Operators and Types

Python's arithmetic looks familiar at first, but two things take getting used to: the order in which operators are evaluated, and the difference between Python's two kinds of numbers.

Content Learning Objectives

After completing this activity, students should be able to:

- Identify and justify the precedence of arithmetic operators.
- Explain differences between integer and floating-point numbers.

Process Skill Goals

During the activity, students should make progress toward:

- Identifying data types by running examples in a shell. (Information Processing)

Model 1 Order of Operations

Python follows the usual order for arithmetic operations. The following table lists operators from highest to lowest ***precedence***.

Operator	Description
`**`	Exponentiation
`+ -`	Positive, Negative (***unary*** operators)
`* /`	Multiplication, Division
`+ -`	Addition, Subtraction (***binary*** operators)
`=`	Assignment

Questions (20 min) Start time:

1. Determine the order of operations in the statement: `y = 9 / 2`

a) First operator to be evaluated:
b) Second operator:
c) Value of `y`:

2. Determine the order of operations in the statement: `x = 5 * -3`

a) First operator to be evaluated:
b) Second operator:
c) Third operator:
d) Value of `x`:

3. Determine the order of operations in the statement: `z = 2 * 4 ** (3 + 1)`

a) First operator to be evaluated:
b) Second operator:
c) Third operator:
d) Fourth operator:
e) Value of `z`:

4. The + and - operators appear twice in the table of operator precedence. For the Python statement `x = 5 * -3`, explain how you know whether the - operator is being used as a unary or binary operator.

5. What do the words "unary" and "binary" mean in this context?

6. What operator has the lowest precedence? Why do you think Python is designed that way?

7. What operator have the highest precedence? Why do you think Python is designed that way?

8. Enter the expressions below into a Python Shell. Why are the results different? Explain your answer in terms of operator precedence.

- `-3 ** 2` Result:
- `(-3) ** 2` Result:

Model 2 Integers and Floats

Every value in Python has a ***type*** which determines what can be done with the value. Consider the following statements and expressions that were entered into a Python Shell.

Python code	Shell output
`integer = 3`	
`type(integer)`	<class 'int'>
`type("integer")`	<class 'str'>
`pi = 3.1415`	
`type(pi)`	<class 'float'>
`word = str(pi)`	
`word`	'3.1415'
`number = float(word)`	
`print(word * 2)`	3.14153.1415
`print(number * 2)`	6.283
`print(word + 2)`	TypeError
`print(number + 2)`	5.14159
`euler = 2.7182`	
`int(euler)`	2
`round(euler)`	3

Questions (20 min) Start time:

9. What is the value and type (`int`, `float`, or `str`) of the following variables?

Variable	Value of Variable	Type of Value
`integer`		
`word`		
`number`		
`euler`		

10. List the function calls that convert a value to another type.

11. How does the behavior of the operators (+ and *) depend on the data type?

12. What is the difference between the `int` function and the `round` function?

13. What is the value of `3 + 3 + 3`? What is the value of `.3 + .3 + .3`? Enter these expressions into a Python Shell—what do you notice about the results?

14. Based on the previous question:

a) In order to store a number with 100% accuracy, what data type is required?

b) How might you precisely represent a bank account balance of $123.45?

15. Try calculating a very large integer in a Python Shell, for example, 123^{456}. Is there a limit to the integers that Python can handle?

16. Try calculating a very large floating-point number in a Python Shell, for example, 123.0^{465}. Is there a limit to the floating-point numbers that Python can handle?

17. Summarize the difference between the numeric data types (`int` and `float`). What are their pros and cons?