**Topic 7 - Math Expressions: Eliminating Ambiguity**

**Introduction**  
When working with arithmetic expressions, the order in which operations are performed can affect the final result. This is a common challenge in algebra and programming alike. In Python, order of operations is determined by specific precedence rules, but ambiguity can still arise. This lesson explores how to use parentheses to control the order of operations, making your code clearer and more predictable.

**What Are Math Expressions and Order of Operations in Python?**

In Python, math expressions are evaluated following a set of precedence rules, which define the sequence in which operations are executed. For example:

***total\_cost = 1 + 3 \* 4***

This expression can yield different results based on the order:

* If added first: 1+3=4, then multiplied by 4, the result is 16.
* If multiplied first: 3∗4=12,then adding 1, the result is 13.

Python’s precedence rules dictate that multiplication is completed before addition, so total\_cost is equal to 13 here.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Why Use Parentheses in Math Expressions?**

Parentheses eliminate ambiguity by specifying the order of operations. This approach has several advantages:

* **Clarity of Intent:** Using parentheses ensures that Python (and readers) interprets the expression as intended.
* **Improved Readability**: Clearly defining the order of operations makes the code easier to understand and reduces misinterpretation.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**How to Use Parentheses to Control Order of Operations in Python**

Using parentheses in math expressions explicitly specifies the order in which operations are carried out. Here’s how it works with examples:

1. Multiplying before adding:

***total\_cost = 1 + (3 \* 4) # Result: 13***

1. Adding before multiplying:

***total\_cost = (1 + 3) \* 4 # Result: 16***

1. Practical Example with Nested Operations:

***result\_of\_computation = (2 \* 4) \* 4 + 2***

To achieve different results, adjust the parentheses based on the desired operation sequence:

* + Multiply first, then add:

***result\_of\_computation = ((2 \* 4) \* 4) + 2 # Result: 34***

* + Multiply by the sum:

***result\_of\_computation = (2 \* 4) \* (4 + 2) # Result: 48***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Conclusion

Using parentheses to specify the order of operations in Python ensures your code executes as expected and enhances its readability. This approach reduces ambiguity and makes your code more reliable and easier to maintain.