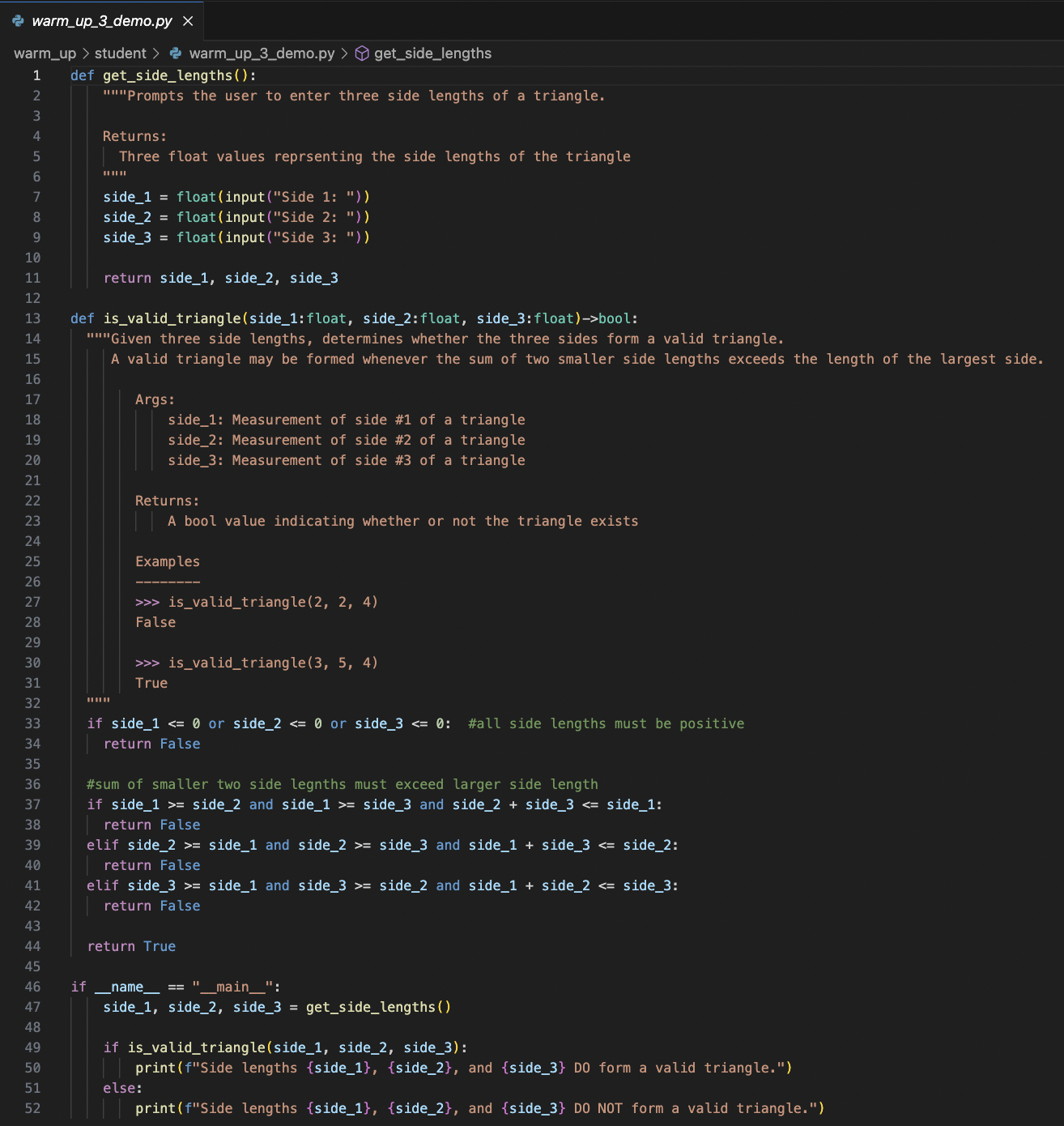
# **Python Warm-Up #3: Selection Statements**

| **Use VSCode to open and read the Python code in** [**selection\_demo.py**](https://drive.google.com/file/d/1eMzSXBYaWk1BXB4k7ql3-dlFosWD5Tlk/view?usp=drive_link) |
| --- |

selection\_demo.py contains the following Python code:

****

**Read through the code. What is the purpose of this program?**

| **Decomposing the problem solution** |
| --- |

This Python program determines whether three given side lengths can form a valid triangle. The problem solution is ***decomposed*** into three sub-problems:

1. Ask the user for three side lengths
2. Determine whether the three side lengths can form a valid triangle
3. Display the resulting determination

The function get\_side\_lengths is responsible for prompting the user to enter three side lengths. Things to note:

* Type casting is employed to convert the string input to a floating-point decimal
* The function returns three separate numeric values, one numeric value for each side length

The function is\_valid\_trinangle is responsible for determining whether three side lengths can form a valid triangle. There are two major criteria to satisfy in order to determine whether three given side lengths can form a valid triangle:

* *Are all three side lengths positive?*
* *Is the sum of the smaller two sides greater than the largest side?*

Displaying the final determination happens in "\_\_main\_\_":

* Users are provided feedback on their original input
* The original problem is restated
* The determination is clearly visible

| **Understanding Boolean Expressions** |
| --- |

Solving this problem requires our code to answer questions. This implies that the outcome of running the Python code can vary depending on the inputs given.

**Selection statements** allow a programming language to ask and answer questions according to well-defined criteria. An **if statement** is the most common form of a selection statement. if statements rely on **Boolean expressions** for their decision-making. A Boolean expression is any expression that evaluates to either True or False

### **Boolean Operators:** <, >, <=, >=, !=, ==

Boolean operators form Boolean expressions because they evaluate to either True or False.

#### **Logical Operators: not**, **and**, **or**

Simple boolean expressions can be combined into more complex boolean expressions with logical operators. Logical operators also form a Boolean expression because they evaluate to either True or False.

* **not**: True -> False, False->True
* **and**: True if both sides of the operator are True, otherwise False
  + **Short-Circuit Evaluation**: If the left side of an **and** evaluates to False, the right side is not evaluated
* **or**: True if either side of the operator is True, otherwise False
  + **Short-Circuit Evaluation**: If the left side of an **or** evaluates to True, the right side is not evaluated

### 

### **Order of Operations**

The Order of Operations from Warm Up #2 can be extended to include Boolean and Logical operators:

| Precedence | Operation | Operator | Example | Result |
| --- | --- | --- | --- | --- |
| 9 | **Comparison** | <, <=, >, >= | 2<=4 | True |
| 10 | **Equality** | ==, != | 2!=4 | True |
| 11 | **Logical NOT** | not | **not** 2 **>** 4 **==** False | True |
| 12 | **Logical AND** | and | **not** 5 > 6 **and** -2 > 0 | False |
| 13 | **Logical OR** | or | **not** 5 > 6 **or** -2 < 0 | True |

Some important considerations:

* Boolean operators have different precedence re: comparison vs equality
* Logical operators have different precedence
* Many other programming languages use ~, &&, and || to represent the same logical operators. It’s an easy mistake to forget that Python uses English-based logical operators.

| **Understanding Selection Statements** |
| --- |

Selection statements use Boolean expressions to create branching paths in code. Based on whether a Boolean expression evaluates to **True** or **False**, code in the body of a selection statement can be skipped or executed.

### **if Statements**

if statements are the most common type of selection statement. Lines 33 - 34 in warm\_up\_3\_demo.py use an if statement to verify whether any side length is negative:

| **if** side\_1 <= 0 **or** side\_2 <= 0 **or** side\_3 <= 0: *#no negative side length allowed!*  **return** **False** |
| --- |

* The keyword if is used to begin the selection statement, followed by a Boolean expression
* The : symbol indicates the start of the body of the if statement.
  + Consistent indentation must be used in the body of the if statement
* If the Boolean expression side\_1 <= 0 **or** side\_2 <= 0 **or** side\_3 <= 0 evaluates to True, then **return** **False** is executed
  + The is\_valid\_triangle function immediately ends. No further code in the function body is executed.
  + The **or** logical operator is used to connect separate Boolean expressions in this **if** statement.
    - Any one of the Boolean Expressions could evaluate to **True** in order to execute the **if** statement body
* Note: Python if statement do not need to enclose Boolean expressions in ()s

### **Negating Boolean Expressions**

A different way of validating the same criteria would use the **and** operator:

| *#all side lengths must be positive*  **if** not(side\_1 > 0 **and** side\_2 > 0 **and** side\_3 > 0):  **return** **False** |
| --- |

* Notice how exp1 OR exp2 OR exp3 is equivalent to NOT(exp1 AND exp2 AND exp3)
  + This is known as **DeMorgan’s Law**
* Similarly, a <= b is equivalent to NOT(a > b)

### 

### **if-else Statements**

Lines 49 - 52 use an **if**-**else** statement to create mutually-exclusive branches in the execution of the code:

| **if** is\_valid\_triangle(side\_1, side\_2, side\_3):  print(f"Side lengths {side\_1}, {side\_2}, and {side\_3} DO form a valid triangle.") **else**:  print(f"Side lengths {side\_1}, {side\_2}, and {side\_3} DO NOT form a valid triangle.") |
| --- |

* ***Mutually-exclusive*** for two alternatives means one, or the other, but not both.
* In an **if**-**else** statement, exactly one of two paths must be followed.
  + The body of the **else** section is sometimes called the ***default***path.

### **Extended if Statements: elif**

An extended-if statement uses the **elif** keyword to create multiple mutually-exclusive branches:

| **if** side\_1 >= side\_2 **and** side\_1 >= side\_3 **and** side\_2 + side\_3 <= side\_1:  **return** **False elif** side\_2 >= side\_1 **and** side\_2 >= side\_3 **and** side\_1 + side\_3 <= side\_2:  **return** **False elif** side\_3 >= side\_1 **and** side\_3 >= side\_2 **and** side\_1 + side\_2 <= side\_3:  **return** **False else**:  **return** **True** |
| --- |

* **elif** connects related criteria into a single selection statement with several mutually-exclusive branches
* The inclusion of an **else** means that exactly one of the branching paths must be followed.
  + If none of the preceding Boolean expressions evaluates to **True**, then the body of the **else** will execute

**Pro Tip**: For this particular scenario, the **elif** isn't strictly necessary. The code could instead be rewritten as:

| **if** side\_1 >= side\_2 **and** side\_1 >= side\_3 **and** side\_2 + side\_3 <= side\_1:  **return** **False** **if** side\_2 >= side\_1 **and** side\_2 >= side\_3 **and** side\_1 + side\_3 <= side\_2:  **return** **False** **if** side\_3 >= side\_1 **and** side\_3 >= side\_2 **and** side\_1 + side\_2 <= side\_3:  **return** **False**  **return** **True** |
| --- |

If one of the Boolean expressions evaluates to **True**, the function returns and immediately stops executing. Even though these are all mutually-exclusive conditions, each **elif** statements could be replaced with if to achieve the same results. In the same vein, the **else** could also be removed given that the function would necessarily have to return **True** when all the Boolean expressions evaluate to **False**:

| **Reading Function Documentation** |
| --- |

The functions in warm\_up3\_demo.py include extensive documentation:

| **def is\_valid\_triangle(side\_1:float, side\_2:float, side\_3:float)->bool:  """Given three side lengths, determines whether the three sides form a valid triangle.  A valid triangle may be formed whenever the sum of two smaller side lengths exceeds the length of**  **the largest side.   Args:  side\_1: Measurement of side #1 of a triangle  side\_2: Measurement of side #2 of a triangle  side\_3: Measurement of side #3 of a triangle   Returns:  A bool value indicating whether or not the triangle exists   Examples  --------  >>> is\_valid\_triangle(2, 2, 4)  False   >>> is\_valid\_triangle(3, 5, 4)  True  """** |
| --- |

These comments specify what programmers can expect regarding:

* The purpose of the function
* Input parameters/arguments
* Return values
* Example behavior ie. simple test cases

### **Type Hints**

Though Python doesn’t include explicit data types when creating variables for defining function parameters, recent versions of Python have support **type hints**, which allow programmers to label expected data types. The function header includes the use of type hints as documentation:

| **def** **is\_valid\_triangle**(side\_1:float, side\_2:float, side\_3:float)->bool: |
| --- |

* Parameters are expected to be floating-point decimals
* The function is expected to return a Boolean value

**Note**: Python type hints are not enforced. They are strictly for documentation purposes.

| **Warm-Up #3 Exercises -** |
| --- |

### 

### **Exercise 3.1: Implement the classify\_triangle function in** [**warm\_up\_3.py**](https://drive.google.com/file/d/10X7YI4x9YIXCJtY4Ng1WazbYkgIU-ydj/view?usp=drive_link)

This problem asks you to classify triangles according to one of four labels:

* "does not exist"
* "acute" - sum of the squares of the two smaller sides is greater than the square of the largest side
* "obtuse" - sum of the squares of the two smaller sides is less than the square of the largest side
* "right" - sum of the squares of the two smaller sides equals the square of the largest side

Unit tests on Gradescope will verify whether your function returns the exact string specified above. So, despite being logically correct, returning "DNE" rather than "does not exist" will cause a Gradescope test to fail.

**Note**: The goal of this assignment is to provide an opportunity to explore Python selection statements. Do not use the built-in Python sort, sorted, max, or min functions in your implementation.

Save your changes, then submit **warm\_up\_3.py** to the Gradescope Warm Up 3 assignment on [Gradescope](https://www.gradescope.com/).

Resubmit via Gradescope until all tests pass.