

# [301] Using Functions

Tyler Caraza-Harter

# Learning Objectives Today

## How to call functions

- input/output

## Modules:

- import styles
- attribute operator (the ".")
- math module

## Inspection:

- discover functions in a module
- learn what function does

**Please read Chapter 3  
of Think Python**

# Learning Objectives Today

## How to call functions

- input/output

## Modules:

- import styles
- attribute operator (the ".")
- math module

## Inspection:

- discover functions in a module
- learn what function does

**Please read Chapter 3  
of Think Python**

**make a battleship game!**

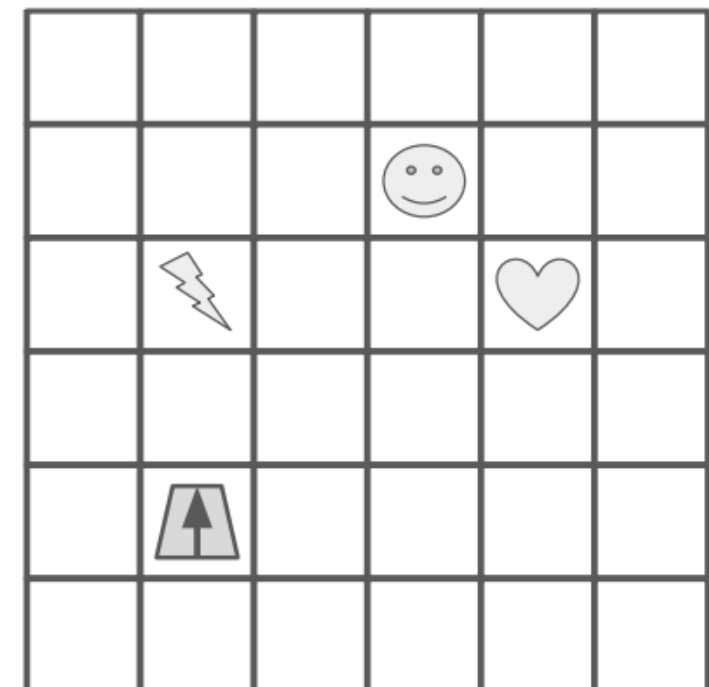
### Main Code:

1. Put 2 in the “moves” box
2. Perform the steps under “Move Code”, then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the “moves” box
5. Perform the steps under “Move Code”, then continue to step 6
6. Whatever symbol the robot is sitting on, write that symbol in the “resut” box

### Move Code:

- A. If “moves” is 0, stop performing these steps in “Move Code”, and go back to where you last were in “Main Code” to complete more steps
- B. Move the robot forward one square, in the direction the arrow is pointing
- C. Decrease the value in “moves” by one
- D. Go back to step A

**Functions are like “mini programs”,  
as in our robot worksheet problem**



### Main Code:

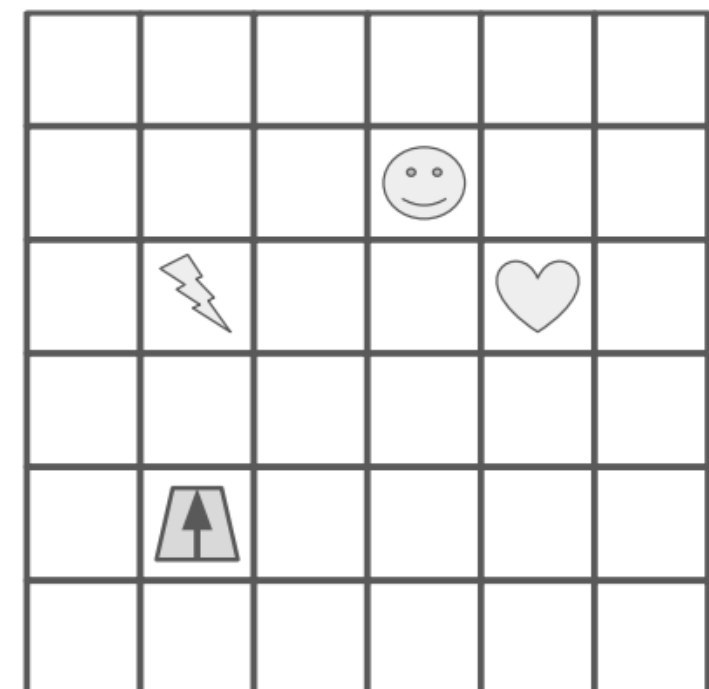
1. Put 2 in the “moves” box
2. Perform the steps under “Move Code”, then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the “moves” box
5. Perform the steps under “Move Code”, then continue to step 6
6. Whatever symbol the robot is sitting on, write that symbol in the “result” box

### Move Code:

- A. If “moves” is 0, stop performing these steps in “Move Code”, and go back to where you last were in “Main Code” to complete more steps
- B. Move the robot forward one square, in the direction the arrow is pointing
- C. Decrease the value in “moves” by one
- D. Go back to step A

*“Move Code” is a function*

**Functions are like “mini programs”,  
as in our robot worksheet problem**



### Main Code:

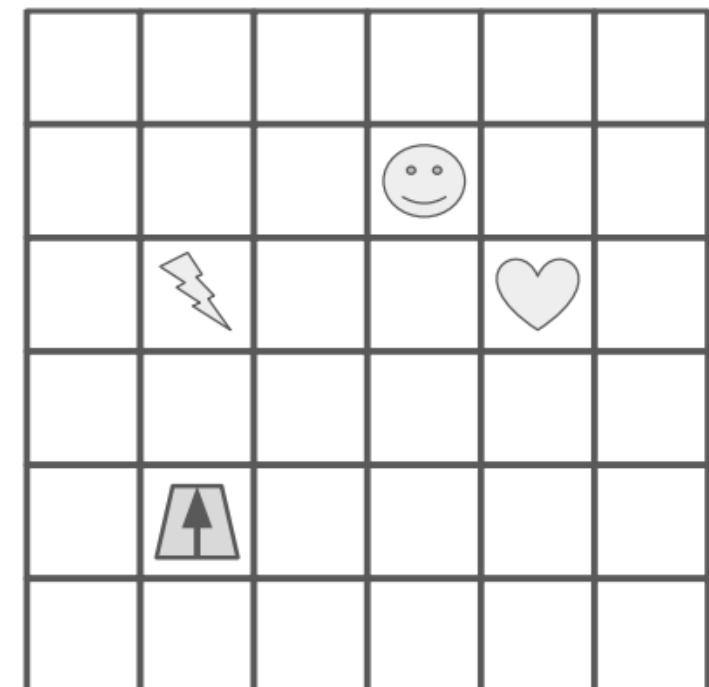
1. Put 2 in the "moves" box
2. Perform the steps under "Move Code", then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the "moves" box
5. Perform the steps under "Move Code", then continue to step 6
6. Whatever symbol the robot is sitting on, write that symbol in the "resut" box

*today we'll learn how to  
use functions in Python*

### Move Code:

- A. If "moves" is 0, stop performing these steps in "Move Code", and go back to where you last were in "Main Code" to complete more steps
- B. Move the robot forward one square, in the direction the arrow is pointing
- C. Decrease the value in "moves" by one
- D. Go back to step A

**Functions are like "mini programs",  
as in our robot worksheet problem**





*we'll learn about how to give functions  
input with "arguments" like "moves"*

### Main Code:

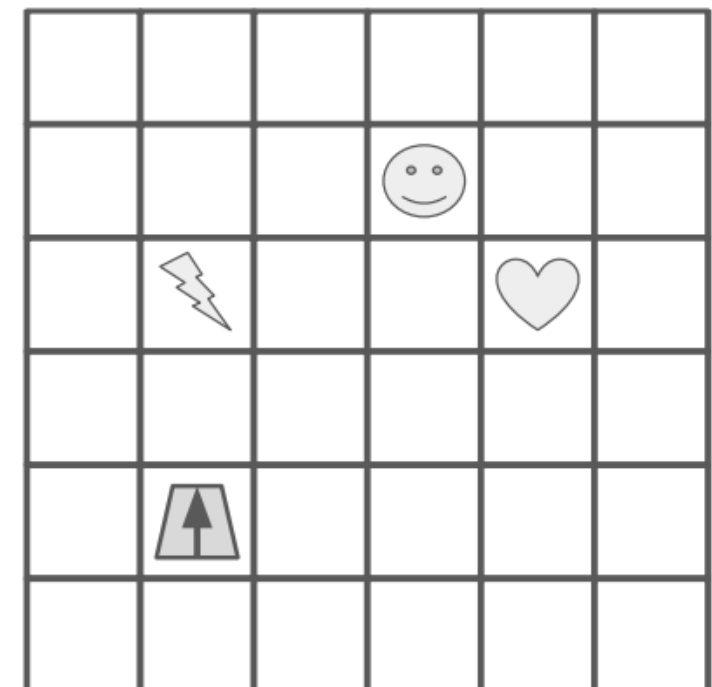
1. Put 2 in the "moves" box
2. Perform the steps under "Move Code", then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the "moves" box
5. Perform the steps under "Move Code", then continue to step 6
6. Whatever symbol the robot is sitting on, write that symbol in the "resut" box

*today we'll learn how to  
use functions in Python*

### Move Code:

- A. If "moves" is 0, stop performing these steps in "Move Code", and go back to where you last were in "Main Code" to complete more steps
- B. Move the robot forward one square, in the direction the arrow is pointing
- C. Decrease the value in "moves" by one
- D. Go back to step A

**Functions are like "mini programs",  
as in our robot worksheet problem**



*we'll learn about how to give functions  
input with "arguments" like "moves"*

### Main Code:

1. Put 2 in the "moves" box
2. Perform the steps under "Move Code", then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the "moves" box
5. Perform the steps under "Move Code", then continue to step 6
6. Whatever symbol the robot is sitting on, write that symbol in the "resut" box

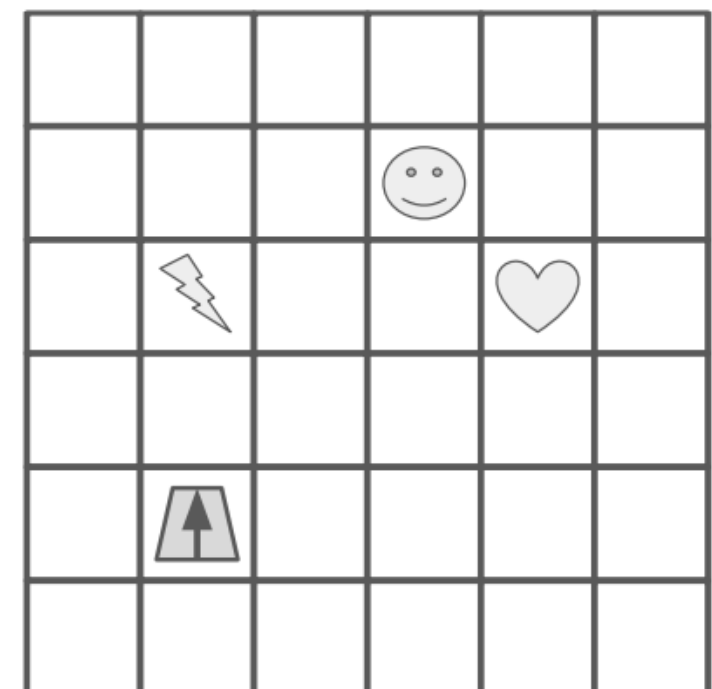
*today we'll learn how to  
use functions in Python*

*we'll also learn how to ask functions  
questions and get answers called return values*

### Move Code:

- A. If "moves" is 0, stop performing these steps in "Move Code", and go back to where you last were in "Main Code" to complete more steps
- B. Move the robot forward one square, in the direction the arrow is pointing
- C. Decrease the value in "moves" by one
- D. Go back to step A

**Functions are like "mini programs",  
as in our robot worksheet problem**





*we'll learn about how to give functions  
input with "arguments" like "moves"*

### Main Code:

1. Put 2 in the "moves" box
2. Perform the steps under "Move Code", then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the "moves" box
5. Perform the steps under "Move Code", then continue to step 6
6. Whatever symbol the robot is sitting on, write that symbol in the "resut" box

*today we'll learn how to  
use functions in Python*

*we'll also learn how to ask functions*

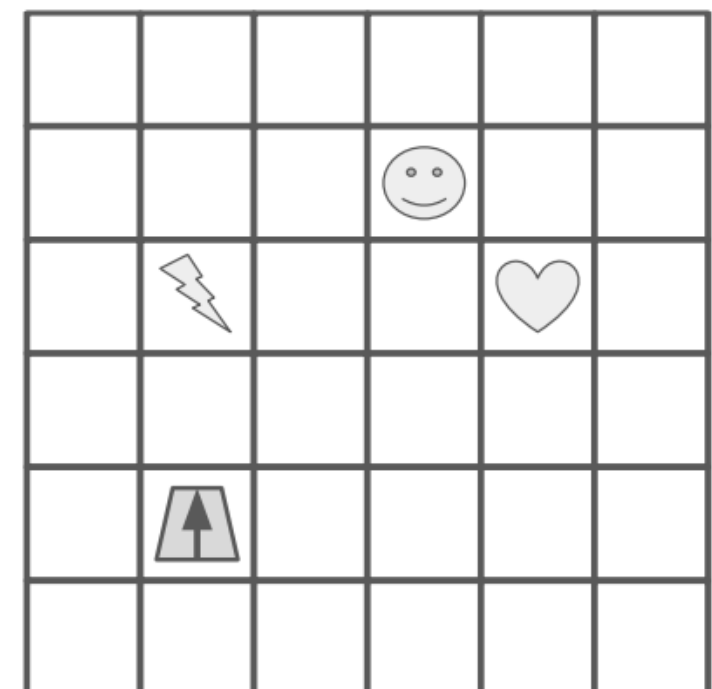
*questions and get answers called return values*

### Move Code:

- A. If "moves" is 0, stop performing these steps in "Move Code", and go back to where you last were in "Main Code" to complete more steps
- B. Move the robot forward one square, in the direction the arrow is pointing
- C. Decrease the value in "moves" by one
- D. Go back to step A

*next lecture, we'll learn how to  
write our own new functions*

**Functions are like "mini programs",  
as in our robot worksheet problem**



# Vocabulary

- ...

## General Function Concepts

Some Code

...

~~~~~

...

~~~~~

...

# Vocabulary

- **refactor**: change organization of code (e.g., to avoid repetition)

## A Function

~~~~~

## Some Code

...

call/invoke

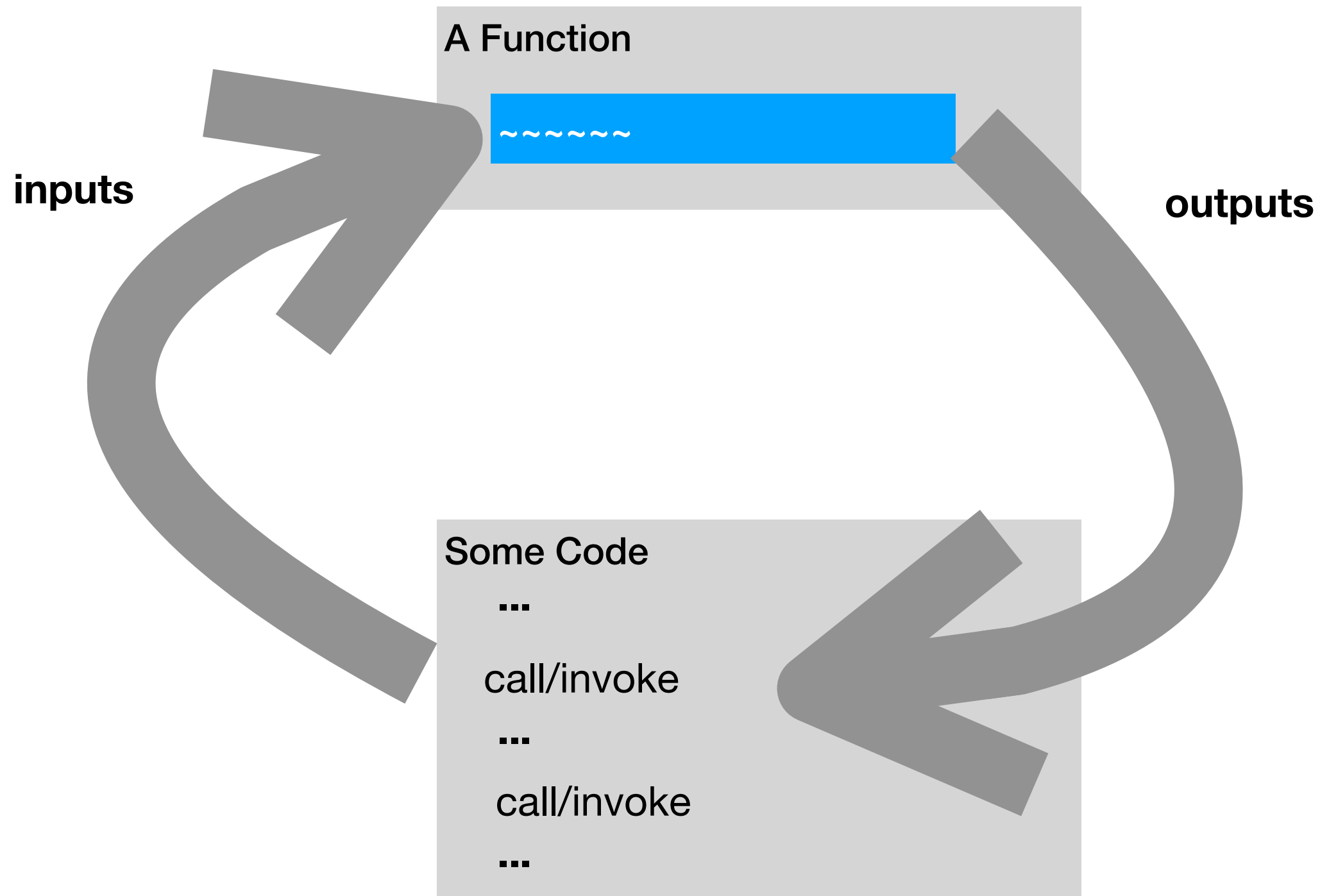
...

call/invoke

...

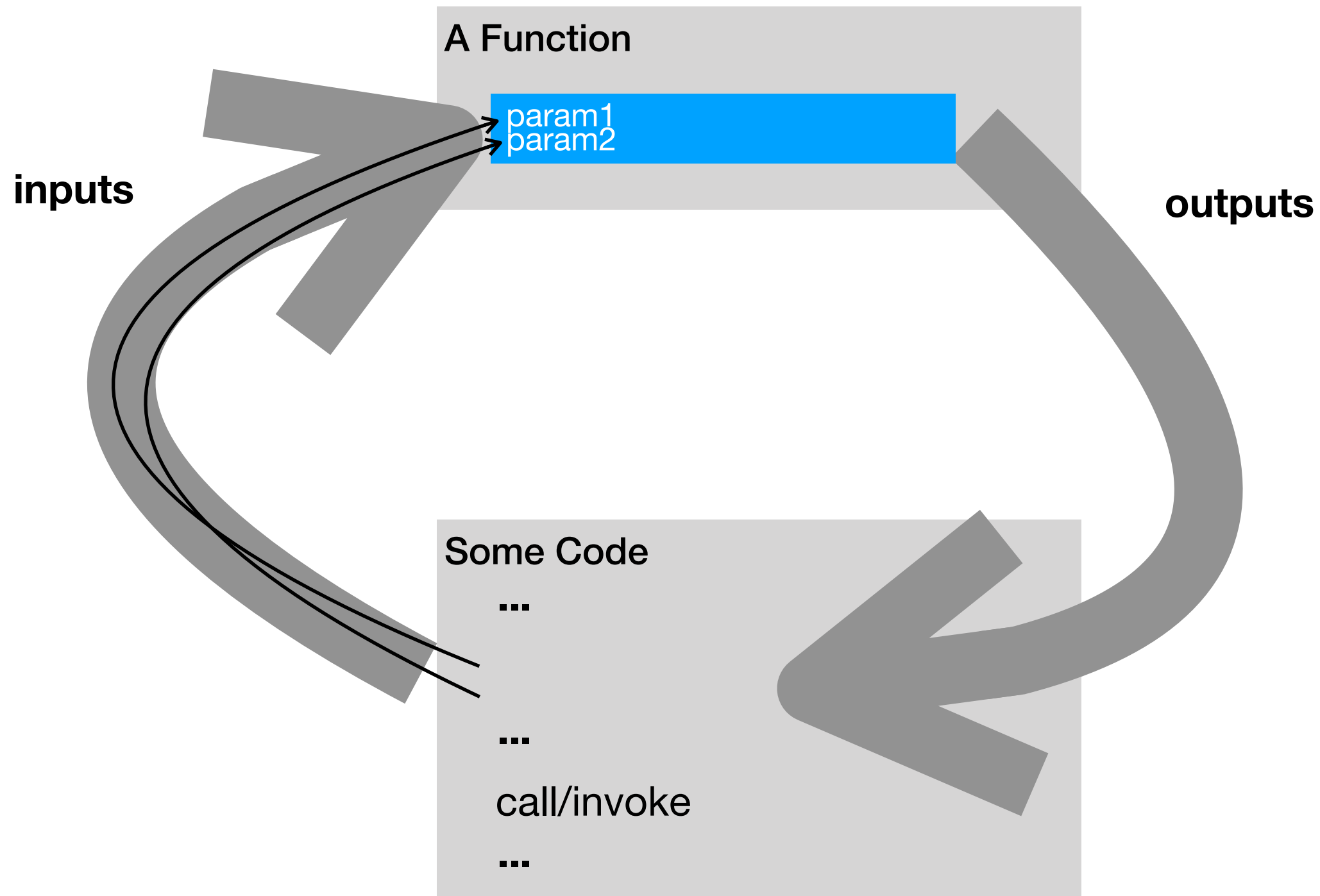
# Vocabulary

- **refactor**: change organization of code (e.g., to avoid repetition)



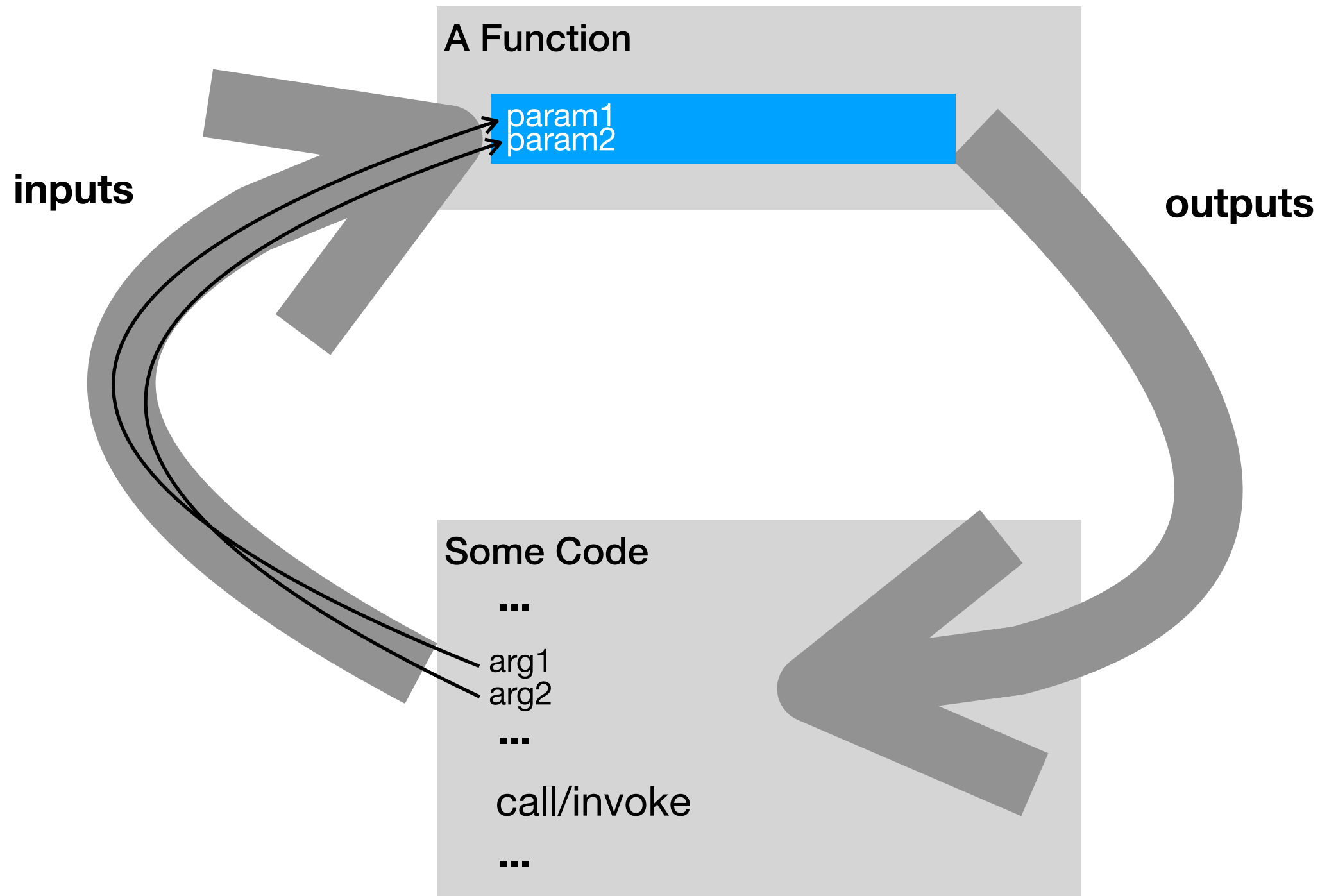
# Vocabulary

- **refactor**: change organization of code (e.g., to avoid repetition)
- **parameter**: variable that receives input to function



# Vocabulary

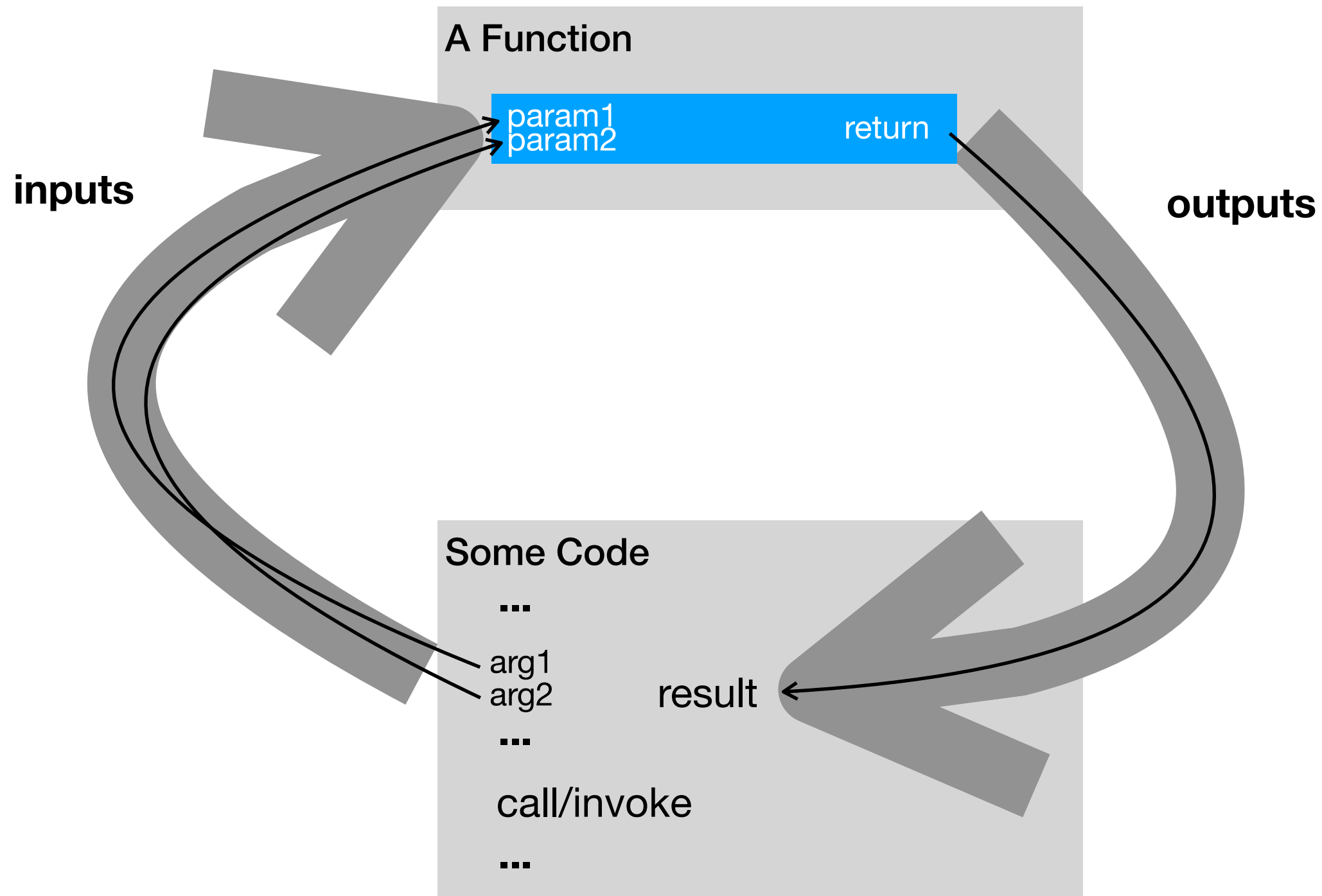
- **refactor**: change organization of code (e.g., to avoid repetition)
- **parameter**: variable that receives input to function
- **argument**: value sent to a function (lines up with parameter)





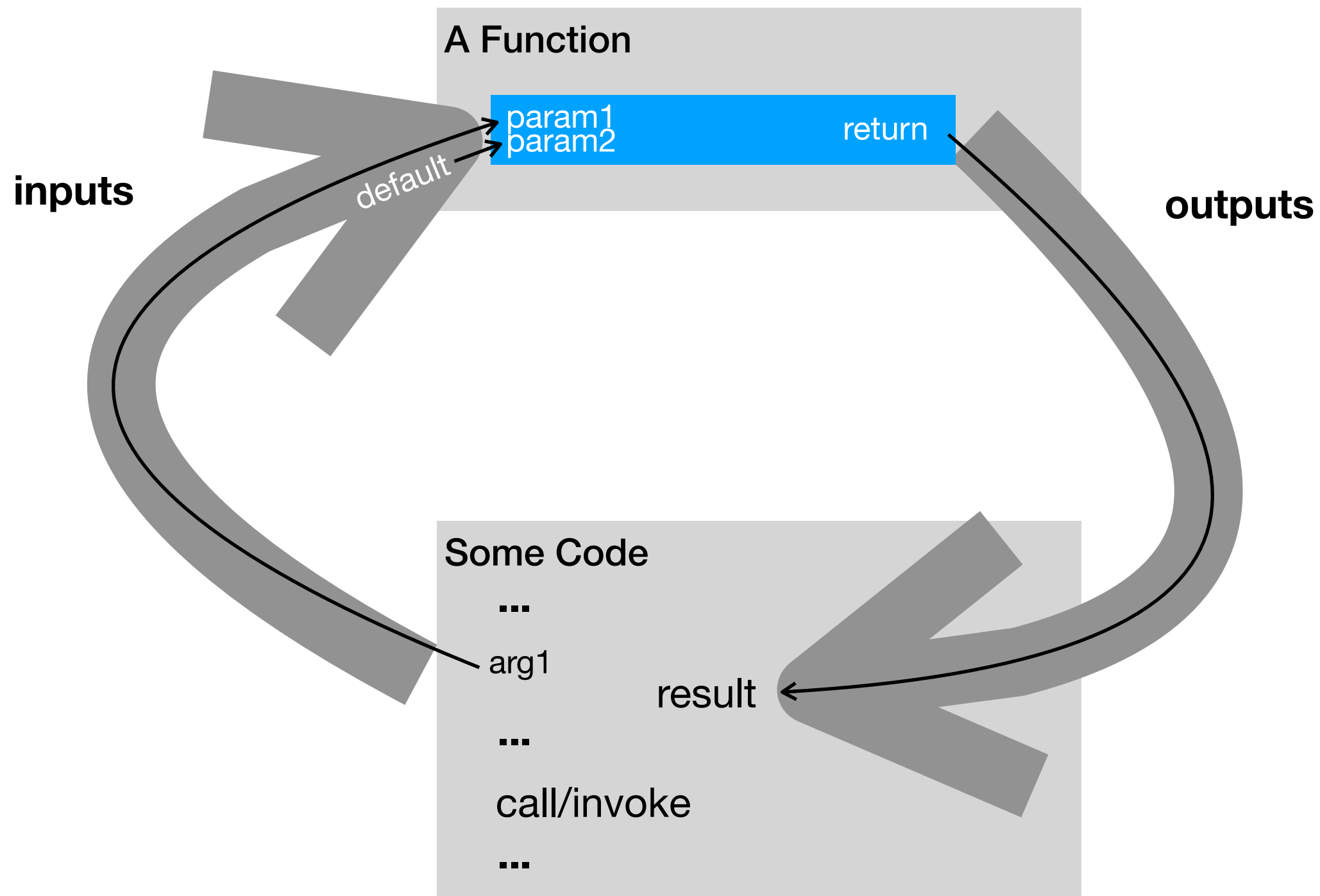
# Vocabulary

- **refactor**: change organization of code (e.g., to avoid repetition)
- **parameter**: variable that receives input to function
- **argument**: value sent to a function (lines up with parameter)
- **return value (or result)**: function output sent back to calling code



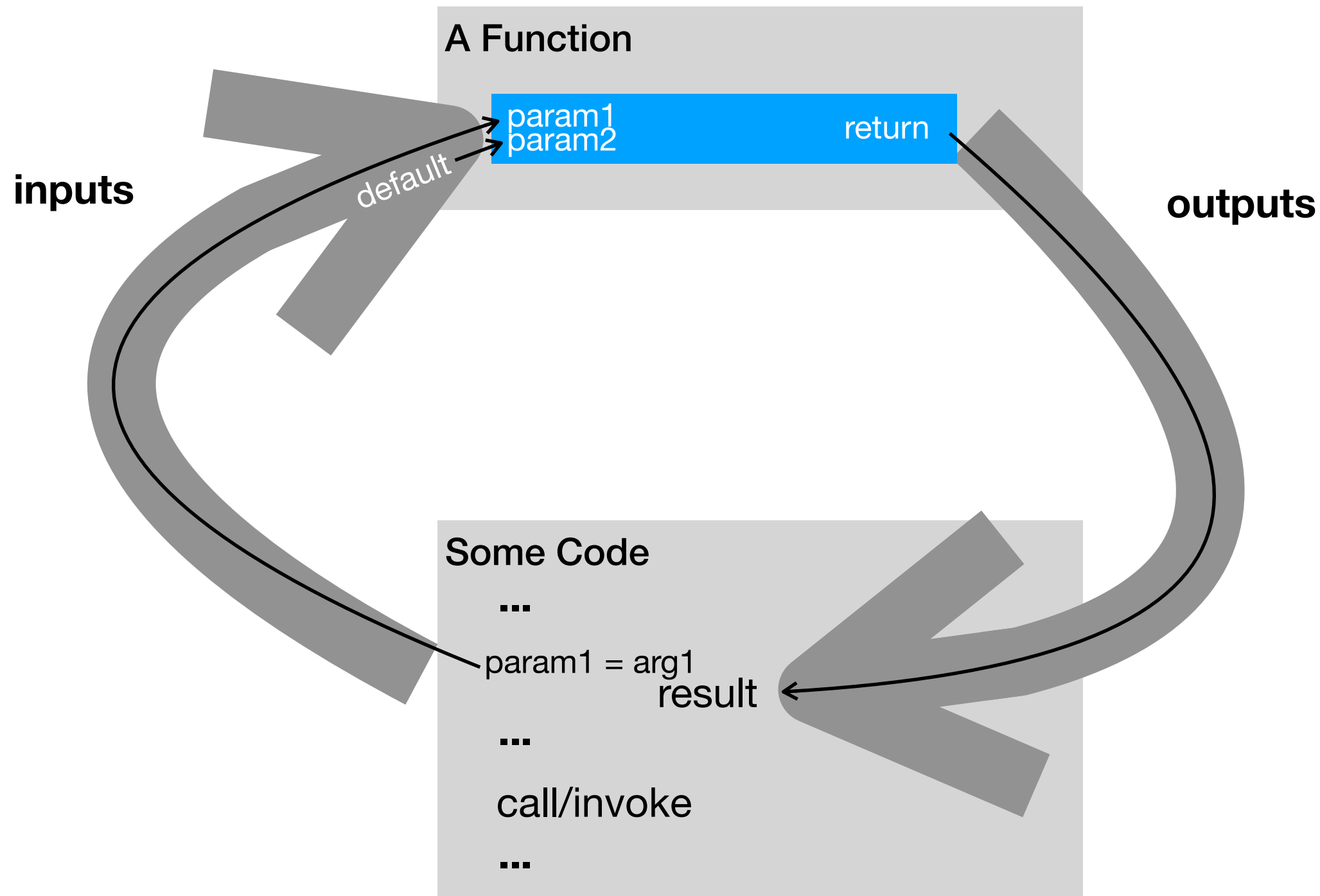
# Vocabulary

- **refactor**: change organization of code (e.g., to avoid repetition)
- **parameter**: variable that receives input to function
- **argument**: value sent to a function (lines up with parameter)
- **return value (or result)**: function output sent back to calling code
- **default argument**: value put in parameter if argument not passed



# Vocabulary

- **refactor**: change organization of code (e.g., to avoid repetition)
- **parameter**: variable that receives input to function
- **argument**: value sent to a function (lines up with parameter)
- **return value (or result)**: function output sent back to calling code
- **default argument**: value put in parameter if argument not passed
- **named/keyword argument**: argument explicitly tied to a parameter



# Calling/Invoking a Function in Python

```
print("hello")  
result = f(x)
```

# Calling/Invoking a Function in Python

```
print("hello")  
result = f(x)
```

**ALWAYS:** function's name

# Calling/Invoking a Function in Python


```
print("hello")  
result = f(x)
```

**ALWAYS:** function's name

**ALWAYS:** followed by parentheses



# Calling/Invoking a Function in Python



**arguments**

```
print("hello")  
result = f(x)
```

**ALWAYS:** function's name

**ALWAYS:** followed by parentheses

**SOMETIMES:** with one or more arguments

# Calling/Invoking a Function in Python

`print("hello")`

`result = f(x)`

 **return value**

**ALWAYS:** function's name

**ALWAYS:** followed by parentheses

**SOMETIMES:** with one or more arguments

**SOMETIMES:** producing a result

# Calling/Invoking a Function in Python

```
print("hello")
```

```
result = f(x)
```



return value

**ALWAYS:** function's name

**ALWAYS:** followed by parentheses

**SOMETIMES:** with one or more arguments

**SOMETIMES:** producing a result

**demos**

**TODO: battleship**

**TODO: modules**



**demos**

# **TODO: improving battleship**

**actual ship, multiple ships, random location**

**TODO: polar coords**