3.2.10 Repetition Statements

Seth asks if you and Tom have covered repetition statements yet. After looking at the dataset more closely, Seth thinks you'll be using repetition statements to retrieve data to complete the audit. Tom agrees and offers to walk you through how to create and use repetition statements.

When writing an algorithm that performs the same task over and over, it's a lot easier to write the code for the algorithm once, and then place that algorithm in a repetition structure that can be repeated as many times as necessary. This type of repetition structure is called a **loop**.

REWIND

Loops tell a computer to repeat lines of code over and over (and over) again and `for` loops tell the computer to repeat the lines of code a specific number of times.

There are two categories of repetition structures: condition-controlled loops and count-controlled loops.

A **condition-controlled loop** uses a true or false condition to control the number of times that it repeats, like asking a user if they want to continue with their online shopping after they add something to their "basket." This type of loop is also called a `while` loop.

A **count-controlled loop** repeats a specific number of times depending on the conditions, such as getting all the items in a list. This type of loop is also called a `for` loop.

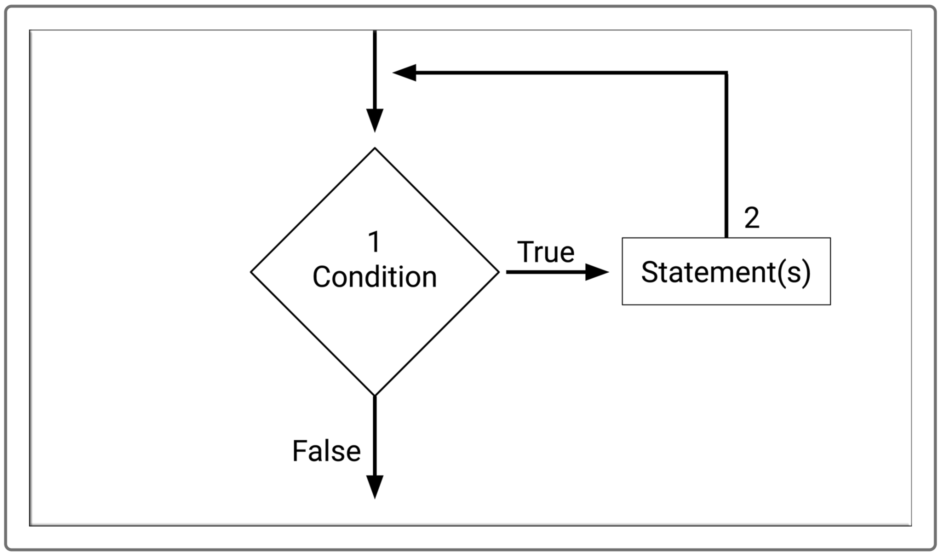
Let's go over each loop type in more detail.

While Loops

`while` loops test if a condition is true. If the condition is true, then the code will perform a task. This type of loop has two parts:

1. A condition that is tested for a true or false value.
2. A statement or statements that are repeated as long as the condition is true.

See the following flowchart that illustrates this process.



Let's practice creating and implementing a `while` loop. Create a new Python file in VS Code and add the following code to your file.

```
x = 0
while x <= 5:
    print(x)
    x = x + 1
```

Let's take a closer look at what this code is doing.

- We set the variable *x* = 0 before we enter the loop.
- We test if *x* is less than or equal to 5.
- If this condition is true, then we print the value of *x*.
- With `x = x + 1`, we increment *x* by 1.
- The condition is tested again

We repeat this process until the condition is false. When *x* is greater than 5, the loop stops.

CAUTION

> If you forget to write code inside the loop that makes the test condition false, the `while` loop will continue to run. This is called an **infinite loop.** An infinite loop continues to repeat until the program is interrupted, either by quitting tor exiting the program. Avoid writing infinite loops!

For Loops

`for` loops iterate, or run through, a program a specific number of times before it stops. A `for` loop can be written like `if` and `if-else` statements. Here's the general format:

```
for item in [items]:
    statement 1
    statement 2
```

Let's break this down step by step.

- There is a `for` statement, followed by an `item` or variable that is found in a list of items of an unknown number. The items can be a list, as in the previous code block example, or a tuple or dictionary.
- Upon the first iteration, the `item` will be the first item in a list. The second iteration will be the second item in the list, and so on.
- The statements that follow will be executed as long as the number of items has not been exhausted. When there are no more items in the list, the `for` loop stops.

Iterate Through Lists and Tuples

Let's practice creating and implementing a `for` loop by iterating through our list of counties. Add the following code to the Python_practice.py file.

```
for county in counties:
    print(county)
```

When we execute this code, the `county` variable is declared and set equal to the first item in the list of counties, "Arapahoe." Then we print the first item, "Arapahoe," to the screen. For the second iteration, the `county` variable is set equal to "Denver," and then "Denver" is printed. This process continues for the number of items in the list of counties.

```
Arapahoe
Denver
Jefferson
```

Python has a built-in function, `range()`, that simplifies the process of writing a `for` loop. The `range()` function creates an iterable object or a list. For example, if we had a list of numbers, we could print each number using a `for` loop like this:

```
numbers = [0, 1, 2, 3, 4]
for num in numbers:
    print(num)
```

When this code is executed, all the numbers in the list would be printed to the screen.

```
0
1
2
3
4
```

We will get the same output if we simplify the code by modifying the `for` loop, using the `range()` function:

```
for num in range(5):
    print(num)
```

Indexing can also be used to iterate through a list. If the list contains strings, we'll need to get the length of the list as an integer for the `range()` function. For example, to iterate through the counties list using the `range()` function, the code should be rewritten as follows:

```
for i in range(len(counties)):
    print(counties[i])
```

Let's break down what's happening in this code.

- The variable `i` is used to indicate the index, or the values 0, 1, and 2, in the length of the counties list. The letter `i` is often used for simplicity, but any variable can be used.
- Inside the `range()` function, we get the length of the list of counties, which is the integer 3.

- Then, we iterate through the list, where the variable `i` is equal to 0 for the first index. The 0 is passed into the `print(counties[i])` statement, where `i = 0`, and "Arapahoe" is printed.
- This process is continued until all three items, or counties, in the list are printed to the screen.

NOTE

> A variable being used to iterate through a `for` loop is chosen arbitrarily and could be anything, but it's a best practice to use a variable that makes sense in the current situation.

We can iterate through a tuple the same way we iterate through a list.

Retake

Iterate Through a Dictionary

We can use a `for` loop to iterate over a dictionary and get all the keys, all the values, or all the keys and values.

To practice getting the keys and values from a dictionary, we will use the counties dictionary that was created earlier:

`counties_dict = {"Arapahoe": 422829, "Denver": 463353, "Jefferson": 432438}`

Get the Keys of a Dictionary

To get only the keys from a dictionary using a `for` loop, the loop can be written like we are iterating over a list or tuple.

```
for county in counties_dict:
    print(county)
```

When we execute this code, we get the counties printed to the screen.

```
Arapahoe
Denver
Jefferson
```

We can also use the `keys()` method to iterate over a dictionary to get the keys. To the previous code, add the `keys()` method to the end of the `counties_dict`, like this:

```
for county in counties_dict.keys():
    print(county)
```

When using the `keys()` method, it doesn't matter what variable name we use in the `for` loop. The `keys()` method will print each key in order.

SKILL DRILL

Print each county from the `counties` dictionary using the `keys()` method.

Get the Values of a Dictionary

To get the values of a dictionary, we iterate over the dictionary using the `values()` method, just like we used the `keys()` method.

```
for voters in counties_dict.values():
    print(voters)
```

Also, when using the `values()` method, it doesn't matter what variable name we use in the `for` loop. The `values()` method will print each value in order.

REWIND

You can also use the format `dictionary_name[key]` to get the value for the key.

When using the format `dictionary_name[key]`, include the key `county` in the `for` loop in the print statement. This will return the value of the key in the output.

For the counties list, modify the `for` loop and use the key, `county` to reference the value.

```
for county in counties_dict:
    print(counties_dict[county])
```

Another method we can use to get the values of a key is to use the `get()` method on the dictionary in the `for` loop.

```
for county in counties_dict:
    print(counties_dict.get(county))
```

The output of both of these for loops will be:

```
422829
463353
432438
```

Get the Key-Value Pairs of a Dictionary

Finally, if we want to print the key-value pair of the dictionary, we use the `items()` method in the `for` loop, which follows this general format:

```
for key, value in dictionary_name.items():
    print(key, value)
```

When we use the `items()` method, we get the key and the value for each dictionary by referencing them as "key" and "value" as in the code above, or we can reference them by name, like "county" and "voters", as in the code below.

```
for county, voters in counties_dict.items():
    print(county, voters)
```

When we execute this code in the VS Code terminal, the output will be each key and value in order.

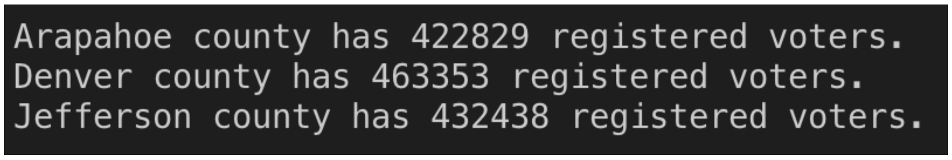
```
Arapahoe 422829
Denver 463353
Jefferson 432438
```

IMPORTANT

> When iterating over a dictionary:
>
> - The first variable declared in the `for` loop is assigned to the keys.
> - The second variable is assigned to the values.

SKILL DRILL

Print each county and registered voter form the `counties` dictionary so that the output looks like this:



```
Arapahoe county has 422829 registered voters.
Denver county has 463353 registered voters.
Jefferson county has 432438 registered voters.
```

Iterate Through a List of Dictionaries

A `for` loop can be used to iterate through a list of dictionaries like our `voting_data` list of dictionaries we created earlier. With a `for` loop we can:

- Retrieve each dictionary in the list
- Retrieve only the values of each dictionary
- Retrieve the key-value pairs of each dictionary

Get Each Dictionary in a List of Dictionaries

To print each dictionary in `voting_data`, use the standard format for iterating over the list of dictionaries with a `for` loop:

```
voting_data = [{"county":"Arapahoe", "registered_voters": 422829},
               {"county":"Denver", "registered_voters": 463353},
               {"county":"Jefferson", "registered_voters": 432438}]

for county_dict in voting_data:
    print(county_dict)
```

When we execute the code, each dictionary is printed on a separate line:

```
{'county': 'Arapahoe', 'registered_voters': 422829}
{'county': 'Denver', 'registered_voters': 463353}
{'county': 'Jefferson', 'registered_voters': 432438}
```

Since this is a list of dictionaries, use the `range()` function to iterate over the list.

Get the Values from a List of Dictionaries

To retrieve only the values from each dictionary in the list of dictionaries, we need to use a **nested** `for` loop. Let's see how this can be done with our `voting_data`.

First, use the `for` loop: `for county_dict in voting_data:` to retrieve each dictionary. Then, in the second `for` loop, use the `values()` method on the variable `county_dict` to reference the data in the second `for` loop in order to get each value.

```
for county_dict in voting_data:
    for value in county_dict.values():
        print(value)
```

When we execute this code in the VS Code terminal, the output is each value from each key:

```
Arapahoe
422829
Denver
463353
Jefferson
432438
```

Retake

If we only want to print the county name from each dictionary, we can use `county_dict['county']` in the print statement for the `for` loop.

```
for county_dict in voting_data:
    print(county_dict['county'])
```

When this code is executed, we retrieve each county in each dictionary.

```
Arapahoe
Denver
Jefferson
```