Expressions and Arithmetic

Goal

The goal of this lesson is understand what expressions are and how they are different from statements.

Objectives

- The student will be able to identify the arithmetic operators and use them to create valid arithmetic expressions
- The student will be able to identify relational operators and use them to create valid boolean expressions
- The student will learn a few string operators and use them to create valid string expressions

Lesson

These concepts will be given interactively, via whiteboard/projector. Students can take notes during the discussion. Estimated time ~15 minutes.

The grammar of imperative languages is composed of statements, expressions, and values. Today we will discuss expressions.

Expressions are *evaluated* to obtain a *value*. They are always evaluated before they are used in a statement. We know three types of values (number, string, boolean), so we will look at three kinds of expressions, focussing on numbers.

An expression is built using variables, values and operators. Each type has a set of operators that can be used to work with that type. For numbers, the operators will look familiar.

Operator	**What it means**
x + y	Addition
x - y	Subtraction
x * y	Multiplication
x / y	Floating point division
x // y	Integer division (compare to math.floor)
x % y	Modular division (remainder)
-x	Negation (unary)

If an expression contains only integers, the value will be an integer (unless you use floating point division).

If an expression contains any floating point numbers, the value will be a floating point number.

Python follows normal math operator precendence (left to right, negation, multiplication/division, addition/subtraction). This can be modified with paranthesis.

```
3 + 6 * 2           # 15
(3 + 6) * 2         # 18
```

String expressions are much simplier than numbers, as there is only a single operator, the +. When this operator is applied to strings, the value is the contatenation of those strings into a single string.

```
'John' + " " + 'Smith'     # "John Smith"
```

While we can mix integers and floating point numbers in an expression, we can't mix numbers and strings:

```
3 + 'dog'   # Returns an error
```

Logical expressions use the logical operators. We will cover this in January with Conditional Execution, but it's useful to compare logical expressions with numeric expressions because they both occur so frequently.

Here are the relational operators, these are used to compare two values and yield a boolean value.

Operator	What it means
x == y	Does x equal y ?
x != y	Does x not equal y (are they different) ?
x < y	Is x less than y ?
x > y	Is x greater than y ?
x >= y	Is x greater than or equal to y ?

All of these operators work with any type that can be compared, including strings, numbers, and in some cases booleans. Remember that the comparison always yields a boolean value.

Here are some examples:

```
9 < 10              # True
9 > 10              # False
'cat' == 'cat'      # True
True == 8 < 10      # False
True == (8 < 10)    # True
0.333333 == (1 / 3) # False
```

Remember that floating point numbers are not precise, it's usually not a good idea to use `==` to compare them.

```
1 - 1/3 - 1/3 - 1/3 == 0      # Should be True, but it's not
1 - 1/3 - 1/3 - 1/3 < 1e-10   # Instead use < or >
```

Activities

Students can work through these activities directly on their Pi. Estimated time ~ 20 minutes.

1. Let the user input a number of seconds and convert this hours and minutes. For example, 11900 seconds is 3h 33m.

2. Modify your previous program to handle the case if the number of seconds is greater than a single day. Change the format to show days, hours and minutes. For example, 3421356 seconds is 39d 14h 22m.

3. If you have 1 - (1 / 2) - (1 / 2) with real numbers, the answer should be zero. Does Python calculate this to zero? What about 1 - (1 / 3) - (1 /3) - (1 / 3) ? What's the difference?

   What if you do 1 - (3 * 1 / 3)? What's the difference?

4. Add to your temperature conversion program from last week to print whether the temperature in celsius is below freezing (0) or above boiling (100).

   If you didn't get that far, you can use this one:

   ```
   fahr = float(input('Enter temperature in F: '))
   cel = 5 / 9 * (fahr - 32)
   print('Celsius:', round(cel, 1))
   ```

5. Run your program from #4, use these values to test:

   ```
   211.99999999999999999999999999999999999999
   32.00000000000000000000000000000001
   ```

   Do they work? If not, how would you fix them?