Python Essentials (Pt. 2)

Overview: In this lesson we'll be looking at lists, dictionaries, tuples, sets, comparison operators, and if/else statements. It's still pretty concept-based at this point, but there's one or two activities you can try out in command prompt/Anaconda included here as well. Happy coding!

Lists

Lists are sequences of elements separated by commas and contained by square brackets. They can take in any data type, like strings, integers, etc.

```
Input:     list = [ 1, 2, 3]
           list.append(4)
           list
Output:    [ 1, 2, 3, 4]
```

The *append* function is particularly useful here, allowing the user to add a value to the end of the list.

Like *string indexing* (which was covered in the previous lesson), you can use indexing to locate values in a list.

```
Input: list[0]
Output: 1 #remember that the first value of a list corresponds
           to an index of zero!
Input: list[1:3]
Output: [2, 3] # when using this notation, remember that your
                 output will start with the lower limit but will NOT
                 include the upper limit (the beginning value is
                 inclusive, but the end value is exclusive)
```

You can also use indexing to reassign values.

```
Input: list[0] = 'one'
      list
Output: [ 'one', 2, 3, 4]
```

When you put a list inside of a list, this is known as *nesting*. In the following example, the value 'prize' is what we will try to "grab" with indexing.

```
Input: nest = [ 'a', 'b', 'c', [ 1, 2, ['prize']]]
        nest[3][2][0]
Output: 'prize' # Without [0], the output would've looked like
                 ['prize']. If you wanted to get prize by itself, you
                 would do print(nest[3][2][0]).
```

Dictionaries

Dictionaries are another way to organize elements, and can contain any data type, including more lists and more dictionaries! They are organized by key-value pairs, so indexing is a bit different.

```
Input: dict = {'key1' : 'value', 'key2' : 135}
```

```
       dict['key1']
Output: 'value' # The use of dict[0] would have produced an error.
Input: d = {'key1' : {'key2' : [1,2,3]}}
```
To obtain the value 3:
```
Input: d['key1']['key2'][2]
Output: 3
```

Tuples

Tuples are very similar to lists, in that they are sequences of elements. Some key differences, however, are that tuples are contained by parentheses and are immutable, meaning values can not be reassigned.
```
Input: tuple = (1, 2, 3)
       t[1]
Output: 2
Input: tuple[0] = 'one'
Output: ERROR
```

Sets

Sets are collections of unique elements. This means that if you were to have multiple elements of the same value, the output would only contain the distinct ones.
```
Input: {1, 2, 2, 3, 3, 3}
Output: {1, 2, 3}
Input: set = {1, 3, 5}
       set.add(7)
       set
Output:{1, 3, 5, 7}
```
Notice what happens if you use the .add function to add an element already present in the set:
```
Input: set.add(7)
       set
Output: {1, 3, 5, 7} # If there are repeated elements, it won't be
                       added.
```

Comparison Operators

Comparison operators can take in any data type, and the result is a boolean value of `True / False`.
```
Input: 1 > 0 # > means greater than
Output: True
Input: 3 < 2 # < means less than
Output: False
Input: 5 >= 4 # >= means greater than or equal to
Output: True
Input: 3 <= 6 # <= means less than or equal to
Output: True
Input: 1 == 1 # == means equal to
```

```
              # if you were to only use one equal sign like 1 = 1,
           you
                would get an error because Python will think you are
                trying to reassign a variable
Output: True
Input: 'hello' != 'hi' # != means not equal to
Output: True
```

When you use *and*, all conditional statements must be true in order for the output to be `True`.

```
Input: (5 > 3) and (4 < 3) # parentheses are optional
Output: False
```

On the other hand, the use of *or* returns true if one or more conditional statements is true.

```
Input: (5 > 3) or (4 < 3)
Output: True
```

If/Else Statements

An if/else statement will perform a certain action in the code while blocking others if it satisfies a certain condition.

To put that in "English," it's similar to making a decision with two options. For example, let's say you're buying a new pair of shoes, and you're looking for the least expensive pair. You have found two that you like, called shoes A and shoes B. The if/else statement in pseudo-code might look something like this:

```
if shoes A is cheaper than shoes B:
     buy shoes A
else:
     buy shoes B;
```

What's in parentheses is called the conditional statement; in an if statement, this is your criteria that you are using to choose A over B, or vice versa. For the shoes example, the condition would be "shoes A is cheaper than shoes B." This could also be "shoes B is cheaper than shoes A"; you would just have to switch what is returned in the if and what is returned in the else.

In code, this would be a little bit complicated. First, you would have to set variables for the prices of shoes A and shoes B along with variables for the shoes themselves (for simplicity's sake, they'll be strings) because that's what you'll use to compare the price. Your code might look something like this:

```
shoesA = "Shoes A"
shoesB = "Shoes B"
priceA = 150
priceB = 75

if priceA < priceB:
```

```
      print(shoesA)
else:
      print(shoesB)
```

You can try this yourself in command prompt or Anaconda. You should see `Shoes B` printed since 150 is not smaller than 75.

In If/Else statements, conditions are set using your comparison operators (that's right, you're going to be reusing everything you learn, and here's a good example of that!). If you were checking if a variable equals a set value (e.g. if you have $20 in cash), you would use the == operator. It might look like this:

```
cash = 19
if (cash == 20):
      print("can pay")
else:
      print("cannot pay")
```

Thus, if your value for cash is $20, you should see "can pay"; otherwise (and in this case), you would get "cannot pay". You can try playing around with different comparison operators as the condition. If you feel up to the challenge, you can incorporate the math operators included in the previous lesson or string methods like .length() and see what happens!

There's also the *elif* (standing for "else if") statement, which you'd include between the if and the else statement. Like the if statement, the elif takes a condition, though it will different from what your if condition is (note that the else statement DOES NOT take a condition because it is essentially the "backup" if nothing else works). Using the shoes example again, the elif statement can be used to check if the shoes cost the same amount since < and > operators are not inclusive. Thus, you'd have:

```
if priceA < priceB:
      print(shoesA)
elif priceA == priceB:
      print("equal")
else:
      print(shoesB)
```

PRACTICE!!!

Below are some prompts that you can try to write a solution for! We'll include an "answer code" of sorts in the next lesson if it's necessary, but we encourage you to try them. They're fairly easy to implement and shouldn't take long.

#1. Create a more complex version of the shoe problem. It doesn't have to be about shoes; the only important thing is practicing use of the if/else statements—we challenge you to use at least three elif statements in your new code.

#2. Make either a list, dictionary, tuple, or set. Add at least one element (if possible—for example, this won't work for tuples since they are generally immutable), and print out every single element inside said list, dictionary, tuple, or set. For now, you'll be using a bunch of print statements. We'll cover loops next week, which will make this much easier if you've created a long list (or whatever you've chosen to try).

BONUS - use something that we covered last week (string indexing, math operators, etc.) in either or both of these prompts.

This concludes the second lesson. Feedback is appreciated, and tell us what you'd like to learn next! As usual, the next lesson will be out either this week or next week.
-Emma and Nicole <3