Types, Objects, and Functions, Oh My!

Today is a Paper + Pencil or Tablet + Pencil day...
please keep laptops stowed away!

COMP110 - CL01

2024/01/16

Office Hours

Drop-in appointments queued via https://Course.Care - Enroll Code: 48A3A6

- All Office Hours usage is through the Course.Care via the enroll code above

- To use office hours:

1. **Come to the Sitterson Lobby and take a seat in the open Lobby Area.**
2. Open Course.Care and Create an Appointment Request. ***Put effort into the questions asked!***
3. The **TAs will call tickets 1-by-1, in-order**. You will go into the drop-in meeting.
4. **Meetings are 15-minutes long** to ensure fairness to other students.
5. There is an **hour-long wait between meetings** and **you are expected to make an hour of progress**.

Why? **Your success in this course depends on your individual understanding and mastery of the material!**

1. Review - Basic Types

Jot down responses to these four questions and discuss with your neighbor.

1. What is the difference between `int` and `float`?

2. Is there a difference between `"True"` and `True`? What **type** of **literal** is each an example of?

3. What is the difference between `1 + 1` and `"1" + "1"`? What is the resulting *value* and *type* of each?

4. What role do **types** play for data in Python?

2. Review - `str` is a *Sequence* Type

Jot down responses to these four questions and discuss with your neighbor.

5. What does `len` function evaluate to when applied to a `str` value?
   What will the expression `len("owl")` evaluate to?

6. Is there a difference between `"True"` and `'True'`? What **type** of **literal** is each an example of?

7. What are the **square brackets** called in the following *expression*? What does the following expression evaluate to? `"BEAR"[3]`

8. Can a string be a number in Python? Explain.

3. Review - Operators and Expressions

Jot down responses to these four questions and discuss with your neighbor.

9. What is the result of evaluating `10 % 3`? What about `10 // 3`? What about `10 ** 3`?

10. Is there an error in the expression `"CAMP" + 110`? If so, how would you fix it such that the `+` symbol is evaluated to be **concatenation**?

11. What is the evaluation of the expression `10 / 4`? What types are the *operands* (10 and 4), what type does the expression evaluate to?

12. What is the evaluation of the expression `2 - 6 / 3 + 4 * 5`?

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
```

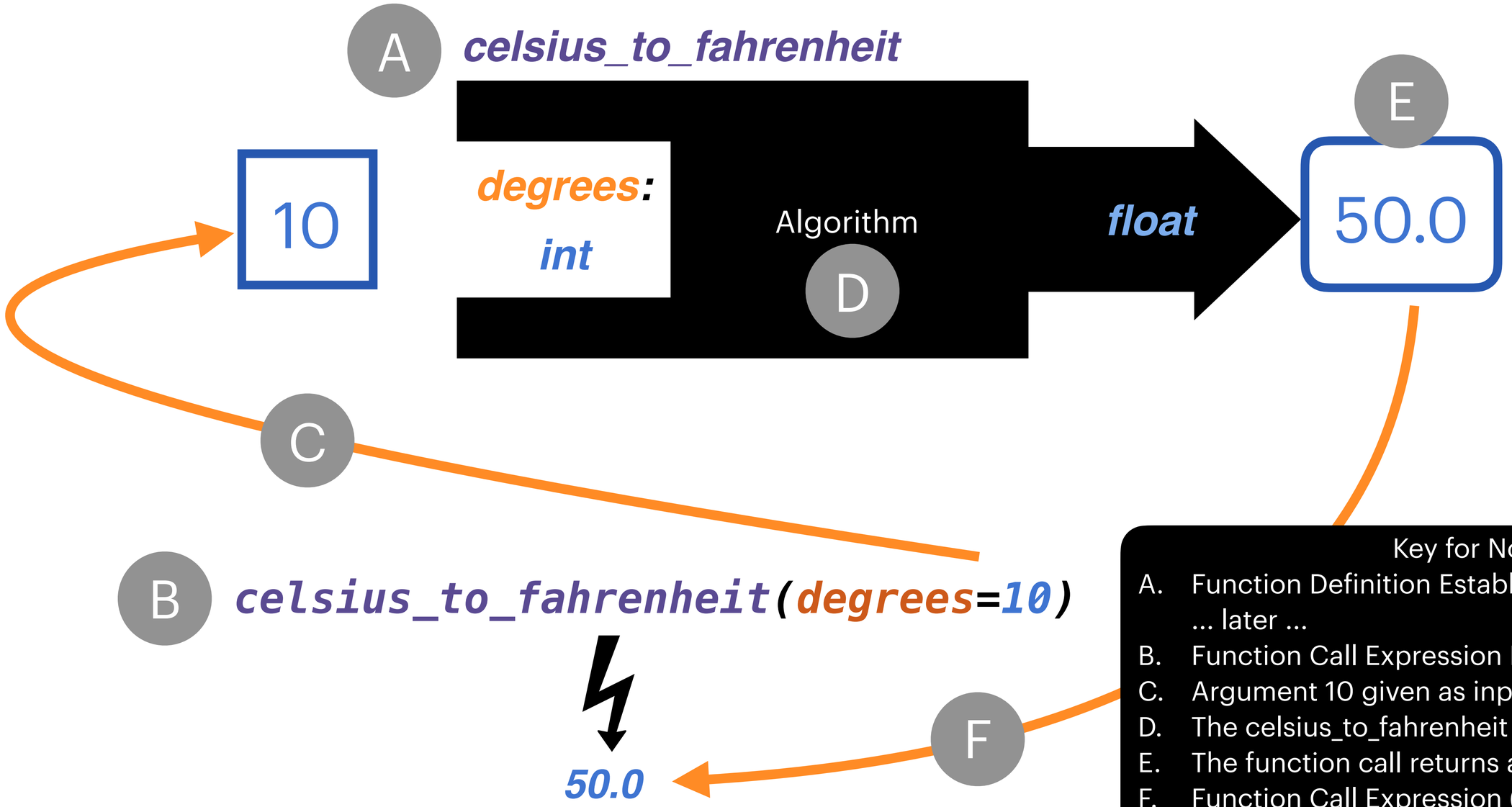
Now consider the following **Function Call Expressions**, which use the definition...

```
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?

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
```

This will be the CL01 in-class submission...

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!"

Submitting CL01 to Gradescope for Participation

- Decide one of the two of you (or three...) to be the SUBMITTER

- From the SUBMITTER's cell phone:

1. Open CL01 on Gradescope and make a submission

2. Upload a wide angle selfie-photo of your pair or group of 3 holding your *anatomy of a function* notes and giving a cozy thumbs up

3. Make your submission, **then add your partner(s) to your submission!!!**