Using Nested Loops

Introduction

In this lesson, we will be looking at how to perform nested iteration (or looping). What does this mean exactly? Well, we know that a nested data structure is having one form of data nested inside another. For example, a nested list would be a list that contains another list, or a dictionary that has a key that points to a list.

```
# nested lists
list_of_lists = [[1,2,3], [4,5,6], [3,5,2]]
dict_nested_list = { 'name': "example", 'colors': ["blue", "green", "yel
low", "red"] }
```

So, with that example, we can infer that a nested loop would be a loop inside another loop. Sounds exciting, right? Let's take a look at some nested loops.

Objectives

You will be able to:

- Use a nested loop when it is appropriate

Writing A Nested Loop

Working with a nested data structure is a little confusing at first, but after doing it a few times it becomes much less intimidating. The same is true for writing nested loops. They will be somewhat commonplace in our programming future and are important to be comfortable with.

Basically what happens with a nested loop is the inner loop runs in its entirety **every** iteration of the outer loop. Let's take a look at an example before diving deeper.

In [1]:
```
outer_numbers = [1,2,3]
inner_words = ["ONE", "TWO", "THREE"]
for number in outer_numbers:
    print(f"this is iteration **{number}** of the OUTER loop")
    for word in inner_words:
        print(f"     this is iteration {word} of the INNER loop")
    print("\n")
```

```
this is iteration **1** of the OUTER loop
     this is iteration ONE of the INNER loop
     this is iteration TWO of the INNER loop
     this is iteration THREE of the INNER loop


this is iteration **2** of the OUTER loop
     this is iteration ONE of the INNER loop
     this is iteration TWO of the INNER loop
     this is iteration THREE of the INNER loop


this is iteration **3** of the OUTER loop
     this is iteration ONE of the INNER loop
     this is iteration TWO of the INNER loop
     this is iteration THREE of the INNER loop
```

How Nested Loops Work

Alright, so, what we see happening here is that the **inner** loop runs through each of its iterations for each iteration of the **outer** loop. If we break this down even further, we can think of the block inside of a loop as a single operation. The current iteration only moves on to the next iteration once it has completed the entire operation.

So, the operation of the outer loop is to print a string and execute a for loop on the `inner_words` collection. The outer block will do this three times. **BUT** the nested loop has to finish before the next iteration of the outer loop. The nested loop's job is to print its string three times, so that happens for each iteration of the outer loop.

We can nest any and all kinds of loops.

In [2]:
```
outer = 0
inner = 0
while outer < 3:
    outer += 1
    print("outer iteration:", outer)
    while inner < 3:
        inner += 1
        print("    inner iteration:", inner)
    inner = 0
    print("\n")
```

```
outer iteration: 1
    inner iteration: 1
    inner iteration: 2
    inner iteration: 3


outer iteration: 2
    inner iteration: 1
    inner iteration: 2
    inner iteration: 3


outer iteration: 3
    inner iteration: 1
    inner iteration: 2
    inner iteration: 3
```

In [ ]:
```
outer = 0
inner = 0
while outer < 5:
    outer += 1
    print('outer one:', outer)
    while
```

Using Nested Loops

Seeing how to use nested loops is great and all, but when do we really use them? Well, as we touched on earlier, nested data structures don't simply provide a convenient way to conceptualize nested loops, they provide a clear use case for them too.

Let's say we have a `list` of `dictionaries` that represent people. People can have attributes that also point to other collections, let's say their pets. So, if we wanted a way to list out the names of all people's pets, this would be a great opportunity to employ a nested loop. Let's take a look at an example.

In [5]:
```
programmers = [{'name': "rachel", 'favorite_languages': ['Ruby', 'JavaScript', 'S
               {'name': "daniel", 'favorite_languages': ['JavaScript', 'Elixir',
               {'name': "greg", 'favorite_languages': ['C#', 'CoffeeScript', 'R']
               {'name': "meryl", 'favorite_languages': ['C++', 'PHP', 'Swift']}
              ]
print(programmers)
```

```
[{'name': 'rachel', 'favorite_languages': ['Ruby', 'JavaScript', 'SQL', 'Jav
a']}, {'name': 'daniel', 'favorite_languages': ['JavaScript', 'Elixir', 'Pytho
n']}, {'name': 'greg', 'favorite_languages': ['C#', 'CoffeeScript', 'R']}, {'na
me': 'meryl', 'favorite_languages': ['C++', 'PHP', 'Swift']}]
```

So, if we wanted to take the above list of `programmers` and dynamically list out everyone's `favorite_languages` we would need to use two separate loops.

In [10]:
```
for programmer in programmers:
    for language in programmer['favorite_languages']:
        print(language)
```

```
Ruby
JavaScript
SQL
Java
JavaScript
Elixir
Python
C#
CoffeeScript
R
C++
PHP
Swift
```

It's possible to get this done without a nested loop, but it would require much more code and would not be nearly as efficient or semantic as the above solution. So, in cases where we're dealing with nested data structures, nesting loops becomes very useful.

Summary

In this lesson, we introduced nested loops. Nested loops are exactly what they sound like. A loop inside of another. Nested iteration is helpful when dealing with nested collections in cases where we would like to dynamically access and use nested data contained in these collections. Nested loops can quickly become hard to read and maintain, so, it is important to use them wisely and sparingly.