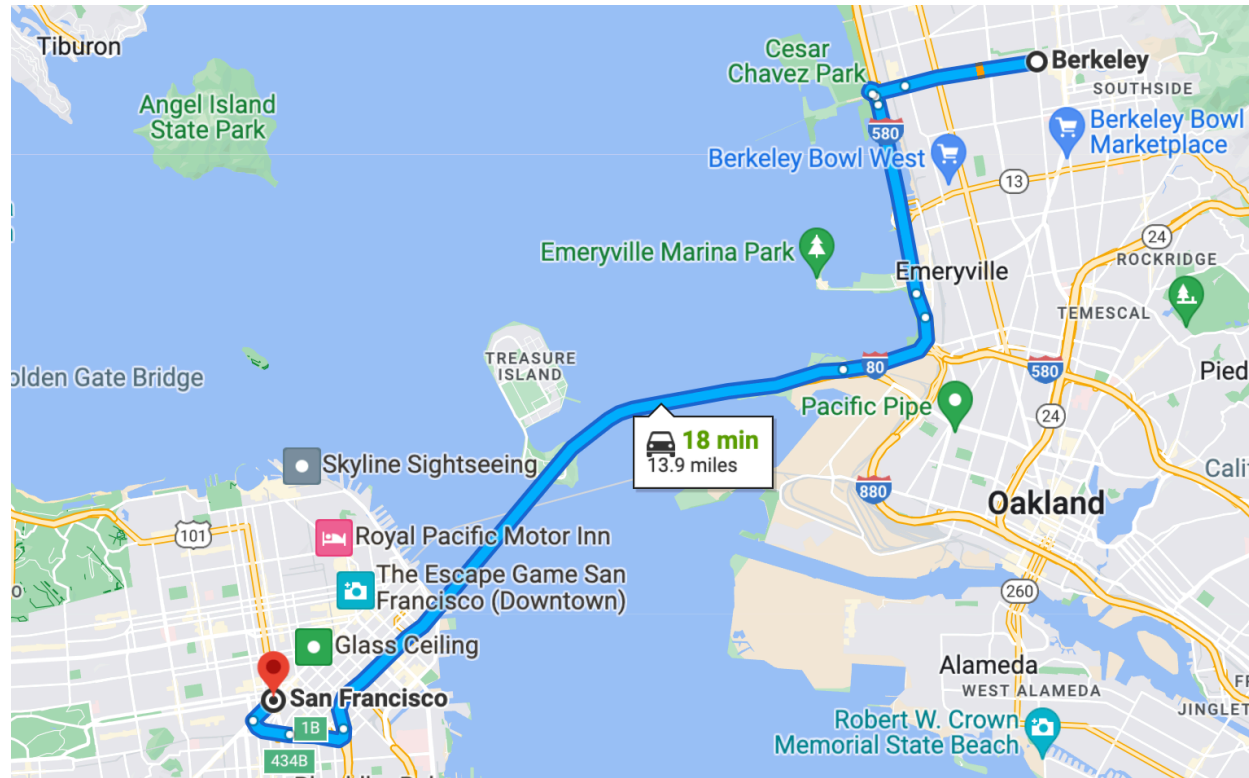
How do you get to **2024**?

```
5 → 10 → 20
5 → 4 → 3 → 2
5 → 4
```

Effective problem solving:

- Understand the problem
- Come up with ideas
- Turn those ideas into solutions

Search



A common strategy: try a bunch of options to see which is best

Computer programs can evaluate many alternatives by repeating simple operations

A Computational Approach

Try all the small expressions with 3 function calls, then 4 calls, then 5 calls, etc.

$f(f(f(5))) \rightarrow 2$	$f(f(h(5,5))) \rightarrow 53$	$h(5,f(f(5))) \rightarrow 53$	$h(g(5),f(5)) \rightarrow 104$
$g(f(f(5))) \rightarrow 6$	$g(f(h(5,5))) \rightarrow 108$	$h(5,g(f(5))) \rightarrow 58$	$h(g(5),g(5)) \rightarrow 1010$
$f(g(f(5))) \rightarrow 7$	$f(g(h(5,5))) \rightarrow 109$	$h(5,f(g(5))) \rightarrow 59$	$h(g(5),h(5,5)) \rightarrow 1055$
$g(g(f(5))) \rightarrow 16$	$g(g(h(5,5))) \rightarrow 220$	$h(5,g(g(5))) \rightarrow 520$	$h(h(5,5),f(5)) \rightarrow 554$
$f(f(g(5))) \rightarrow 8$	$f(h(5,f(5))) \rightarrow 53$	$h(5,f(h(5,5))) \rightarrow 554$	$h(h(5,5),g(5)) \rightarrow 5510$
$g(f(g(5))) \rightarrow 18$	$g(h(5,f(5))) \rightarrow 108$	$h(5,g(h(5,5))) \rightarrow 5110$	$h(h(5,5),h(5,5)) \rightarrow 5555$
$f(g(g(5))) \rightarrow 19$	$f(h(5,g(5))) \rightarrow 509$	$h(5,h(5,f(5))) \rightarrow 554$	$h(f(f(5)),5) \rightarrow 35$
$g(g(g(5))) \rightarrow 40$	$g(h(5,g(5))) \rightarrow 1020$	$h(5,h(5,g(5))) \rightarrow 5510$	$h(g(f(5)),5) \rightarrow 85$
	$f(h(5,h(5,5))) \rightarrow 554$	$h(5,h(5,h(5,5))) \rightarrow 5555$	$h(f(g(5)),5) \rightarrow 95$
	$g(h(5,h(5,5))) \rightarrow 1110$	$h(5,h(f(5),5)) \rightarrow 545$	$h(g(g(5)),5) \rightarrow 205$
	$f(h(f(5),5)) \rightarrow 44$	$h(5,h(g(5),5)) \rightarrow 5105$	$h(f(h(5,5)),5) \rightarrow 545$
	$g(h(f(5),5)) \rightarrow 90$	$h(5,h(h(5,5),5)) \rightarrow 5555$	$h(g(h(5,5)),5) \rightarrow 1105$
	$f(h(g(5),5)) \rightarrow 104$	$h(f(5),f(5)) \rightarrow 44$	$h(h(5,f(5)),5) \rightarrow 545$
	$g(h(g(5),5)) \rightarrow 210$	$h(f(5),g(5)) \rightarrow 410$	$h(h(5,g(5)),5) \rightarrow 5105$
	$f(h(h(5,5),5)) \rightarrow 554$	$h(f(5),h(5,5)) \rightarrow 455$	$h(h(5,h(5,5)),5) \rightarrow 5555$
	$g(h(h(5,5),5)) \rightarrow 1110$		$h(h(f(5),5),5) \rightarrow 455$
			$h(h(g(5),5),5) \rightarrow 1055$
			$h(h(h(5,5),5),5) \rightarrow 5555$

Reminder: $f(x)$ subtracts 1; $g(x)$ doubles; $h(x, y)$ concatenates

A Computational Approach

Try all the small expressions with 3 function calls, then 4 calls, then 5 calls, etc.

`g(h(g(5),g(g(f(f(5))))))` -> 2024 has 7 calls and 24 characters.
`g(g(f(f(f(f(h(5,g(5)))))))` -> 2024 has 8 calls and 27 characters.
`g(h(g(5),g(f(f(g(f(5))))))` -> 2024 has 8 calls and 27 characters.
`f(h(g(g(5)),h(f(f(f(5))),5))` -> 2024 has 8 calls and 29 characters.
`f(h(f(f(f(h(g(g(5))),5))),5)` -> 2024 has 8 calls and 29 characters.
`f(h(f(f(h(g(g(5))),f(5))),5)` -> 2024 has 8 calls and 29 characters.
`f(h(f(h(g(g(5))),f(f(5))),5)` -> 2024 has 8 calls and 29 characters.
`f(h(h(g(g(5))),f(f(f(5))),5)` -> 2024 has 8 calls and 29 characters.
`h(g(g(5)),g(g(g(f(f(5))))))` -> 2024 has 8 calls and 27 characters.
`h(g(g(5)),f(h(f(f(f(5))),5))` -> 2024 has 8 calls and 29 characters.
`h(g(g(5)),h(f(f(f(5))),f(5))` -> 2024 has 8 calls and 29 characters.
`h(f(f(f(h(g(g(5))),5))),f(5)` -> 2024 has 8 calls and 29 characters.
`h(f(f(h(g(g(5))),f(5))),f(5)` -> 2024 has 8 calls and 29 characters.
`h(f(h(g(g(5))),f(f(5))),f(5)` -> 2024 has 8 calls and 29 characters.
`h(h(g(g(5))),f(f(f(5))),f(5)` -> 2024 has 8 calls and 29 characters.

Reminder: `f(x)` subtracts 1; `g(x)` doubles; `h(x, y)` concatenates

A Computational Approach

Try all the small expressions with 3 function calls, then 4 calls, then 5 calls, etc.

```
def f(x):  
    return x - 1  
def g(x):  
    return 2 * x  
def h(x, y):  
    return int(str(x) + str(y))  
  
class Number:  
    def __init__(self, value):  
        self.value = value  
  
    def calls(self):  
        return 0  
  
    def __str__(self):  
        return str(self.value)  
  
class Call:  
    """A call expression."""  
    def __init__(self, f, operands):  
        self.f = f  
        self.operands = operands  
        self.value = f(*[e.value for e in operands])  
  
    def calls(self):  
        return 1 + sum(o.calls() for o in self.operands)  
  
    def __str__(self):  
        return f'{self.f.__name__}({",".join([str(o) for o in self.operands])})'
```

2: Functions

10: Containers

12: Objects

13: Attributes

9: Sequences

```
def smalls(n):  
    if n == 0:  
        return [Number(5)]  
    else:  
        results = []  
        for operand in smalls(n-1):  
            results.append(Call(f, [operand]))  
            results.append(Call(g, [operand]))  
        for k in range(n):  
            for first in smalls(k):  
                for second in smalls(n-k-1):  
                    if first.value > 0 and second.value > 0:  
                        results.append(Call(h, [first, second]))  
        return results  
  
def print_smallest():  
    result = []  
    for i in range(9):  
        result.extend([e for e in smalls(i) if e.value == 2024])
```

7: Recursion

11: Mutability

8: Tree Recursion

3: Control

By the Midterm, you can do this.