Functions and Function Scope

Guttag Chapter 4

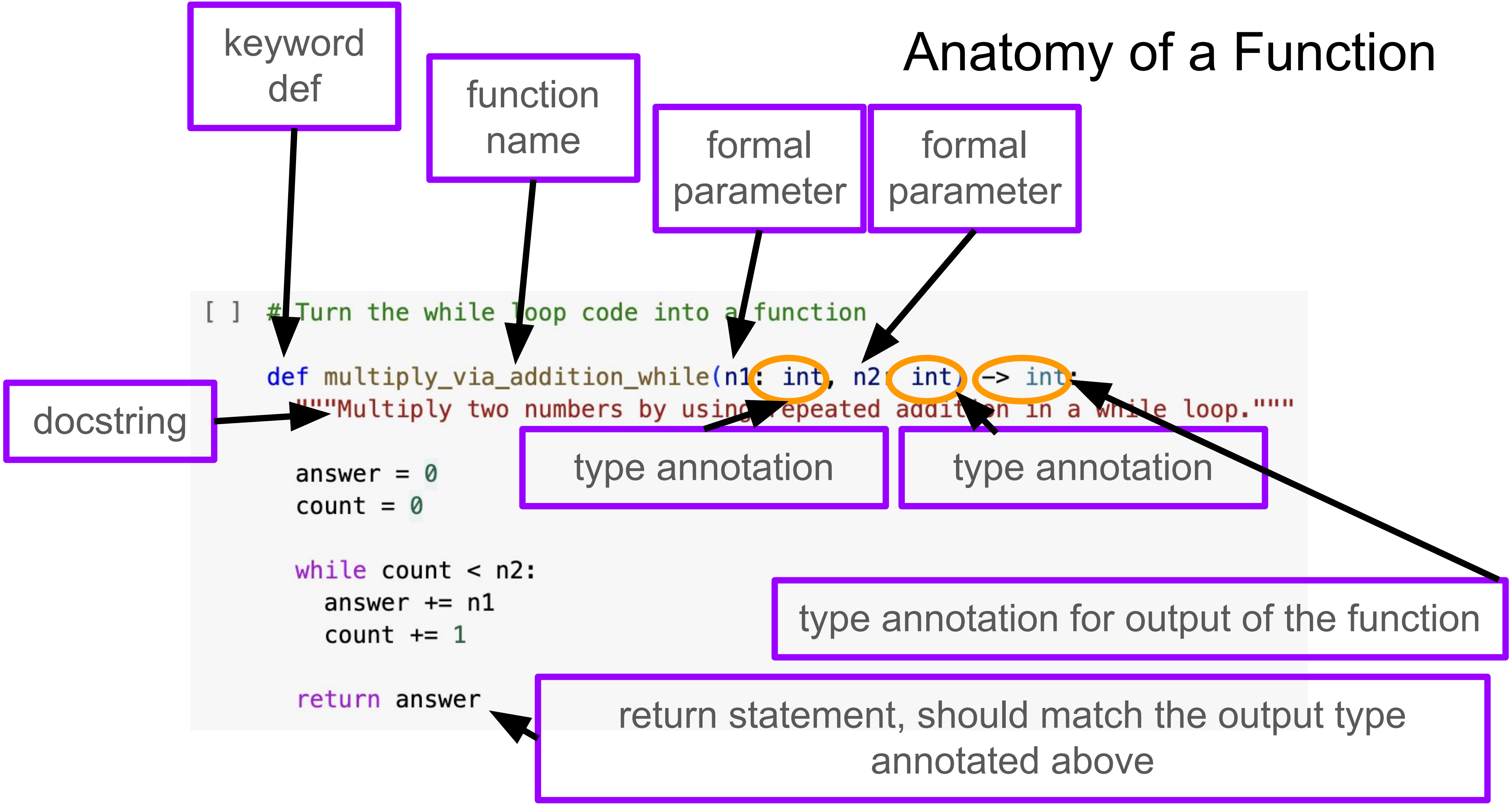
Goals

- Reminders about defining a function vs calling a function
- Specify all parts of a function
    - formal parameter
    - actual parameter
- learn vocab
    - point of execution
    - stack frame
    - local
    - global
    - pass by assignment

Defining vs Calling

Which one is "a set of instructions"?	definition
Which one contains a return statement?	definition
Which one stores the return value?	call
Where should printing go?	call
When does the function get executed?	call

Anatomy of a Function



```
[ ]  # 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
```

keyword def

function name

formal parameter

formal parameter

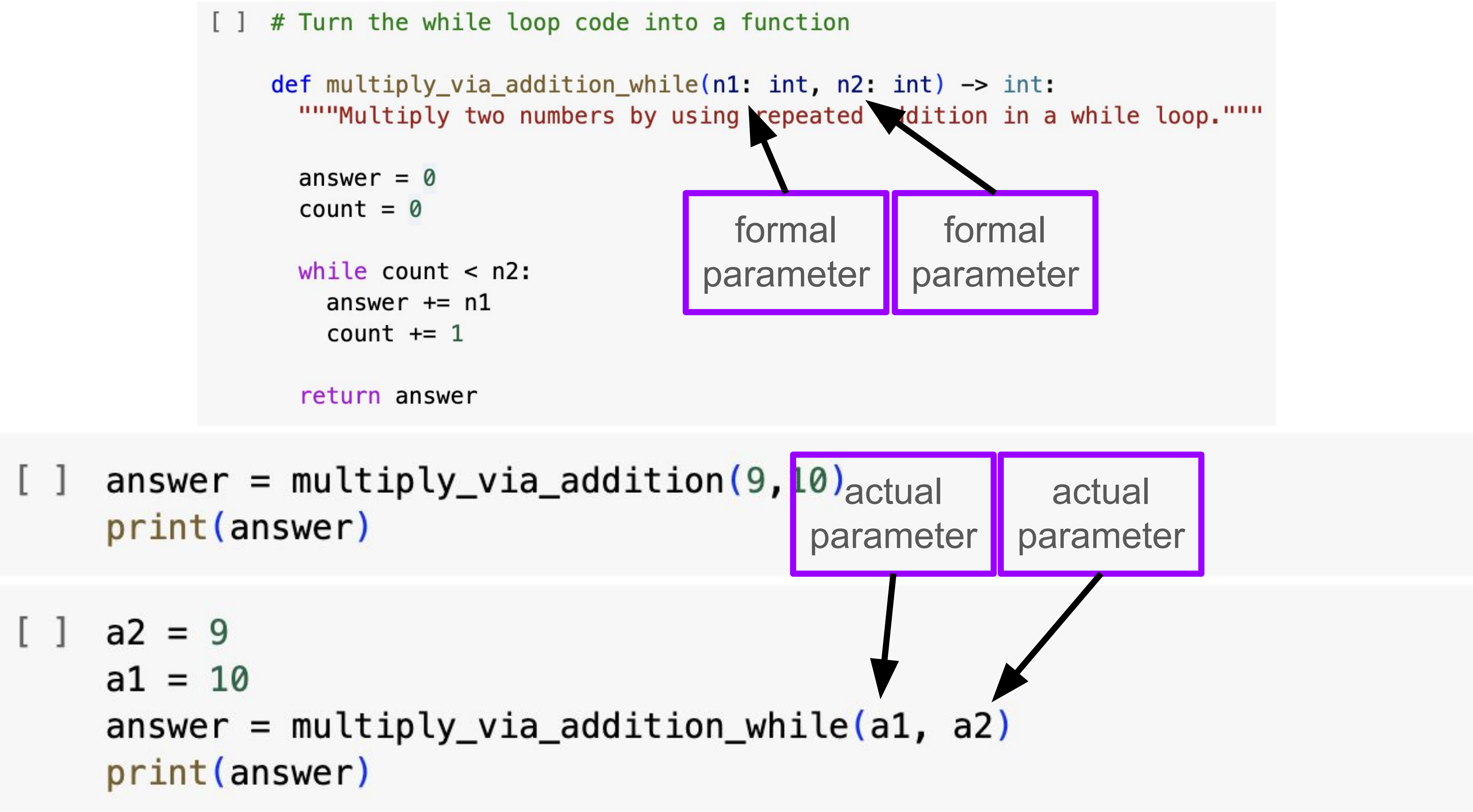
docstring

type annotation

type annotation

type annotation for output of the function

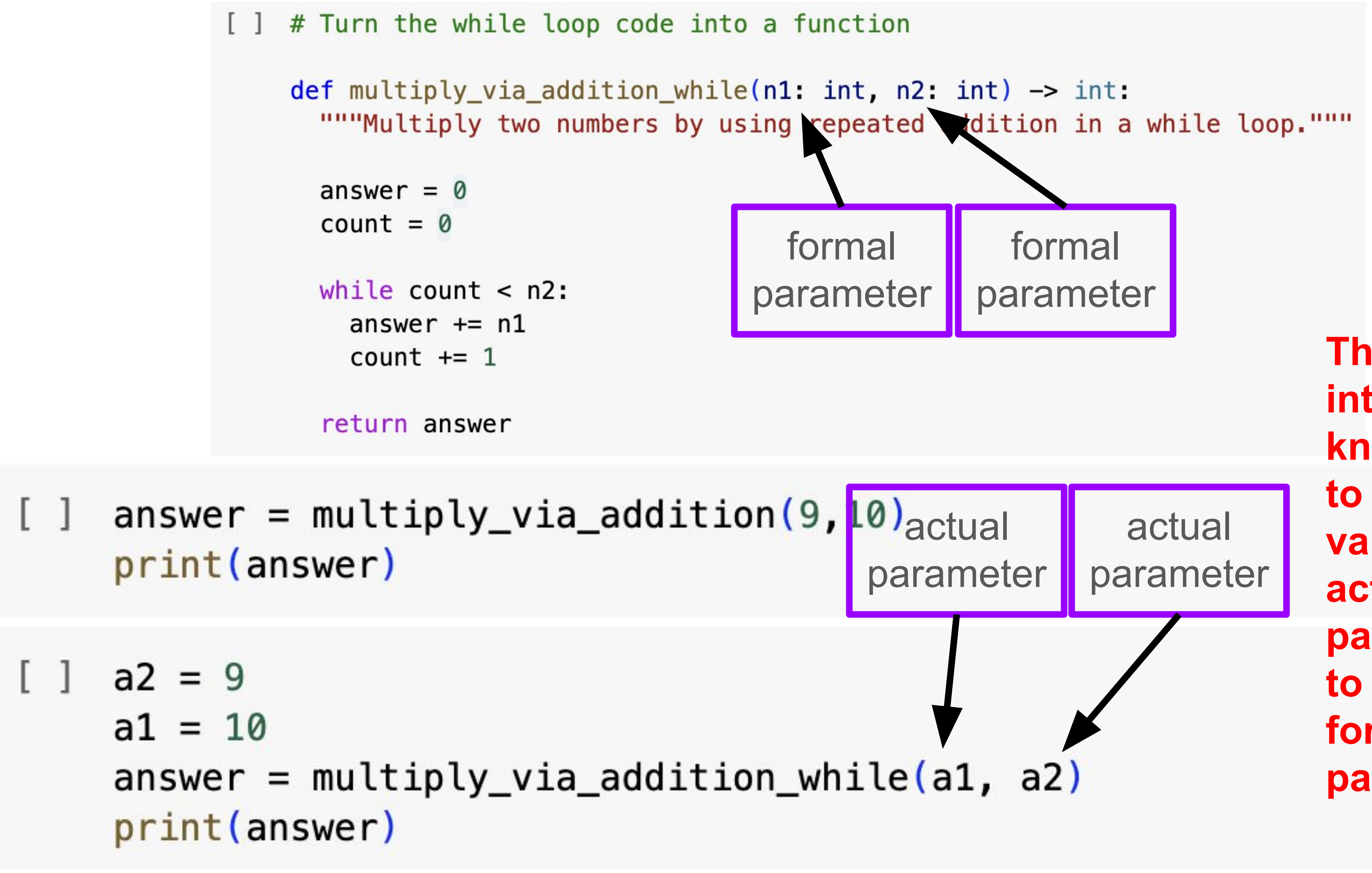
return statement, should match the output type annotated above



```
# 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
```

```
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)
```

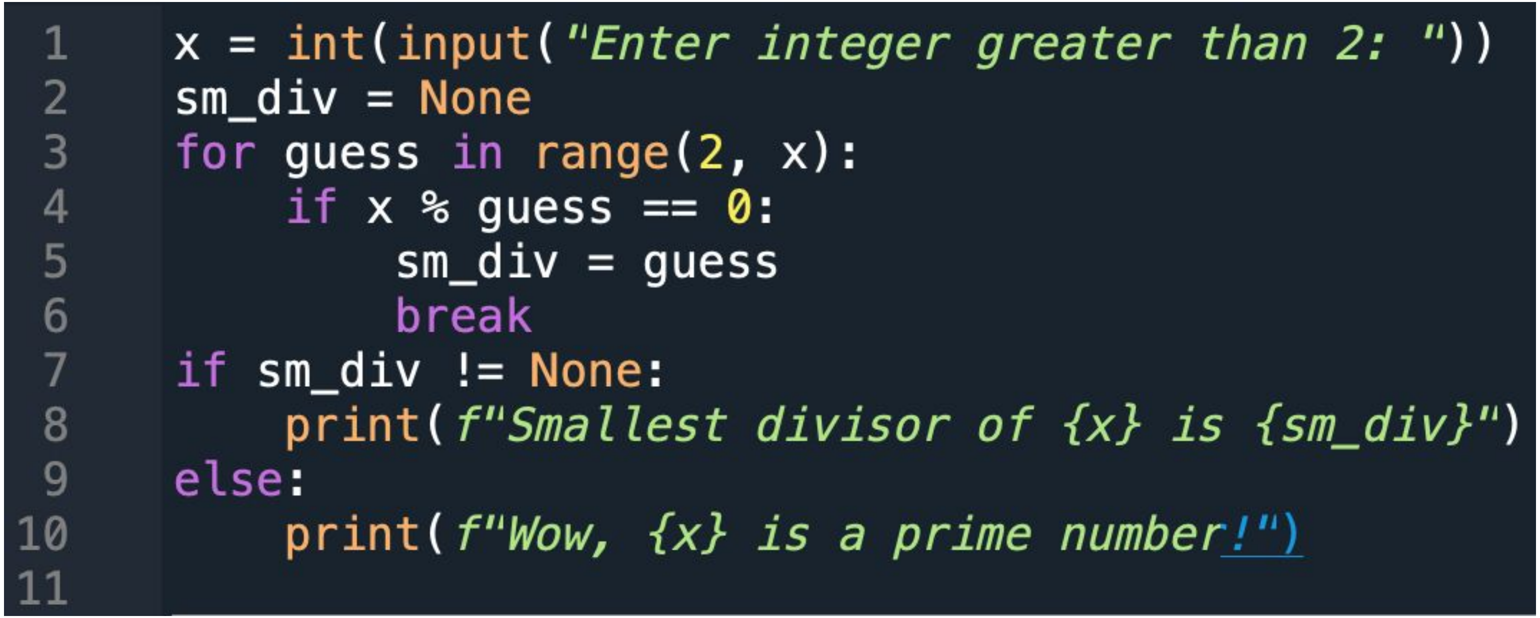
```
[ ]  a2 = 9
     a1 = 10
     answer = multiply_via_addition_while(a1, a2)
     print(answer)
```

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

Other Vocab

Point of Execution (pretend to be the python interpreter)

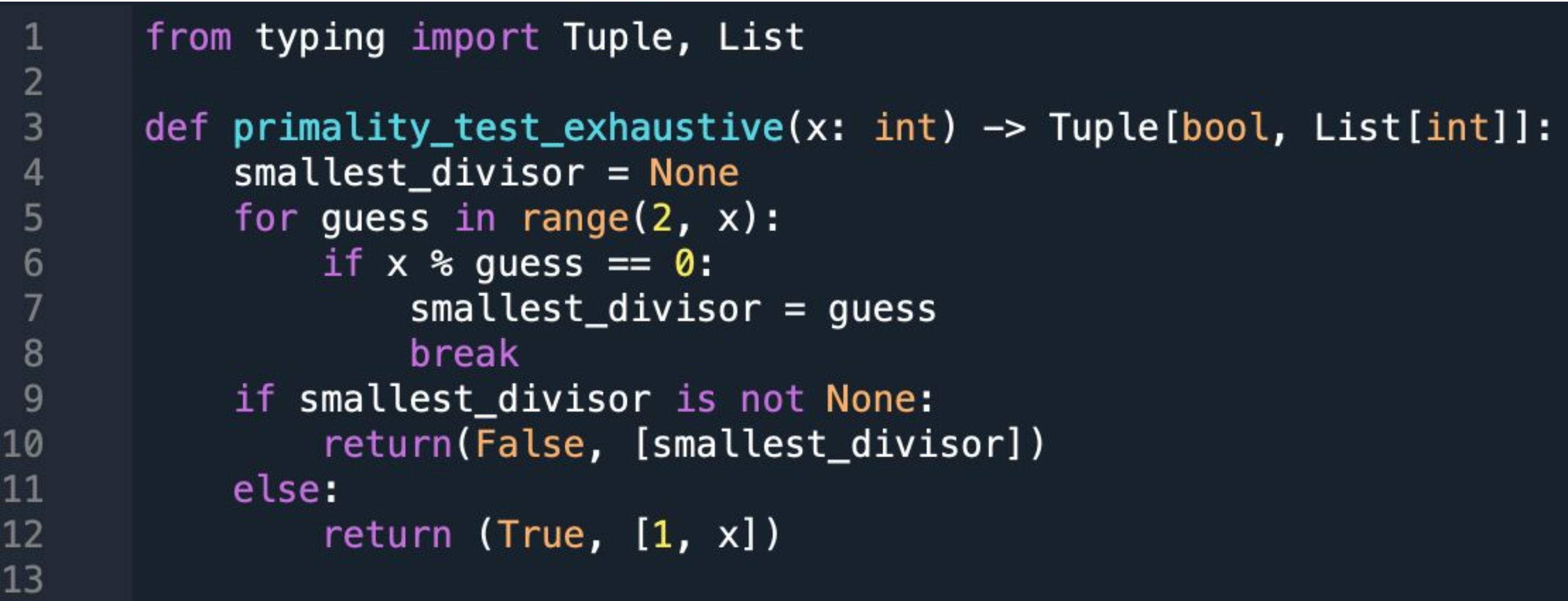
Group activity: Write out the line numbers that get executed in the following code:



```
1   x = int(input("Enter integer greater than 2: "))
2   sm_div = None
3   for guess in range(2, x):
4       if x % guess == 0:
5           sm_div = guess
6           break
7   if sm_div != None:
8       print(f"Smallest divisor of {x} is {sm_div}")
9   else:
10      print(f"Wow, {x} is a prime number!")
11
```

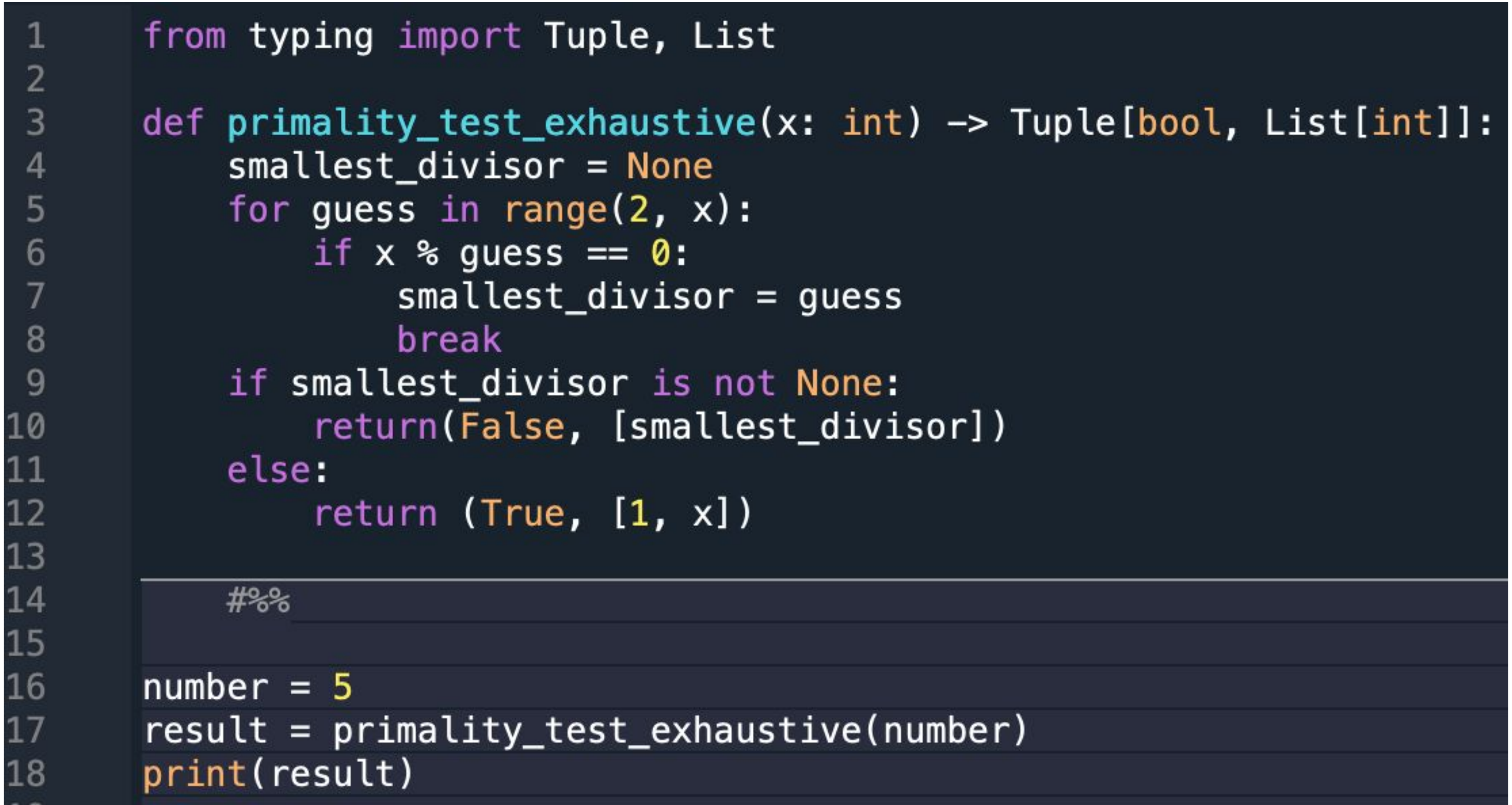
Point of Execution (pretend to be the python interpreter)

Group activity: Write out the line numbers that get executed in the following code:



```
1   from typing import Tuple, List
2
3   def primality_test_exhaustive(x: int) -> Tuple[bool, List[int]]:
4       smallest_divisor = None
5       for guess in range(2, x):
6           if x % guess == 0:
7               smallest_divisor = guess
8               break
9       if smallest_divisor is not None:
10          return(False, [smallest_divisor])
11      else:
12          return (True, [1, x])
13
```

Point of Execution (pretend to be the python interpreter)



```python
from typing import Tuple, List

def primality_test_exhaustive(x: int) -> Tuple[bool, List[int]]:
    smallest_divisor = None
    for guess in range(2, x):
        if x % guess == 0:
            smallest_divisor = guess
            break
    if smallest_divisor is not None:
        return(False, [smallest_divisor])
    else:
        return (True, [1, x])

    #%%

number = 5
result = primality_test_exhaustive(number)
print(result)
```

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!

Scope

global?

formal param?

actual param?

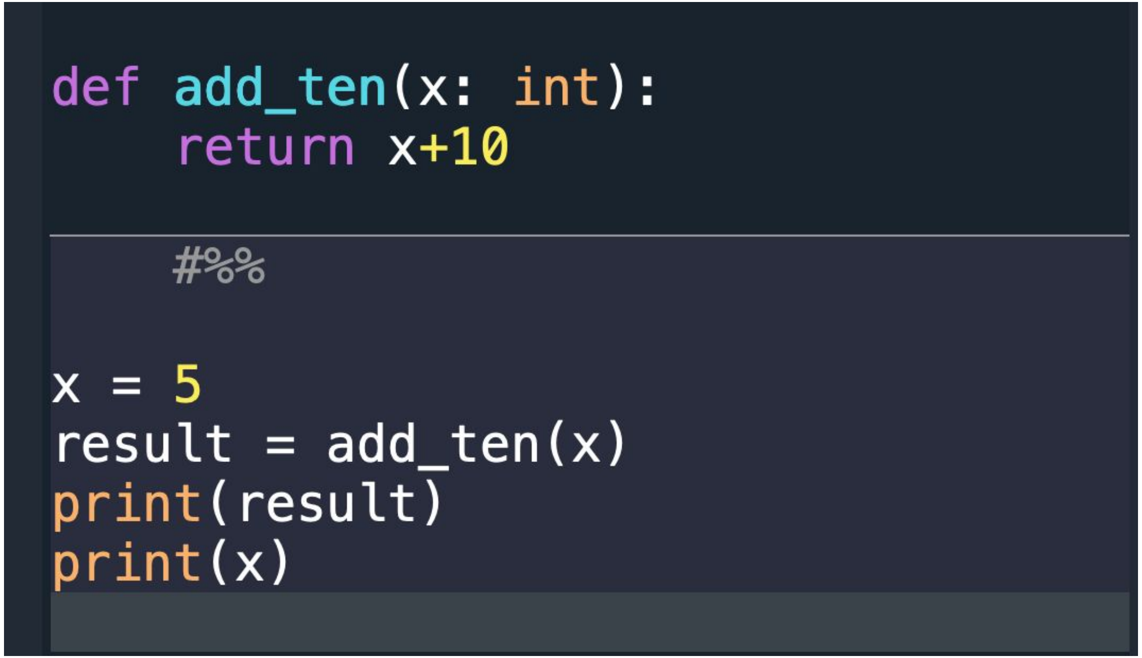
what prints?

globale x (5)

formal param: x in the function!

actual param: global x

"pass by assignment"



```
def add_ten(x: int):
    return x+10
```

```
    #%%

x = 5
result = add_ten(x)
print(result)
print(x)
```