Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 6 – 17, 2020

Python for Data Analytics

Functions

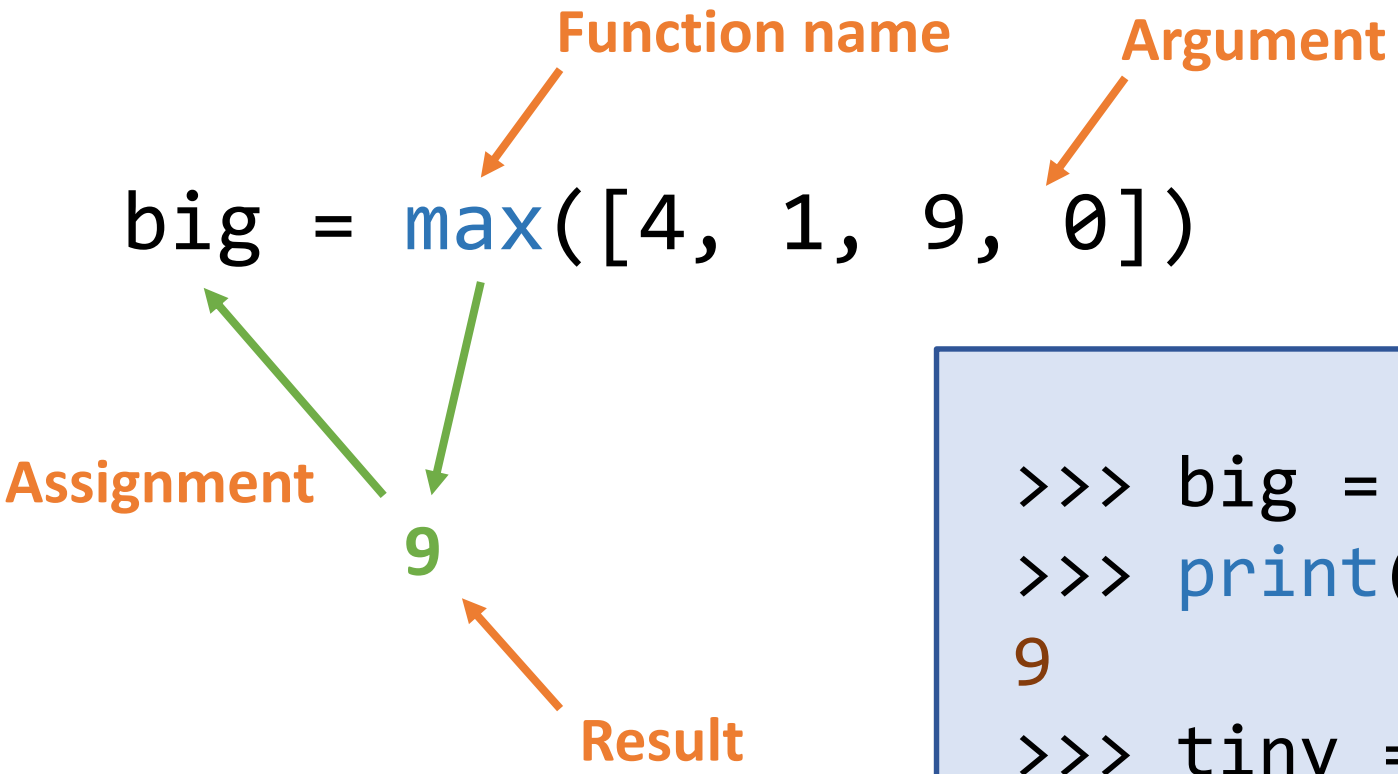
Python Functions

- A function is some reusable code that takes argument(s) as input, does some computation, and then returns a result or results.

- Built-in functions
  - Provided as part of Python
  - `print()`, `input()`, `type()`, `float()`, `int()`, `max()`, ...

- User-defined functions
  - Functions that we define ourselves and then use
  - A function can be defined using the `def` reserved word
  - A function is called (or invoked) by using the function name, parentheses, and arguments in an expression

Function Example

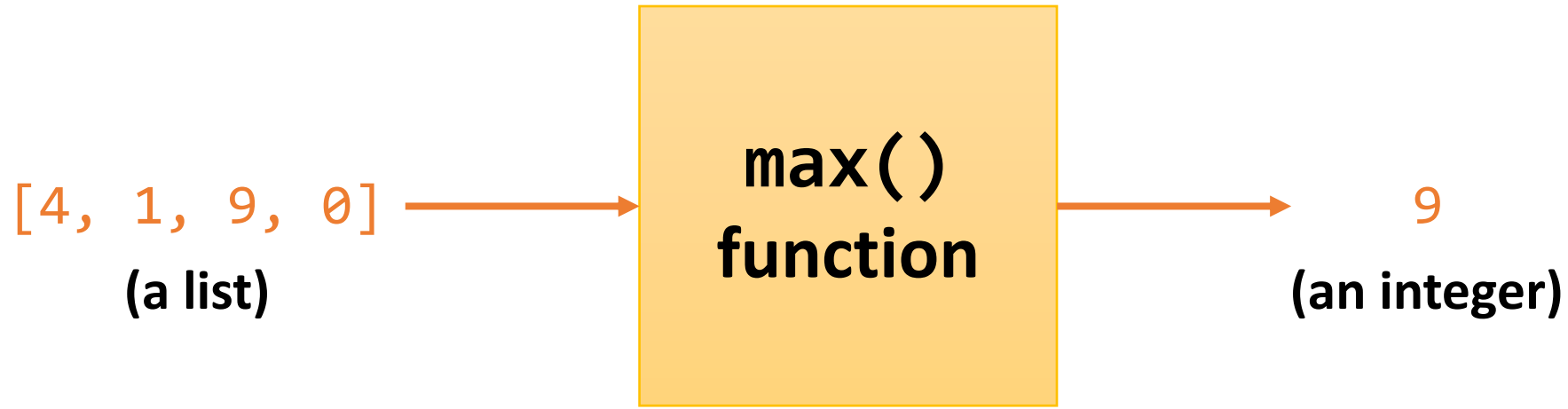



```
>>> big = max([4, 1, 9, 0])
>>> print(big)
9
>>> tiny = min([4, 1, 9, 0])
>>> print(tiny)
0
```

max()

- A function is some stored code that we use.
- A function takes some input and produces an output.

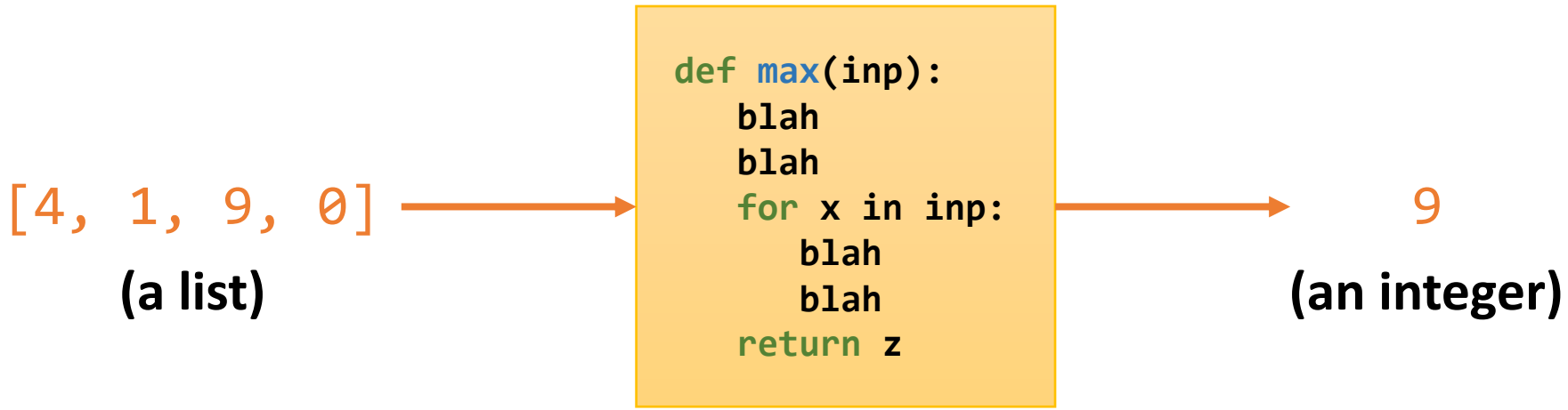
```
>>> big = max([4, 1, 9, 0])
>>> print(big)
9
```



max()

- A function is some stored code that we use.
- A function takes some input and produces an output.

```
>>> big = max([4, 1, 9, 0])
>>> print(big)
9
```



Building Our Own Functions

- We create a new function using the `def` keyword followed by optional parameters in parentheses
- We indent the body of the function
- This defines the function but does not execute the body of the function

```
def print_lyrics():
    print('I bless the day I found you')
    print('I want to stay around you')
```

Defining a Function

```
x = 5
print('Hello')

def print_lyrics():
    print('I bless the day I found you')
    print('I want to stay around you')

print('World')
x = x + 2
print(x)
```

```
Hello
World
7
```

Definitions and Uses

- Once we have defined a function, we call (or invoke) it as many times as we like

```
x = 5
print('Hello')

def print_lyrics():
    print('I bless the day I found you')
    print('I want to stay around you')

print('World')
print_lyrics()
x = x + 2
print(x)
```

```
Hello
World
I bless the day I found you
I want to stay around you
7
```

Arguments

- An argument is a value we pass into the function as its input when we call the function
- We use arguments so we can direct the function to do different kinds of work when we call it at different times
- We put the arguments in parentheses after the name of the function

```
big = max([4, 1, 9, 0])
```

Parameters

- A parameter is a variable which we use in the function definition

- It is a "handle" that allows the code in the function to access the arguments for a particular function invocation.

```
>>> def greet(lang):
...     if lang == 'kr':
...         print('안녕하세요')
...     elif lang == 'fr':
...         print('Bonjour')
...     elif lang == 'es':
...         print('Hola')
...     else:
...         print('Hello')

>>> greet('en')
Hello
>>> greet('es')
Hola
>>> greet('kr')
안녕하세요
>>>
```

Return Value

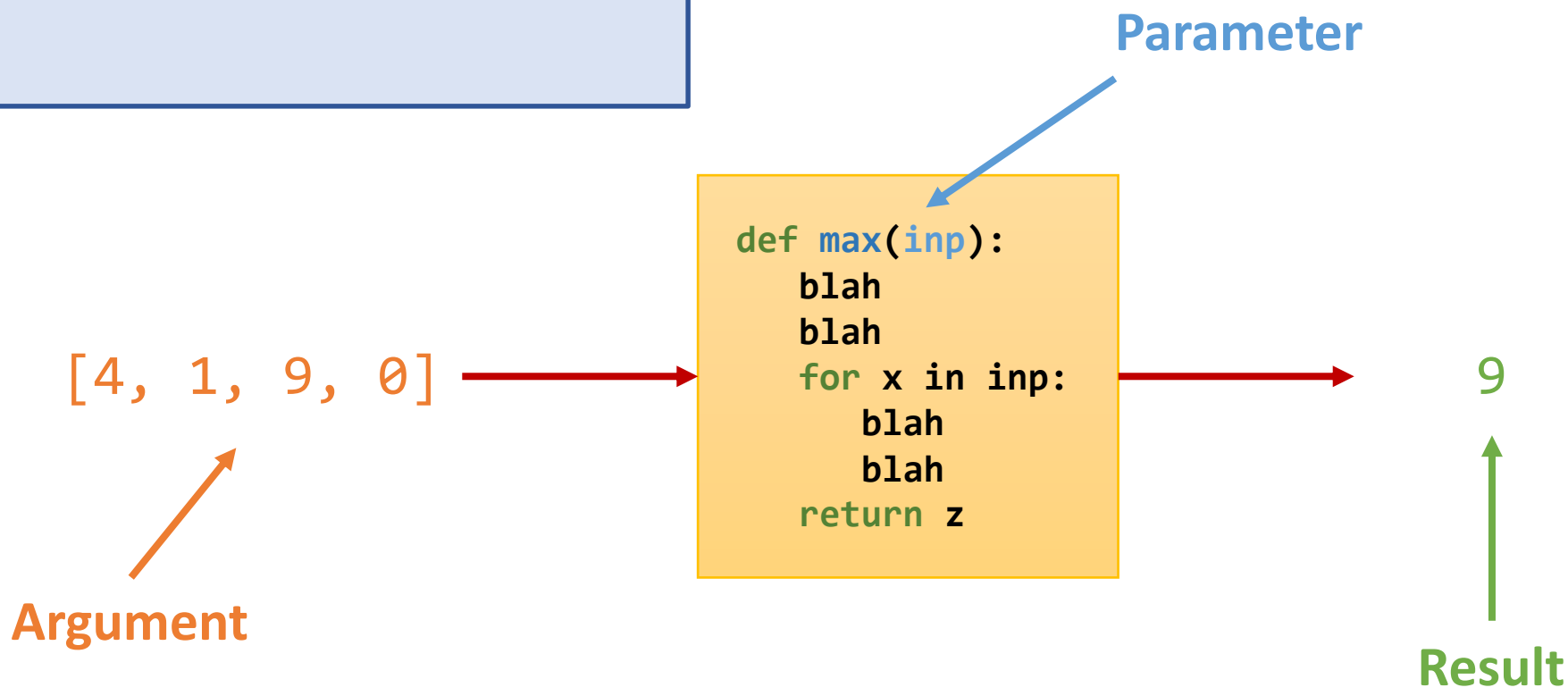
- A "fruitful" function is one that produces a result (or return value)

- The `return` statement ends the function execution and "sends back" the result of the function

```
>>> def greet(lang):
...     if lang == 'kr':
...         return '안녕하세요'
...     elif lang == 'fr':
...         return 'Bonjour'
...     elif lang == 'es':
...         return 'Hola'
...     else:
...         return 'Hello'

>>> print(greet('en'), 'Jack')
Hello Jack
>>> print(greet('es'), 'Sally')
Hola Sally
>>> print(greet('kr'), '홍길동')
안녕하세요 홍길동
>>>
```

Arguments, Parameters, and Results

```
>>> big = max([4, 1, 9, 0])
>>> print(big)
9
```



Multiple Parameters / Arguments

- We can define more than one parameter in the function definition
- We simply add more arguments when we call the function
- We match the number and order of arguments and parameters

```
def mymax(a, b):
    if a > b:
        return a
    else:
        return b

x = mymax(3, 5)
print(x)
5
```

Default and Keyword Arguments

- Default arguments
  - You can specify default values for arguments that aren't passed
- Keyword arguments
  - Callers can specify which argument in the function to receive a value by using the argument's name in the call

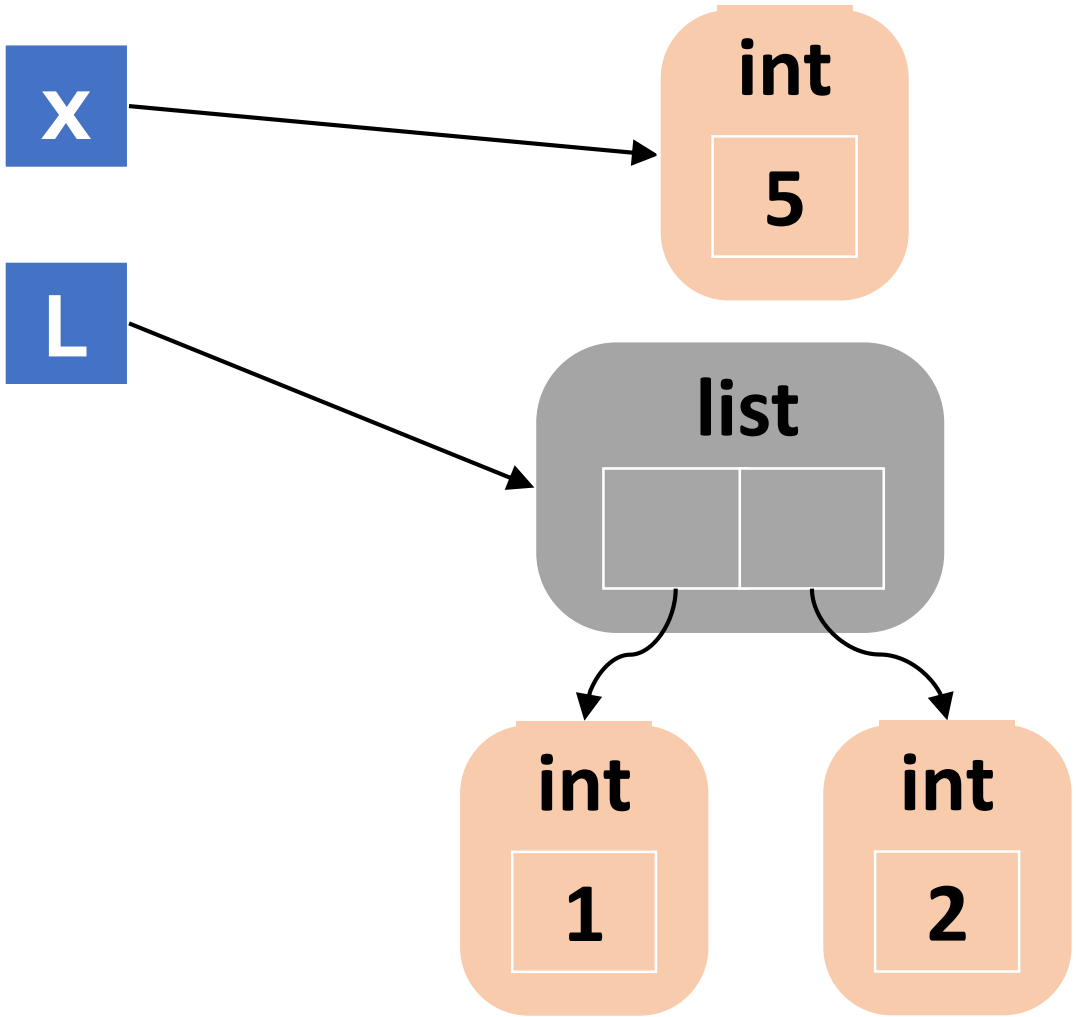
```
def student(name, id='00000', dept='CSE'):
    print(name, id, dept)

student('John')
student('John', '00001')
student(name='John')
student(id='20191', dept='EE', name='Jack')
```

Passing Arguments

```
>>> def f(a, b):
...     a = 9
...     b[0] = 'spam'

>>> x = 5
>>> L = [1, 2]
>>> f(x, L)
>>> print(x)
5
>>> print(L)
['spam', 2]
```

Passing Arguments

```
>>> def f(a, b):
...     a = 9
...     b[0] = 'spam'

>>> x = 5
>>> L = [1, 2]
```

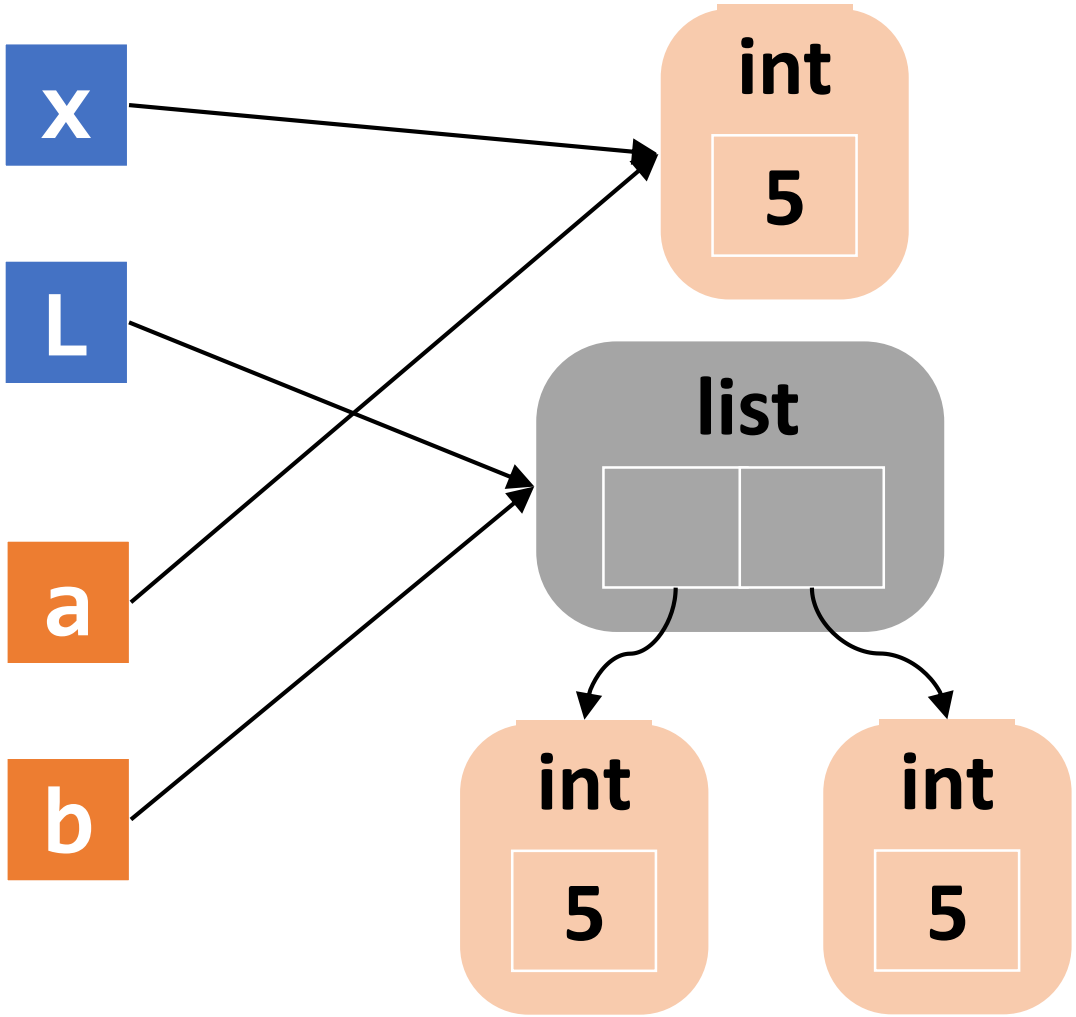


Passing Arguments

```
>>> def f(a, b):
...     a = 9
...     b[0] = 'spam'

>>> x = 5
>>> L = [1, 2]
>>> f(x, L)
```

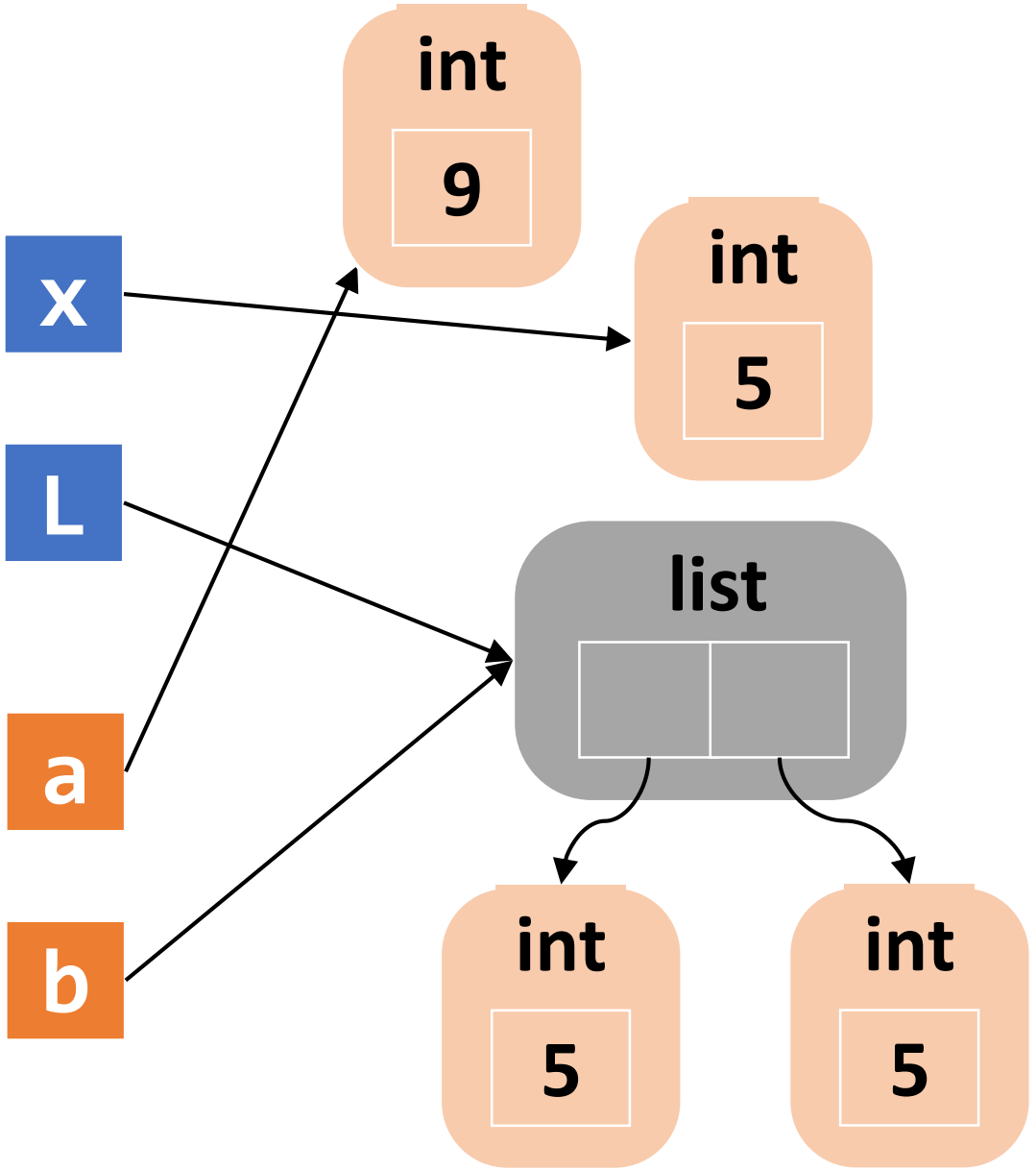
b = L

a = x



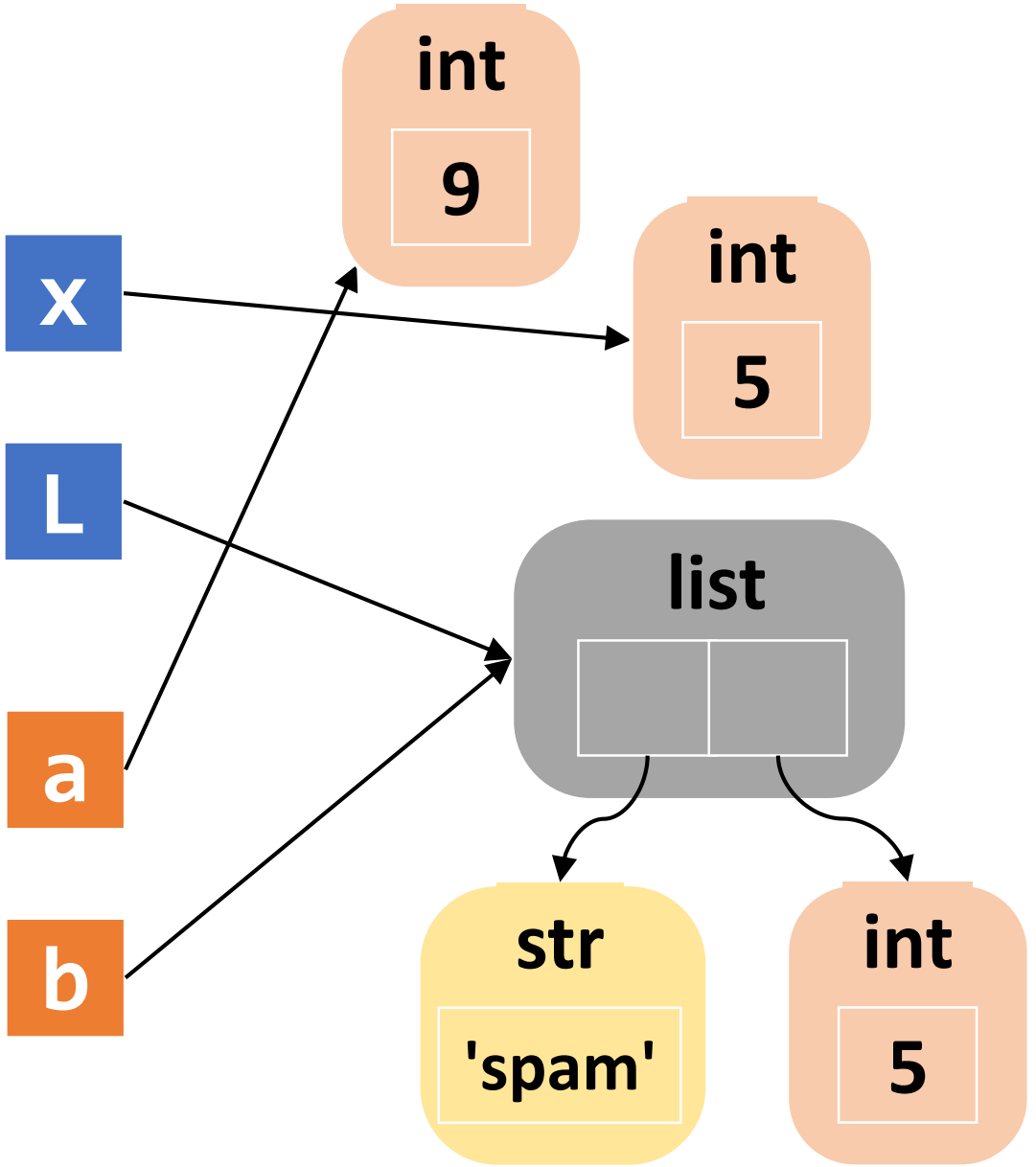
Passing Arguments

```
>>> def f(a, b):
...     a = 9
...     b[0] = 'spam'

>>> x = 5
>>> L = [1, 2]
>>> f(x, L)
```



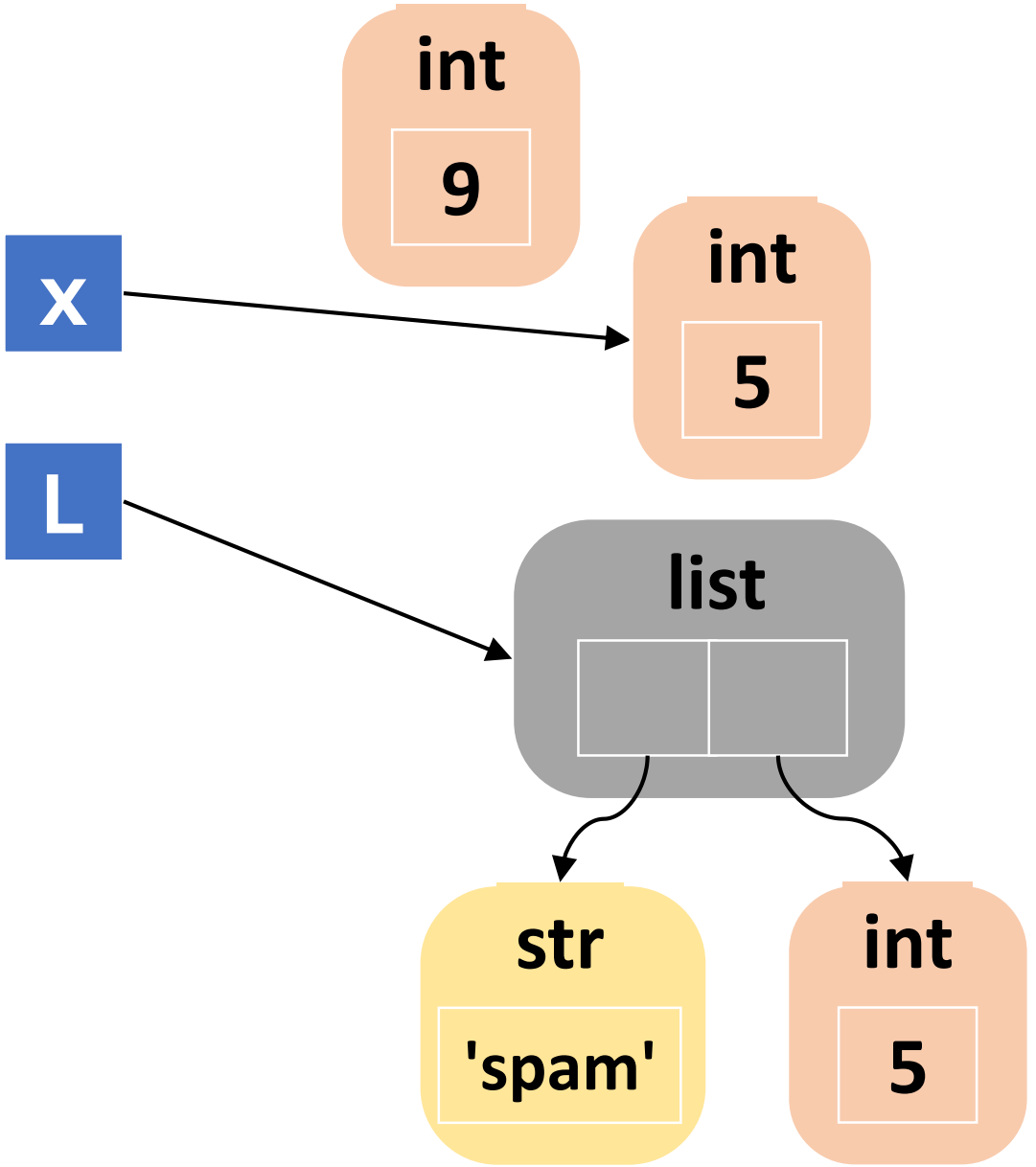
Passing Arguments

```
>>> def f(a, b):
...     a = 9
...     b[0] = 'spam'

>>> x = 5
>>> L = [1, 2]
>>> f(x, L)
```



Passing Arguments

```
>>> def f(a, b):
...     a = 9
...     b[0] = 'spam'

>>> x = 5
>>> L = [1, 2]
>>> f(x, L)
>>> print(x)
5
>>> print(L)
['spam', 2]
```



Local vs. Global Variables

- Local variables
  - If a variable is assigned a value anywhere within the function's body, it is assumed to be a local
  - Visible only to code inside the function def and exists only while the function runs

- Global variables
  - Variables defined outside a function
  - Variables that are only referenced inside a function are implicitly global
  - Use global keyword to use a global variable inside a function
  - There is no need to use the global keyword outside a function

Local vs. Global Variables: Examples (1)

```
def f():
    print(s)

s = 'Spam is delicious'
f()
```

```
Spam is delicious
```

```
def f():
    s = 'Egg is better'
    print(s)

s = 'Spam is delicious'
f()
print(s)
```

```
Egg is better
Spam is delicious
```

Local vs. Global Variables: Examples (2)

```
def f():
    print(s)
    s = 'Egg! Egg!'
    print(s)

s = 'Spam is delicious'
f()
print(s)
```

```
Traceback (most recent call last):
  File "local.py", line 7, in <module>
    f()
  File "local.py", line 2, in f
    print(s)
UnboundLocalError: local variable 's' referenced
before assignment
```

```
def f():
    global s
    print(s)
    s = 'Egg! Egg!'
    print(s)

s = 'Spam is delicious'
f()
print(s)
```

```
Spam is delicious
Egg! Egg!
Egg! Egg!
```

Recursive Function

- Functions that call themselves either directly or indirectly

$$f(n) = \begin{cases} f(n-1) + f(n-2) & n \geq 2 \\ 1 & n = 0, n = 1 \end{cases}$$

```
def fib(n):
    if (n < 2):
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Why Functions?

- Make the program modular and readable
- Can be reused later
- You can even package them as a library (or a module)