Functions

ENDG 233 – Programming with Data

Instructor: Dean Richert

Introduction to programming
Creating and running Python programs in JupyterLab
Programming style
Variables, operators, and basic data structures
Flow control (if, else, for, etc.)

Advanced data structures
Data manipulation
Data visualization
Algorithms

* I would like to acknowledge the recommendations provided by chatGPT in developing this lesson. The content and ideas have been validated by me.

Learning outcomes and outline

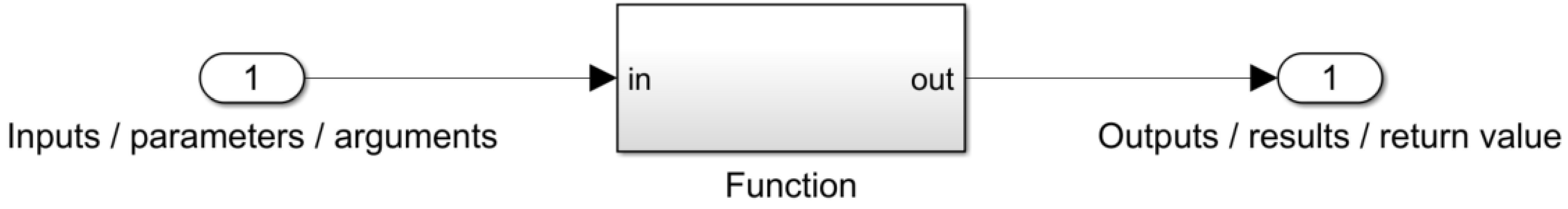
At the end of this lesson and accompanying active learning session you will be able to:

1. **Motivate** the use of functions in computer programming
2. **Define functions** in Python and the variants available
3. Call or **invoke functions** in Python
4. Understand some of the common **misconceptions** relative to Python functions, including the **distinction** between function definition and invocation
5. Apply your understanding of functions to decision-making using **machine learning**

Introduction

What is a function?

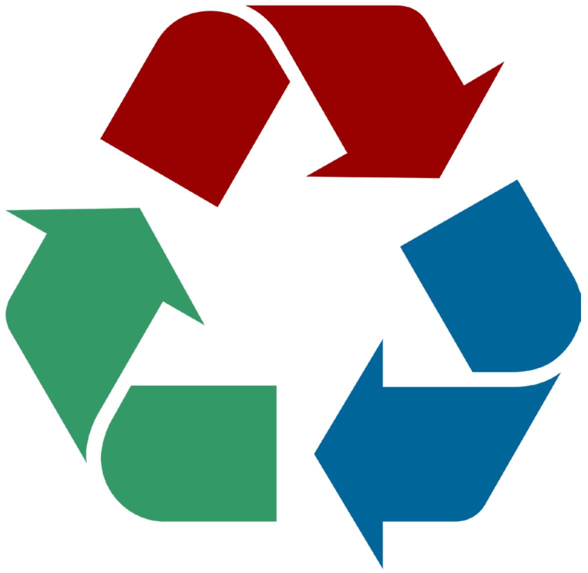
Ans: A chunk or block of code that **receives** information, **performs** tasks, and **reports** results back


1
Inputs / parameters / arguments
in
out
Function
1
Outputs / results / return value

Introduction

Why do we use function in programming?

1. Code reuse 2. Modularity 3. Readability



Introduction

Why do we use function in programming?

1. Code reuse

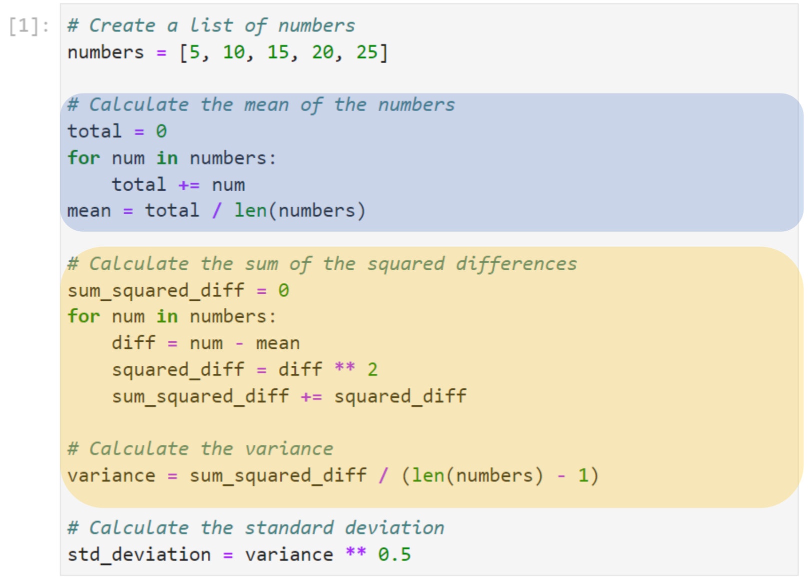
2. Modularity

3. Readability

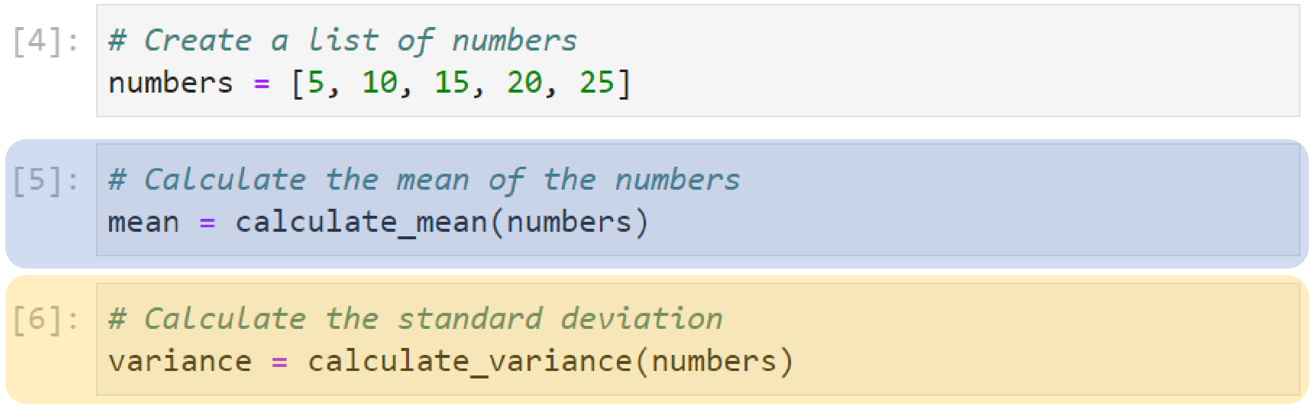
Introduction

Why do we use function in programming?

1. Code reuse   2. Modularity   **3. Readability**



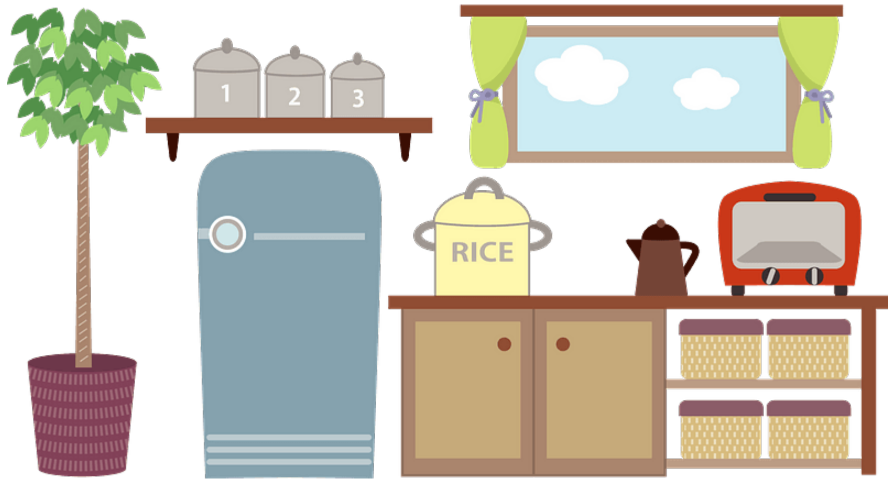
```
[1]: # Create a list of numbers
     numbers = [5, 10, 15, 20, 25]

     # Calculate the mean of the numbers
     total = 0
     for num in numbers:
         total += num
     mean = total / len(numbers)

     # Calculate the sum of the squared differences
     sum_squared_diff = 0
     for num in numbers:
         diff = num - mean
         squared_diff = diff ** 2
         sum_squared_diff += squared_diff

     # Calculate the variance
     variance = sum_squared_diff / (len(numbers) - 1)

     # Calculate the standard deviation
     std_deviation = variance ** 0.5
```



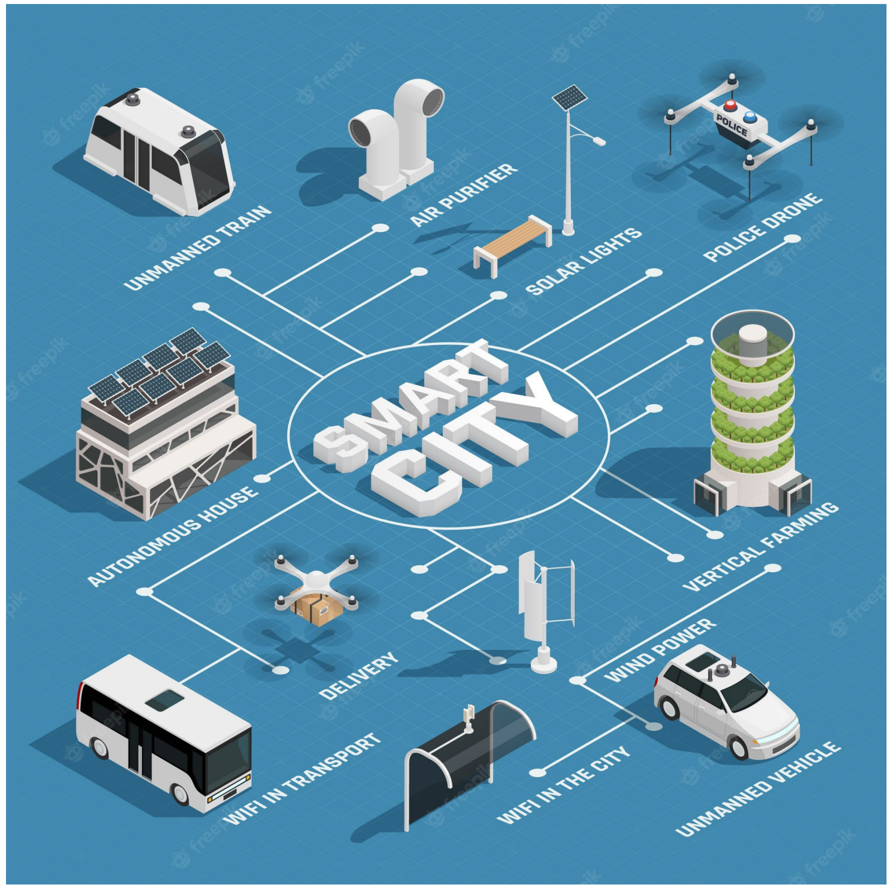
```
[4]: # Create a list of numbers
     numbers = [5, 10, 15, 20, 25]

[5]: # Calculate the mean of the numbers
     mean = calculate_mean(numbers)

[6]: # Calculate the standard deviation
     variance = calculate_variance(numbers)
```

Introduction

Analogy of a function:



Engineering example:



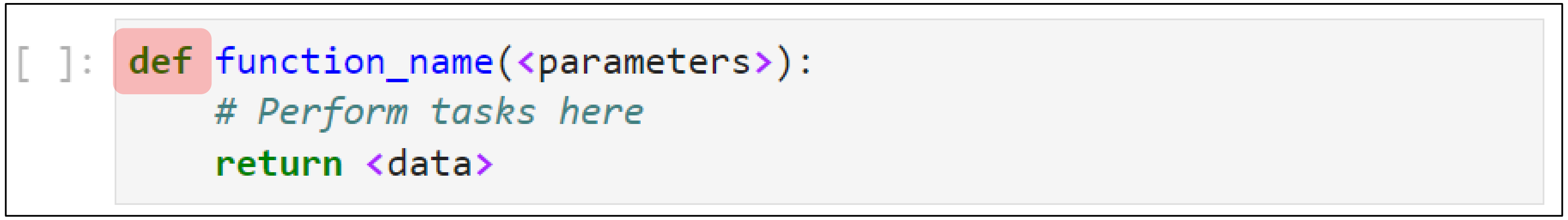
Function Definition

Functions must be defined **before** they can be used!

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:

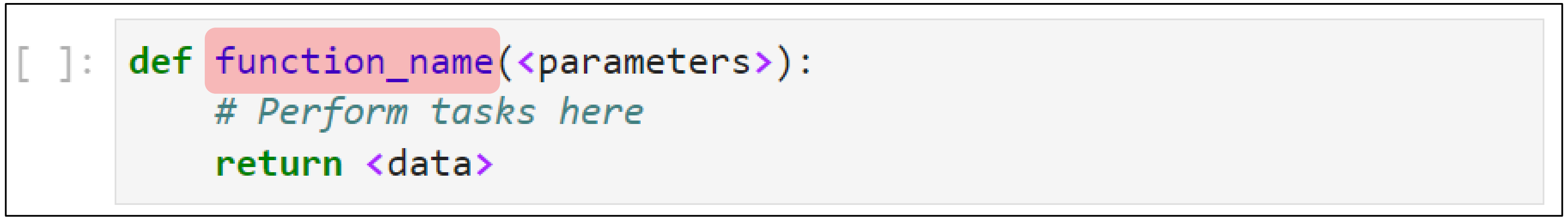


```
[ ]: def function_name(<parameters>):
         # Perform tasks here
         return <data>
```

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:

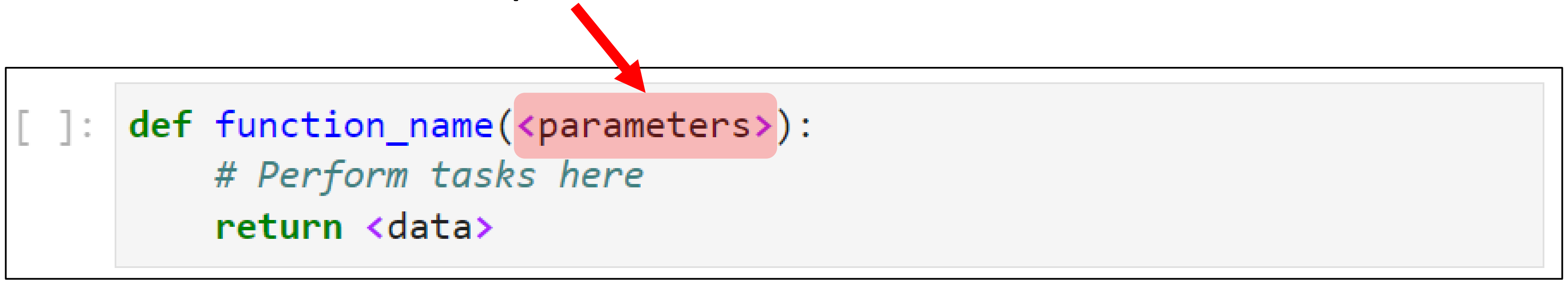

```
[ ]: def function_name(<parameters>):
         # Perform tasks here
         return <data>
```

Function names should be lower case, meaningful, and have underscores to separate words

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:

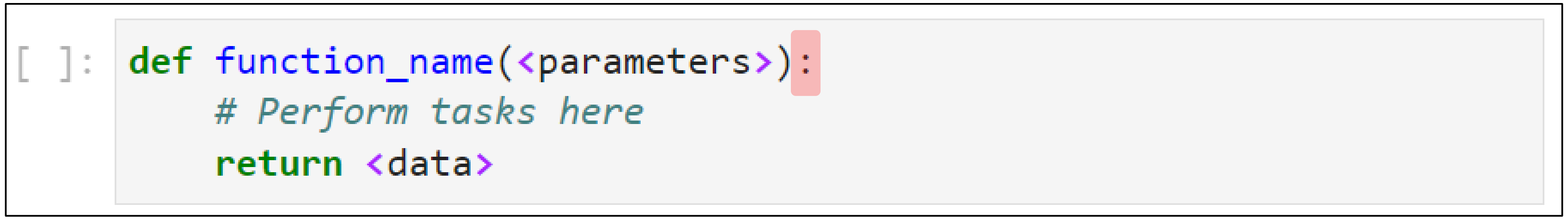


```
[ ]: def function_name(<parameters>):
         # Perform tasks here
         return <data>
```

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:

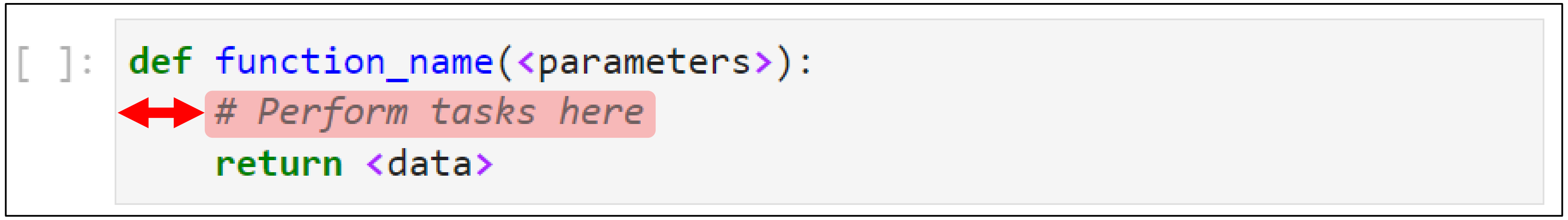


```
[ ]: def function_name(<parameters>):
         # Perform tasks here
         return <data>
```

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:

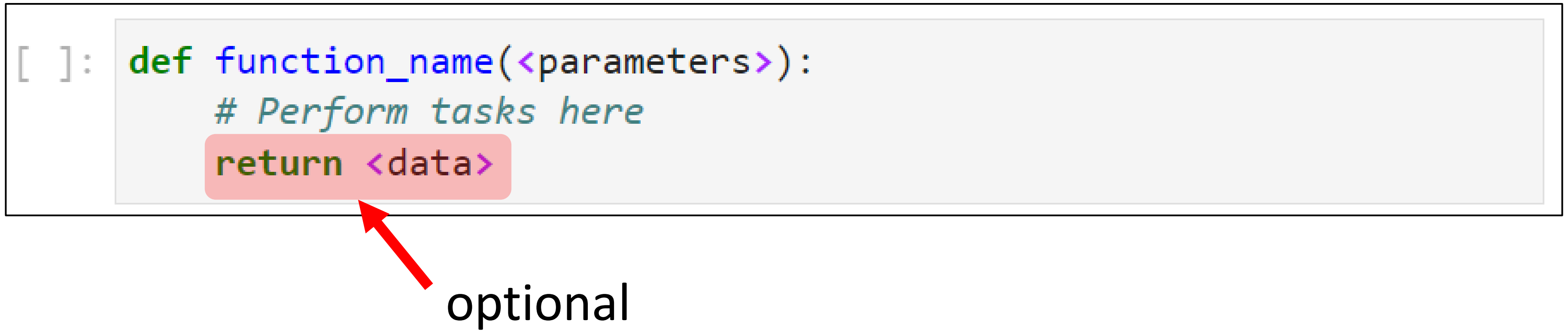


```
[ ]: def function_name(<parameters>):
         # Perform tasks here
         return <data>
```

Function Definition

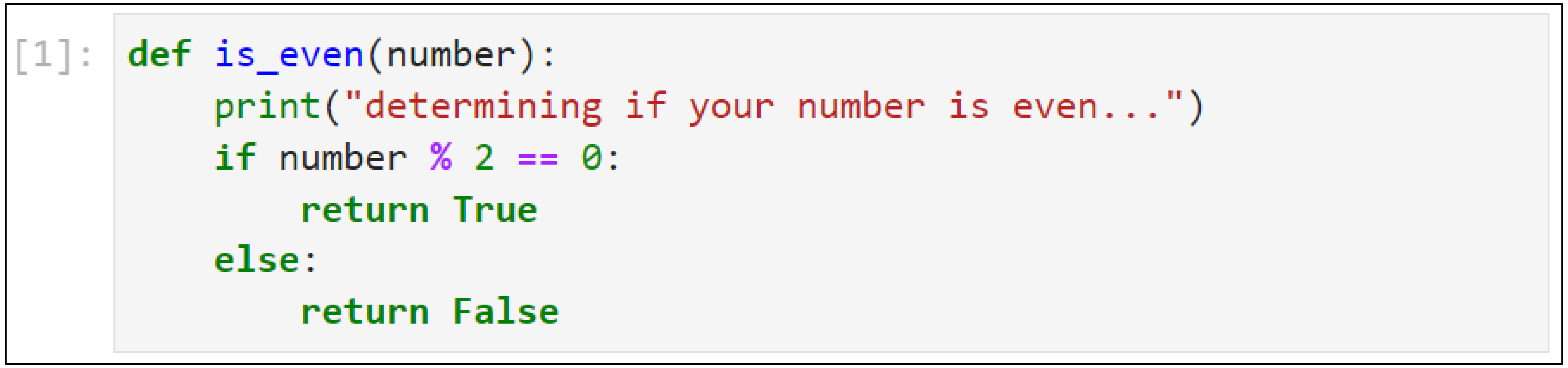
Functions must be defined before they can be used!

Anatomy of a function definition:



```
[ ]: def function_name(<parameters>):
         # Perform tasks here
         return <data>
```

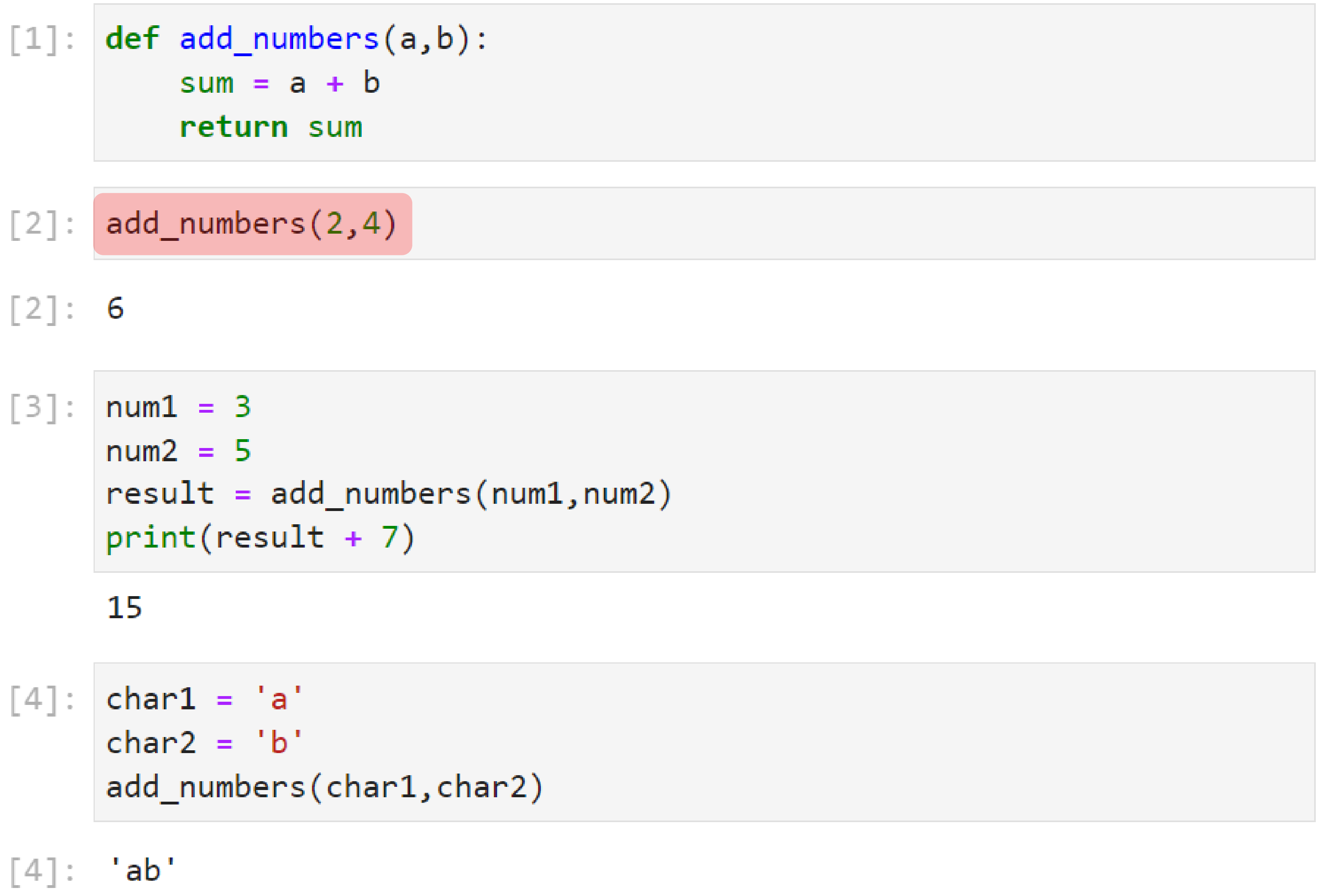
Function Definition - Example



```
[1]: def is_even(number):
         print("determining if your number is even...")
         if number % 2 == 0:
             return True
         else:
             return False
```

Function Invocation

Functions must be called/invoked to execute its code!



```
[1]: def add_numbers(a,b):
         sum = a + b
         return sum
```

```
[2]: add_numbers(2,4)
```

```
[2]: 6
```

```
[3]: num1 = 3
     num2 = 5
     result = add_numbers(num1,num2)
     print(result + 7)
```

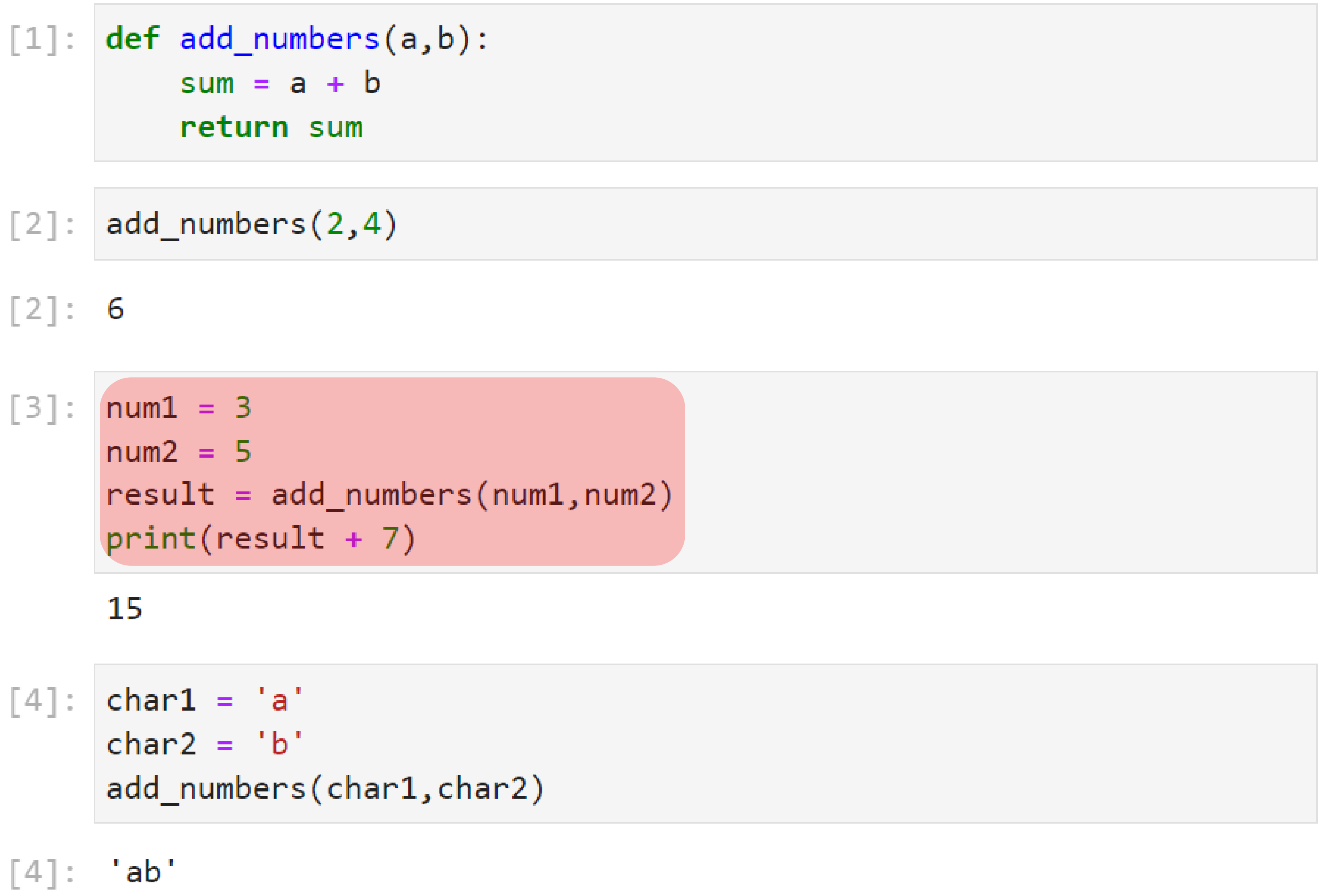
```
     15
```

```
[4]: char1 = 'a'
     char2 = 'b'
     add_numbers(char1,char2)
```

```
[4]: 'ab'
```

Function Invocation

Functions must be called/invoked to execute its code!



```
[1]: def add_numbers(a,b):
         sum = a + b
         return sum
```

```
[2]: add_numbers(2,4)
```

```
[2]: 6
```

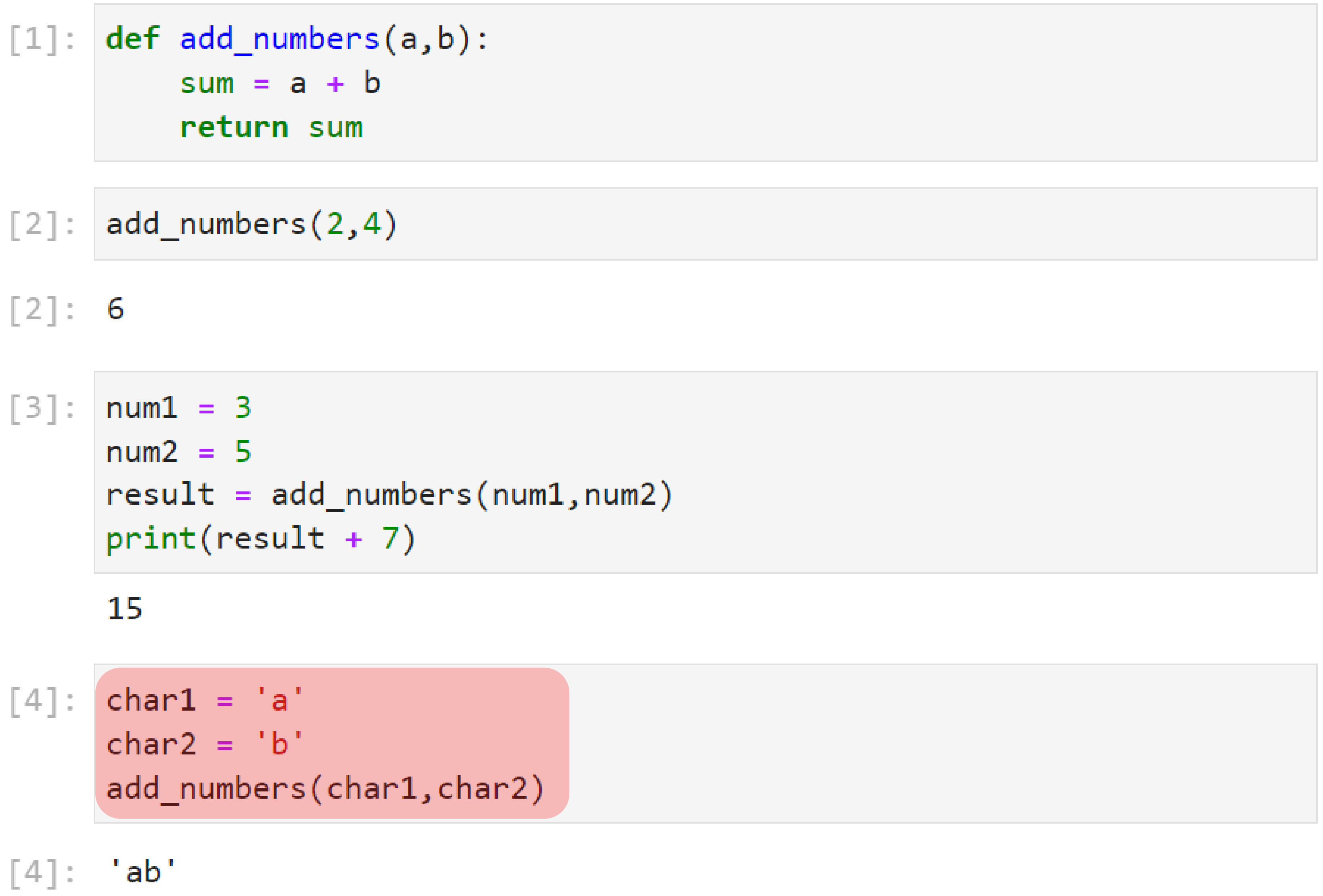
```
[3]: num1 = 3
     num2 = 5
     result = add_numbers(num1,num2)
     print(result + 7)

     15
```

```
[4]: char1 = 'a'
     char2 = 'b'
     add_numbers(char1,char2)
```

```
[4]: 'ab'
```

Function Invocation

Functions must be called/invoked to execute its code!



```
[1]: def add_numbers(a,b):
         sum = a + b
         return sum
```

```
[2]: add_numbers(2,4)
```

```
[2]: 6
```

```
[3]: num1 = 3
     num2 = 5
     result = add_numbers(num1,num2)
     print(result + 7)
```

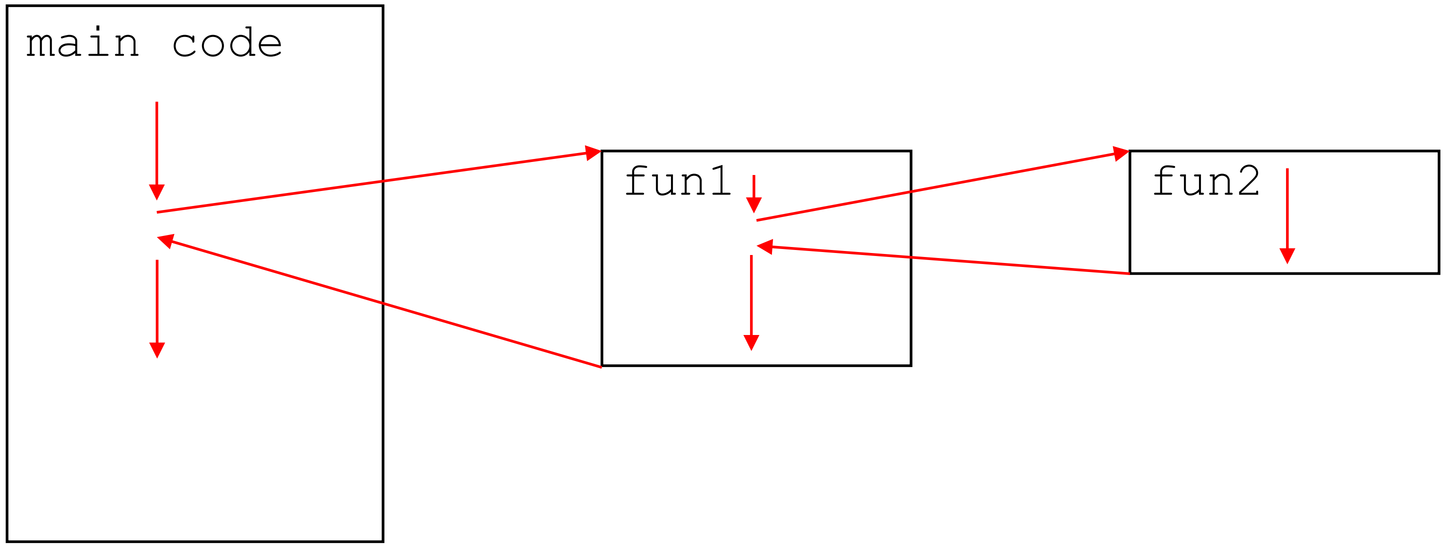
```
15
```

```
[4]: char1 = 'a'
     char2 = 'b'
     add_numbers(char1,char2)
```

```
[4]: 'ab'
```

Function Invocation

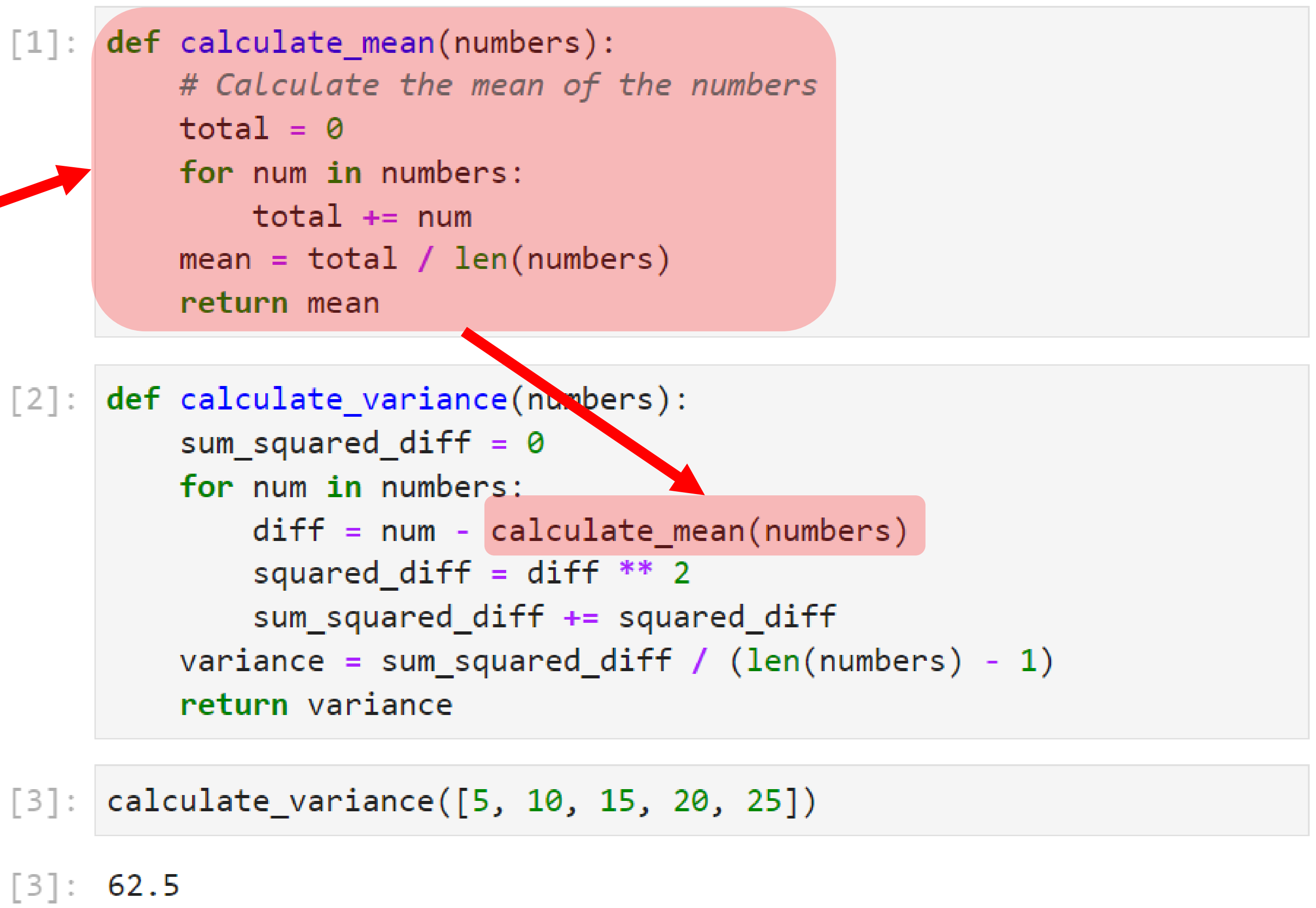
Functions can call other functions (or even themselves)



Function Invocation

Functions can call other functions (or even themselves)

$$S^2 = \frac{\sum(x_i - \boxed{\bar{x}})^2}{n-1}$$



```
[1]: def calculate_mean(numbers):
         # Calculate the mean of the numbers
         total = 0
         for num in numbers:
             total += num
         mean = total / len(numbers)
         return mean
```

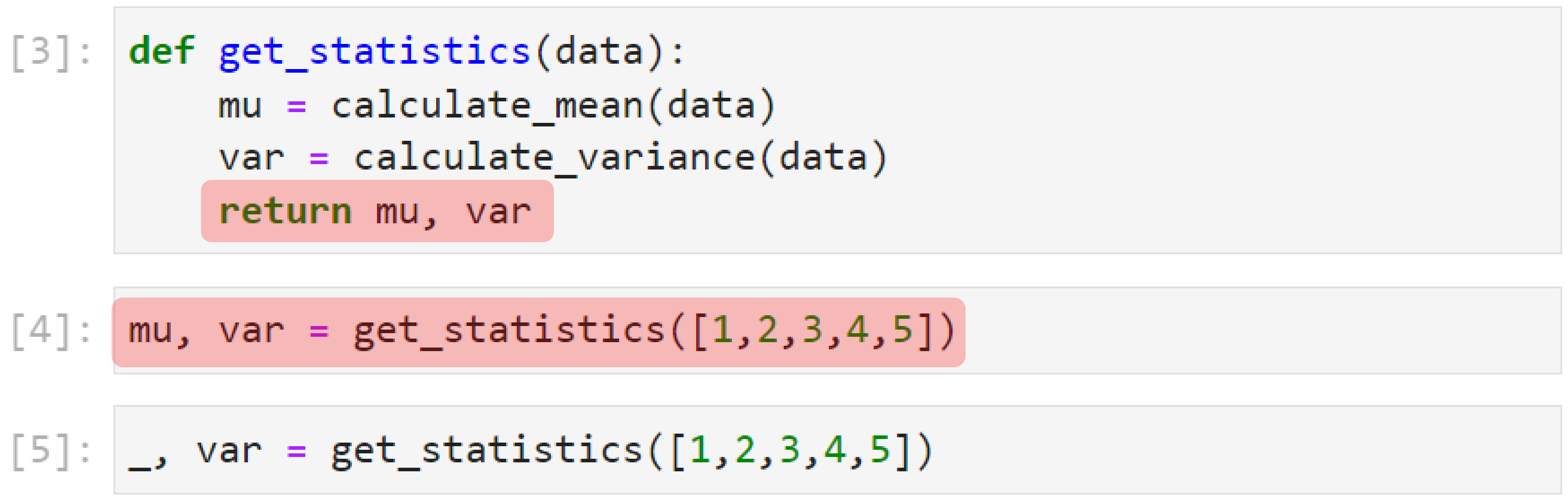
```
[2]: def calculate_variance(numbers):
         sum_squared_diff = 0
         for num in numbers:
             diff = num - calculate_mean(numbers)
             squared_diff = diff ** 2
             sum_squared_diff += squared_diff
         variance = sum_squared_diff / (len(numbers) - 1)
         return variance
```

```
[3]: calculate_variance([5, 10, 15, 20, 25])
```

```
[3]: 62.5
```

Advanced topics

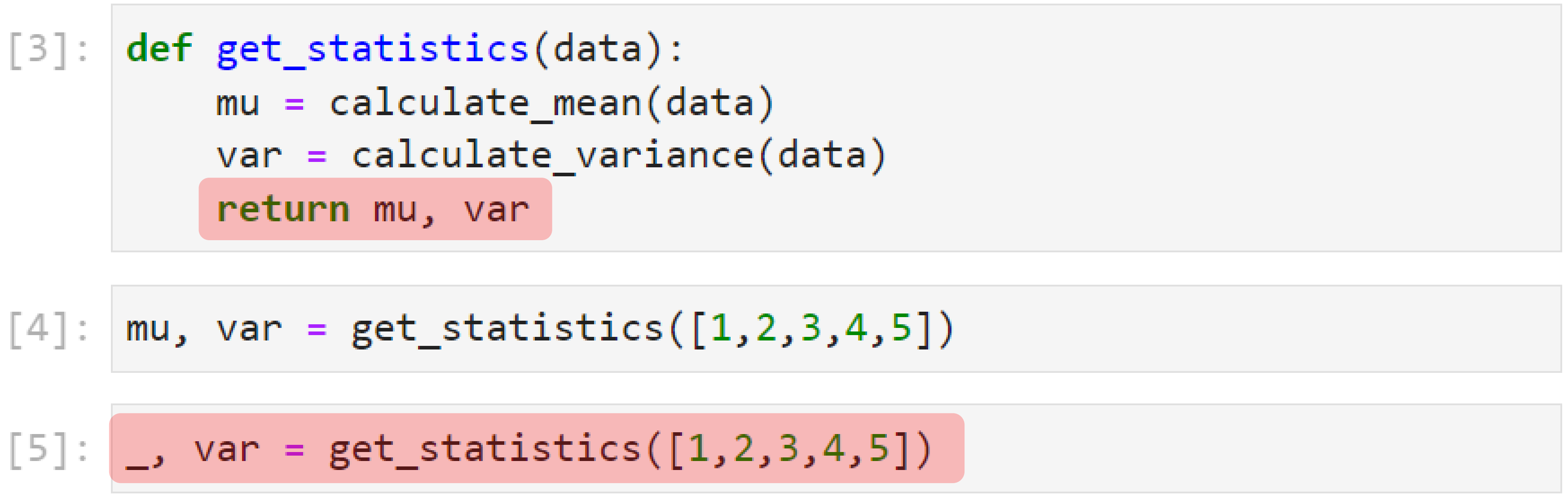
Multiple return values



```
[3]: def get_statistics(data):
         mu = calculate_mean(data)
         var = calculate_variance(data)
         return mu, var

[4]: mu, var = get_statistics([1,2,3,4,5])

[5]: _, var = get_statistics([1,2,3,4,5])
```

Advanced topics

Multiple return values



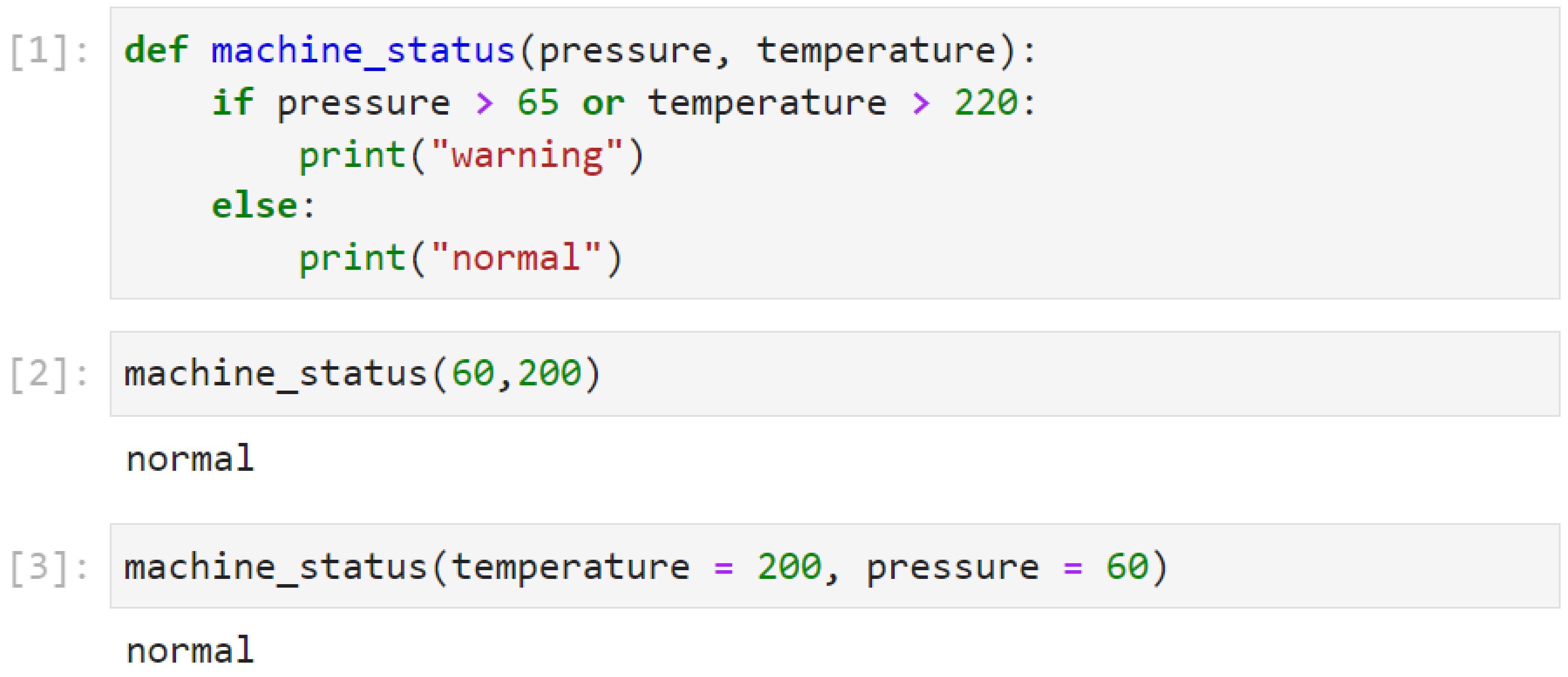
```
[3]: def get_statistics(data):
         mu = calculate_mean(data)
         var = calculate_variance(data)
         return mu, var
```

```
[4]: mu, var = get_statistics([1,2,3,4,5])
```

```
[5]: _, var = get_statistics([1,2,3,4,5])
```

Advanced topics

Keyword arguments allow parameters to be sent out of order



```
[1]: def machine_status(pressure, temperature):
         if pressure > 65 or temperature > 220:
             print("warning")
         else:
             print("normal")
```

```
[2]: machine_status(60,200)
```

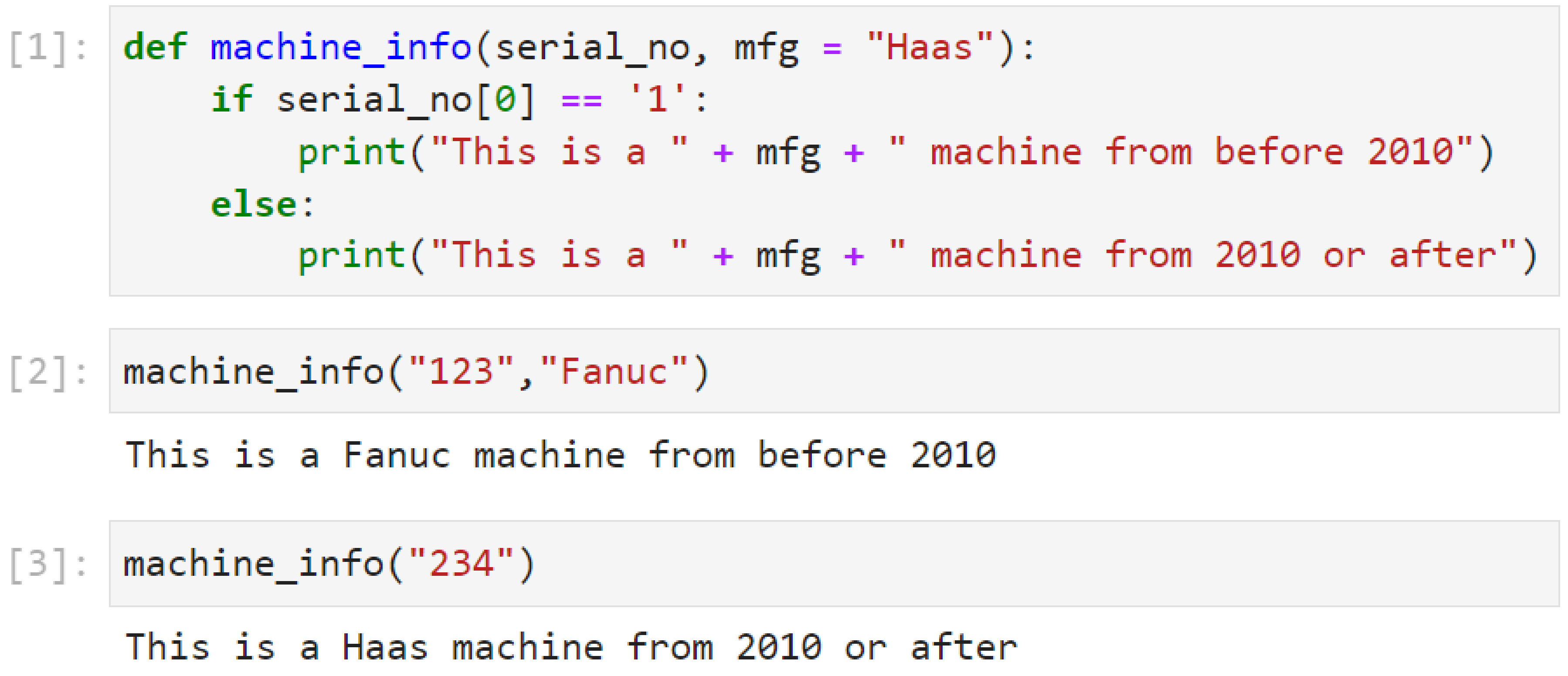
```
normal
```

```
[3]: machine_status(temperature = 200, pressure = 60)
```

```
normal
```

Advanced topics

Default argument values allow parameters to omitted



```
[1]: def machine_info(serial_no, mfg = "Haas"):
         if serial_no[0] == '1':
             print("This is a " + mfg + " machine from before 2010")
         else:
             print("This is a " + mfg + " machine from 2010 or after")
```

```
[2]: machine_info("123","Fanuc")
```

```
This is a Fanuc machine from before 2010
```

```
[3]: machine_info("234")
```

```
This is a Haas machine from 2010 or after
```

Resources

- Programming with Mosh YouTube channel – Python Functions video
- W3Schools Python Functions page
    - For further study:
        - Arbitrary Arguments (`*args`)
        - Arbitrary Keyword Arguments (`**kwargs`)
        - `pass` statement
        - Recursion
- chatGPT to help with debugging or generating sample code