CSC110 Lecture 5: Function Scope and Testing Functions

Print this handout

Exercise 1: Function scope and local variables

Suppose we define the following Python function in a file called `lecture5.py`:

```
# In file lecture5.py

def add(x: int, y: int) -> int:
    """Return the sum of x and y.

    >>> add(5, 6)
    11
    """
    return x + y
```

Then, we run this file in the Python console, and type in the following, we see an error:

```
>>> eleven = add(5, 6)
>>> twelve = x + 7
Traceback (most recent call last):
  ... [some output omitted] ...
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

1. Which assignment statement is causing the error, `eleven = ...` or `twelve = ...`?

2. What does `NameError: name 'x' is not defined` mean?

3. Why is `x` not defined?

4. What are the *local variables* in the `add` function?

Exercise 2: Function scope and the value-based memory model

Recall the function definitions for `square` and `calculate_distance`, with some "markers" adde
in comments:

```
def square(x: float) -> float:
    """Return x squared.

    >>> square(3.0)
    9.0
    >>> square(2.5)
    6.25
    """
    # MARKER C
    return x ** 2


def calculate_distance(x1: float, y1: float, x2: float, y2: float) ->
        float:
    """Return the distance between points (x1, y1) and (x2, y2),
    rounded to two decimal places.

    >>> calculate_distance(0.0, 0.0, 3.0, 4.0)
    5.0
    """
    # MARKER B
    dx_squared = square(x1 - x2)
    dy_squared = square(y1 - y2)
    return round((dx_squared + dy_squared) ** 0.5, 2)
```

Suppose we run a file containing these definitions in the Python console, and then execute the following code:

```
>>> p = [3.0, 4.0]
>>> # MARKER A
>>> distance = calculate_distance(0.0, 0.0, p[0], p[1])
>>> # MARKER D
```

1. Complete the value-based memory model diagram to show the current state of the variables when "MARKER A" is reached (*before* `calculate_distance` is called).

   (*Note*: we've given you more rows than necessary.)

   `__main__` (Python console)

Variable	*Value*

Variable	*Value*

2. Complete the value-based memory model diagram to show the current state of the variables when "MARKER B" is reached (*inside* `calculate_distance`, and *before* `square` is called).

   (*Note*: we've given you more rows than necessary.)

   `__main__` (Python console)

Variable	*Value*

   `calculate_distance`

Variable	*Value*

3. Complete the value-based memory model diagram to show the current state of the variables th *first* time "MARKER C" is reached (*inside* `square`, and *before* the return statement).

(*Note*: we've given you more rows than necessary.)

`__main__` (Python console)

Variable	*Value*

`calculate_distance`

Variable	*Value*

`square`

Variable	*Value*

Variable	*Value*

4. Complete the value-based memory model diagram to show the current state of the variables when "MARKER D" is reached (after `calculate_distance` has returned).

`__main__` (Python console)

Variable	*Value*

Exercise 3: Writing test cases

Suppose your friend Levi has defined the following function:

```
def rank_absolute_values(numbers: set) -> list:
    """Return a list that contains the absolute values of the given
        numbers,
    in non-decreasing order.
    """
    absolute_values = [abs(number) for number in numbers]
    return sorted(absolute_values)
```

Levi comes to you for help writing test cases for this function.

1. Write one *doctest example* for this function. Then, show where you would put it inside the above function definition.

2. What is the code Levi needs to add to the bottom of his Python file in order to run his doctests automatically?

3. Next, show how to turn the doctest example into a *unit test* that you would run using `pytest`.

4. Finally, write one additional unit test that checks a different kind of input than what you used the previous unit test. Make sure to include an English description of the test case in the docstring that makes it clear why it's different.

Additional exercises

1. *Practice with importing modules:* `math`. Use the Function Design Recipe to implement the following function: Given three side lengths of a triangle (as `float`s), calculate the angles in tl triangle.

   Include an `import math` statement at the top of your Python file so that you can use definitions from `math` in your function implementation for this question.

   Hints:

   - The *Cosine Law* from trigonometry states that for a triangle with side lengths a, b, and c, with angle θ opposite the side with length c, the following equality holds:

     $$c^2 = a^2 + b^2 - 2ab\cos\theta$$

   - The sum of all angles in a triangle add up to π radians (180 degrees).
   - The `math` module has functions both for calculating the trigonometric functions and thei inverses: `sin` and `asin` (short for "arcsin"), `cos` and `acos`, etc.

     To learn more about these functions in the Python console, try importing the `math` module and then executing (for example) `help(math.sin)`.

     The `math` module also has a variable `pi` that you can access in your code as `math.pi`.

2. *Practice with importing modules:* `datetime`. You are renting a car to make a road trip across Canada. The car rental company you plan to use charges a fee of $50 plus $15 per day you rent the car.

   Given the starting and ending dates of your trip (represented in Python as `datetime.date` values), calculate the total cost of renting a car. (*Note*: the start and end dates are both counte in the rental cost.)

   For this function, you should read Section 2.5 Importing Modules (https://www.teach.cs.toronto.edu/~csc110y/fall/notes/02-functions/05-importing-modules.html) for more information about the `datetime` module.