



# CS61A Lecture 2

Amir Kamil  
UC Berkeley  
January 25, 2013

# Announcements



- Reminder: hw0 due tonight!
  - Read html file for instructions
  - Use our .py templates for homework and projects
- Find answers to your questions on Piazza
  - Use search functionality
  - Don't post private question unless it is not of general interest
- In-class quiz next Friday
  - Bring a writing implement

# Review: Expressions



# Review: Expressions



---

Primitive expressions:

# Review: Expressions



Primitive expressions:

2

Number

# Review: Expressions



Primitive expressions:

2

add

Number

Name

# Review: Expressions



Primitive expressions:

2

Number

add

Name

'hello'

String

# Review: Expressions



Primitive expressions:

2

add

'hello'

Number

Name

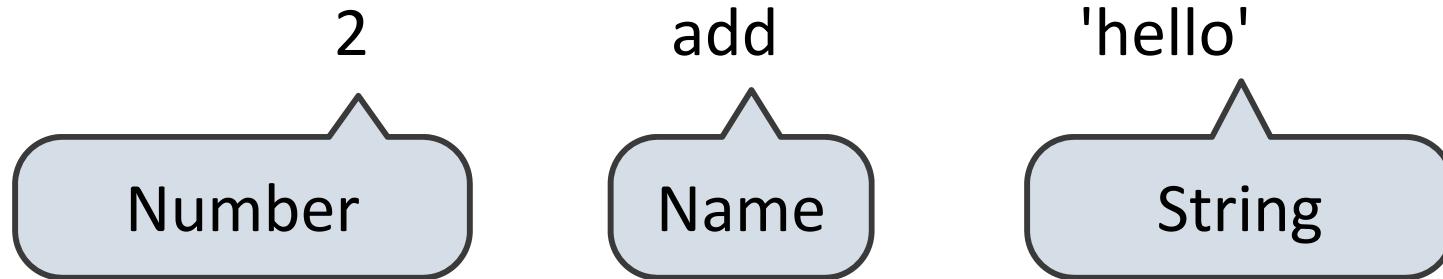
String

Call expressions:

# Review: Expressions



Primitive expressions:



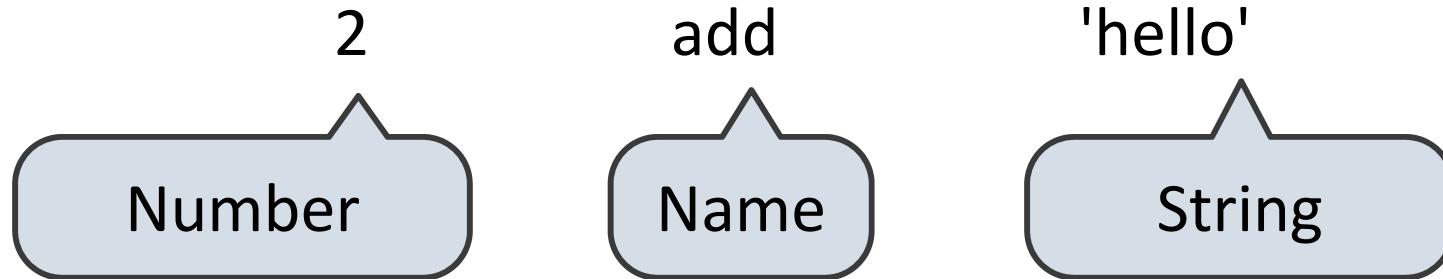
Call expressions:



# Review: Expressions



Primitive expressions:



Call expressions:



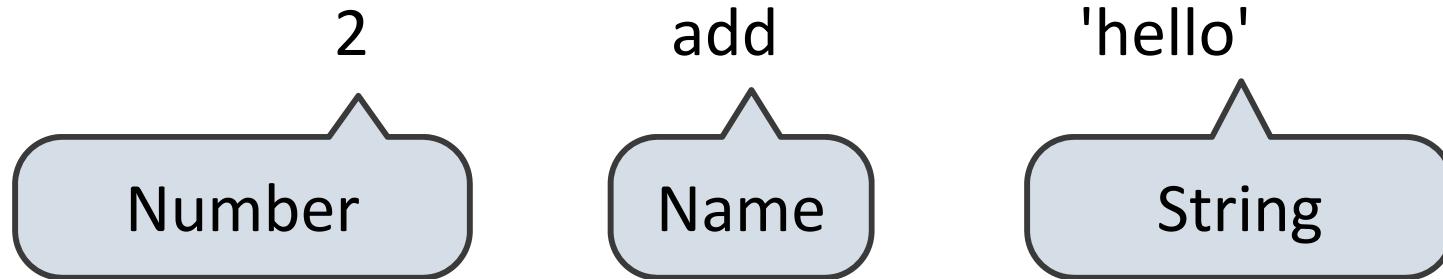
One big nested  
call expression

`max(min(pow(3, 5), -4), min(1, -2))`

# Review: Expressions



Primitive expressions:



Call expressions:



One big nested  
call expression

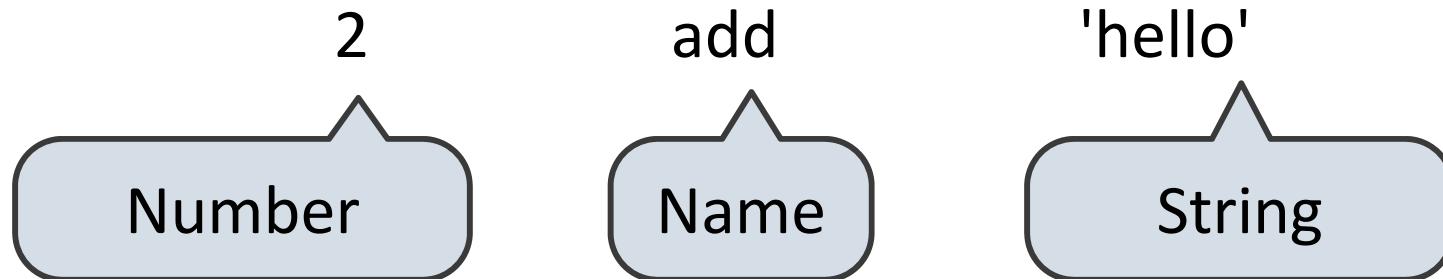
`max(min(pow(3, 5), -4), min(1, -2))`

Infix operators represent implicit call expressions

# Review: Expressions



Primitive expressions:



Call expressions:



One big nested  
call expression

`max(min(pow(3, 5), -4), min(1, -2))`

Infix operators represent implicit call expressions

$2 + 3 \rightarrow \text{add}(2, 3)$

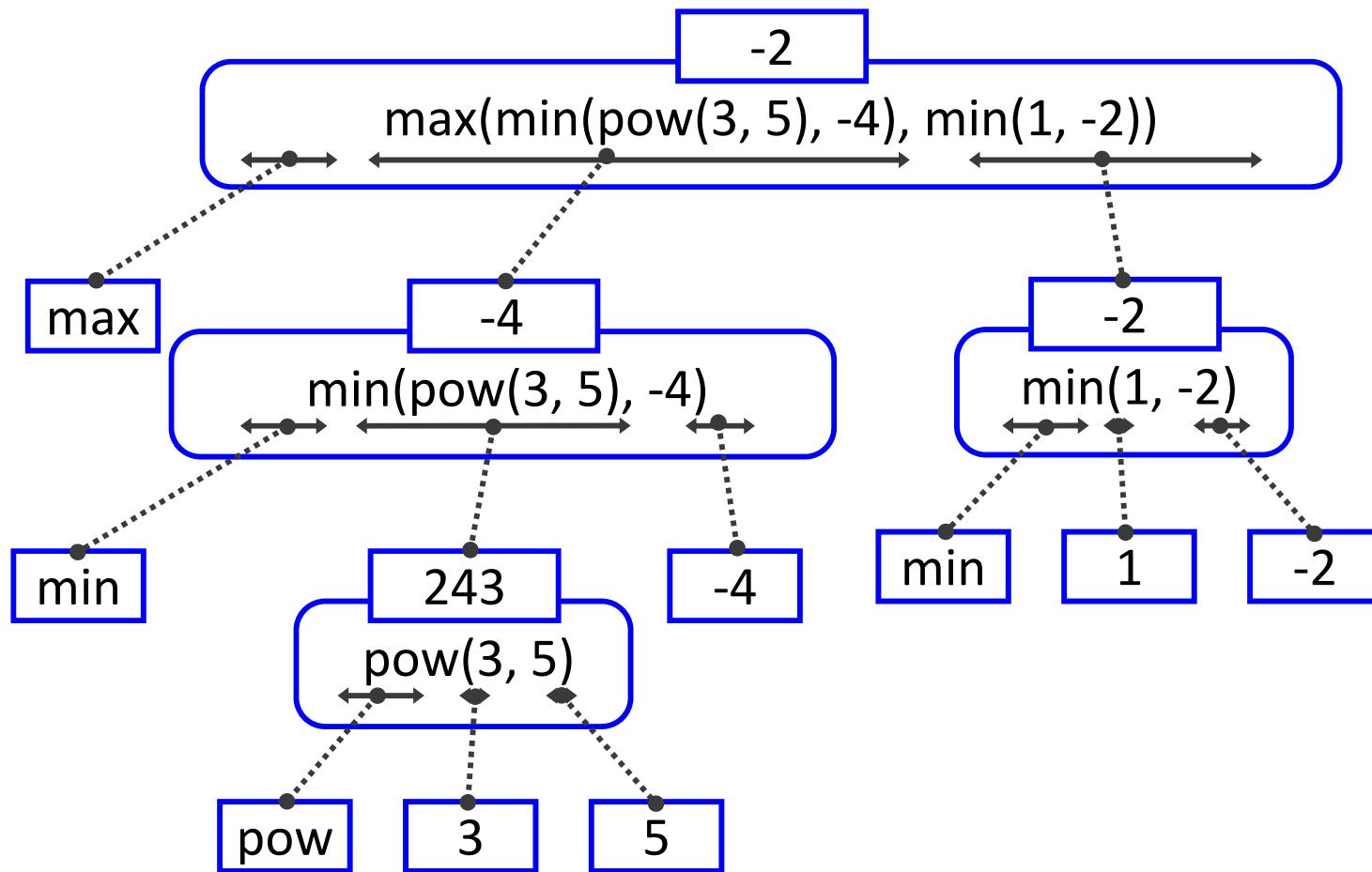
# Review: Evaluation Procedure



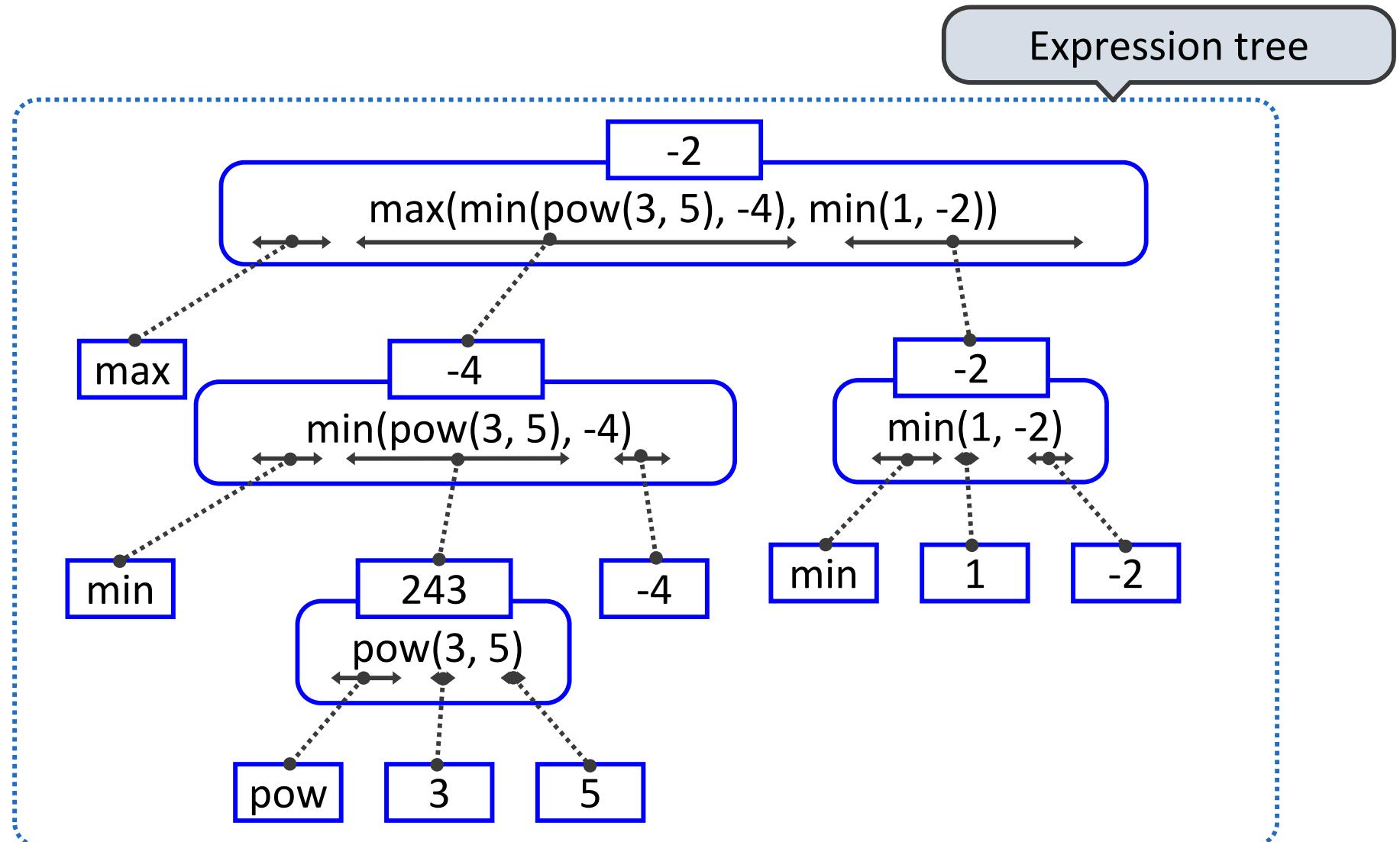
---

`max(min(pow(3, 5), -4), min(1, -2))`

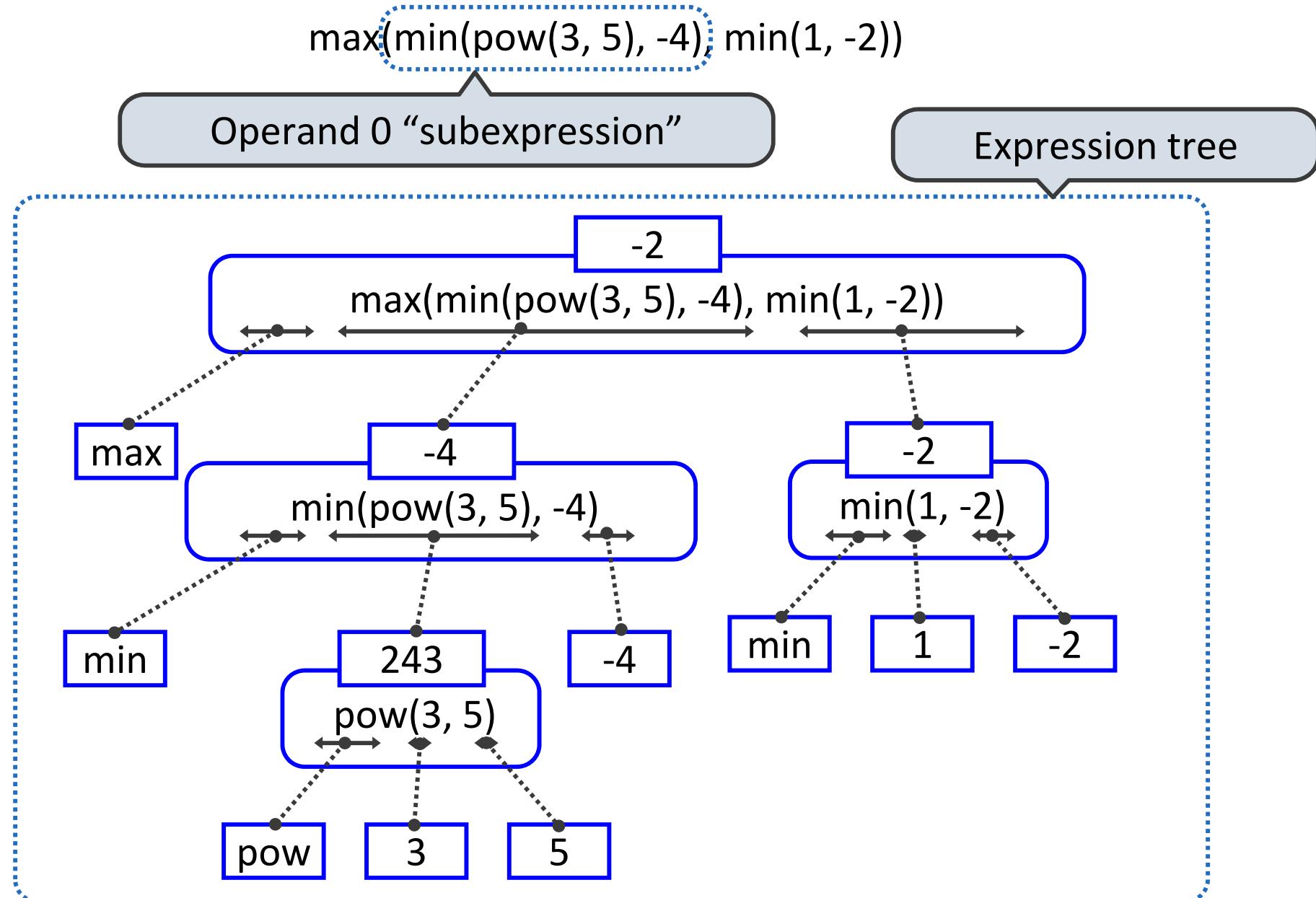
# Review: Evaluation Procedure

$$\max(\min(\text{pow}(3, 5), -4), \min(1, -2))$$


# Review: Evaluation Procedure


$$\max(\min(\text{pow}(3, 5), -4), \min(1, -2))$$


# Review: Evaluation Procedure



# Review: Evaluation Procedure

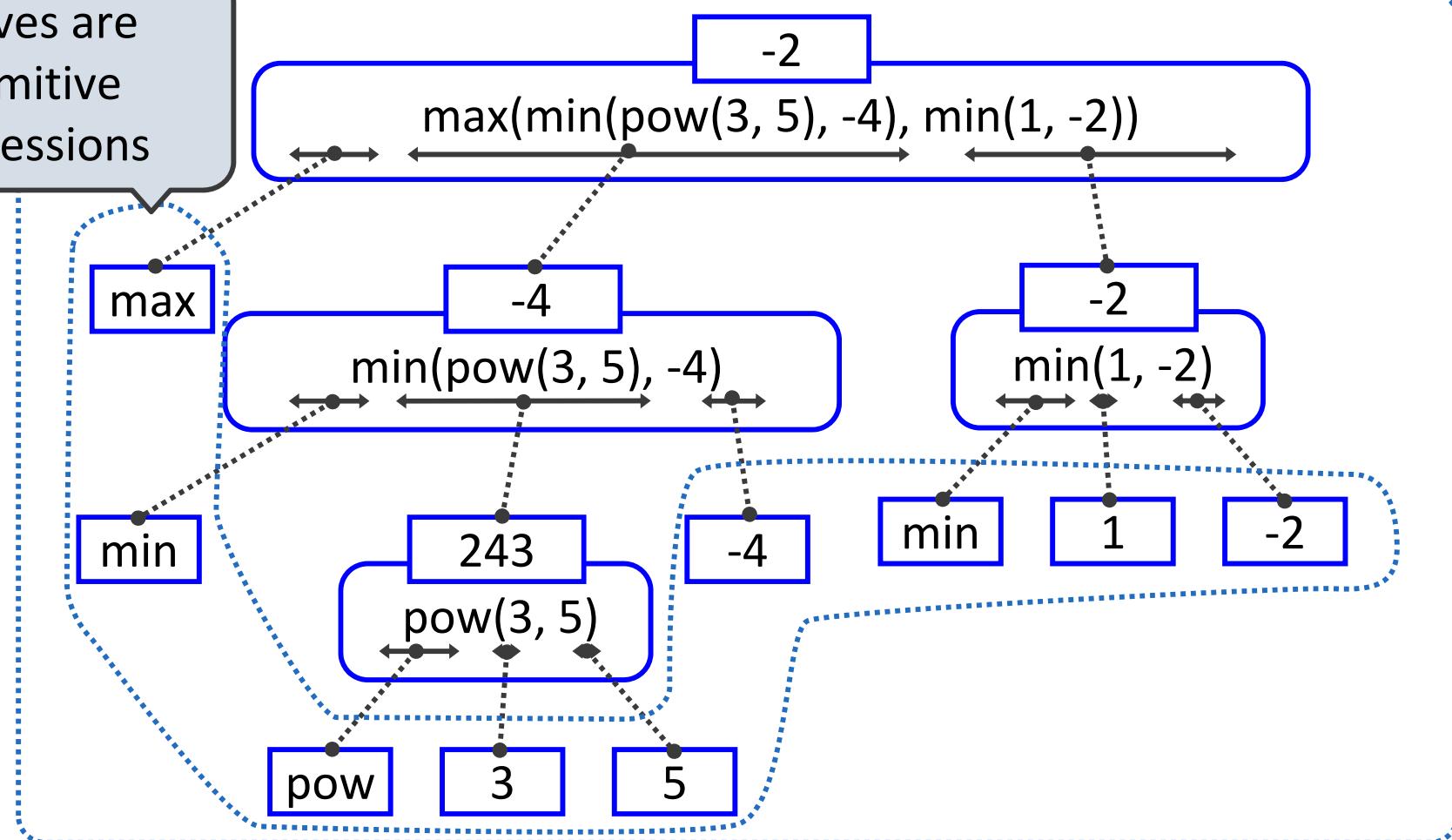


max(min(pow(3, 5), -4), min(1, -2))

Operand 0 “subexpression”

Expression tree

Leaves are primitive expressions



# Types of Functions



---

Pure Functions

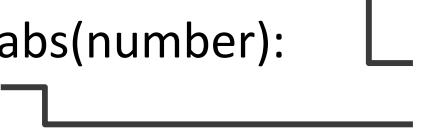
Non-Pure Functions

# Types of Functions



## Pure Functions

abs(number):



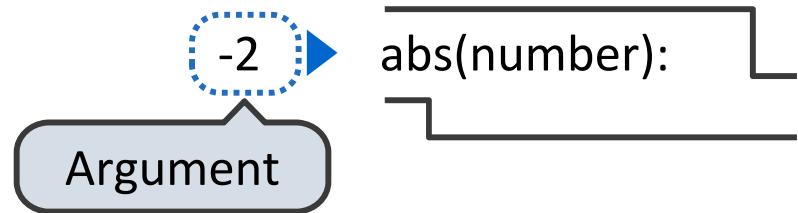
A diagram illustrating a function call. It shows a horizontal line with a bracket underneath it. The word "abs(number):" is positioned above the line. A bracket is drawn below the line, starting from the left end and ending at the right end, indicating that the entire expression is a single function call.

## Non-Pure Functions

# Types of Functions



## Pure Functions

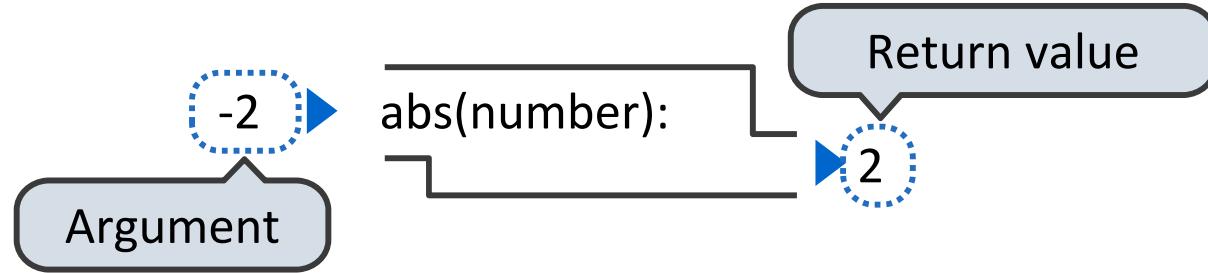


## Non-Pure Functions

# Types of Functions



## Pure Functions

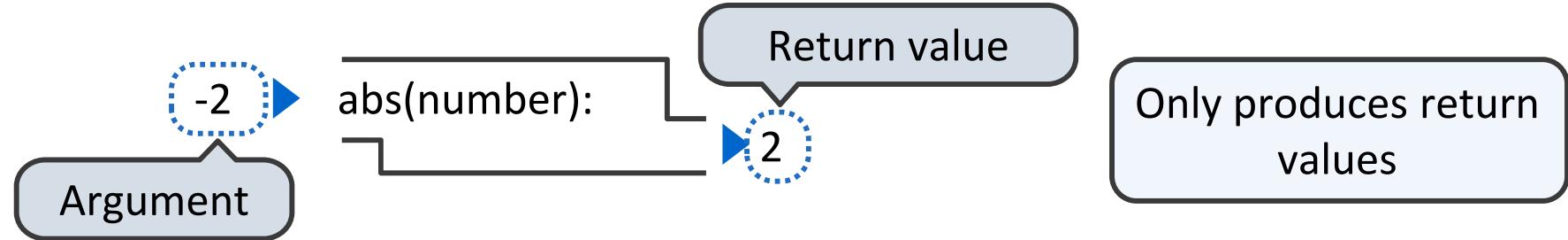


## Non-Pure Functions

# Types of Functions



## Pure Functions

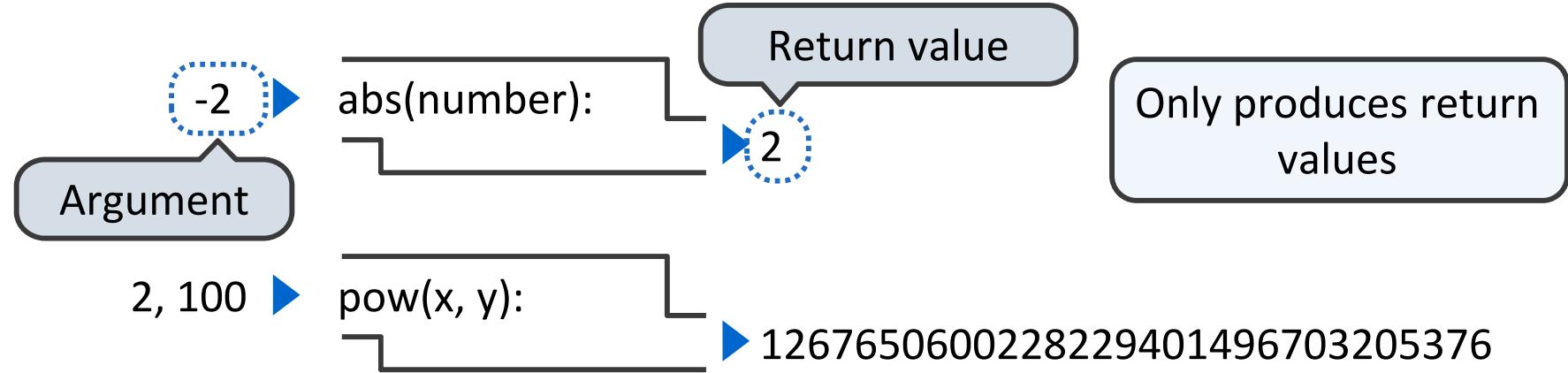


## Non-Pure Functions

# Types of Functions



## Pure Functions

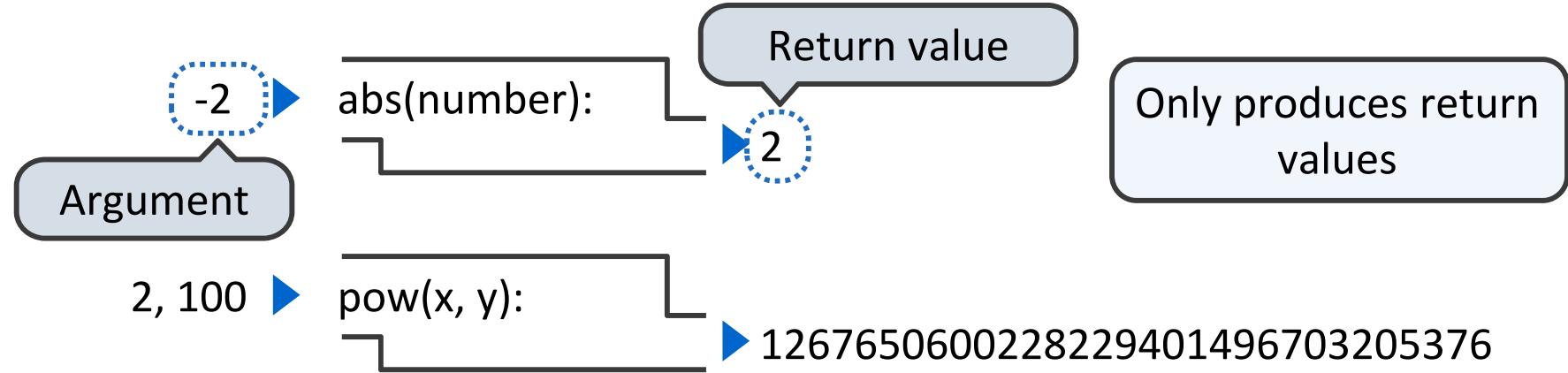


## Non-Pure Functions

# Types of Functions



## Pure Functions



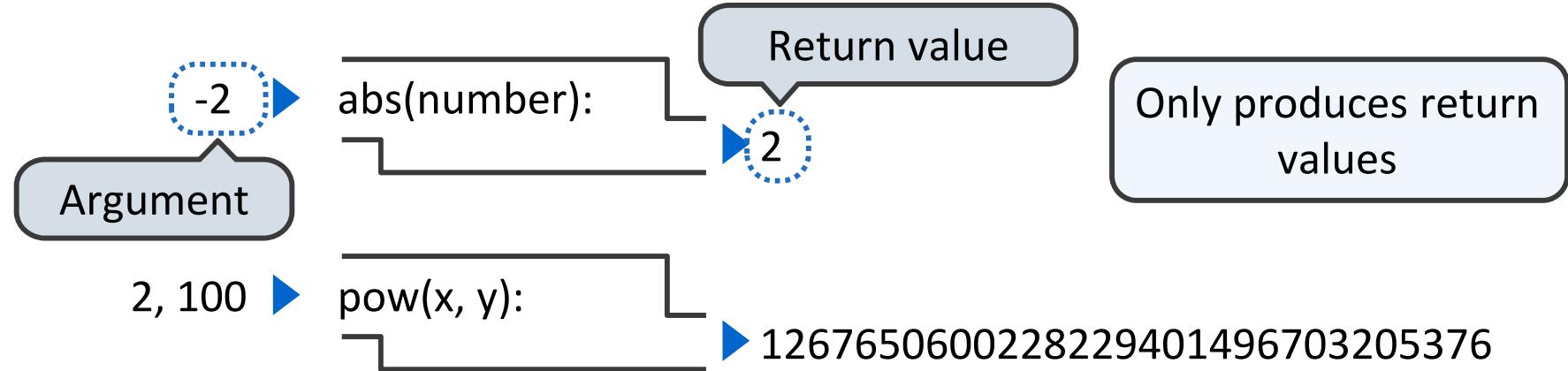
## Non-Pure Functions



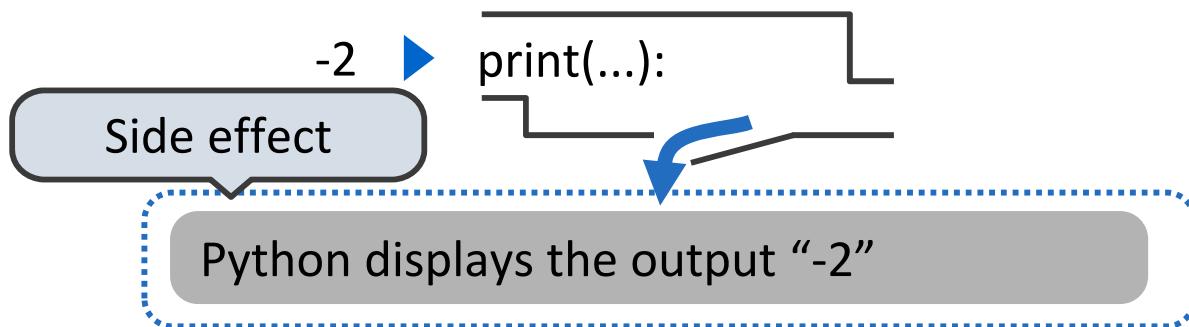
# Types of Functions



## Pure Functions



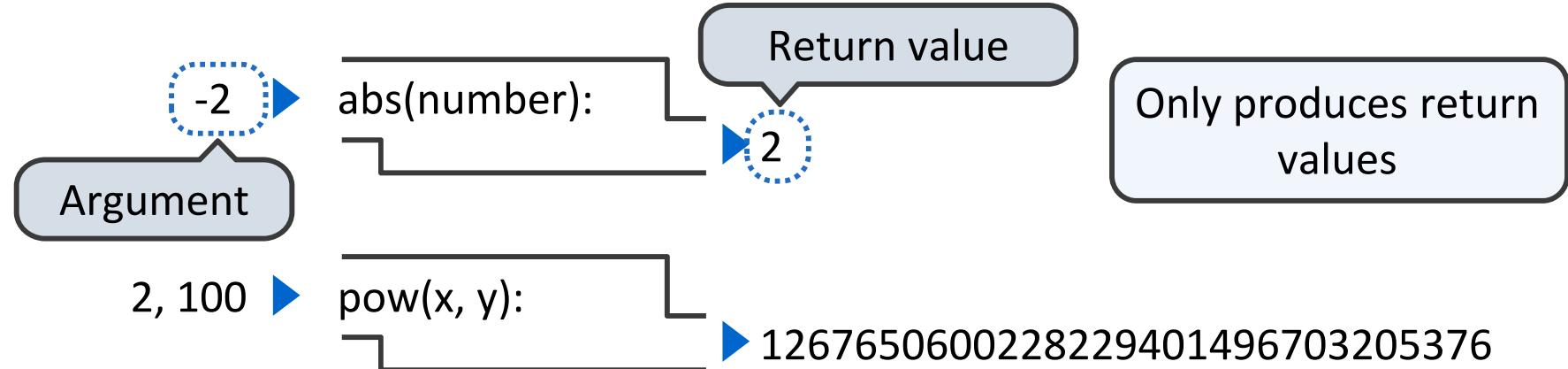
## Non-Pure Functions



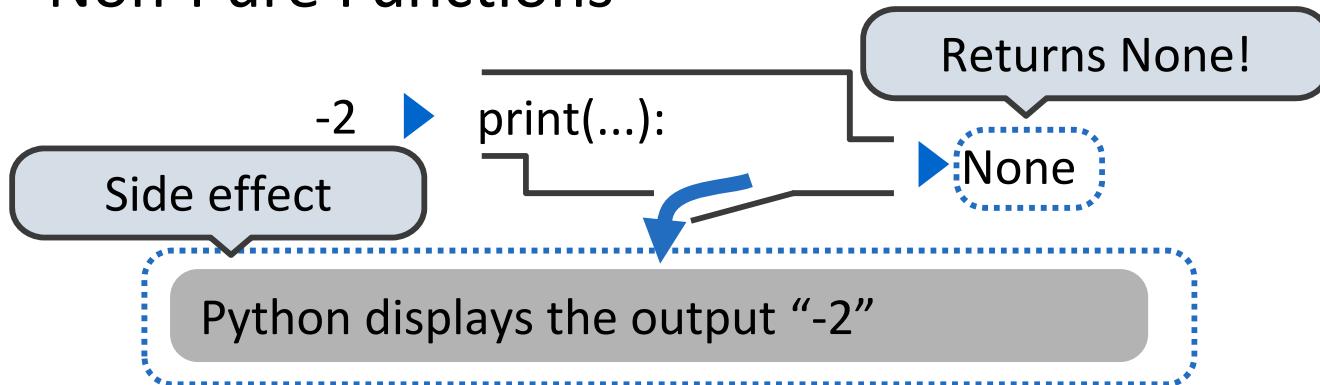
# Types of Functions



## Pure Functions



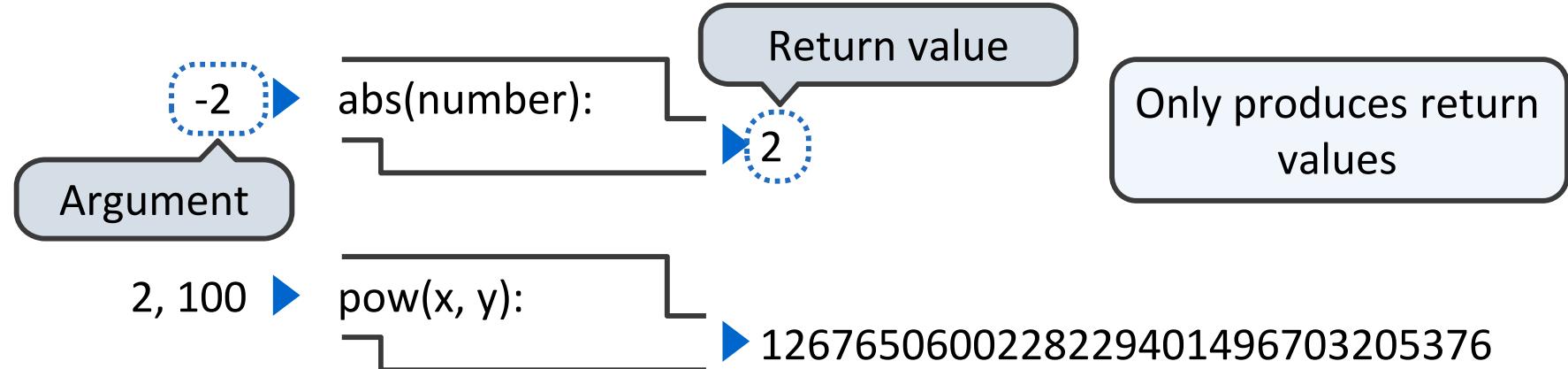
## Non-Pure Functions



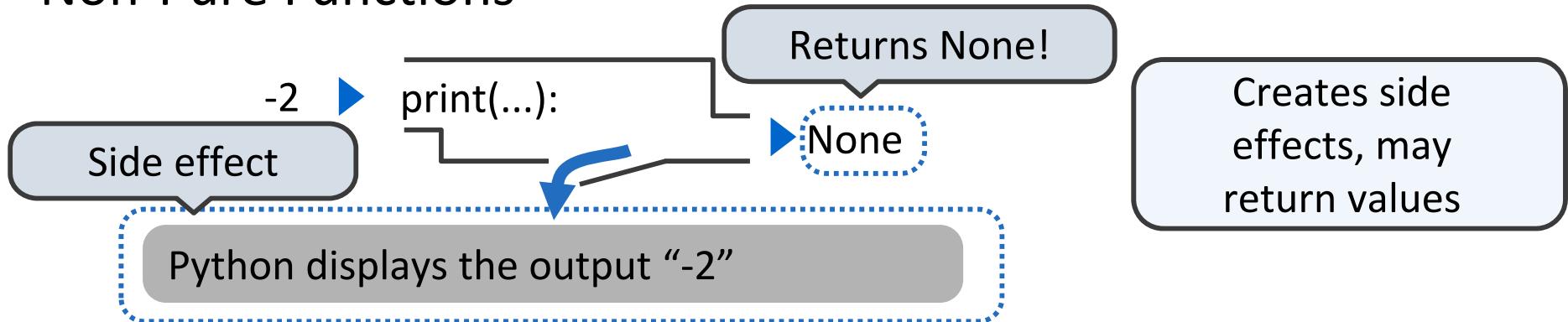
# Types of Functions



## Pure Functions



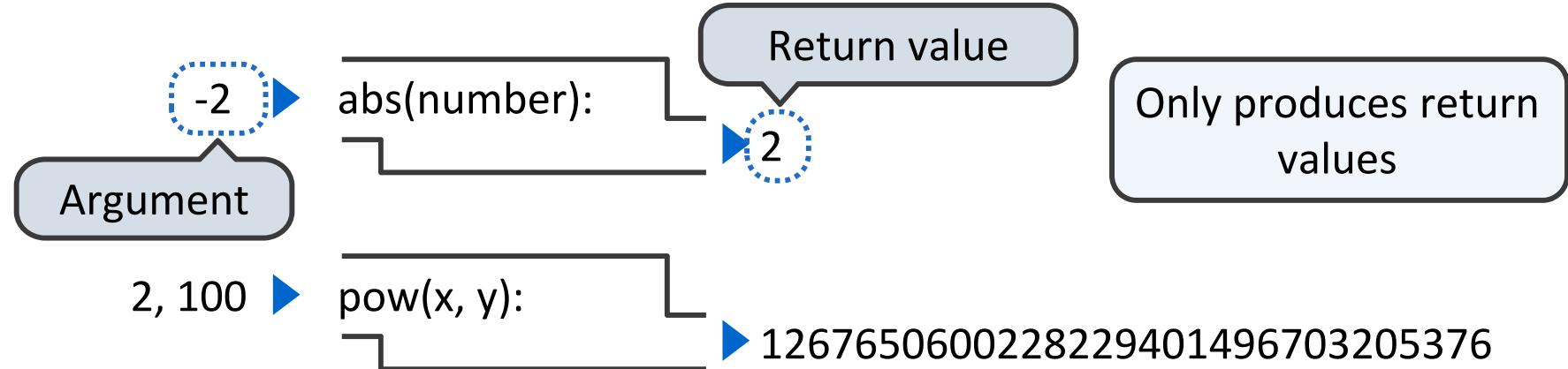
## Non-Pure Functions



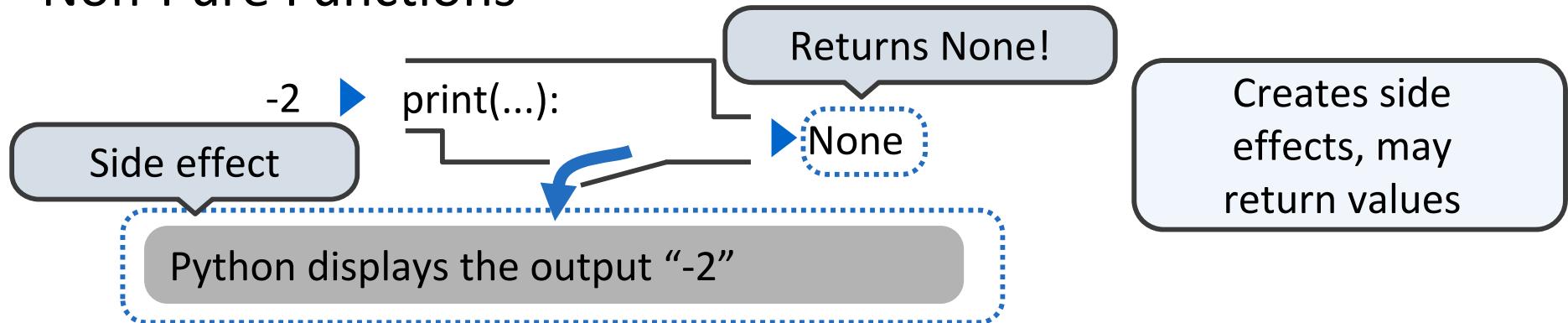
# Types of Functions



## Pure Functions



## Non-Pure Functions



The interactive interpreter displays all return values except `None`.

# Nested Print Expressions



None, None ▶

```
print(...):  
    ↗  
    ↘  
    ↗ ↘
```

```
>>> print(print(1), print(2))
```

# Nested Print Expressions



None, None ➤

```
print(...):  
  ↗  
  ↘  
    ↗  
    ↘
```

```
>>> print(print(1), print(2))
```

```
print(print(1), print(2))
```

# Nested Print Expressions



None, None ➤

```
print(...):
```

A calligraphic brace is drawn around the word "print" in the code. It has two horizontal strokes at the top and bottom, with a vertical stroke connecting them. The vertical stroke ends in a small hook pointing right.

```
>>> print(print(1), print(2))
```

A calligraphic brace is drawn around the entire nested expression "print(print(1), print(2))". It has a horizontal stroke at the top and bottom, with a vertical stroke on the left side. The vertical stroke ends in a small hook pointing right.

```
print(print(1), print(2))
```

A calligraphic brace is drawn around the innermost expression "print(1)". It has a horizontal stroke at the top and bottom, with a vertical stroke on the left side. The vertical stroke ends in a small hook pointing right.

```
print
```

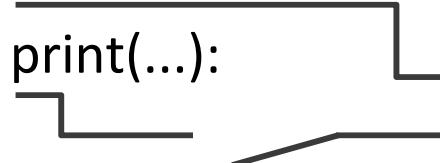
A calligraphic brace is drawn around the outermost expression "print(print(1), print(2))". It has a horizontal stroke at the top and bottom, with a vertical stroke on the left side. The vertical stroke ends in a small hook pointing right.

A dotted arrow points from the "print" box down to the innermost "print" box, indicating the flow of execution from the outer function call to the inner one.

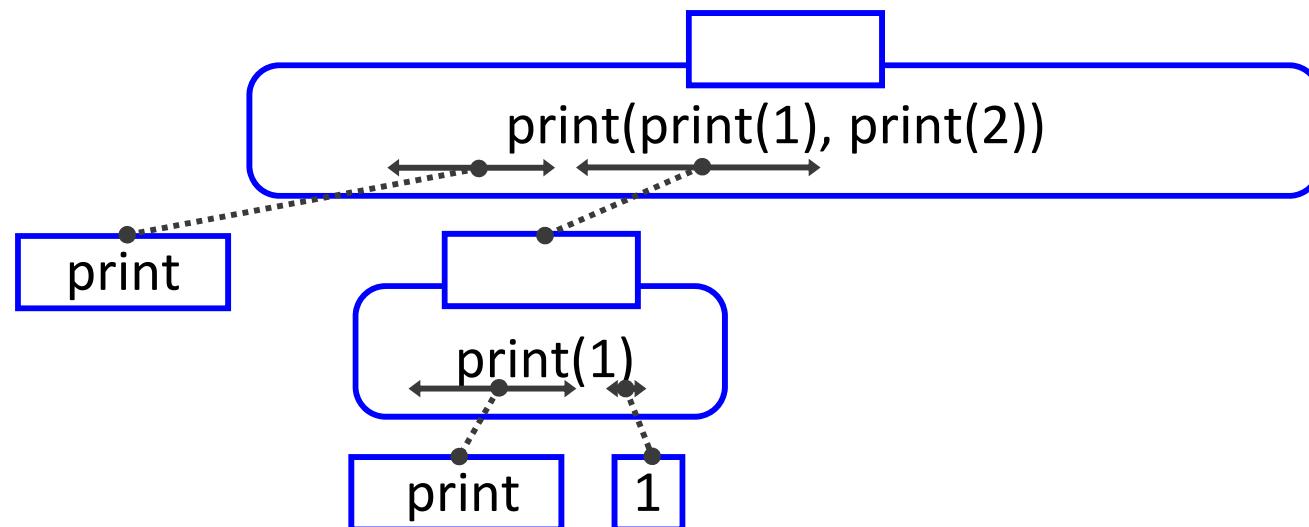
# Nested Print Expressions



None, None ➤



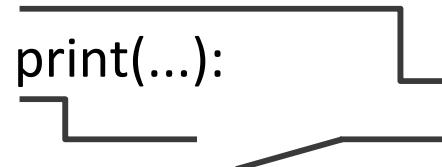
```
>>> print(print(1), print(2))
```



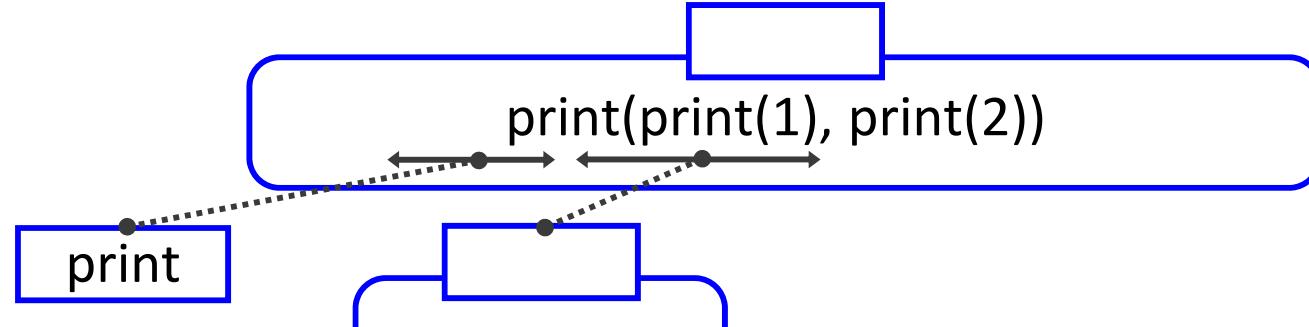
# Nested Print Expressions



None, None ➤



```
>>> print(print(1), print(2))  
1
```



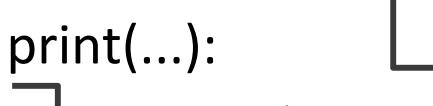
print

print(1)

print

1

1 ➤



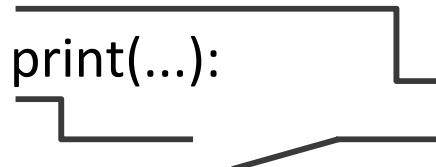
None

display "1"

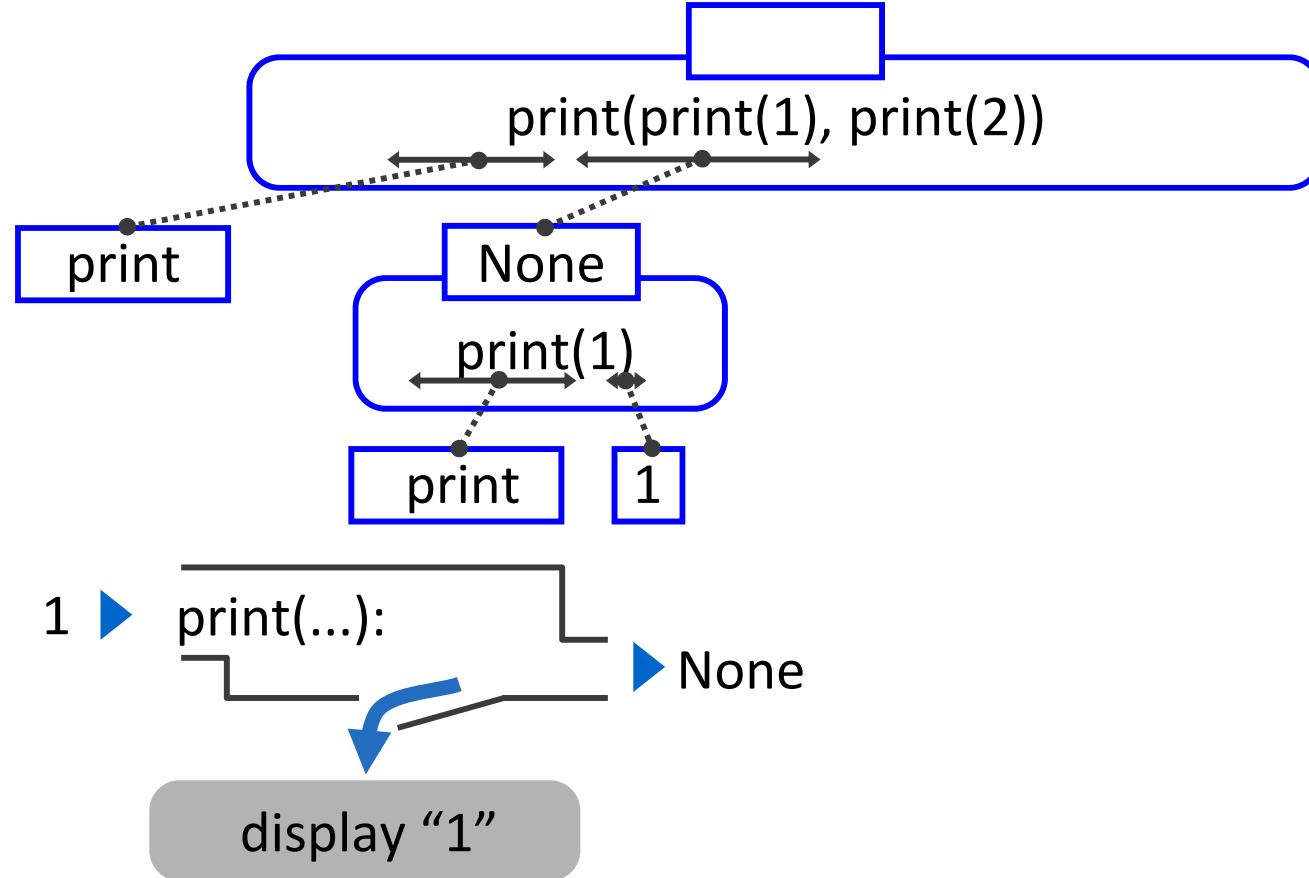
# Nested Print Expressions



None, None ➤



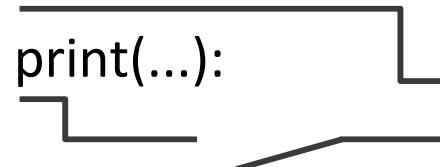
```
>>> print(print(1), print(2))  
1
```



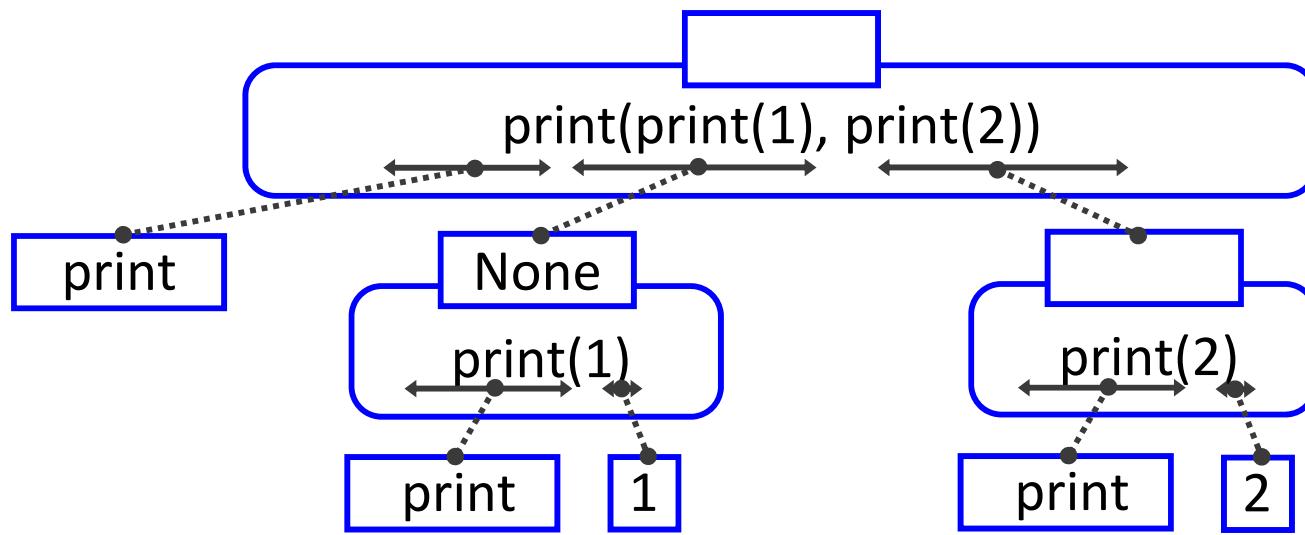
# Nested Print Expressions



None, None ➤



```
>>> print(print(1), print(2))  
1
```



1 ➤

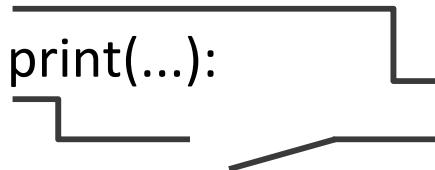
```
graph TD; A[print(...):] --> B[None]; A --> C[None]
```

display "1"

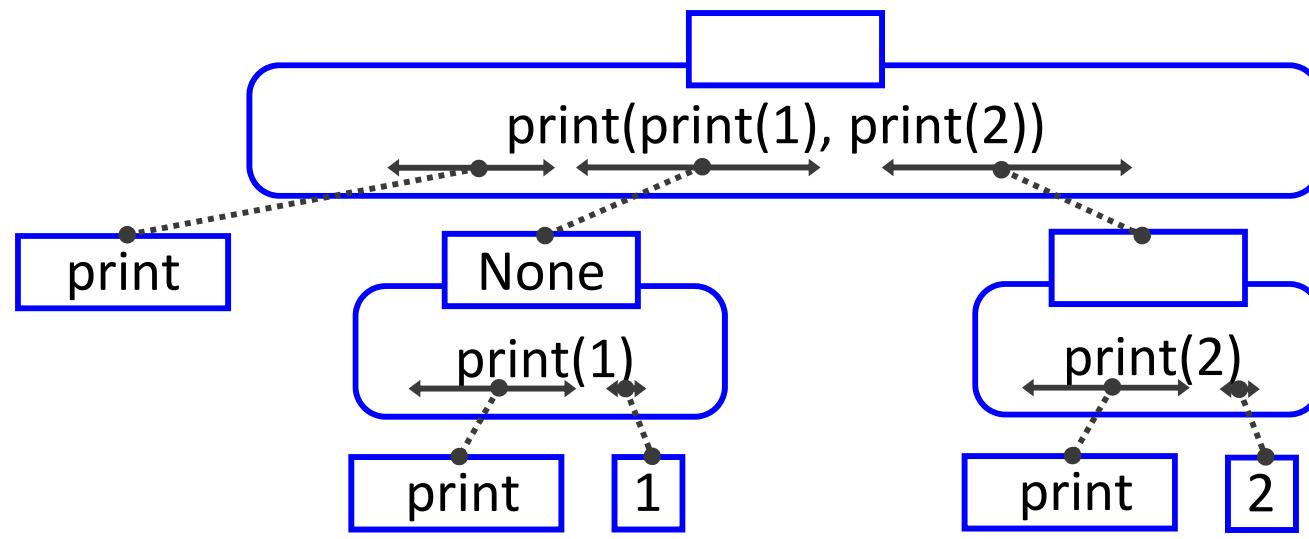
# Nested Print Expressions



None, None ➤ print(...):



```
>>> print(print(1), print(2))  
1  
2
```



1 ➤ print(...): ➤ None

display “1”

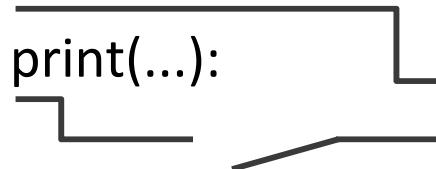
2 ➤ print(...): ➤ None

display “2”

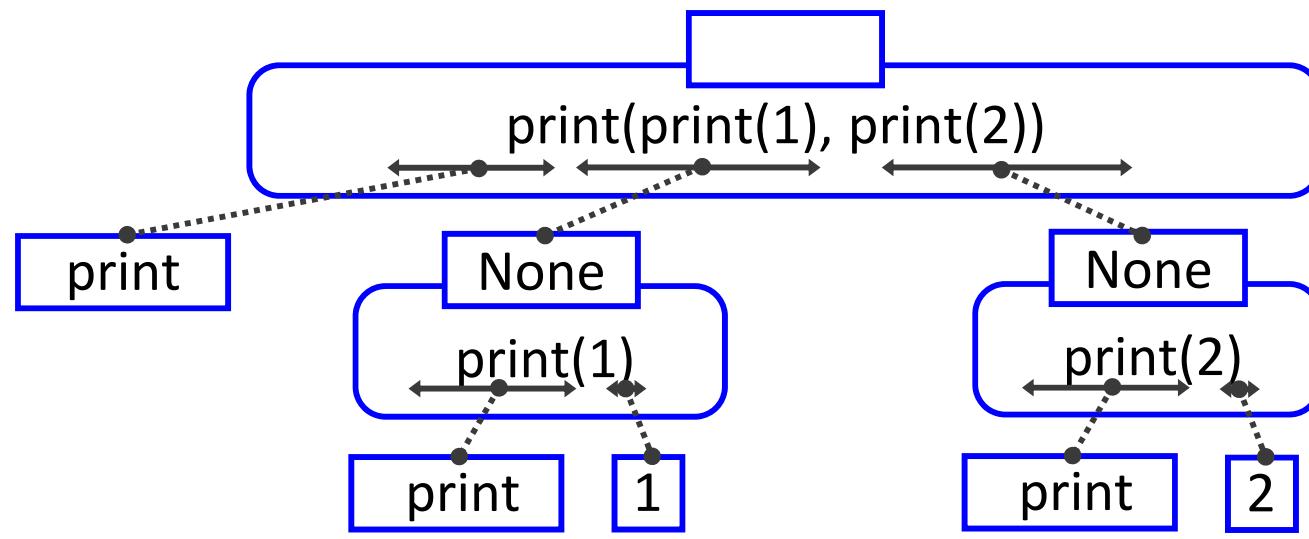
# Nested Print Expressions



None, None ➤ print(...):



```
>>> print(print(1), print(2))  
1  
2
```



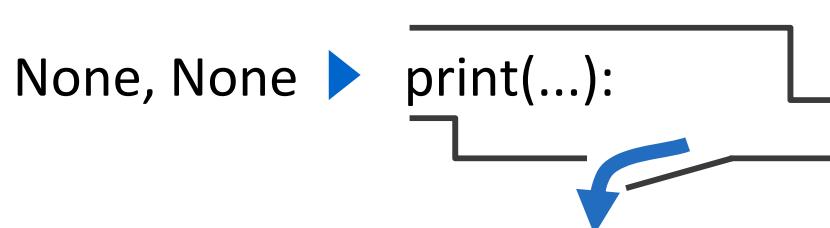
1 ➤ print(...): ➤ None

display “1”

2 ➤ print(...): ➤ None

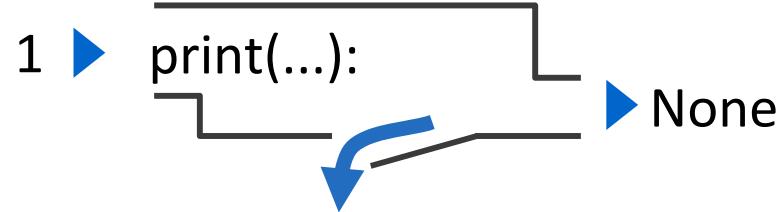
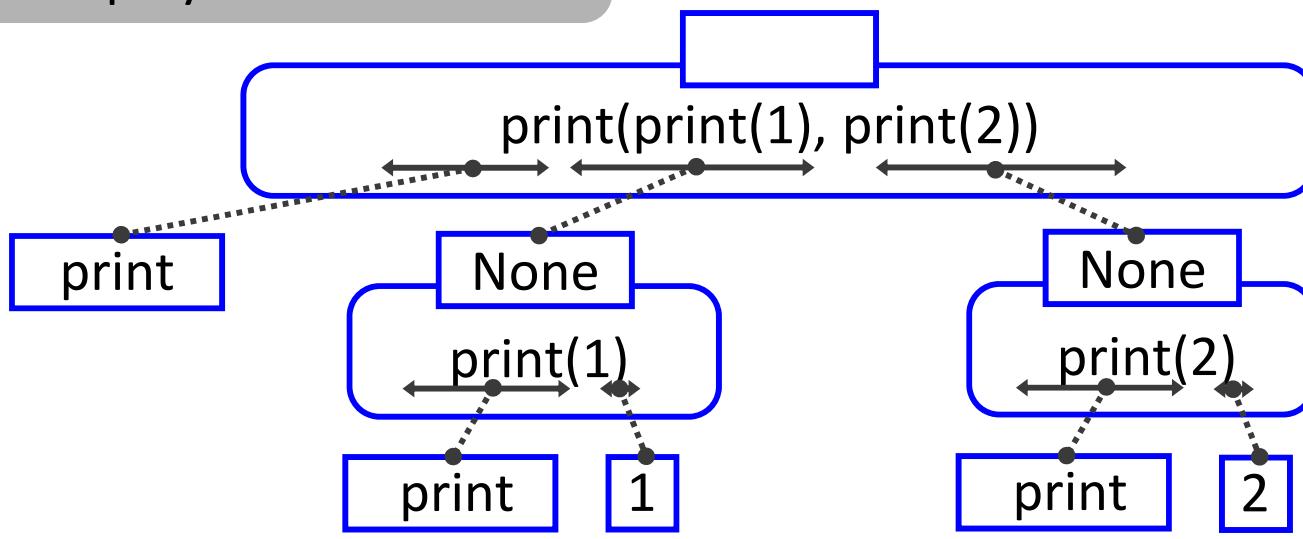
display “2”

# Nested Print Expressions

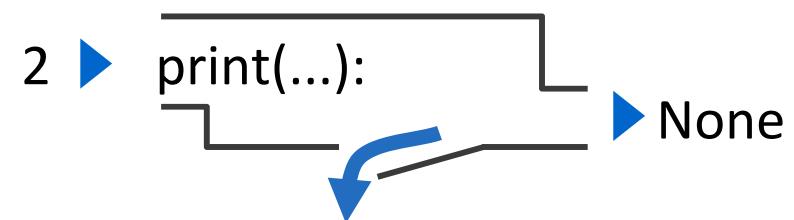


display "None None"

```
>>> print(print(1), print(2))  
1  
2  
None None
```

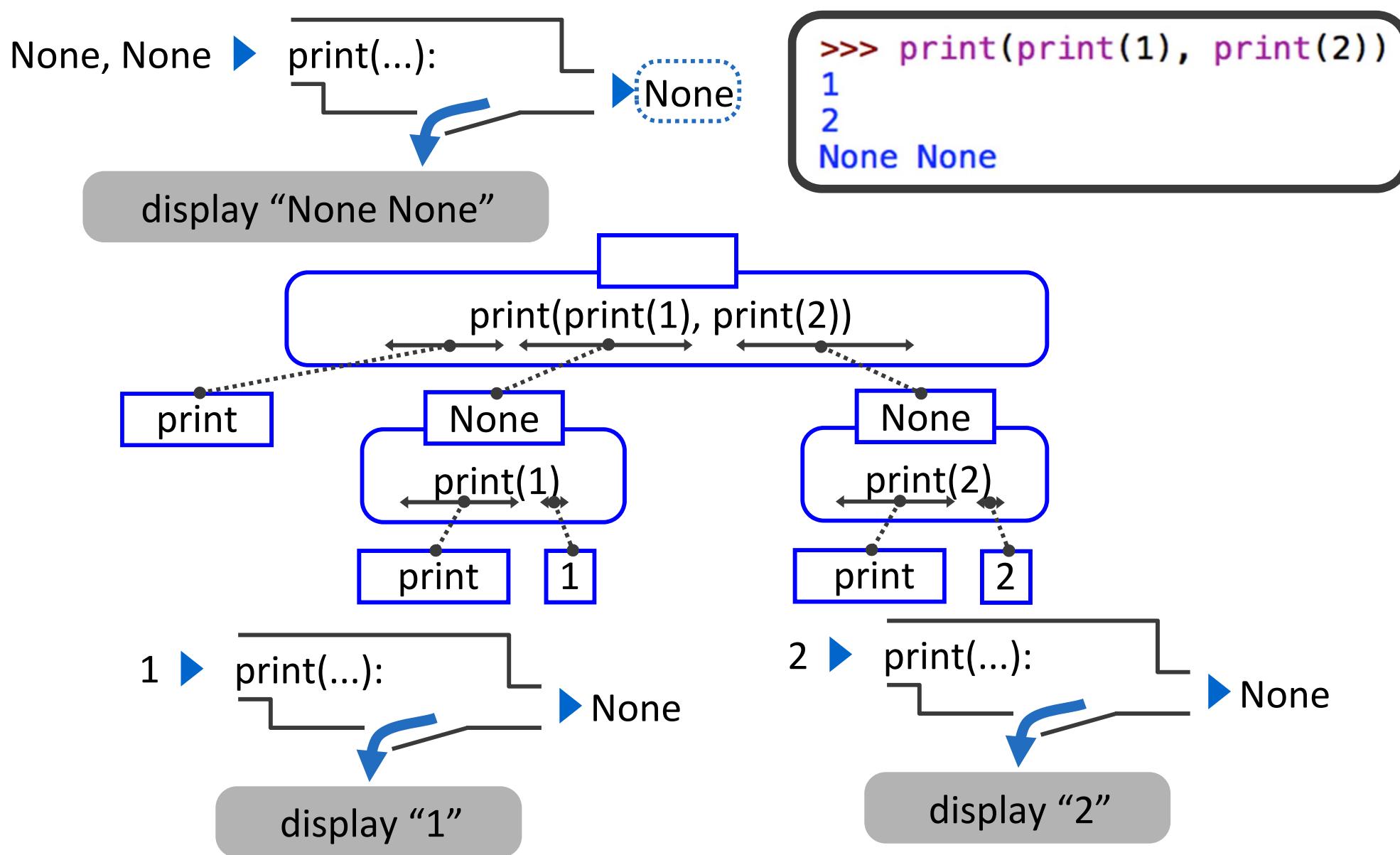


display "1"

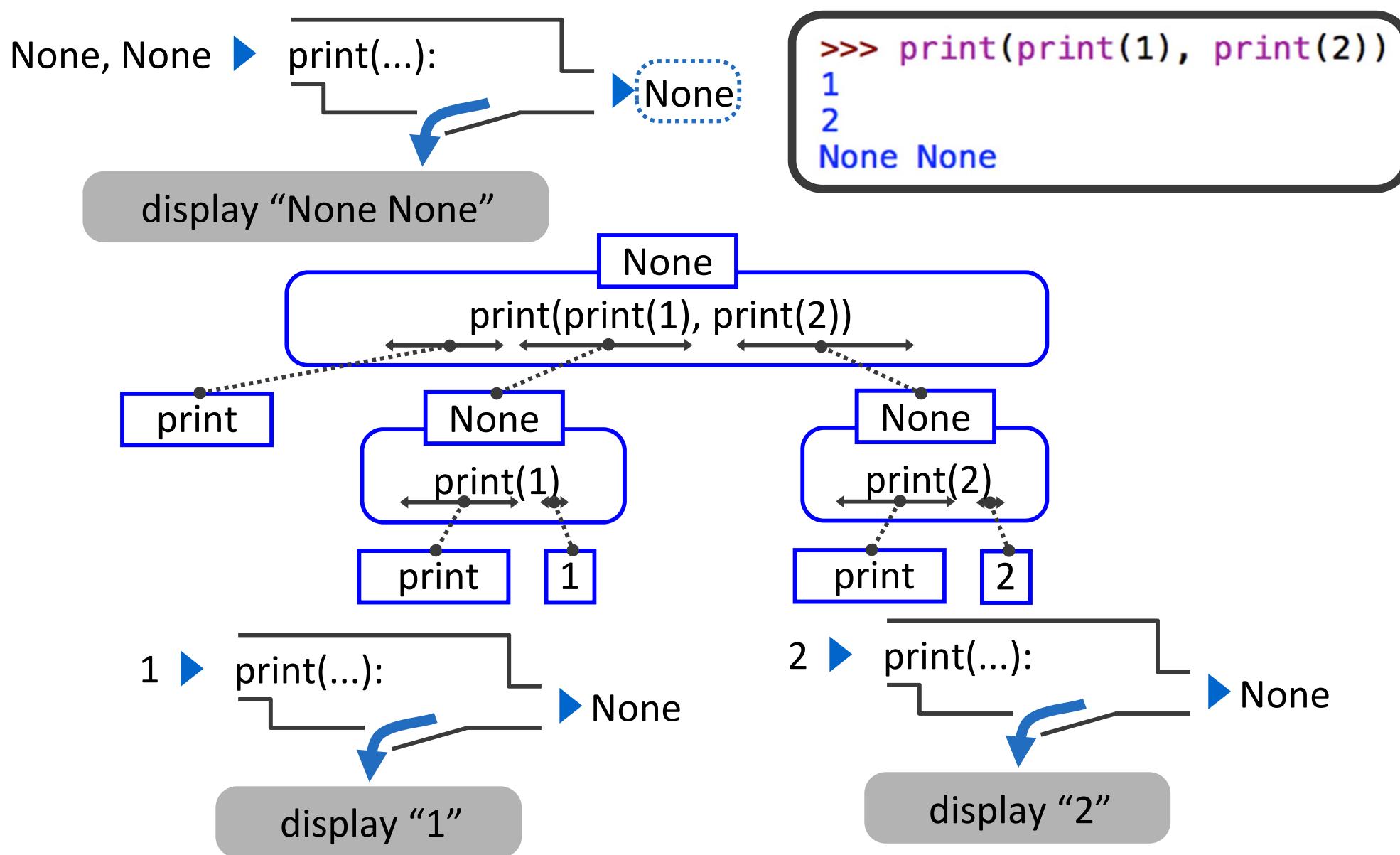


display "2"

# Nested Print Expressions



# Nested Print Expressions



# The Elements of Programming



# The Elements of Programming



- Primitive Expressions and Statements
  - The simplest building blocks of a language

# The Elements of Programming



- Primitive Expressions and Statements
  - The simplest building blocks of a language
- Means of Combination
  - Compound elements built from simpler ones

# The Elements of Programming



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