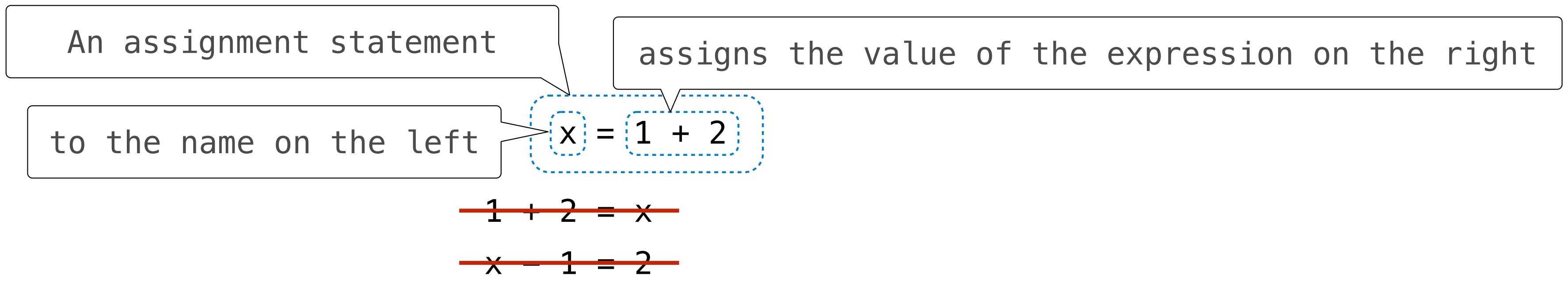
Functions

Announcements

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
```

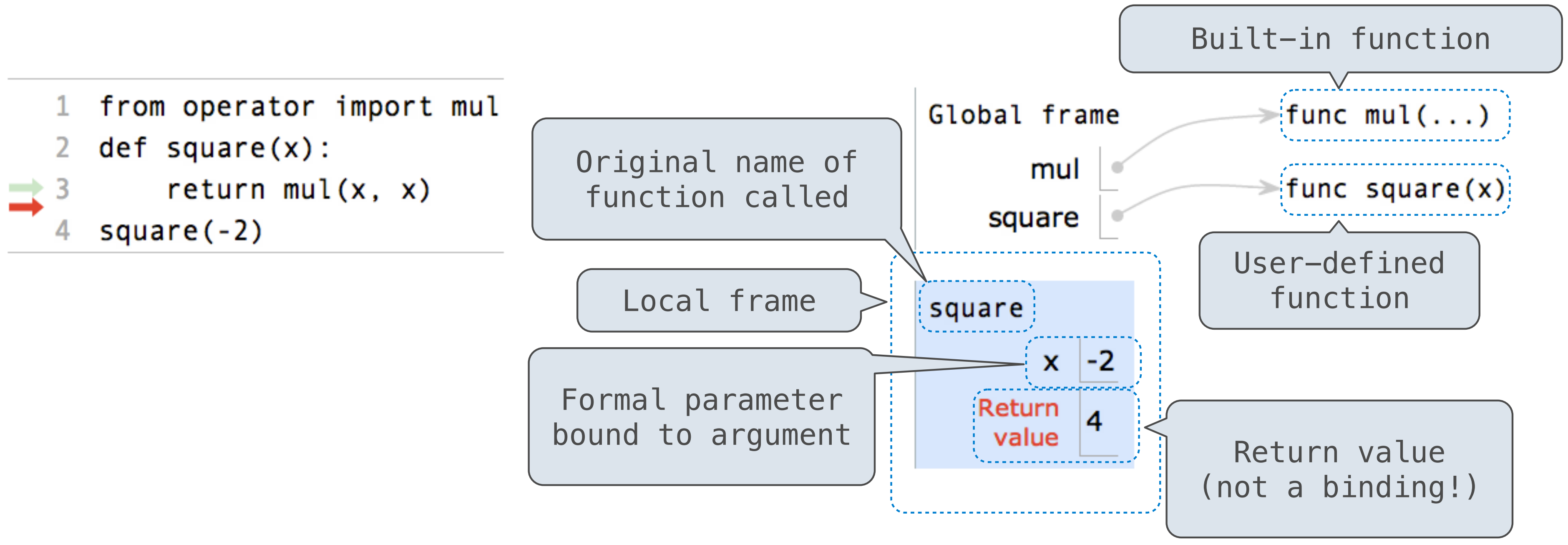
(Demo)

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



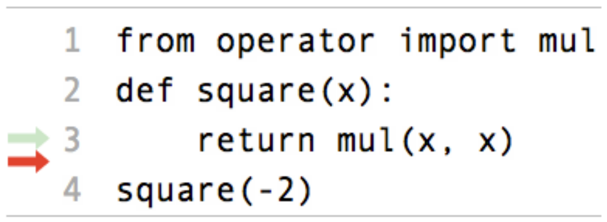
```python
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

http://pythontutor.com/composingprograms.html#code=from%20operator%20import%20mul%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20mul%28x,%20x%29%0Asquare%28-2%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Calling User-Defined Functions

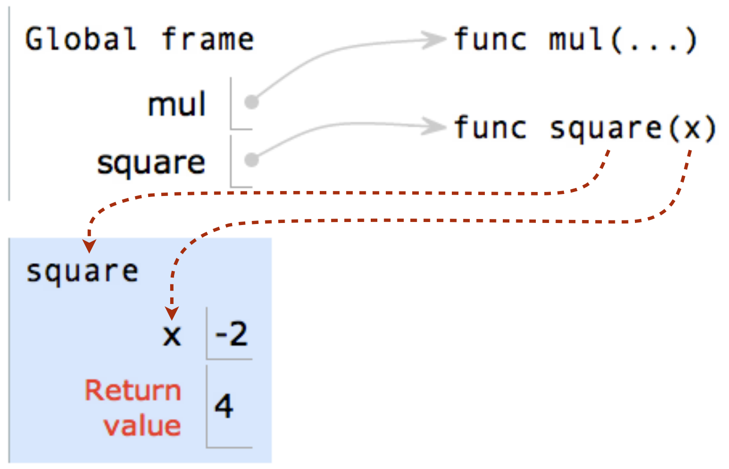
Procedure for calling/applying user-defined functions (version 1):

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



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

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



http://pythontutor.com/composingprograms.html#code=from%20operator%20import%20mul%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20mul%28x,%20x%29%0Asquare%28-2%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

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.

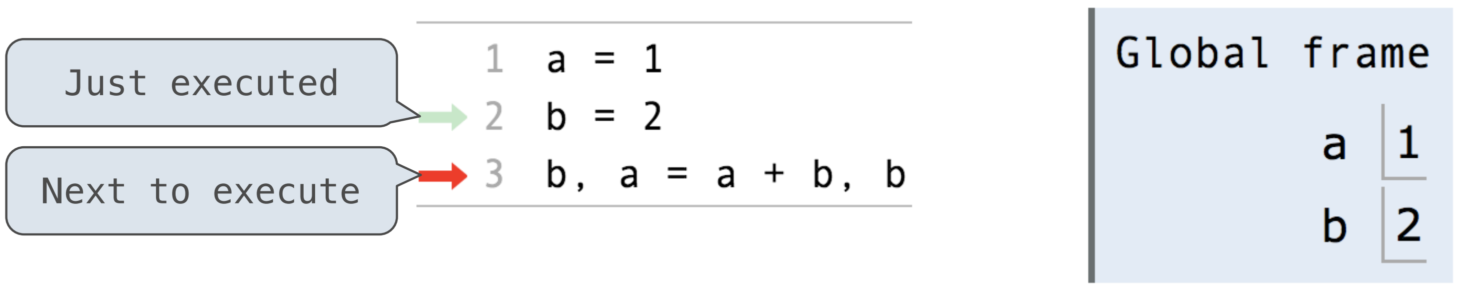
E.g., to look up some name in the body of the square function:

- 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.)

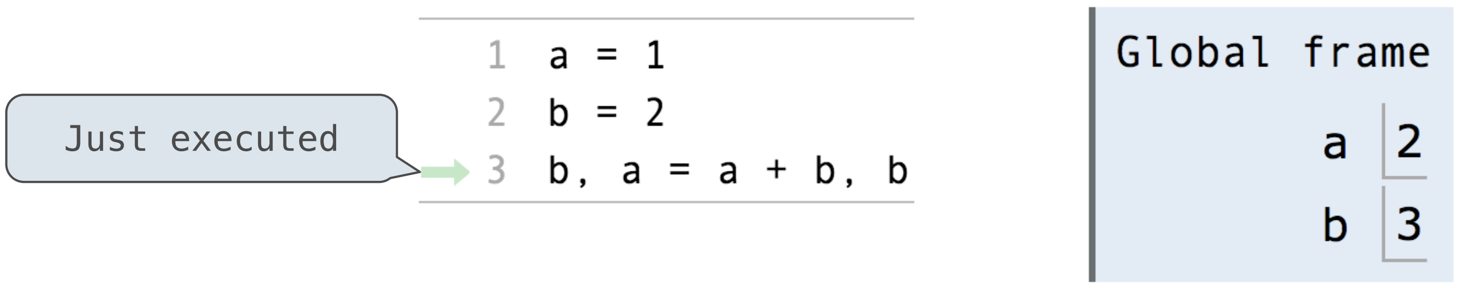
(Demo)

Multiple Assignment

Multiple Assignment



```
a = 1
b = 2
b, a = a + b, b
```



```
a = 1
b = 2
b, a = a + b, b
```

Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.

2. Bind all names to the left of = to those resulting values in the current frame.

(Demo)

http://pythontutor.com/composingprograms.html#code=a%20%3D%201%0Ab%20%3D%202%0Ab,%20a%20%3D%20a%20%2B%20b,%20b&cumulative=false&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Print and None

(Demo)

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 2023?

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 2023?

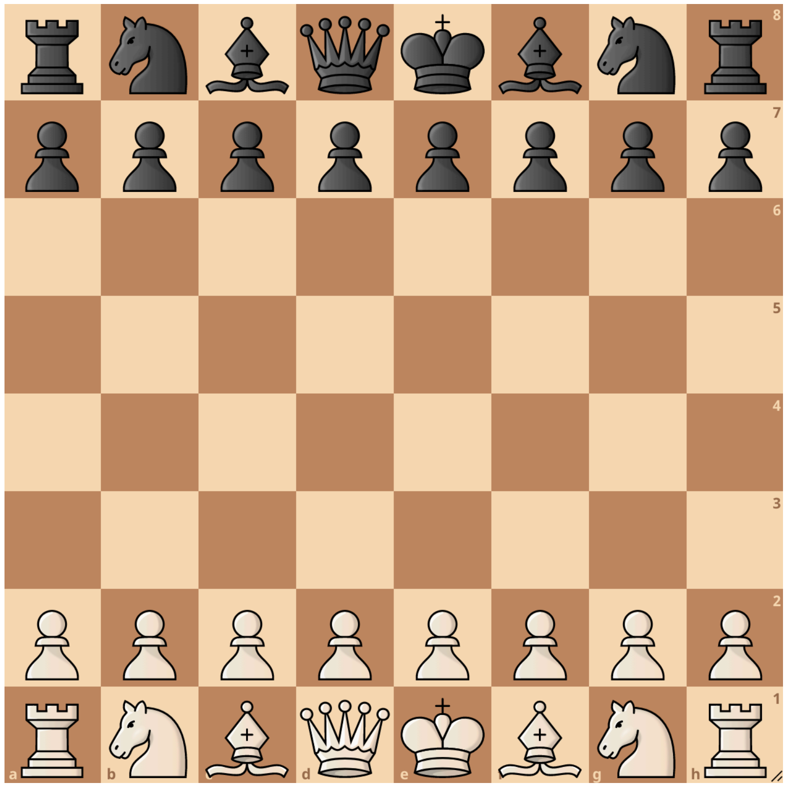
5➡️10➡️20

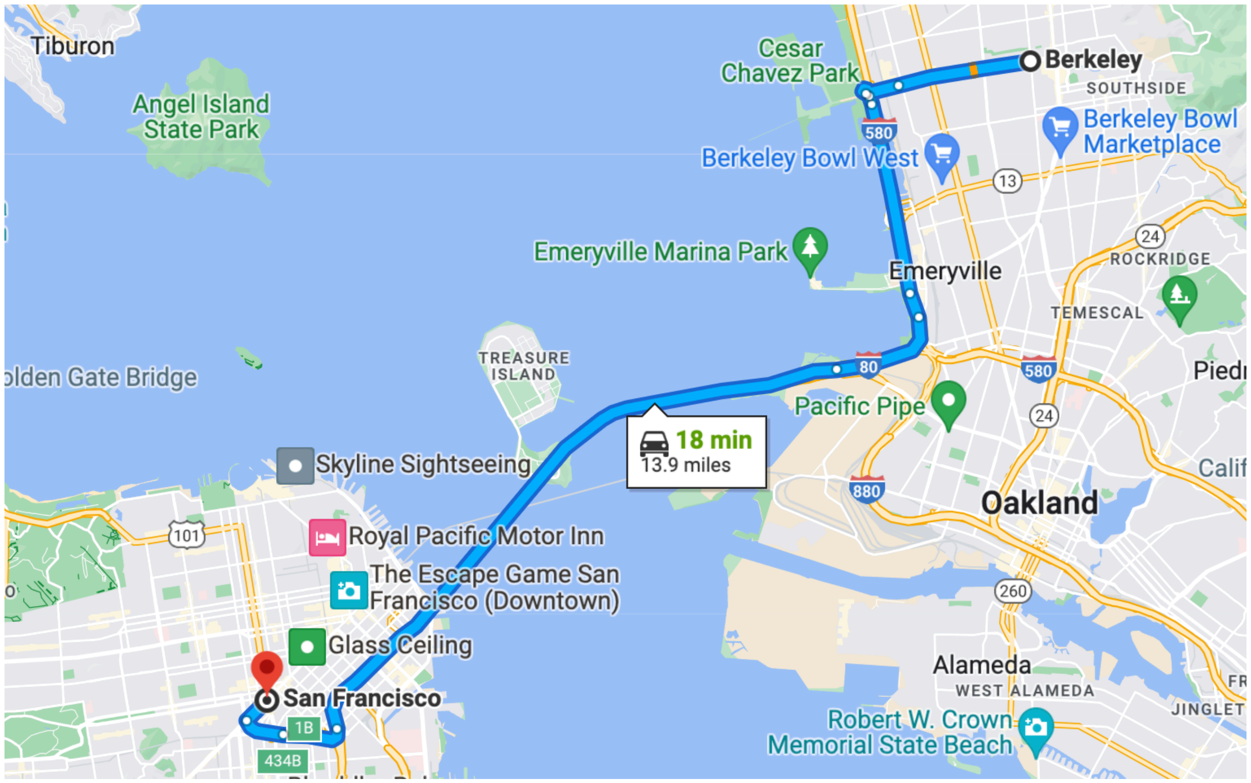
5➡️4➡️3➡️2

5➡️4➡️3

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

Searching for the Shortest





```
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 4 function calls, then 5 calls, then 6 calls, etc.
```

```
f(f(f(f(5)))) -> 1
g(f(f(f(5)))) -> 4
f(g(f(f(5)))) -> 5
g(g(f(f(5)))) -> 12
f(f(g(f(5)))) -> 6
g(f(g(f(5)))) -> 14
f(g(g(f(5)))) -> 15
g(g(g(f(5)))) -> 32
f(f(f(g(5)))) -> 7
g(f(f(g(5)))) -> 16
f(g(f(g(5)))) -> 17
g(g(f(g(5)))) -> 36
f(f(g(g(5)))) -> 18
g(f(g(g(5)))) -> 38
f(g(g(g(5)))) -> 39
g(g(g(g(5)))) -> 80
```

```
f(h(f(5),f(5))) -> 43
g(h(f(5),f(5))) -> 88
f(h(f(5),g(5))) -> 409
g(h(f(5),g(5))) -> 820
f(h(g(5),f(5))) -> 103
g(h(g(5),f(5))) -> 208
f(h(g(5),g(5))) -> 1009
g(h(g(5),g(5))) -> 2020
```

```
h(f(5),f(f(5))) -> 43
h(f(5),g(f(5))) -> 48
h(f(5),f(g(5))) -> 49
h(f(5),g(g(5))) -> 420
h(g(5),f(f(5))) -> 103
h(g(5),g(f(5))) -> 108
h(g(5),f(g(5))) -> 109
h(g(5),g(g(5))) -> 1020
h(f(f(5)),f(5)) -> 34
h(f(f(5)),g(5)) -> 310
h(g(f(5)),f(5)) -> 84
h(g(f(5)),g(5)) -> 810
h(f(g(5)),f(5)) -> 94
h(f(g(5)),g(5)) -> 910
h(g(g(5)),f(5)) -> 204
h(g(g(5)),g(5)) -> 2010
```

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

A Computational Approach

Try all the small expressions with 4 function calls, then 5 calls, then 6 calls, etc.

```
f(g(h(g(5),g(g(f(f(5))))))) -> 2023 has 8 calls and 27 characters.
f(g(h(g(5),g(f(f(g(f(5)))))))) -> 2023 has 9 calls and 30 characters.
f(h(g(g(5)),g(g(g(f(f(5))))))) -> 2023 has 9 calls and 30 characters.
f(h(g(g(5)),h(f(f(f(5))),f(5)))) -> 2023 has 9 calls and 32 characters.
f(h(f(f(h(g(g(5)),f(5)))),f(5))) -> 2023 has 9 calls and 32 characters.
f(h(f(h(g(g(5)),f(f(5)))),f(5))) -> 2023 has 9 calls and 32 characters.
f(h(h(g(g(5)),f(f(f(5)))),f(5))) -> 2023 has 9 calls and 32 characters.
h(g(g(5)),f(g(g(g(f(f(5))))))) -> 2023 has 9 calls and 30 characters.
h(g(g(5)),f(h(f(f(f(5))),f(5)))) -> 2023 has 9 calls and 32 characters.
h(g(g(5)),h(f(f(f(5))),f(f(5)))) -> 2023 has 9 calls and 32 characters.
h(f(f(h(g(g(5)),f(5)))),f(f(5))) -> 2023 has 9 calls and 32 characters.
h(f(h(g(g(5)),f(f(5)))),f(f(5))) -> 2023 has 9 calls and 32 characters.
h(h(g(g(5)),f(f(f(5)))),f(f(5))) -> 2023 has 9 calls and 32 characters.
```

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

A Computational Approach

Try all the small expressions with 4 function calls, then 5 calls, then 6 calls, etc.

```
def f(x):
    return x - 1                                   2
def g(x):
    return 2 * x
def h(x, y):
    return int(str(x) + str(y))                    12

class Number:
    def __init__(self, value):                     18
        self.value = value

    def __str__(self):                             21
        return str(self.value)

    def calls(self):
        return 0

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])    11

    def __str__(self):
        return f'{self.f.__name__}({",".join(map(str, self.operands))})'    4

    def calls(self):
        return 1 + sum(o.calls() for o in self.operands)    16
```

```
@memo                                                  22
def smalls(n):
    """A list of all small expressions with n calls."""
    return list(smalls_gen(n))

def smalls_gen(n):
    if n == 0:
        yield Number(5)                                17
    else:
        for operand in smalls(n-1):                    9
            yield Call(f, [operand])
            yield Call(g, [operand])
        for k in range(1, n-1):
            for first in smalls(k):                    10
                for second in smalls(n-k-1):
                    if first.value > 0 and second.value > 0:    3
                        yield Call(h, [first, second])

result = []
for i in range(11):
    result.extend([e for e in smalls(i) if e.value == 2023])    15
```

By Midterm 2, you can do this.