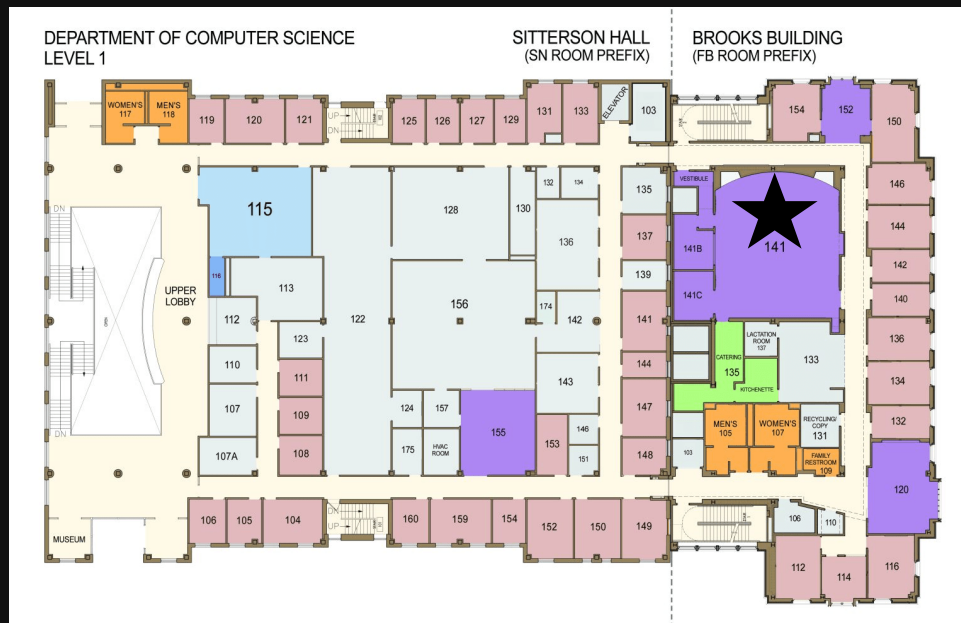




CL03: Intro to Functions

Announcements

- EX00 deadline has been extended to tomorrow at 11:59pm! Please visit Office Hours (Sitterson Hall (SN) 008) for help!
 - The [CSXL website](#) is ready for you! Log in to see our hours, or submit a ticket for help if you're in Sitterson Hall
- Today's tutoring (longer-form help) moved to Fred Brooks 141 from 6-7pm
 - Wednesday's typical hours are from 5-7pm in SN 011



Functions by Intuition

Consider the following **function definition** (a new concept!):



```
1 def celsius_to_fahrenheit(degrees: int) -> float:  
2     """Convert degrees Celsius to degrees Fahrenheit."""  
3     return (degrees * 9 / 5) + 32
```

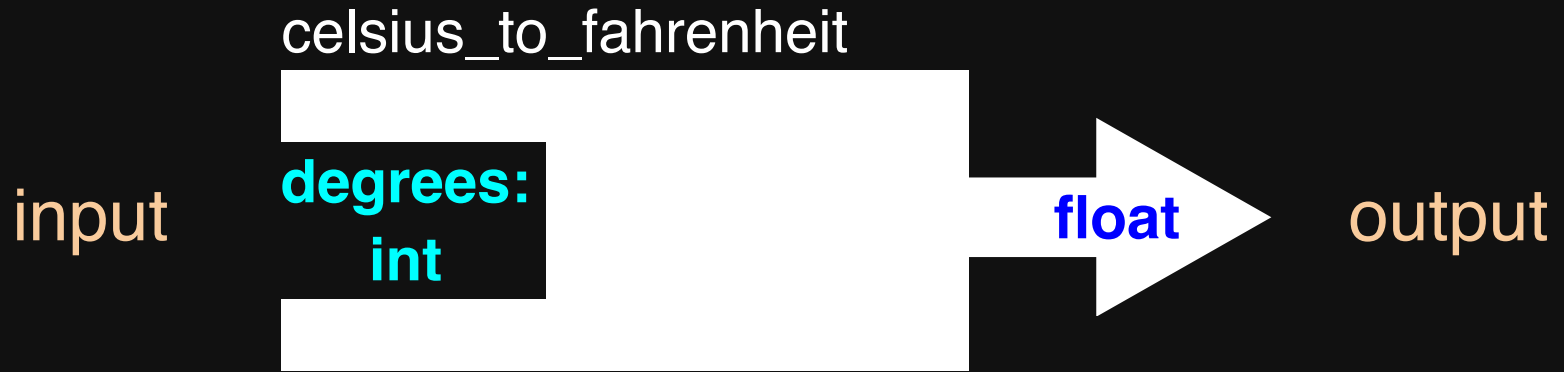
Now, consider the following **function call expressions**, which use the definition:



```
1 celsius_to_fahrenheit(degrees=0)  
2  
3 celsius_to_fahrenheit(degrees=10)
```

What **value** and **type** does each function call expression evaluate to? How many connections between the **definition** and the **call** can you identify intuitively?

The fundamental pattern of functions



```
1 celsius_to_fahrenheit(degrees=10)
```



50.0

Function definitions are like recipes

- A **recipe** in a book *does not result in a meal until you cook it*
- A **function definition** in your program *does not result in a value until you call it*
- An **adaptable recipe** is one where you can substitute ingredients, follow the same steps, and get different, but intentional, results
- A **parameterized function definition** is one where you can substitute input *arguments*, follow the same steps, and get different, but intentional, results.
 - Such as converting Celsius degree values to Fahrenheit!
- **Recipes** and **function definitions** are written down once with dreams of being cooked (called) tens, hundreds, thousands, ... billions of times over!

The anatomy of a function definition

define... a function with this which will return
named this... parameter list a value of this
of "inputs" return type

```
def name_of_function(parameter: type) -> returnType:
```

```
    """Docstring description of the function."""
```

```
    return expression_of_type_returnType
```

function signature specifies how you and others will make use of the function from elsewhere in a program

- What is its **name**?
- What input **parameter(s) type(s)** does it need?
 - (Think: ingredients)
- What **type of return value** will calling it result in?
 - (Think: meal)

The anatomy of a function definition

define... a function with this which will return
named this... parameter list a value of this
of "inputs" return type

↓ ↓ ↓ ↓

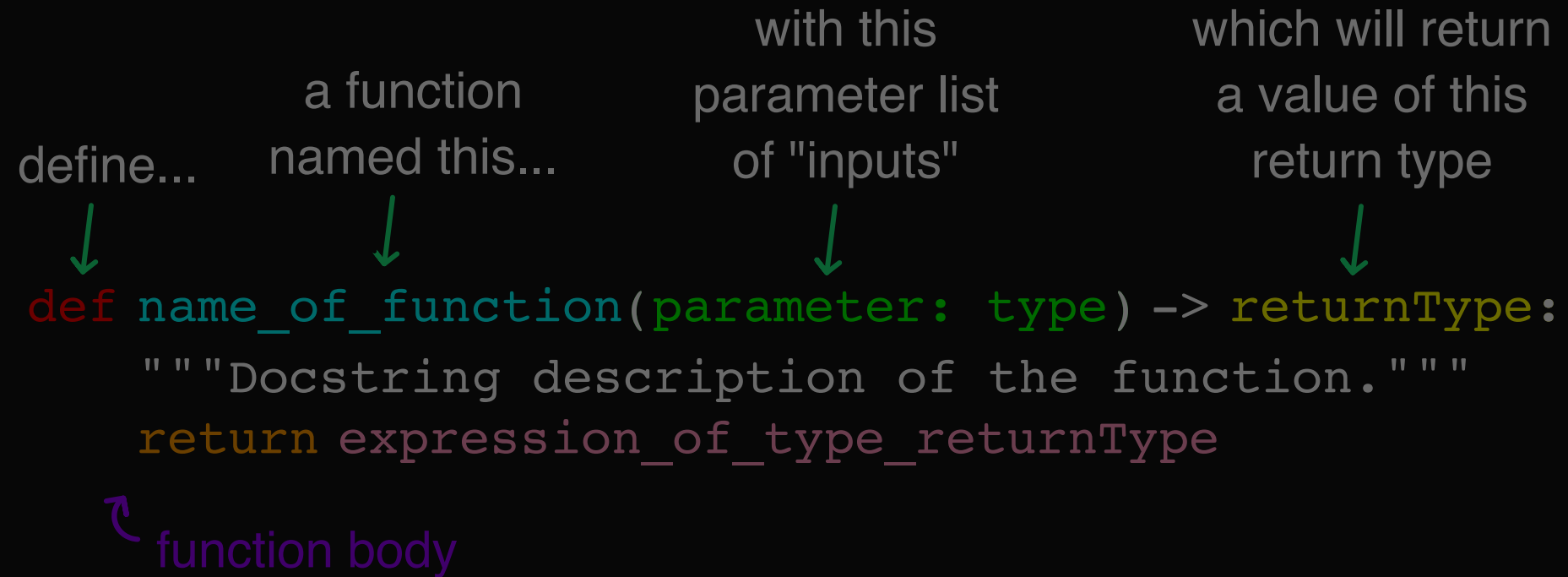
```
def name of function(parameter: type) -> returnType:
```

```
    """Docstring description of the function."""  
    return expression_of_type_returnType
```

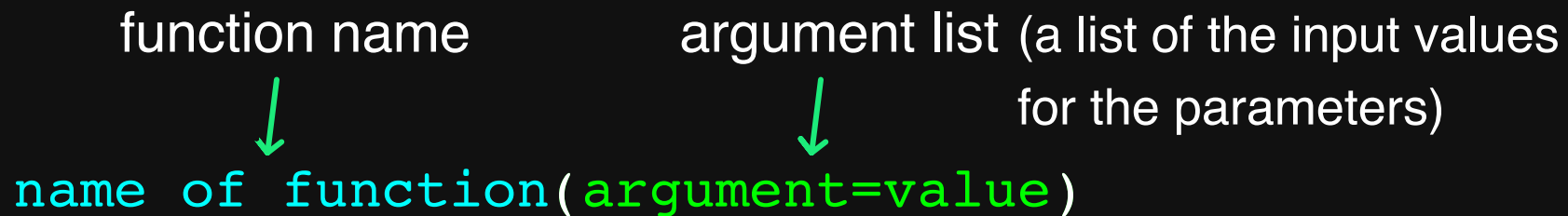
↻ **function body** specifies the subprogram, or set of steps, which will be carried out every time a function calls the definition:

- Each statement in the body is **indented** by (at least) one level
- The **Docstring** describes the purpose and, often, usage of a function *for people*
- The function body contains one or more **statements**. For now, our definitions will be simple, one-statement functions
- **Return statements** are special and written inside of function definitions
 - When a function definition is called, a return statement indicates, "**stop following this function here and send my caller the result of evaluating this return expression!**"

The anatomy of a function definition



The anatomy of a function call



Fill in the blank to complete the missing expression

Say you want to hang string lights around your dorm room. How long of a strand of string lights will you need?

```
1 def perimeter(length: float, width: float) -> float:
2     """Calculate the perimeter of a rectangle."""
3     return _____
```

This is an example function call expression that calls the perimeter function definition above. What value and type will this expression evaluate to?

```
1 perimeter(length=10.0, width=8.0)
```