61A Lecture 2

Monday, August 29

The Elements of Programming

- Primitive Expressions and Statements
  - *The simplest building blocks of a language*
- Means of Combination
  - *Compound elements are built from simpler ones*
- Means of Abstraction
  - *Compound elements can be named and manipulated as units*

Programming languages allow us to communicate, too

Functions and Data

Data: Stuff we want to manipulate

"The Art of Computer Programming"

2

This slide

Donald Knuth
(Ka-NOOTH)

Functions: Rules for manipulating data

Count the words in a line of text

Add numbers

Load the next slide

Pronounce someone's name

Types of expressions

An expression

describes a computation

and evaluates to a value

$18 + 69$

$\frac{6}{23}$

$\sin \pi$

$f(x)$

$\sqrt{3493161}$

$\sum_{i=1}^{100} i$

$|-1869|$

$\binom{69}{18}$

Call Expressions in Python

All expressions can use function call notation

(Demo)

Anatomy of a Call Expression

add	(	2	,	3	)
Operator		*Operand 0*		*Operand 1*	

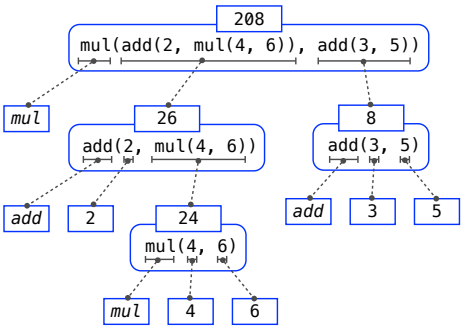
Operators and operands are expressions

So they evaluate to values

Evaluation procedure for call expressions:

1. Evaluate the operator and operand subexpressions
2. Apply the function that is the value of the operator subexpression to the arguments that are the values of the operand subexpression

Evaluating Nested Expressions



The Print Function

(Demo)

Pure Functions & Non-Pure Functions

Pure Functions

Function signature: how many parameters

−2 ▶ *abs*(number): ▶ 2

Only produces return values

2, 100 ▶ *pow*(x, y): ▶ 126765060022822940149670320537 6

Non-Pure Functions

−2 ▶ *print*(...): ▶ None

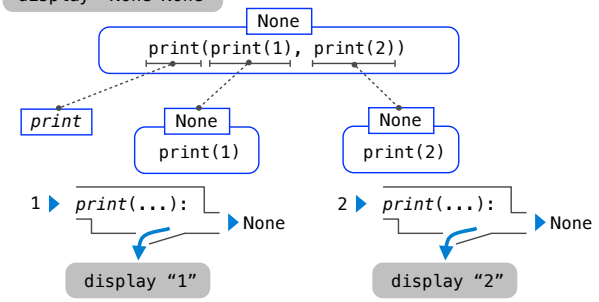
display "−2"

May create side effects

Nested Expressions with Print

None, None ▶ *print*(...): ▶ None

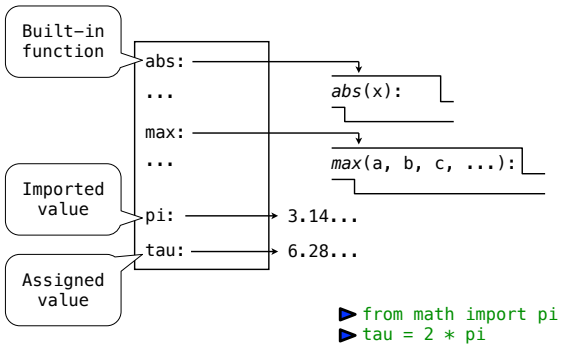
display "None None"



Names and Assignment

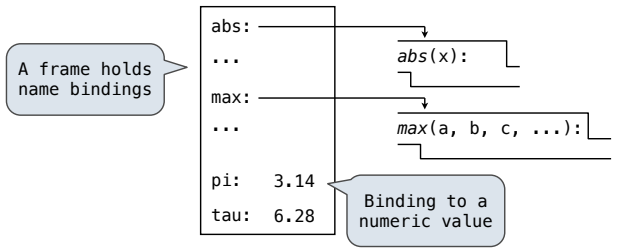
(Demo)

Environments



The environment does not track where names came from

Environments




```
from math import pi
tau = 2 * pi
```

The environment does not track where names came from

User-Defined Functions

Named values are a simple means of abstraction

Named *expressions* are a more powerful means of abstraction

def expressions:
- Create a new function
- Bind a name to it

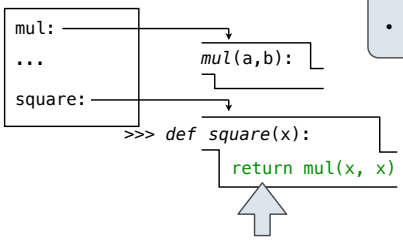
```
>>> def <name>(<formal parameters>):
        return <return expression>
```


User-Defined Functions

Named values are a simple means of abstraction

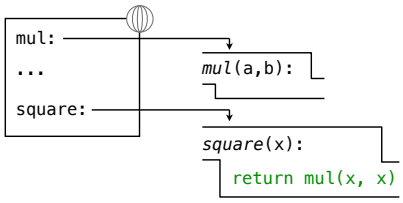
Named *expressions* are a more powerful means of abstraction

def expressions:
- Create a new function
- Bind a name to it



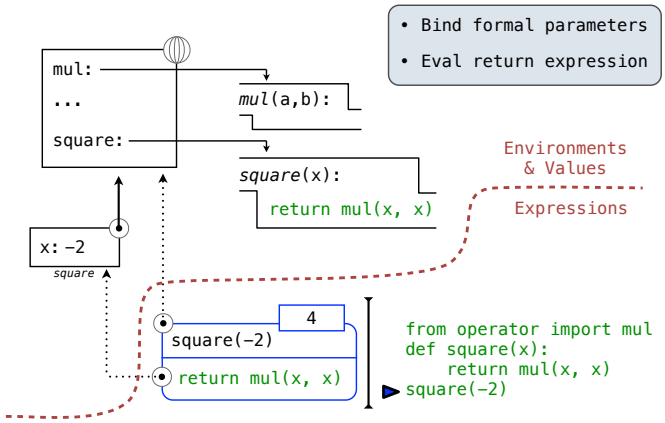
(Demo)

Calling User-Defined Functions

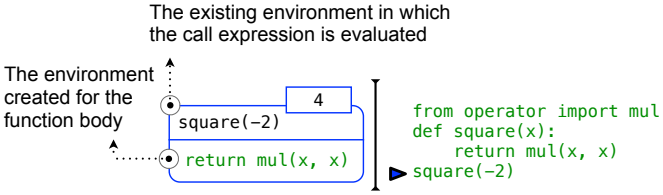



```
from operator import mul
def square(x):
    return mul(x, x)
```

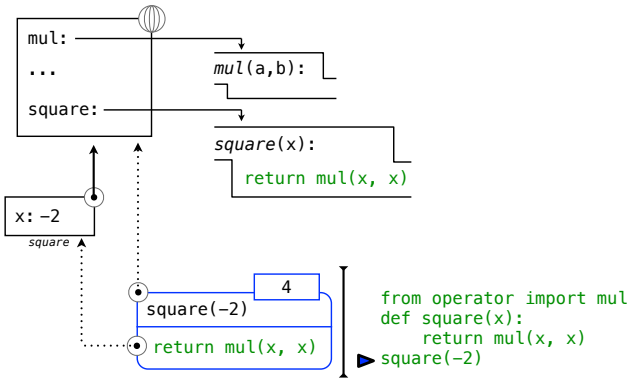

Calling User-Defined Functions



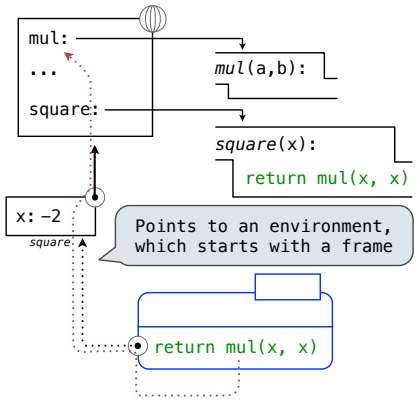
Calling User-Defined Functions



Calling User-Defined Functions



Evaluating a Name in an Environment

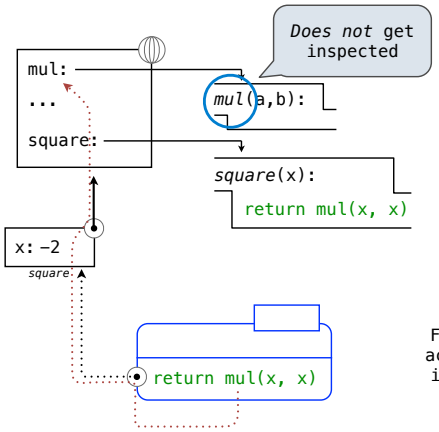


A name evaluates to the **value bound** to that name

...in the **earliest** frame of the **current environment**

...in which that **name is found**

Intrinsic Function Names Don't Play a Role

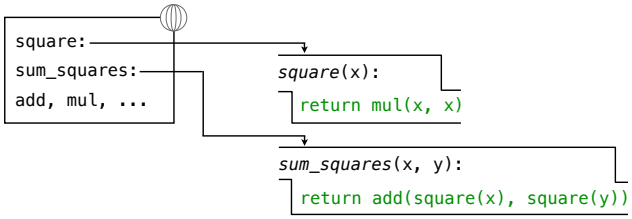


Witness Protection Program

Functions aren't accessed by their intrinsic names!

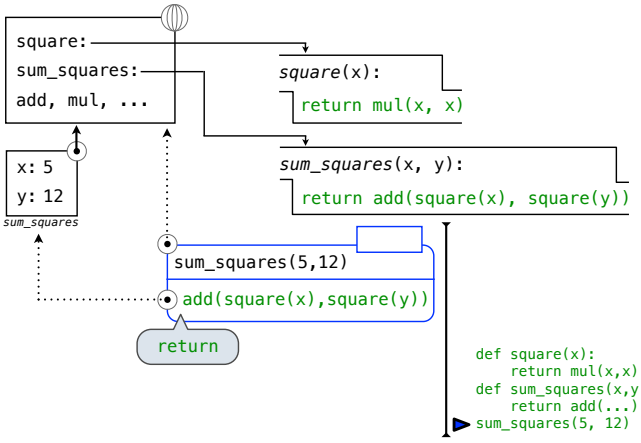
(Demo)

Example: Function Application



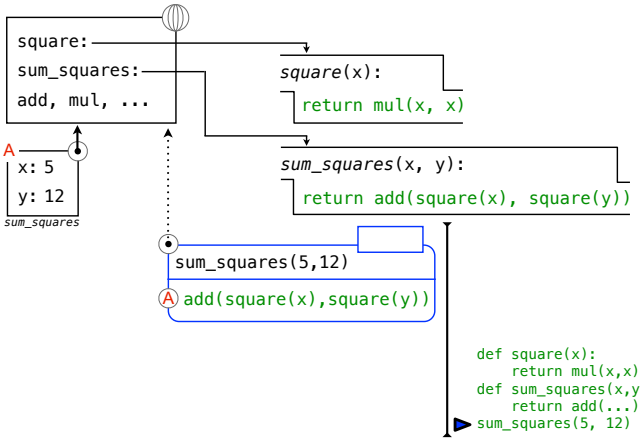

```
def square(x):
    return mul(x,x)
def sum_squares(x, y):
    return add(square(x),square(y))
```


Example: Function Application



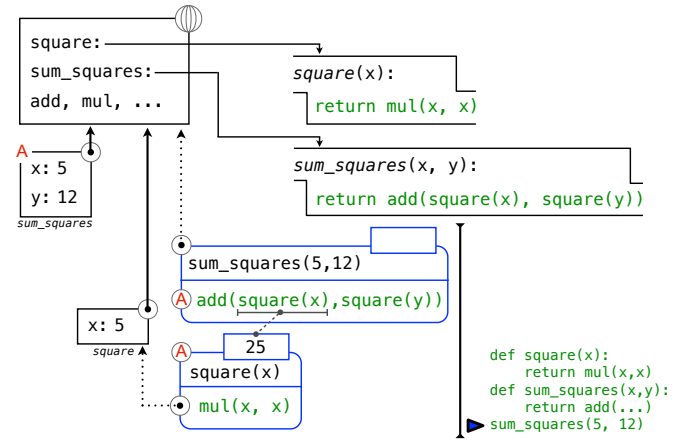

```
def square(x):
    return mul(x,x)
def sum_squares(x,y):
    return add(...)
sum_squares(5, 12)
```


Example: Function Application




```
def square(x):
    return mul(x,x)
def sum_squares(x,y):
    return add(...)
sum_squares(5, 12)
```

Example: Function Application



Example: Function Application