Introduction to Functions

Functions by Intuition...

Consider the following **Function Definition**, which is a new concept to you...

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

Now consider the following **Function Call Expressions**, which use the definition...

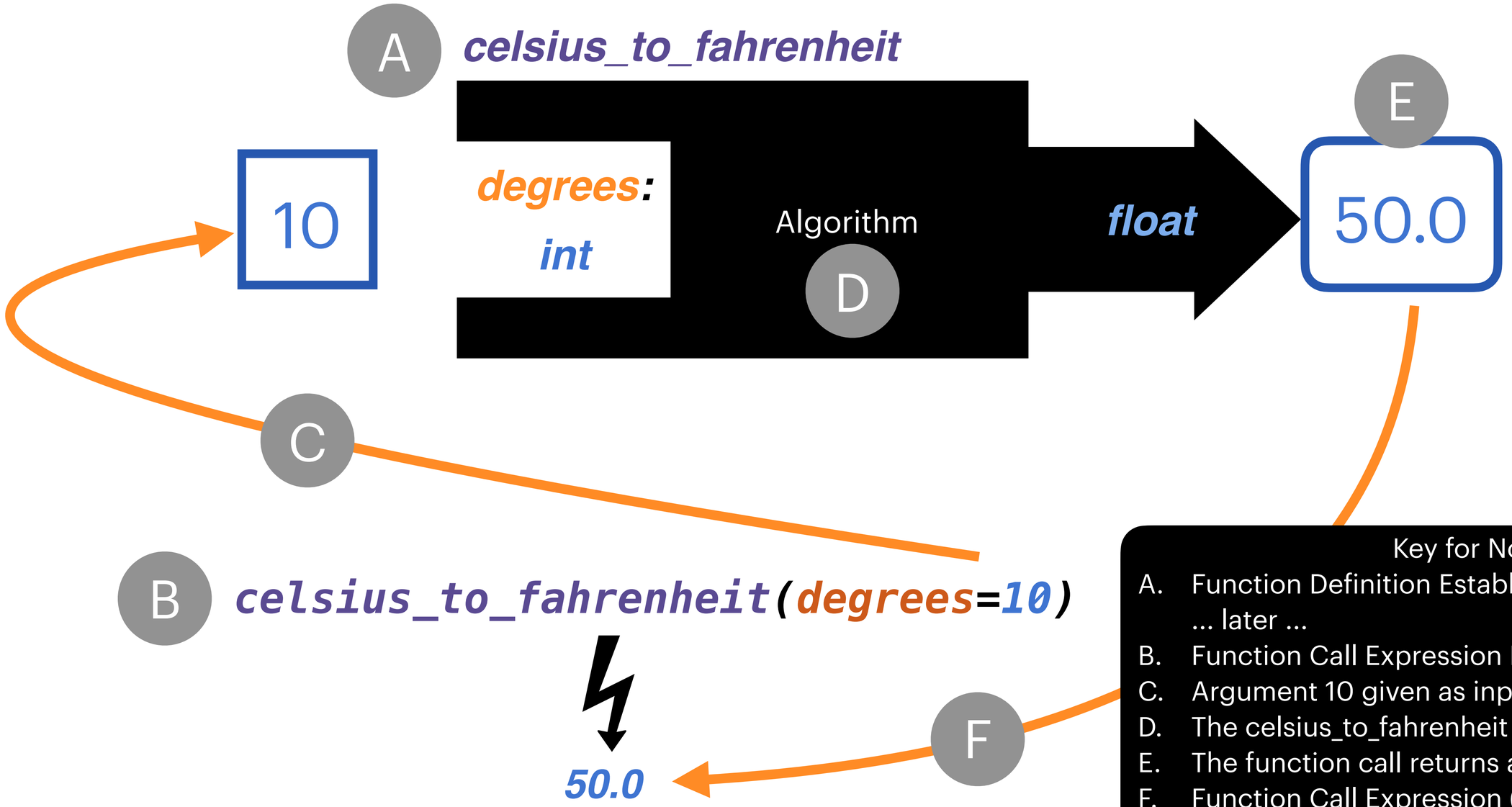
```
celsius_to_fahrenheit(degrees=0)
```

```
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?

Follow-along in VSCode

Functions and the Fundamental Pattern



Key for Notes

A. Function Definition Established
... later ...
B. Function Call Expression Evaluation Begins
C. Argument 10 given as input to degrees parameter
D. The celsius_to_fahrenheit "algorithm" evaluates
E. The function call returns a value of 50.0
F. Function Call Expression (B) results in a float value of 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 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. Such as blueberry biscuits, cinnamon biscuits, sage biscuits, and so on.
- 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 different Celsius degree values to Fahrenheit degree values.
- **Recipes** and **function definitions** are written down once with dreams of being cooked and called tens, hundreds, thousands, ... billions of times over!

The Anatomy of a Function Definition

```
def name_of_function(parameter: type) -> returnType:
    """Docstring description of function for people"""
    return expression_of_type_returnType
```

Function Definition *Signature*

```
def name_of_function(parameter: type) -> returnType:
    """Docstring description of function for people"""
    return expression_of_type_returnType
```

The **signature** of a function definition 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: biscuits)*

Function Definition *Body* or *Implementation*

```
def name_of_function(parameter: type) -> returnType:
    """Docstring description of function for people"""
    return expression_of_type_returnType
```

The **body** or implementation a function definition 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 one-level to visually denote it.

The **Docstring** describes the purpose and, often, usage of a function *for people*

The function body then 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 right here and send my caller the result** of evaluating this return expression!"