Lecture 3a - Functional Abstraction

LT3a Objectives

- Understand the idea of a function in programming

General form of a function definition

```
def <name>(<formal parameters>):
    <body>
```

- name
  - The symbol to be associated with the function
- formal parameters
  - The names used in the body to refer to the actual arguments (e.g. values) of the function
- body
  - The expression to be evaluated to yield the result of the function application
  - Has to be indented (standard is 4 spaces)
  - <u>Can</u> return values as output

define

name

Parameter (s)

```
def square(x):
    result = x * x
    return result
```

indentation

indentation

body

The expression compute the result (may or may not involve the parameter)

```
square(2)
```

4

```
square(2 + 5)
```

49

```
square(square(3))
```

81

Should we use ‘print’ or ‘return’ ?

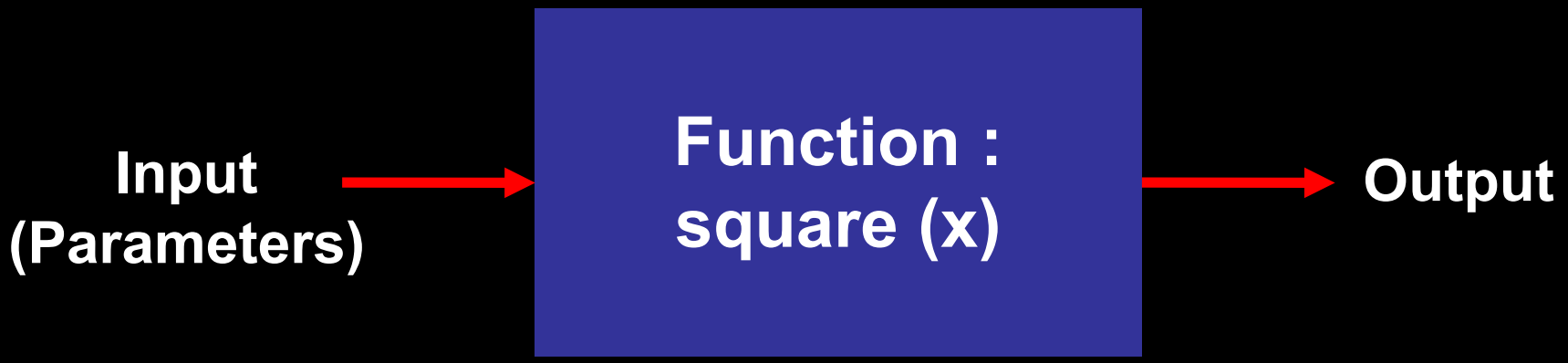
```
def square(x):
        result = x * x
        print(result)

def square(x):
        result = x * x
        return result
```

Try running this :

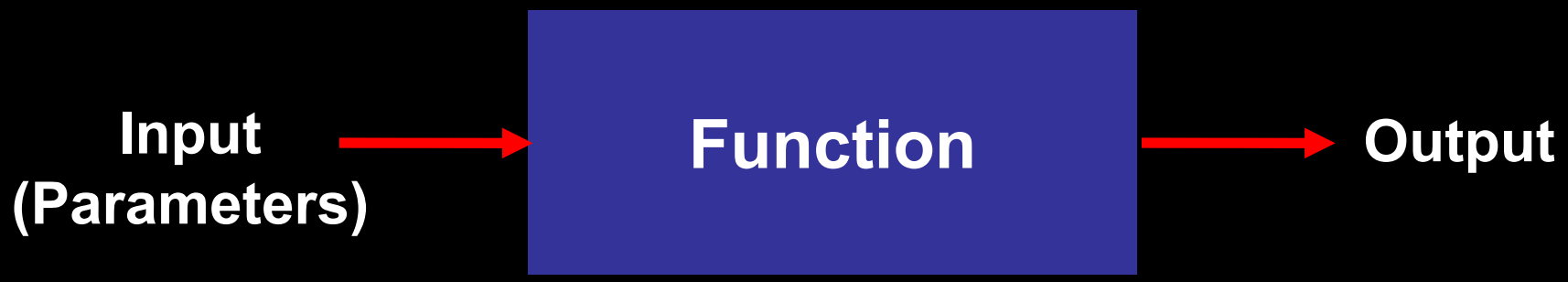
```
>>> square(square(2))
```

Black Box



x		Output
2	>>> square(2)	4
3	>>> square(3)	9
4	>>> square(4)	16
5	>>> square(5)	???

Black Box



No need to know HOW it does the job!

Just need to know WHAT it does. ☺

Different Implementations

```
def square(x):
        result = < expression >
        return result
```

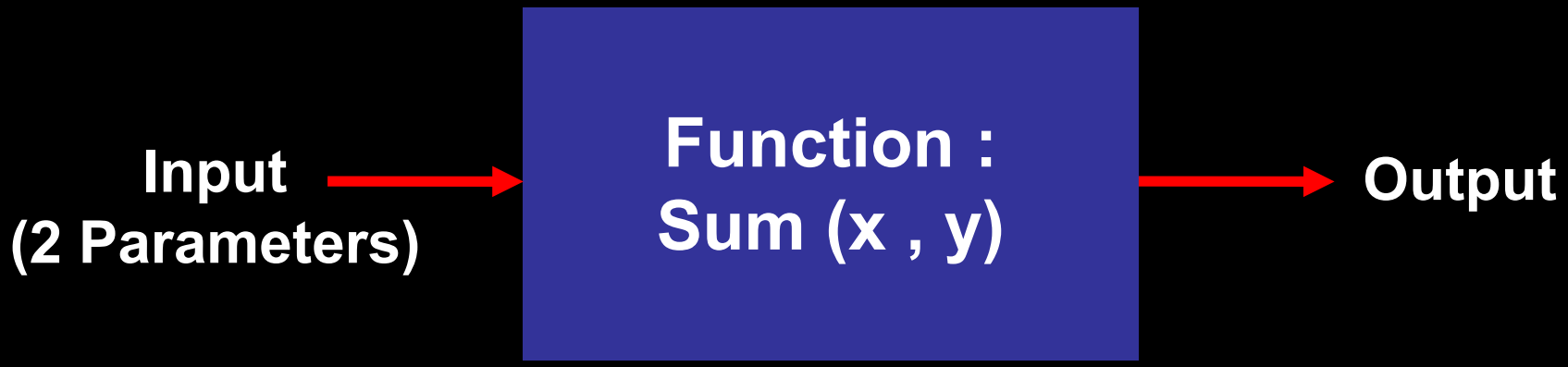
Possible < expression > :

```
  result = x * x

  result = x ** 2

  result = e ** ( 2 * ln (x))
```

Black Box



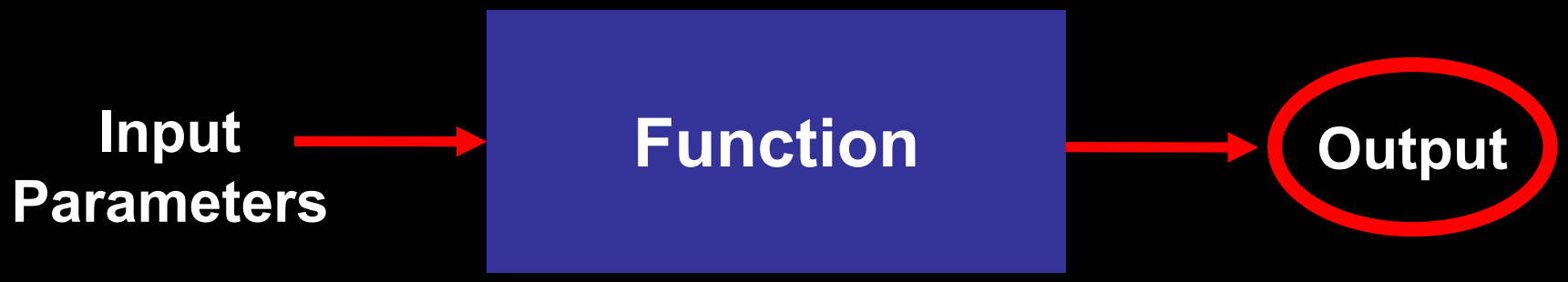
x	y	Output
1	2	3
2	3	5
3	4	7
5	6	???

Black Box

Input (Parameters) → **Function : is_leap_yr (year)** → **Output (Boolean type)**

year	Output
2000	True
2100	False
2104	True
2018	???

The Output



The Output is returned with **`return`**.
The Output type could be **`None`**.

The Output is a NoneType.

```
def square(x):
    result = x * x
    print(result)
    return None     # Python
                      will
                      assume
                      this!
```

About ‘Parameter(s)’

```
def square(x):
    result = x * x
    return result

# Python will forget ‘result’
after the function returns!

>>> print(result)
```

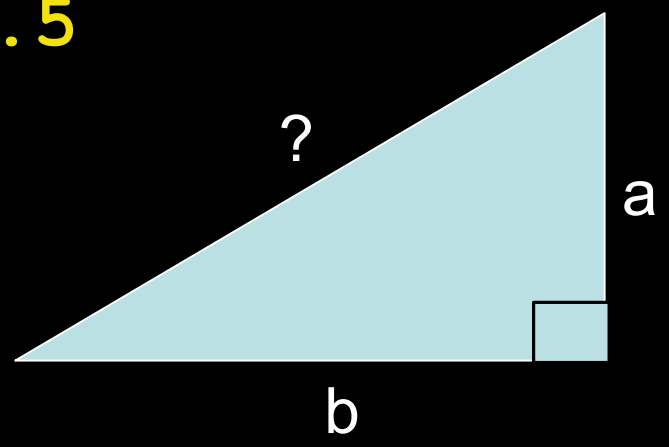
result is not defined

Local and Global Scope

Suppose you want to write a function to compute hypothenuse

This is a lousy way to code !!!

```
def hypotenuse(a, b):
    return (a**2 + b**2)**0.5
```



Wishful Thinking

Top-down approach:
Pretend you have whatever you need!

```
def hypothenuse (a, b):
  result = sqrt (square(a) + square(b))
  return result
```

What is “square”?
Pretend we know!

What is “sqrt”?
Pretend we know!

```
def square (x):
    result = x ** 2
    return result
```

What is “square”?
Now we know!

```
def sqrt (y):
    result = y ** 0.5
    return result
```

What is “sqrt”?
Now we know!

```
def square (x):
   result = x ** 2
   return result

def sqrt (y):
   result = y ** 0.5
   return result


def hypothenuse (a, b):
  result = sqrt (square(a) + square(b))
  return result
```

This is a better way to code !!!

```
def hypothenuse (a, b):

    def square (x):
        result = x ** 2
        return result

    def sqrt (y):
        result = y ** 0.5
        return result

    result = sqrt (square(a) + square(b))
    return result
```

Another way of writing

```
def hypothenuse (a, b):

    def square (x):
        result = x ** 2
        return result

    def sqrt (y):
        result = y ** 0.5
        return result

    result = sqrt (square(a) + square(b))
    return result
```

Local Scope 1

Local Scope 2

Global Scope

Local and Global Scope

1. Code in the global scope cannot use any local variables.
2. A local scope can access global variables.
3. Code in a Function’s local scope cannot use variables in any other local scope.
4. You can use the same name for different variables if they are in different scopes.

Local and Global Scope

1. Code in the global scope cannot use any local variables.

a and b are Global variables

```
def hypothenuse (a, b):
```

```
def square (x):
    result = x ** 2
    return result
```

Local Scope 1

```
result = x + a + b
return result
```

x is not defined

Local and Global Scope

2. A local scope can access global variables.

```
def hypothenuse (a, b):
   k = 5

   def square (x):
        result = x + k
        return result

   result = square(a)
   return result
```

k is a Global variable

Local Scope 1

Local and Global Scope

3. Code in a Function's local scope cannot use variables in any other local scope.

```
def hypothenuse (a, b):
    def square (x):
        result = x + y
        return result
    def sqrt (y):
        result = y ** 0.5
        return result

    result = square(a)
    return result
```

Local Scope 1

Local Scope 2

y is not defined

Local and Global Scope

4. You can use the same name for different variables if they are in different scopes.

```
def hypothenuse (a, b):
    def square (x):
        result = x ** 2
        return result
    def sqrt (x):
        result = x ** 0.5
        return result

    result = sqrt (square(a) + square(b))
    return result
```

Local Scope 1

Local Scope 2