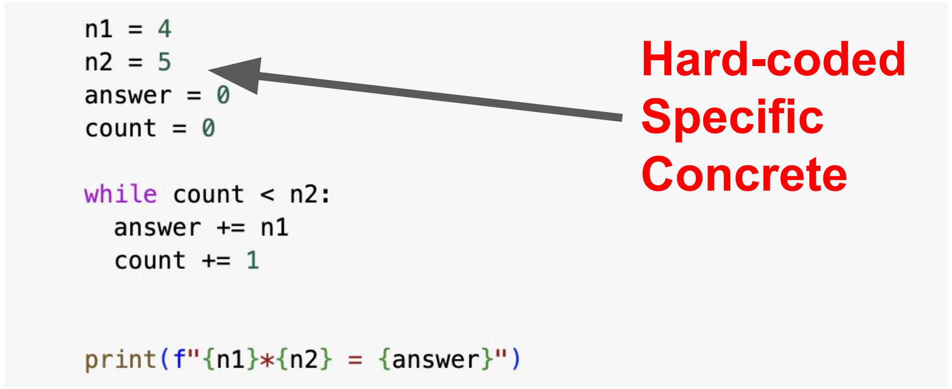
Functions

and Function Scope

Functions abstract tasks in the code

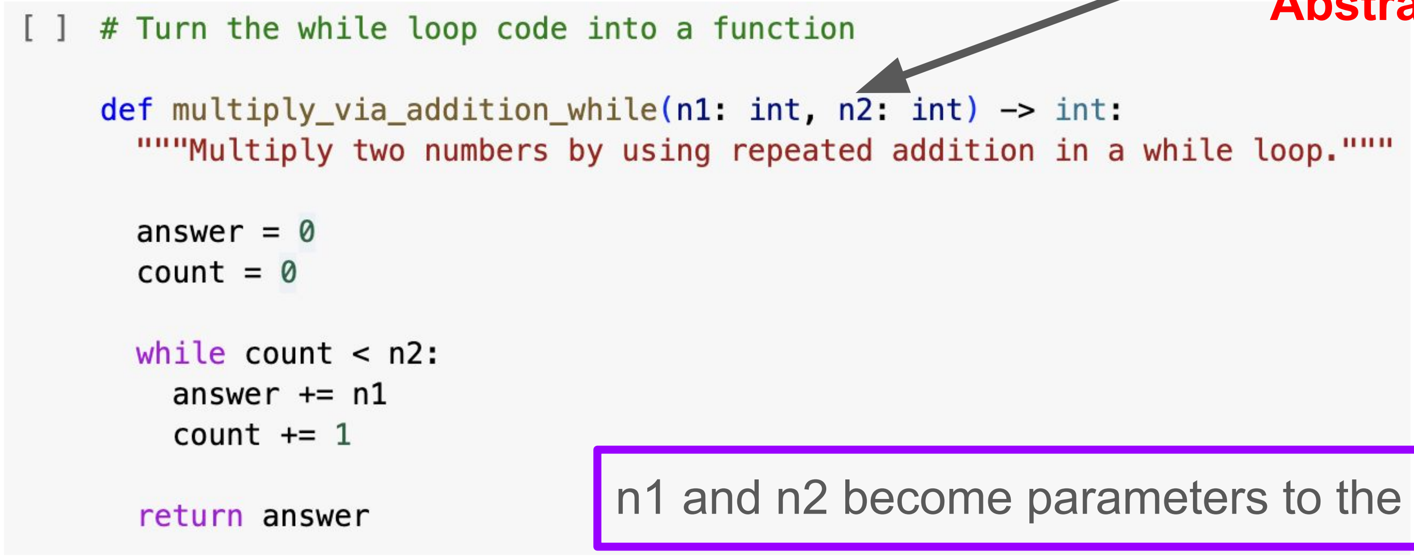
- consider task of multiplying two numbers together



```python
n1 = 4
n2 = 5
answer = 0
count = 0

while count < n2:
  answer += n1
  count += 1


print(f"{n1}*{n2} = {answer}")
```

Hard-coded
Specific
Concrete

Generic,
Unspecified
Abstract



```python
[ ]  # Turn the while loop code into a function

     def multiply_via_addition_while(n1: int, n2: int) -> int:
       """Multiply two numbers by using repeated addition in a while loop."""

       answer = 0
       count = 0

       while count < n2:
         answer += n1
         count += 1

       return answer
```

n1 and n2 become parameters to the function

Abstraction

- Functions are defined with abstract parameters (not hard coded)
- Functions are abstractions of tasks/concepts
    - could be useful even if the implementation is unknown

Example

Do you need to know how to make a projector in order to make it work?

This course

- demystify common functions
    - searching
    - sorting
    - storing information

When you see functions or other abstractions, you have some idea how they work on the inside

You can make abstractions for others (or you) to use

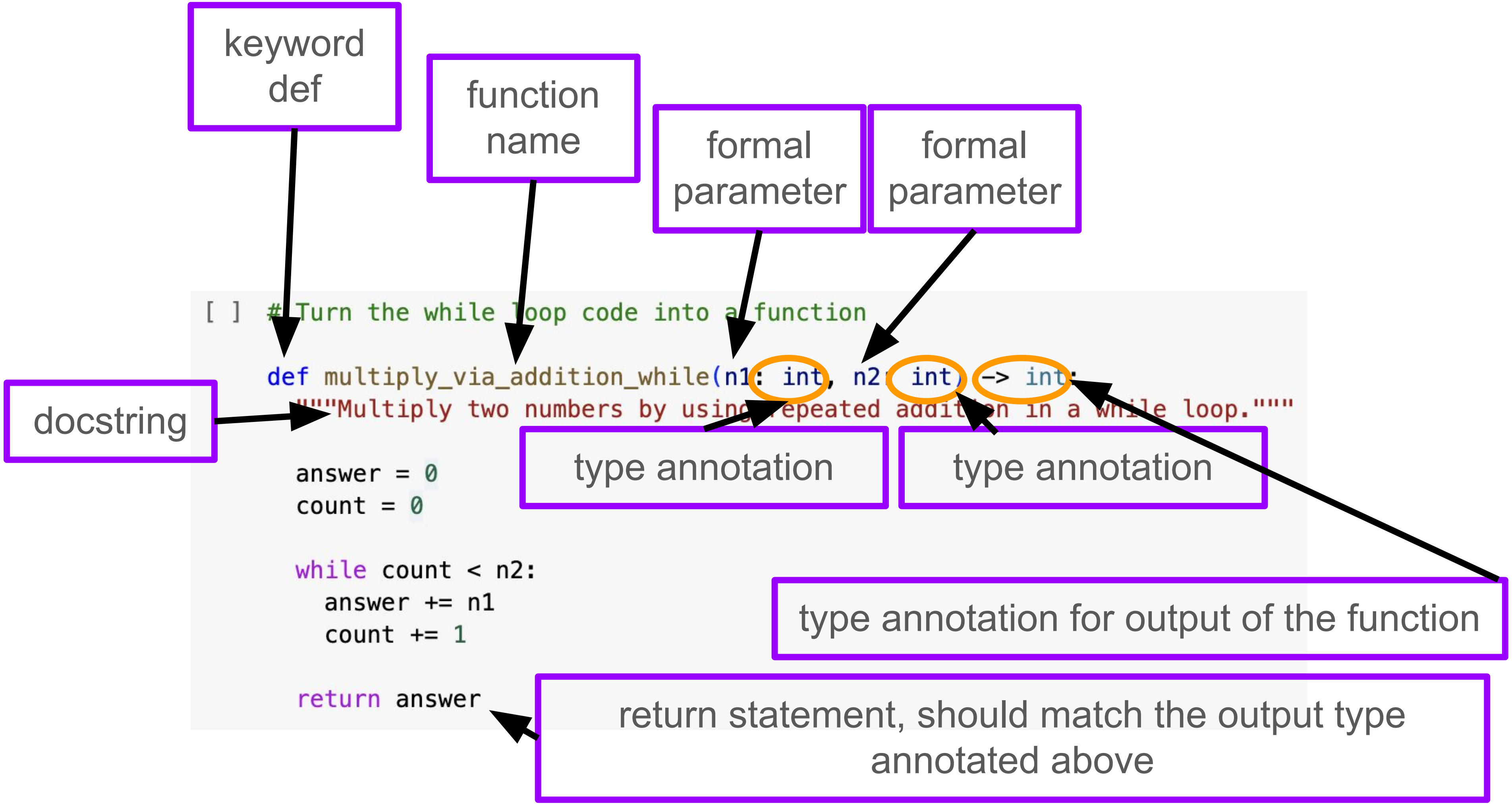
Decomposition

Concept that large problems can be broken up into smaller problems

- smaller problems/tasks could be solved by implementing function
    - then calling the functions

So, functions can be thought of as mini programs

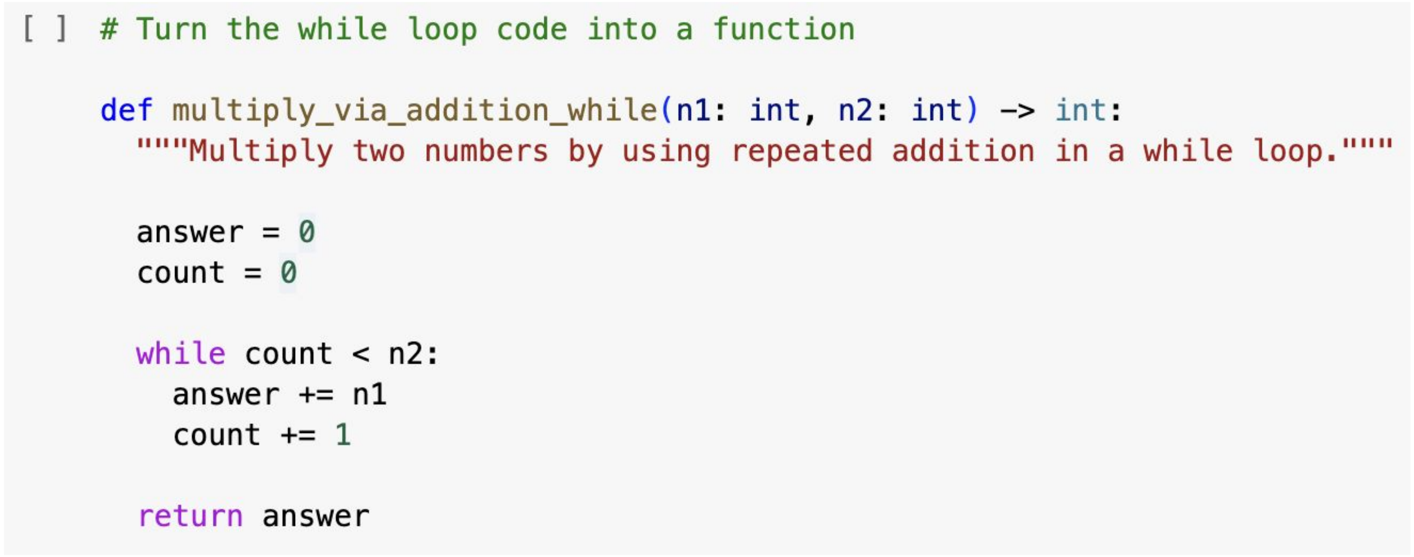
- once a function is written
    - The same function can be (re)used with different inputs to get different outputs
    - all the instructions are only written ONCE
        - you only have to debug that function in ONE location


keyword def
function name
formal parameter
formal parameter
docstring
type annotation
type annotation
type annotation for output of the function
return statement, should match the output type annotated above

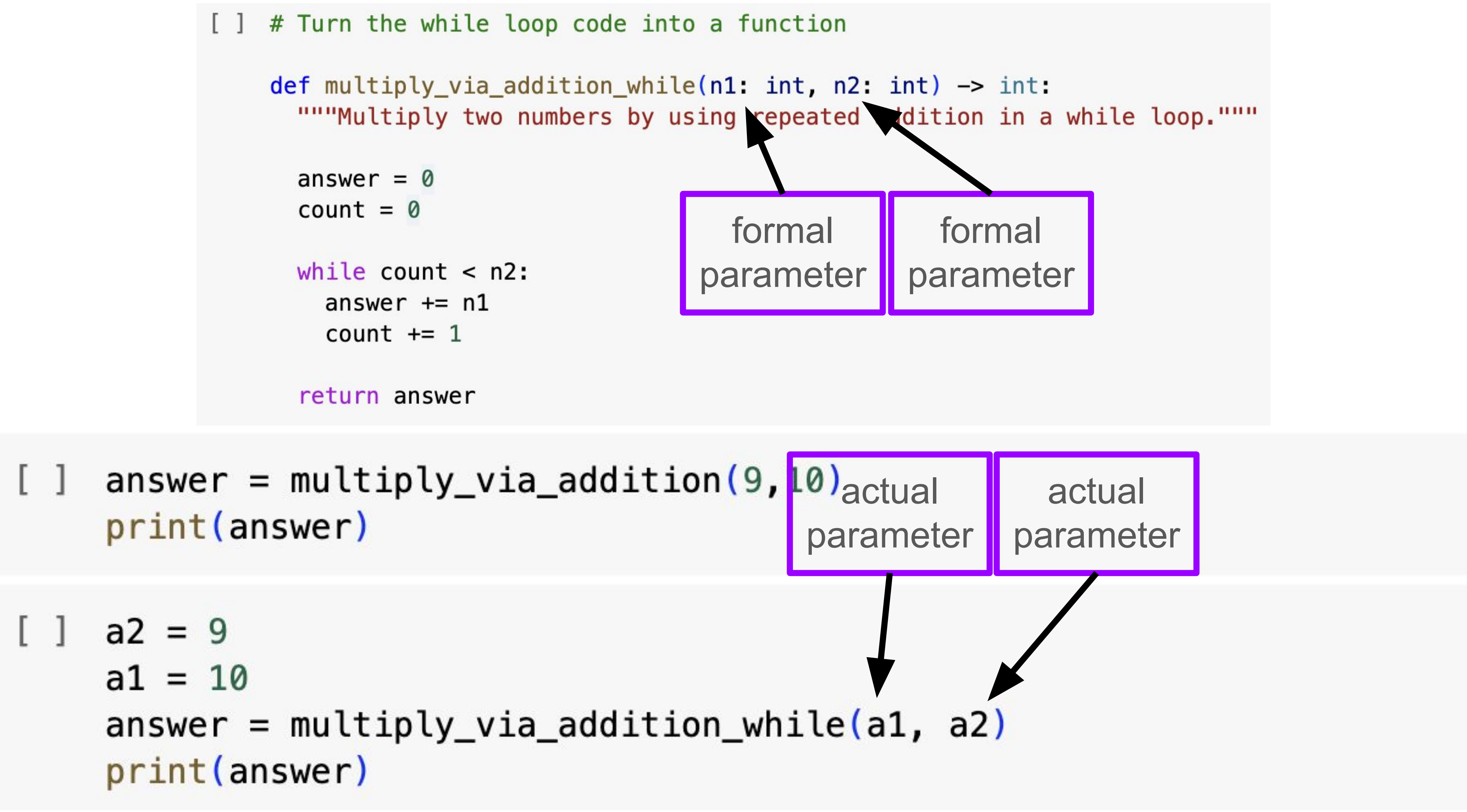
```
[ ] # Turn the while loop code into a function

    def multiply_via_addition_while(n1: int, n2: int) -> int:
      """Multiply two numbers by using repeated addition in a while loop."""

      answer = 0
      count = 0

      while count < n2:
        answer += n1
        count += 1

      return answer
```

Call this function

https://forms.gle/VjCKmkT9acNPdHiUA



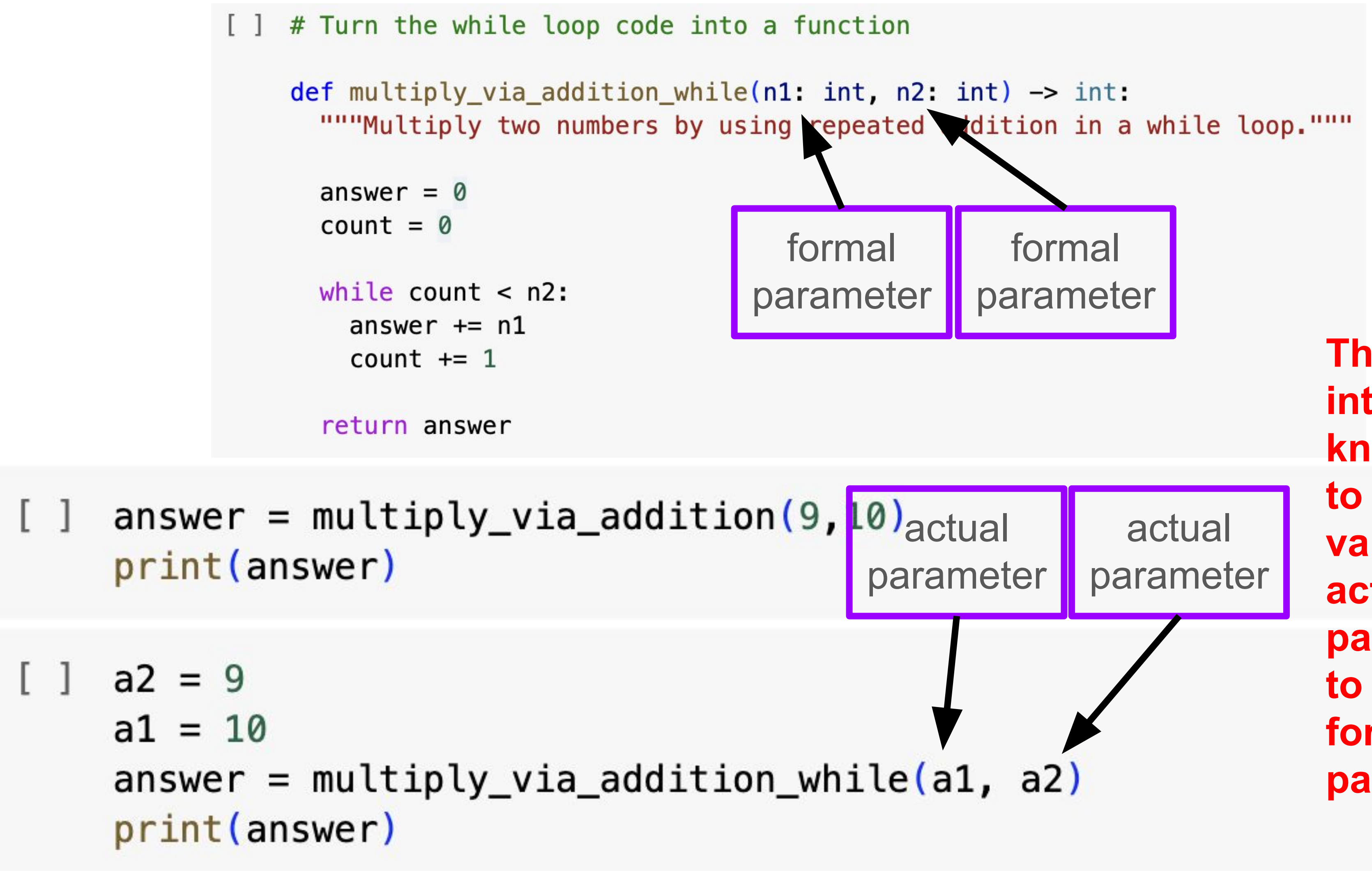
```
[ ] # Turn the while loop code into a function

    def multiply_via_addition_while(n1: int, n2: int) -> int:
      """Multiply two numbers by using repeated addition in a while loop."""

      answer = 0
      count = 0

      while count < n2:
        answer += n1
        count += 1

      return answer
```



```
[ ] # Turn the while loop code into a function

    def multiply_via_addition_while(n1: int, n2: int) -> int:
      """Multiply two numbers by using repeated addition in a while loop."""

      answer = 0
      count = 0

      while count < n2:
        answer += n1
        count += 1

      return answer
```

```
[ ] answer = multiply_via_addition(9,10)
    print(answer)
```

```
[ ] a2 = 9
    a1 = 10
    answer = multiply_via_addition_while(a1, a2)
    print(answer)
```



```
[ ] # Turn the while loop code into a function

    def multiply_via_addition_while(n1: int, n2: int) -> int:
      """Multiply two numbers by using repeated addition in a while loop."""

      answer = 0
      count = 0

      while count < n2:
        answer += n1
        count += 1

      return answer
```

```
[ ] answer = multiply_via_addition(9,10)
    print(answer)
```

```
[ ] a2 = 9
    a1 = 10
    answer = multiply_via_addition_while(a1, a2)
    print(answer)
```

The python interpreter knows how to bind the value of the actual parameters to the formal parameters

Scope

When a function is called, it is executed in a temporary isolated environment called a **stack frame**

The function should not need to know about the outside world (with rare exceptions)

- Everything that the function needs should be passed in or computed
- the actual parameters' values are bound to the formal parameters
- globals can technically be accessed inside a function

Check out ipynb for function scope. Take and save notes on this file!

Schedule up to Midterm on Monday Feb 19th

Monday Feb 12

- Higher order functions
- named parameters
- lambda functions

Wednesday Feb 14

- review/question session

Thursday Feb 15

- Study / Make cheat sheet for exam / finish labs

Friday Feb 16

- review/question session