ECOR 1041 Lecture 1
Python demo script: types and expressions.
-------------------------------------------------------------------

Some Experiments to Learn About Python Arithmetic Expressions

We are going to create some experiments to help us learn how to write Python expressions that describe calculations with integers and floating-point numbers. (Floating-point is one way of representing real numbers in a programming languages). To do this, we are going to run the Python interpreter in "interactive mode".

When the interpreter starts, it displays a message that indicates the Python release; e.g.,

```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC
v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The message is followed by a prompt (`>>>`), which indicates that Python is waiting for us to type an expression. After we have typed the expression and pressed the Enter key, Python reads the expression, evaluates it, and prints (displays) the result. Python then redisplays the prompt and waits for us to enter another expression.

Aside 1: this set of steps is often referred to as a REPL, which is short for “Read-Evaluate-Print Loop”.

Aside 2: the interactive interpreter is sometimes referred to as the “Python shell”, because it operates somewhat like the software that lets users use an operating system by typing commands interactively; for example, a Unix shell, a Windows command-prompt window, or a macOS shell running in a Terminal window.

Here is the simplest experiment I can think of to determine if a Python expression that adds two integers sets the same notation the we learned in elementary school. After the >>> prompt, type `1 + 2`, followed by the Enter key

```
>>> 1 + 2
3
```

Clearly, `1 + 2` is a valid Python expression. Python evaluates the expression (adds the two integers) and displays the result (3).

Some vocabulary: 1 and 2 are *literal values*. A literal is a fixed value that can be directly represented in a programming language. Here, 1 and 2 are integer literals.

The + is the symbol that represents addition. It is called the *addition operator*. The 1 and 2 are the *operands* of the operation specified by the + operator.

What are the expressions for subtracting, multiplying and dividing integers? Let us try subtraction first.

```
>>> 4 - 1
3
```

I do not see a “times” symbol on the keyboard, so what is the symbol for the multiplication operator? Lowercase letter ‘x’? Uppercase letter ‘X’? Something else? Let us try some experiments:

```
>>> 2 x 3
Traceback (most recent call last):
  Python Shell, prompt 4, line 1
Syntax Error: invalid syntax: <string>, line 1, pos 3
```

The message displayed by Python tells us that 2 x 3 is not a valid Python expression. Specifically, it has a *syntax error*, because the character x is not the symbol for an operator.

Let us try uppercase X as the multiplication operator:

```
>>> 2 X 3
Traceback (most recent call last):
  Python Shell, prompt 5, line 1
Syntax Error: invalid syntax: <string>, line 1, pos 3
```

Looks like it is time to “read the manual”. For example, when we read Chapter 2 of *Practical Programming*, we learn that the symbol for the multiplication operator is the asterisk, `*`. So, to multiply 2 by 3, we type:

```
>>> 2 * 3
6
```

By reading the textbook, we learn that the symbol for the division operator is the forward slash, /.

Predict the result when Python evaluates this expression:

```
6 / 2
```

Elementary school math suggests that the result might be the integer 3. Students who have programmed in C, Java or Python 2 know that, in those languages, dividing and integer by an integer always yields an integer, so 6 / 2 evaluates to 3. Let us see if this is also true for Python 3:

```
>>> 6 / 2
3.0
```

Notice that the result is 3.0, not 3. A literal numeric value that has a decimal point, followed by 0 or more digits, is a *floating point* number. Floating point numbers are a subset of the real numbers.

Let us try a division expression where we expect the result to have digits after the decimal point:

```
>>> 7 / 2
3.5
```

If we try several experiments like this, we can conclude that / always yields a floating point quotient when the operands are integers, even when the result could be represented as an integer; for example, as 3.0 instead of 3. (In contrast, in C, Java and Python 2, dividing and integer by an integer always yields an integer.)

What does Python do if we type the simplest expression, a literal integer?

```
>>> 42
42
```

So, we see that a literal value evaluates to itself.

Can we have expressions that have more than one operator? Suppose we type:

```
>>> 1 + 2 * 3
```

If this is a valid Python expression, it leads to another question: Are expressions evaluated left to right (in which case 1 + 2 * 3 will evaluate to 9) , or do some operators have higher precedence than others (like the arithmetic we learned in grade school)? If multiplication has higher precedence than addition, 1 + 2 * 3 will evaluate to 7. Try some experiments:

```
>>> 1 + 2 * 3
7

>>> 2 * 3 + 1
7
```

It looks like multiplication has higher precedence than addition.

What if we wanted the addition operation to be evaluated before the multiplication. Our grade school math suggests that we can use parentheses to change the order in which subexpressions are evaluated:

```
>>> (1 + 2) * 3
9

>>> 2 * (3 + 1)
8
```

We can conclude that the subexpressions enclosed in parentheses are evaluated first.

Sometimes, math teachers use brackets, [], or braces, {}, instead of parentheses to enclose subexpressions. Does this work in Python? Let us try it:

```
>>> [1 + 2] * 3
[3, 3, 3]
```

The characters we typed form a valid Python expression, which the interpreter evaluated, but the result is certainly not an integer. It is a list of three 3's - you'll learn about lists later in the course.

Now try braces:

```
>>> {1 + 2} * 3
Traceback (most recent call last):
  Python Shell, prompt 3, line 1
builtins.TypeError: unsupported operand type(s) for *: 'set' and 'int'
```

Clearly, this is not a valid Python expression. You will learn in ECOR 1042 that braces are used to construct containers for data known as sets and dictionaries.

Let us evaluate a few expressions with floating point operands:

```
>>> 7.0 + 2.0
9.0

>>> 4.1 + 5.7
9.8

>>> 2.3 * 5.0
11.5

>>> 2.0 / 3.0
0.6666666666666666
```

It appears that an expression in which both operands are floating point numbers evaluates to a floating-point number.

Every value supported by a programming language has a *type*, which defines a set of values and the operations that can be applied to those values. We can determine a value's type by typing the word type, followed by an expression enclosed in parenthesis:

```
>>> type(1)
<class 'int'>
```

The result tells us that the type of 1 is `int` (short for integer).

Aside 1: We have just applied a built-in function named `type`, to the argument 1. Another way of saying this is, “We have called the built-in function named `type`, passing it the argument 1”. You will learn a lot more about functions in Python, soon.

Aside 2: In Python, types are implemented using a programming construct known as a *class*. We will not cover classes in this course, so you can treat the word 'class' as a synonym for 'type'.

Let us try some more experiments:

```
>>> type(2)
<class 'int'>

>>> type(-1)
<class 'int'>

>>> type(0)
<class 'int'>
```

What is the type of a large integer?

```
>>> type(10000000000000000000000000000000000)
<class 'int'>
```

What is the type of a small integer?

```
>>> type(-10000000000000000000000000000000000)
<class 'int'>
```

It is beginning to look like all integers are values of type `int`. (Which is true.)

What about floating-point numbers?

```
>>> type(3.0)
<class 'float'>

>>> type(-4.7)
<class 'float'>

>>> type(0.0)
<class 'float'>
```

In Python, floating-point numbers have type `float`.

There is a lot more to learn about arithmetic expressions. For example, does Python let us construct expressions that mix values of type `int` with values of type `float`? If so, what is the type of the values produced by the +, -, * and / operators?

Python has other arithmetic operators: //, % and **. What types of operands can be used with these operators? What operations do they perform? What is the type of the result of each operation?

We can think of many more experiments that would help us learn about computation with integers and floating-point values. That will be the topic of Lab 1.

**Extra Practice**

Experiment 1: Without using Python, predict the value Python will calculate when it evaluates this expression:

```
>>> 3 + 7 * 2 – 5
```

Did you predict 15? 12? -18? Something else? Will Python display an error message?

Now check your prediction.

```
>>> 3 + 7 * 2 - 5
12
```

Explanation: Python evaluates the expression this way:

```
3 + 7 * 2 - 5 ==> 3 + 14 - 5
              ==> 17 - 5
              ==> 12
```

Experiment 2: Without using Python, predict the value Python will calculate when it evaluates this expression:

```
>> 5 + 3 * 3 + 7 / 2
```

Did you predict 15? 15.5? 17? 17.5? Something else? Will Python display an error message?

Now check your prediction.

```
>>> 5 + 3 * 3 + 7 / 2
17.5
```

Explanation: Python evaluates the expression this way:

```
5 + 3 * 3 + 7 / 2 ==> 5 + 9 + 7 / 2
                  ==> 5 + 9 + 3.5
                  ==> 14 + 3.5
                  ==> 17.5
```