Python

Fruitful Functions

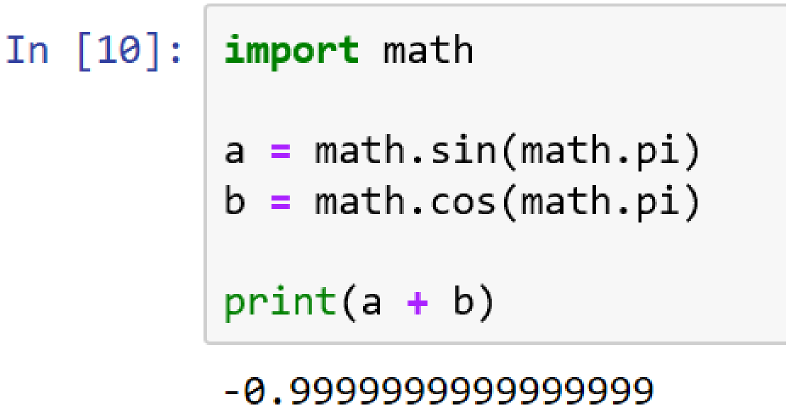
Learning Objectives

- Revisit functions and discuss return values
- Incremental development
- None as a value

Revisiting Functions

- **Recall:** Functions are named sequences of instructions which perform some action
  - Functions accept parameters in a parameter list and return values
  - A function call is the activation of a function
  - Python has many built in functions and many additional useful modules
  - Modules are collections of functions with similar purpose
    - Example: the math module with sin, cos, tan, etc.
- Functions are defined with the def keyword

Fruitful Functions

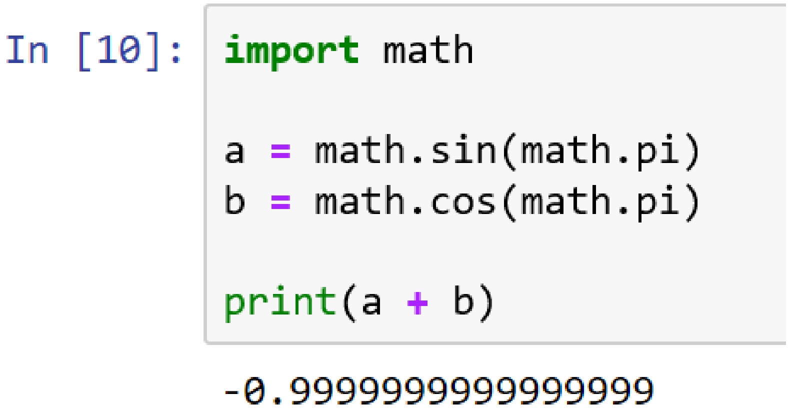


```
In [10]: import math

         a = math.sin(math.pi)
         b = math.cos(math.pi)

         print(a + b)

         -0.9999999999999999
```

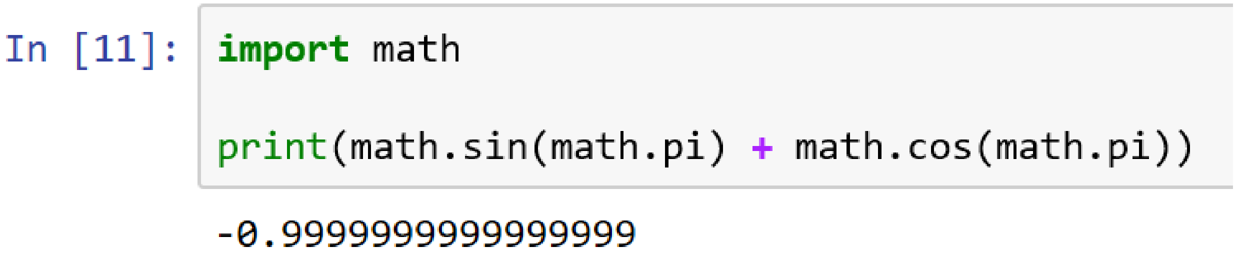
- A fruitful function is defined as a function which returns a value defined by the programmer
- The value a function returns is simply called the return value
- Some examples
  - ex: math.sin(angle) and math.cos(angle) are both fruitful functions

Fruitful Functions

- A fruitful function is defined as a function which returns a value defined by the programmer
- The value a function returns is simply called the return value
- Some examples
  - ex: math.sin(angle) and math.cos(angle) are both fruitful functions
  - **Recall**: one can use a function call anywhere the return value can be used



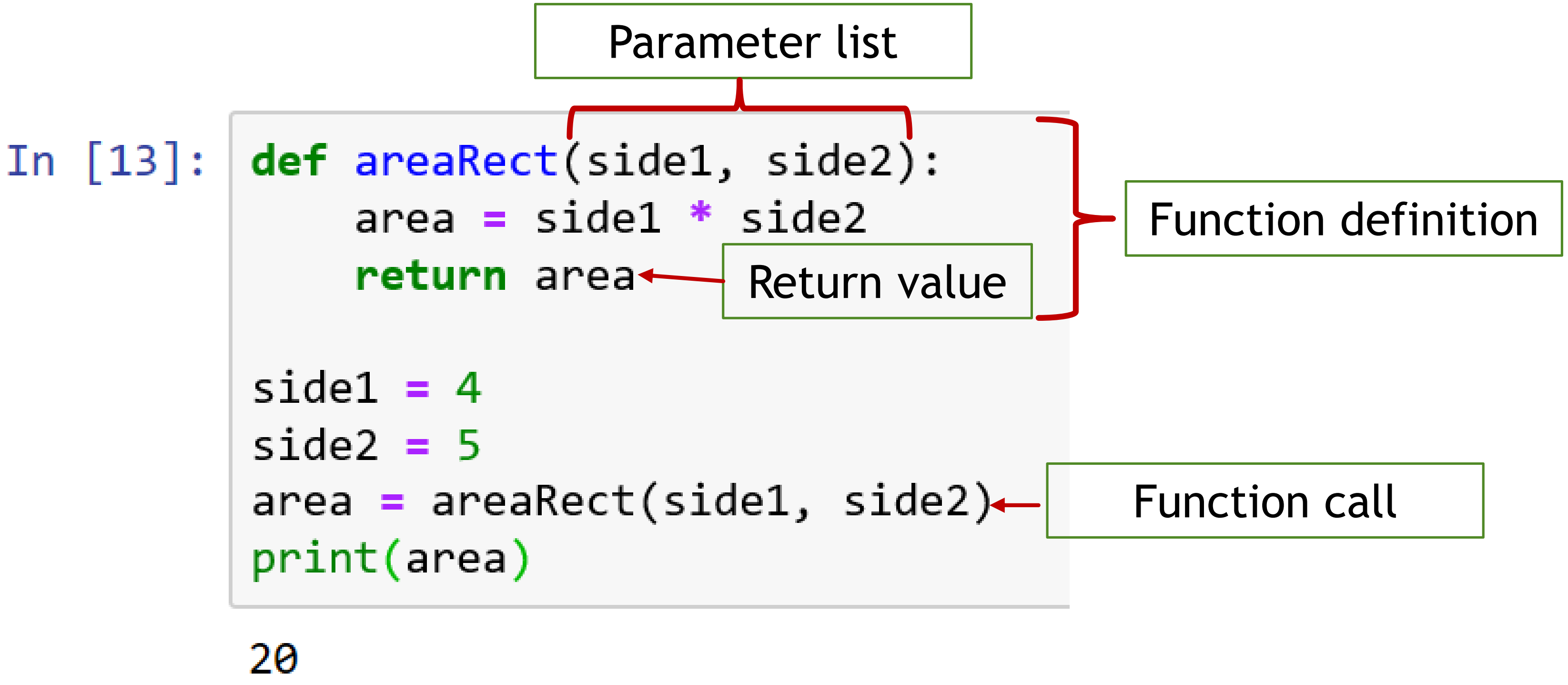
```
In [10]: import math

         a = math.sin(math.pi)
         b = math.cos(math.pi)

         print(a + b)

-0.9999999999999999
```



```
In [11]: import math

         print(math.sin(math.pi) + math.cos(math.pi))

-0.9999999999999999
```

Defining Fruitful Functions

- A return value is identified using the return keyword



```
In [13]: def areaRect(side1, side2):
             area = side1 * side2
             return area

         side1 = 4
         side2 = 5
         area = areaRect(side1, side2)
         print(area)

         20
```

Parameter list

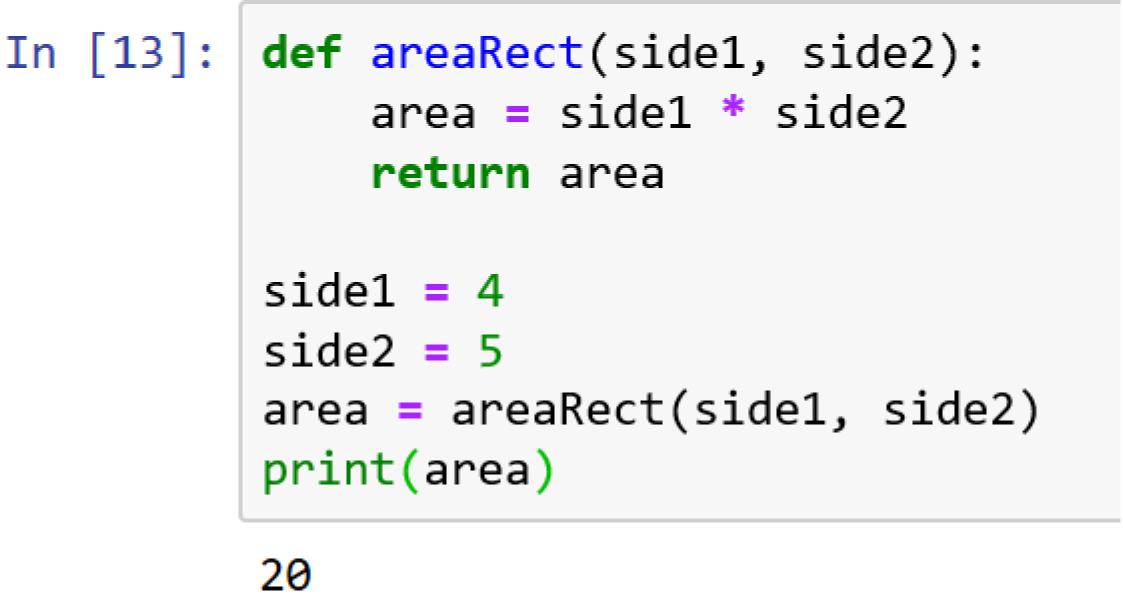
Function definition

Return value

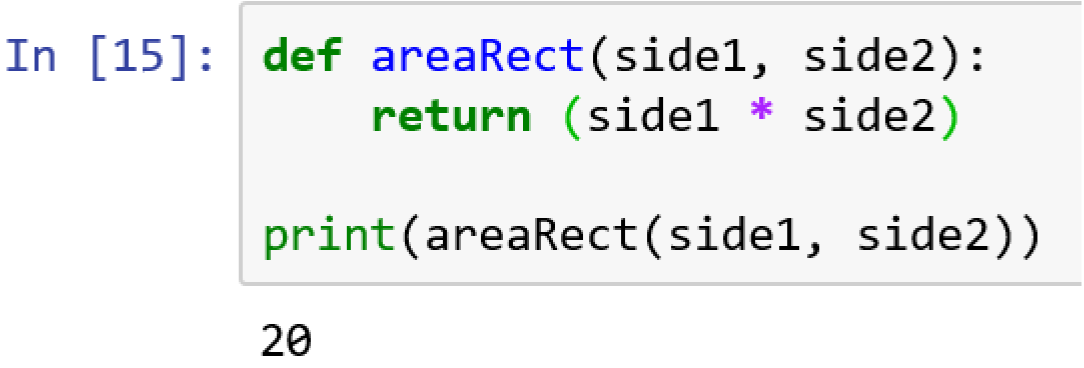
Function call

Refactoring

- Refactoring is the process of restructuring some set of code without changing its function



```
In [13]: def areaRect(side1, side2):
             area = side1 * side2
             return area

         side1 = 4
         side2 = 5
         area = areaRect(side1, side2)
         print(area)

         20
```

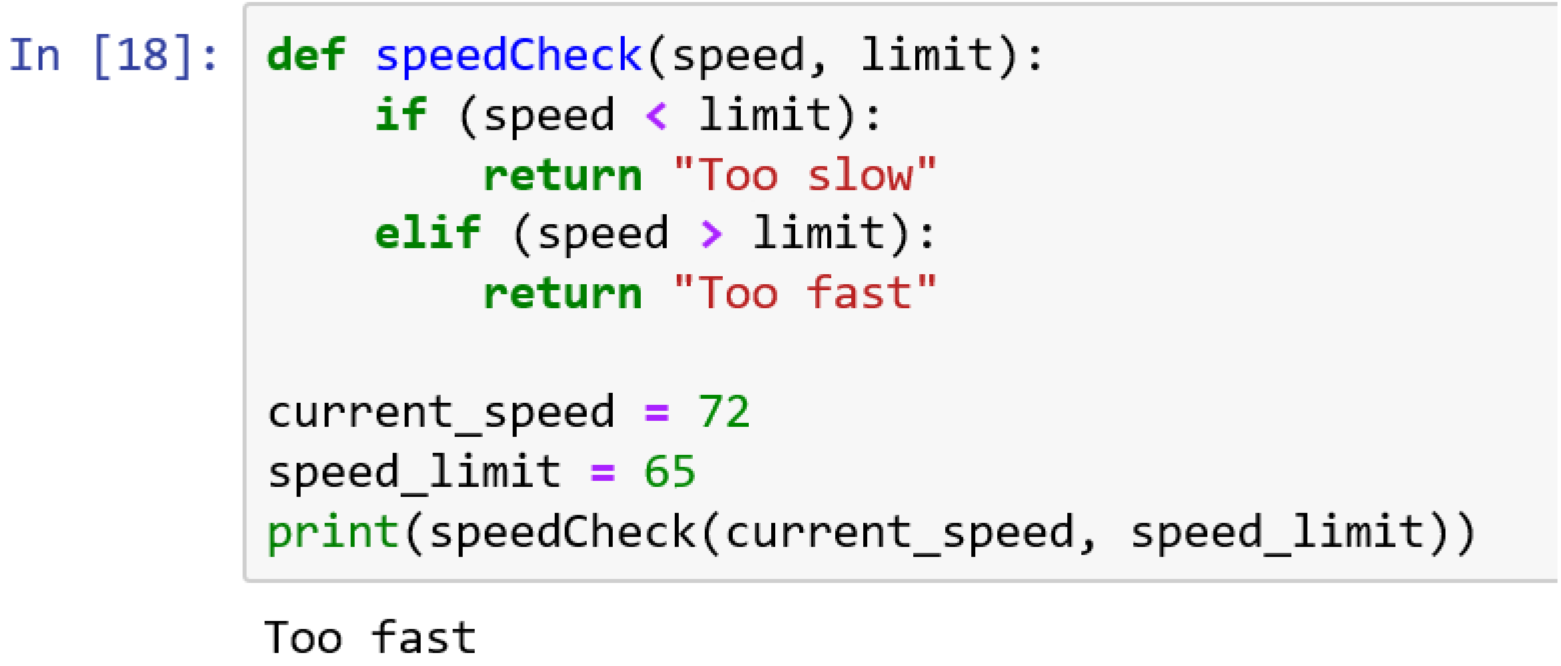


```
In [15]: def areaRect(side1, side2):
             return (side1 * side2)

         print(areaRect(side1, side2))

         20
```

- In this example, I’ve refactored both the areaRect function and __main__. Which is superior?

Return values

- Multiple return statements are allowed, though the first return executed ends the function and returns the return value



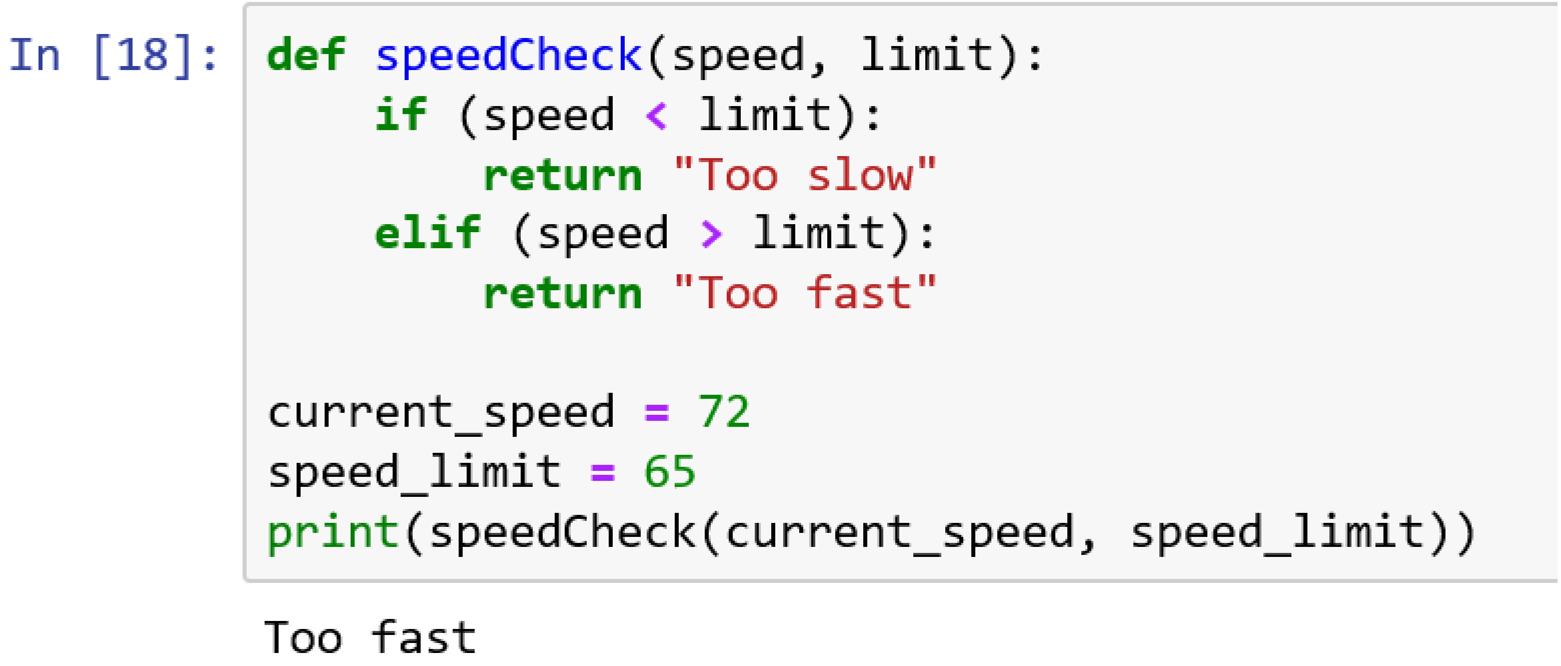
In [18]:

```
def speedCheck(speed, limit):
    if (speed < limit):
        return "Too slow"
    elif (speed > limit):
        return "Too fast"

current_speed = 72
speed_limit = 65
print(speedCheck(current_speed, speed_limit))
```

```
Too fast
```

Return values

- Multiple return statements are allowed, though the first return executed ends the function and returns the return value



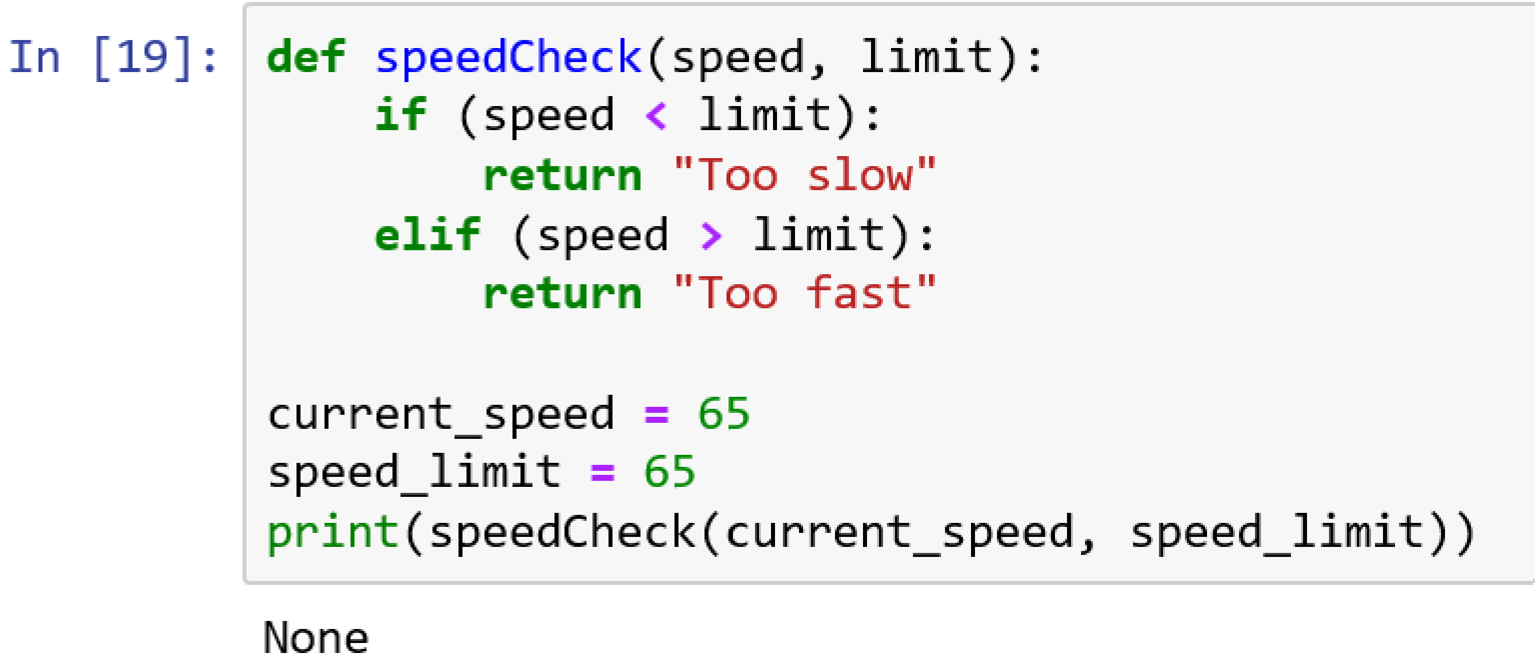
```
def speedCheck(speed, limit):
    if (speed < limit):
        return "Too slow"
    elif (speed > limit):
        return "Too fast"

current_speed = 72
speed_limit = 65
print(speedCheck(current_speed, speed_limit))
```

```
Too fast
```

- What’s the return value if the current speed == the speed limit?

Return values

- Multiple return statements are allowed, though the first return executed ends the function and returns the return value



In [19]:

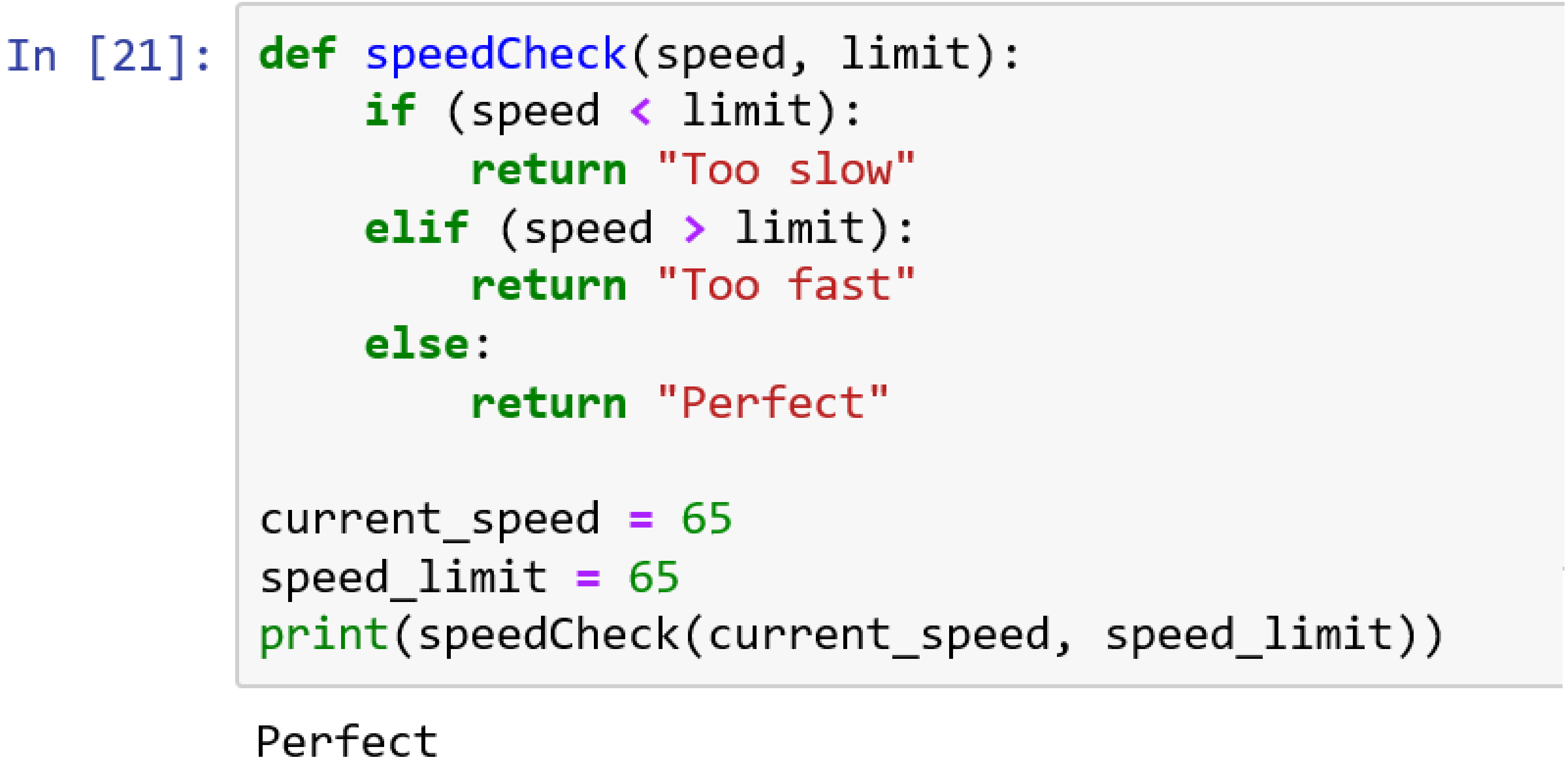
```
def speedCheck(speed, limit):
    if (speed < limit):
        return "Too slow"
    elif (speed > limit):
        return "Too fast"

current_speed = 65
speed_limit = 65
print(speedCheck(current_speed, speed_limit))
```

```
None
```

- None is the default return value. All void functions actually have a return value: None

Return values

- All branches in a function should return a value



In [21]:

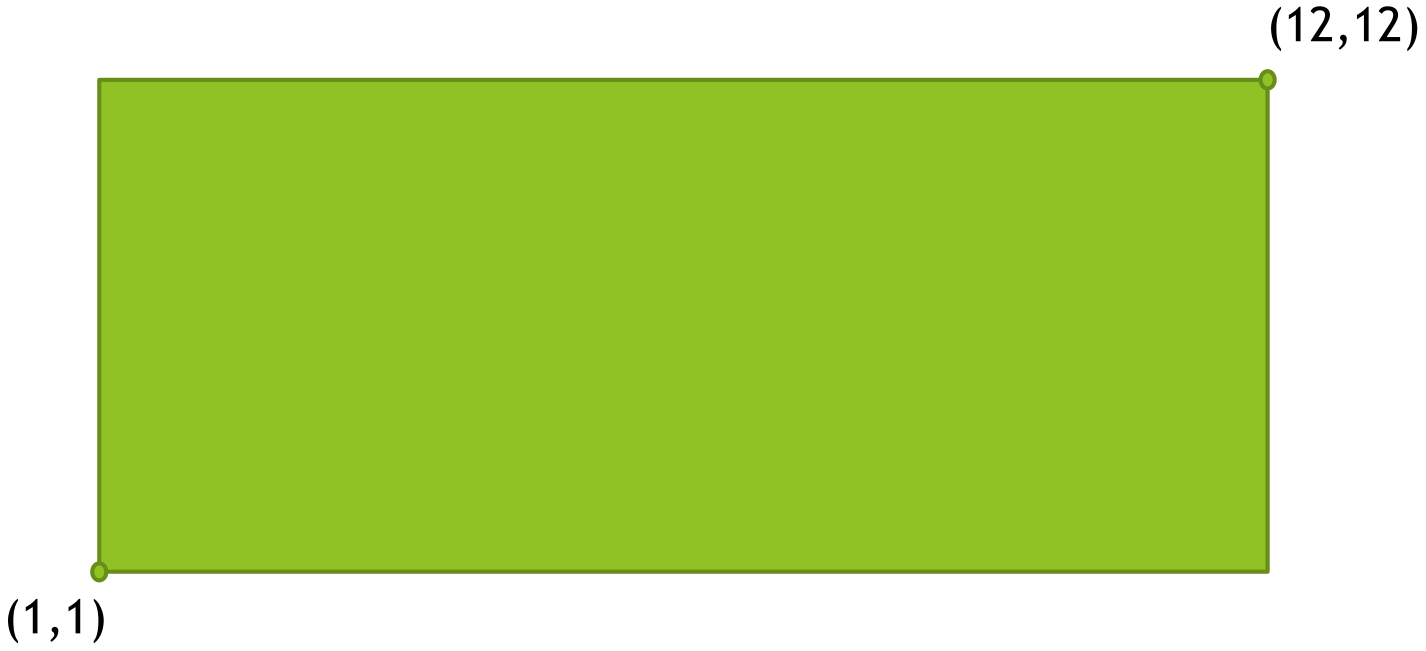
```
def speedCheck(speed, limit):
    if (speed < limit):
        return "Too slow"
    elif (speed > limit):
        return "Too fast"
    else:
        return "Perfect"

current_speed = 65
speed_limit = 65
print(speedCheck(current_speed, speed_limit))
```

```
Perfect
```

a

Composition

- **Recall:** a function can be called from within another function
- **Problem:** find area of a rectangle, given coordinates of opposite corners



Algorithm

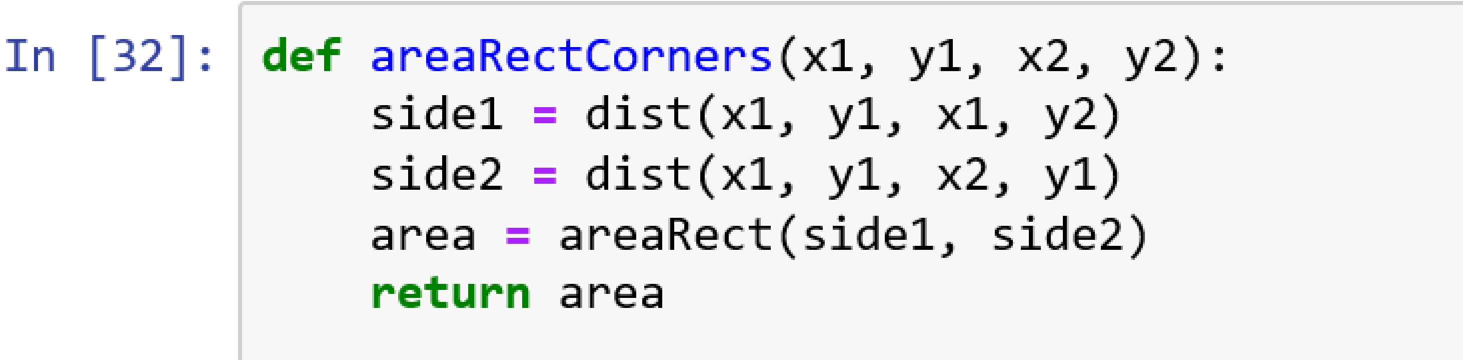
- **Recall**: An algorithm is an ordered set of instructions defining some process
- What is the algorithm necessary to find the area of a rectangle, given the points of the corners?

Algorithm

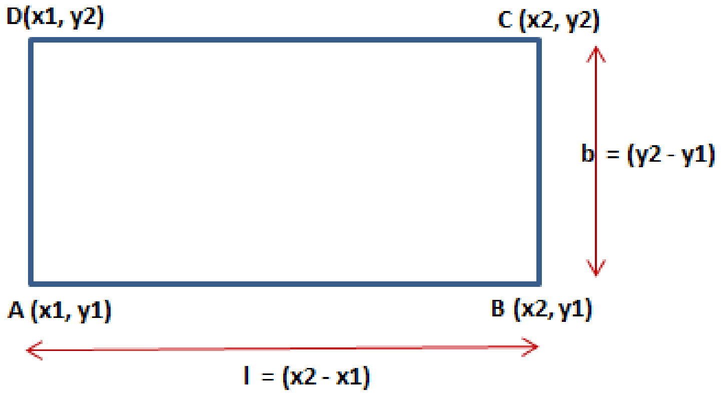
- **Recall**: An algorithm is an ordered set of instructions defining some process
- What is the algorithm necessary to find the area of a rectangle, given the points of the corners?
  1. Obtain the two points
  2. Calculate the length of each side
  3. Multiply the lengths of the two sides together to obtain the area
  4. Return the area

Algorithm

- **Recall:** An algorithm is an ordered set of instructions defining some process
- What is the algorithm necessary to find the area of a rectangle, given the points of the corners?
    1. Obtain the two points
    2. Calculate the length of each side
    3. Multiply the lengths of the two sides together to obtain the area
    4. Return the area

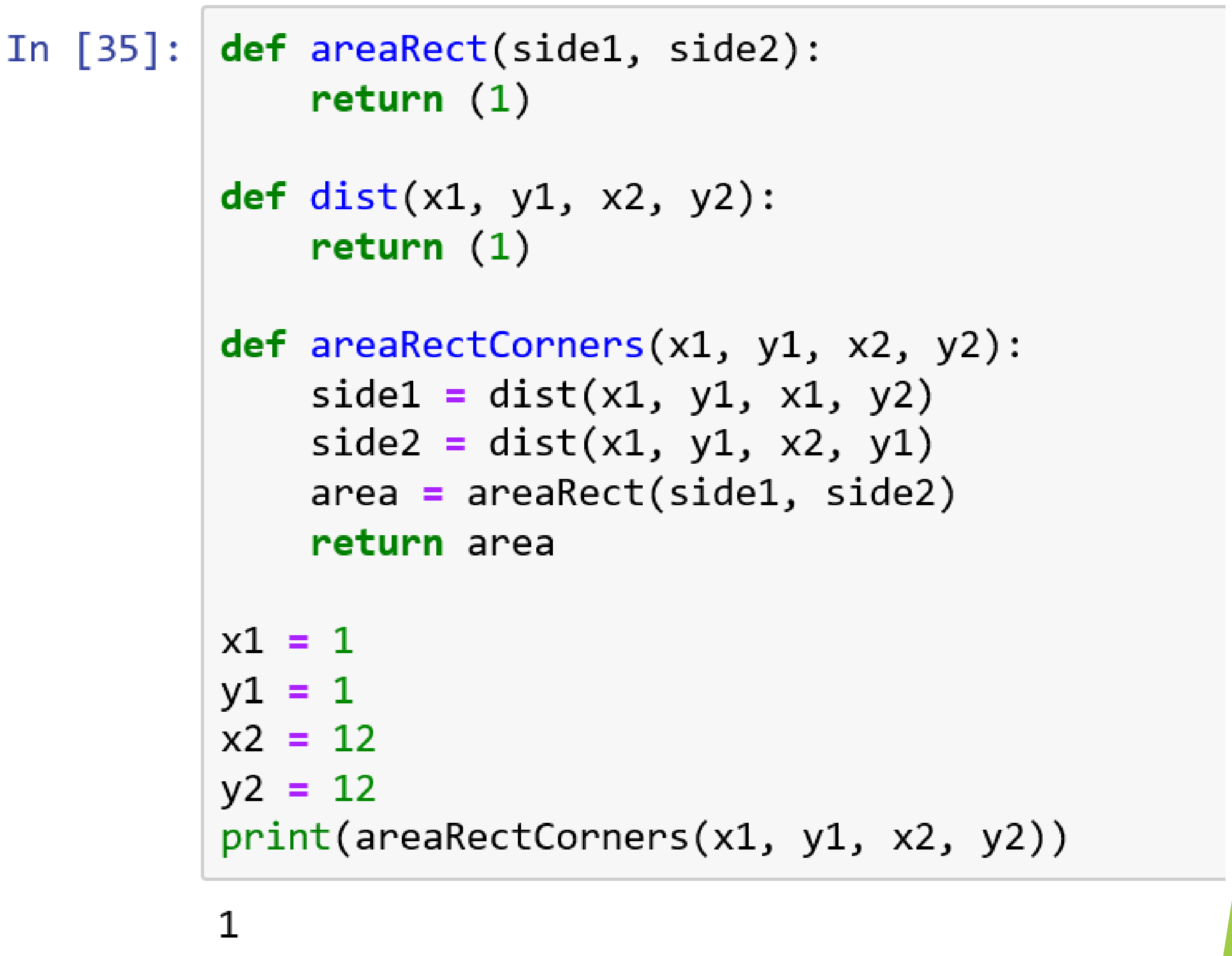


```
In [32]: def areaRectCorners(x1, y1, x2, y2):
             side1 = dist(x1, y1, x1, y2)
             side2 = dist(x1, y1, x2, y1)
             area = areaRect(side1, side2)
             return area
```



Incremental Development

- Incremental development is the process of developing a program in small chunks (increments)
- Stub functions are functions which only implement the interfaces (parameter lists and return values) to allow for incremental development
- Note how areaRect and dist do not do anything, but they do accept the proper values and the do return a value of the proper type

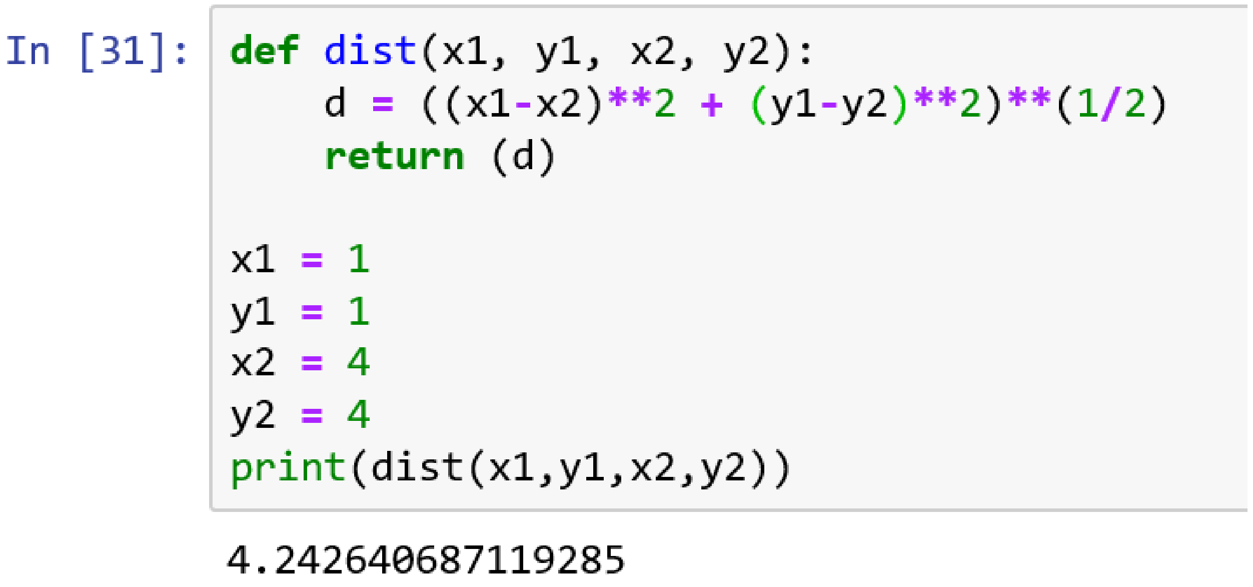


```
In [35]: def areaRect(side1, side2):
             return (1)

         def dist(x1, y1, x2, y2):
             return (1)

         def areaRectCorners(x1, y1, x2, y2):
             side1 = dist(x1, y1, x1, y2)
             side2 = dist(x1, y1, x2, y1)
             area = areaRect(side1, side2)
             return area

         x1 = 1
         y1 = 1
         x2 = 12
         y2 = 12
         print(areaRectCorners(x1, y1, x2, y2))

         1
```

Algorithm

- The distance between two points uses the distance formula

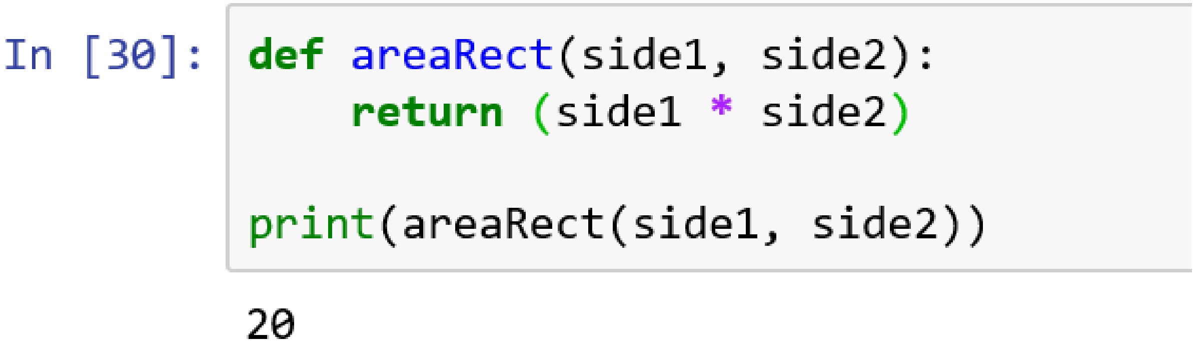
$$distance = \sqrt{(X_1 - X_2)^2 - (Y_1 - Y_2)^2}$$



```
In [31]: def dist(x1, y1, x2, y2):
             d = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
             return (d)

         x1 = 1
         y1 = 1
         x2 = 4
         y2 = 4
         print(dist(x1,y1,x2,y2))

         4.242640687119285
```

Algorithm

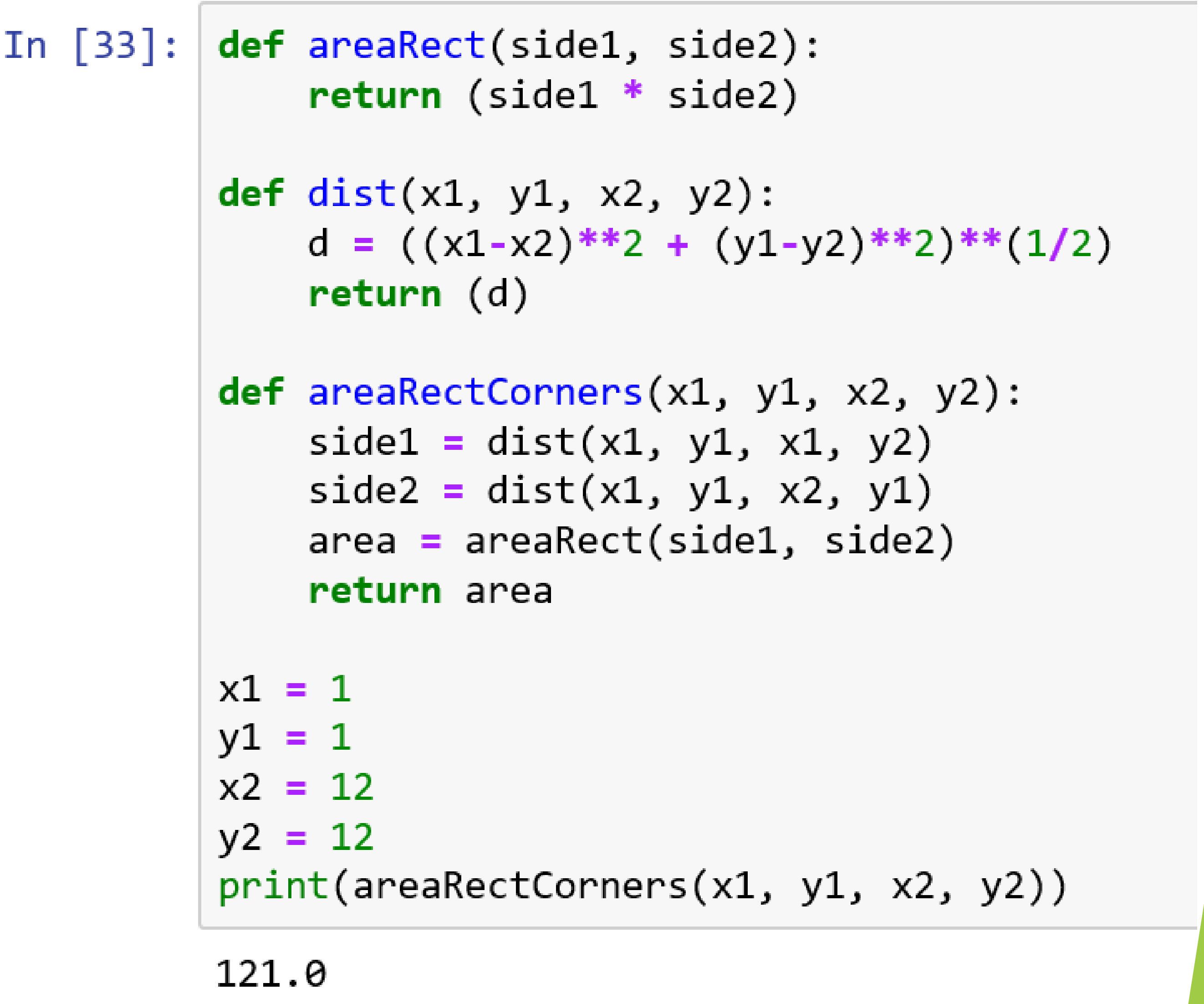
- Area of a rectangle is height * width
- Note how each function is being tested independently



```
In [30]: def areaRect(side1, side2):
             return (side1 * side2)

         print(areaRect(side1, side2))

         20
```

Incremental Development

- By building up the program in increments we can test each function separately
- This allows us to focus on one part at a time
- Get one thing working before moving on to the next



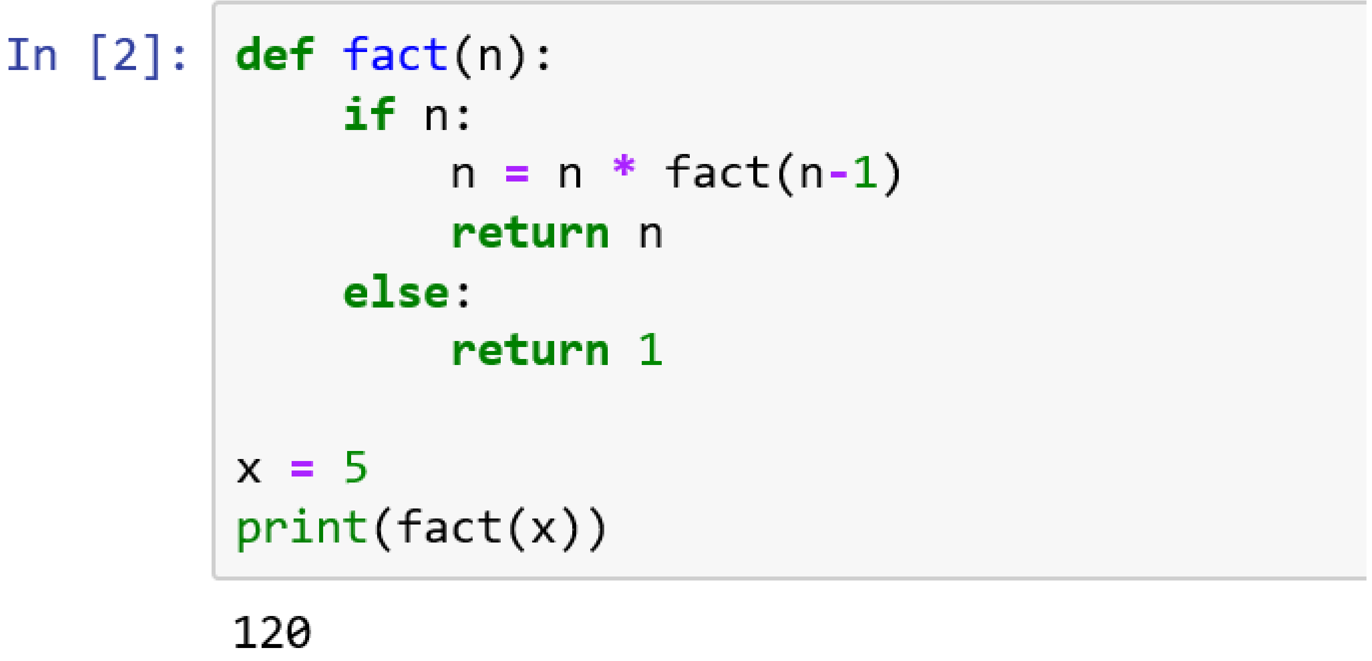
```
In [33]: def areaRect(side1, side2):
             return (side1 * side2)

         def dist(x1, y1, x2, y2):
             d = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
             return (d)

         def areaRectCorners(x1, y1, x2, y2):
             side1 = dist(x1, y1, x1, y2)
             side2 = dist(x1, y1, x2, y1)
             area = areaRect(side1, side2)
             return area

         x1 = 1
         y1 = 1
         x2 = 12
         y2 = 12
         print(areaRectCorners(x1, y1, x2, y2))

         121.0
```

Recursion Revisited

- Recursion becomes useful, once each call can return values to the previous call
- What's the general algorithm to calculate a factorial
  - n == 0? Return 1
  - otherwise return n * factorial(n-1)

Recursion Revisited

- Recursion becomes useful, once each call can return values to the previous call
- What’s the general algorithm to calculate a factorial
  - n == 0? Return 1
  - otherwise return n * factorial(n-1)
- How good is this?
  - What is fact(1.5)?
  - What is fact(-1)



```
In [2]: def fact(n):
            if n:
                n = n * fact(n-1)
                return n
            else:
                return 1

        x = 5
        print(fact(x))

        120
```

Resources

- Bryan Burlingame's notes
- Downey, A. (2016) *Think Python, Second Edition* Sebastopol, CA: O'Reilly Media
- (n.d.). 3.7.0 Documentation. *6. Expressions — Python 3.7.0 documentation.* Retrieved September 11, 2018, from http://docs.python.org/3.7/reference/expressions.html