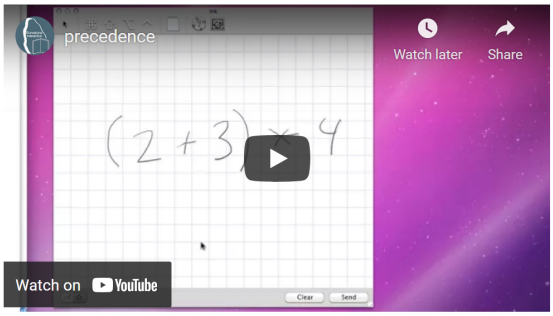
2.11. Order of Operations



Activity: 1 -- Video: (precedencevid)

When more than one operator appears in an expression, the order of evaluation depends on the **rules of precedence**. Python follows the same precedence rules for its mathematical operators that mathematics does.

1. *Parentheses* have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, `2 * (3-1)` is 4, and `(1+1)**(5-2)` is 8. You can also use parentheses to make an expression easier to read, as in `(minute * 100) / 60`: in this case, the parentheses don't change the result, but they reinforce that the expression in parentheses will be evaluated first.
2. *Exponentiation* has the next highest precedence, so `2**1+1` is 3 and not 4, and `3*1**3` is 3 and not 27. Can you explain why?
3. *Multiplication and both division* operators have the same precedence, which is higher than addition and subtraction, which also have the same precedence. So `2*3-1` yields 5 rather than 4, and `5-2*2` is 1, not 6.
4. Operators with the *same* precedence are evaluated from left-to-right. In algebra we say they are *left-associative*. So in the expression `6-3+2`, the subtraction happens first, yielding 3. We then add 2 to get the result 5. If the operations had been evaluated from right to left, the result would have been `6-(3+2)`, which is 1.

Note

Due to some historical quirk, an exception to the left-to-right left-associative rule is the exponentiation operator `**`. A useful hint is to always use parentheses to force exactly the order you want when exponentiation is involved:

Save & Run 4/17/2021, 9:48:40 AM - 2 of 2 Show in CodeLens

```
1 print(2 ** 3 ** 2)     # the right-most ** operator gets done first!
2 print((2 ** 3) ** 2)   # use parentheses to force the order you want!
3
```

```
512
64
```

Activity: 2 -- ActiveCode (ac2_11_1)

Note

This is a second way that parentheses are used in Python. The first way you've already seen is that () indicates a function call, with the inputs going inside the parentheses. How can Python tell when parentheses specify to call a function, and when they are just forcing the order of operations for ambiguous operator expressions?

The answer is that if there's a an expression to the left of the parentheses that evaluates to a function object, then the parentheses indicate a function call, and otherwise not. You will have to get used to making the same inference when you see parentheses: is this a function call, or just specifying precedence?

Check your understanding

data-11-1: What is the value of the following expression:

```
16 - 2 * 5 // 3 + 1
```

- (•) A. 14
- ( ) B. 24
- ( ) C. 3
- ( ) D. 13.667

Check me Compare me

✔ Using parentheses, the expression is evaluated as (2*5) first, then (10 // 3), then (16-3), and then (13+1).

Activity: 3 -- Multiple Choice (question2_11_1)

Here is an animation for the above expression:

Next Step Reset

```
16 - 2 * 5 // 3 + 1
```

```
16 - 2 * 5 // 3 + 1
```

Activity: 4 -- ShowEval (se_ac2_11_1)

You have attempted 5 of 4 activities on this page

2.10. Statements and Expressions">
atements and Expressions">
2.12. Reassignment">

✔ Completed. Well Done!

| Back to top