Lesson 6: Loops

Table of Contents

6.1. Objectives

- Examine the implications of using computation to solve a problem
  - Discuss best practices for using computation to solve a problem
  - Suggest types of problems that can be solved through computation

    - Show how computation can solve a problem

- Recognize key computer science concepts
    - Identify data types used in Python scripting
    - Identify data structures used in Python scripting
    - Define variables and strings
    - Recognize how queries operate

- Demonstrate the ability to build basic scripts using Python scripting language
    - Use various data types and structures in Python scripting
    - Collect data using Python scripting
    - Extract data using Python scripting
    - Develop advanced data structures using Python scripting

---

6.2. Overview

The following material is divided into the following parts:

- Lesson
- Guided Exercise
- Practical Exercises

Instructor Guidance: Refer back to Lesson 1 and relate the four steps of problem-solving using Computational Thinking (Decomposition, Pattern Recognition, Abstraction, & Algorithm Design) to lessons, exercises, examples, student questions/comments, etc., as appropriate throughout this lesson.

6.3. Review

6.3.1. Lists

Exercise: Create a list of at least 5 elements, then print the last three elements using slicing.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
my_list = [1, 2, 3, 4, 5, 'six']
my_list[-3:]
```

Exercise: Change the first element of the list you just created using indexing and assignment.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
my_list[0] = 100
my_list
```

Exercise: Use `.append()` to add an item to your list and then find its length using `len()`.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
my_list.append(24)
len(my_list)
```

Exercise: Use indexing to access the element `'here'` in the list below.

In [ ]:

```
my_list = [[1, 2, 3, 4], [5, 6], [7, 8, 'here', 10]]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
my_list[-1][2]
```

6.3.2. Sets

Exercise: Deduplicate the following list by casting it to a set.

In [ ]:

```
num_list = [1, 1, 2, 3, 6, 8, 1, 2, 5, 6, 8]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
num_set = set(num_list)
num_set
```

Exercise: Find the maximum and minimum values of the set you just created using the `max()` and `min()` functions.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
max(num_set), min(num_set)
```

6.3.3. Dictionaries

Exercise: Initialize the dictionary in the cell below. Then, in a separate code cell, add the following key-value pair to that dictionary: `{'Germany': '+49'}`

In [ ]:

```
country_codes = {'USA': '+1', 'UK': '+44', 'Mexico': '+52',
                 'India': '+91', 'China': '+86'}
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
country_codes['Germany'] = '+49'
country_codes
```

6.4. Lesson: Loops

6.4.1. For Loops

References:

- Python: The for statement (https://docs.python.org/3.4/reference/compound_stmts.html?highlight=while%20statement#the-for-statement)

The power of computation is the ability to do a lot of actions very quickly. Imagine you have thousands of data points to analyze. First, you write some `if` statements to intelligently parse, cleanse, and account for the data, perhaps looking at some individual data points as examples to guide you. Now you have a script that works. But how do you actually execute your code on each data point in the set?

Python makes iterating, or looping, through a collection of data points easy and intuitive. Python has a statement called a `for` loop that steps through a data collection, and allows you to execute code on its individual elements. A `for` loop allows us to, for example, run a set of `if` blocks on each of the data points in a large data set. The following is a template for a `for` loop in Python.

```
for item in collection:
    do_something()
```

Below, we start with a simple list of integers. The `for` loop takes each integer in the list in order and stores it temporarily in the variable `num`. This gives us the ability to operate on that variable in the indented space under the `for` statement. Because it temporarily holds each item in the list, `num` is called a *temporary variable*.

Let's begin by simply printing out each item in our list, one by one.

In [ ]:

```
num_list = [9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39]
```

In [ ]:

```
for num in num_list:
    print(num)
```

From here, we can add more complexity and begin using our temporary variable to perform calculations on each item in the list.

In [ ]:

```
num_list = [9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39]
```

In [ ]:

```
for num in num_list:
    new_num = num - 100
    print(num, new_num)
```

When you are working with data, you often want to store your processed data in new collections that you create. Above, all we did was print the data on the screen. What if we want to store it instead in another list? First, we need to create an empty list to store the new values. Then, we can use `.append()` inside of a `for` loop to add items to the new list.

In [ ]:

```
new_list = []

for num in num_list:
    new_list.append(num - 100)

new_list
```

Discussion Question: What is the variable `num` doing in the `for` loop above?

We could have accomplished the same thing by accessing each item in the list separately. But as you can see below, the code to do this is messier and more fragile than the `for` loop above. This method also requires us to know how many items are in our collection, whereas with a `for` loop, Python automatically stops at the end of the collection.

In [ ]:

```
new_list = []

new_list.append(num_list[0] - 100)
new_list.append(num_list[1] - 100)
new_list.append(num_list[2] - 100)
new_list.append(num_list[3] - 100)
new_list.append(num_list[4] - 100)
new_list.append(num_list[5] - 100)
new_list.append(num_list[6] - 100)
new_list.append(num_list[7] - 100)
new_list.append(num_list[8] - 100)
new_list.append(num_list[9] - 100)
new_list.append(num_list[10] - 100)

new_list
```

6.4.1.1. Temporary Variables

Python `for` loops require us to define a temporary variable to store each element from our sequence as it iterates through. We saw this above with `num`. Temporary variable names follow the same naming conventions as normal Python variables, and you can name them whatever you want. Just like with other variables, it helps to choose descriptive temporary variable names. You can put any code you want in a `for` loop's indented space, including conditionals and other `for` loops.

In [ ]:

```
animals_list = ['dog', 'cat', 'snake', 'rabbit', 'lion', 'mouse']
danger_animals = ['snake', 'lion']

for animal in animals_list:

    if animal in danger_animals:
        print(animal.upper() + '!')

    else:
        print(animal)
```

Exercise: Write a `for` loop to find all the countries in the list below that do not contain the letter `'a'`.

In [ ]:

```
countries = ['USA', 'Mexico', 'Canada', 'Belgium', 'Denmark', 'France',
             'Iceland', 'Italy', 'Luxembourg', 'Netherlands', 'Norway',
             'Portugal', 'UK', 'Greece', 'Turkey', 'Germany', 'Spain']
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
for country in countries:
    if 'a' not in country.lower():
        print(country)
```

Exercise: Create a new list of numbers by looping through the list below and multiplying each number by negative one (`-1`). Use `.append()` to add each new item to your new list.

In [ ]:

```
nums = [-91, -88, -85, -82, -79, -76, -73, -70, -67, -64, -61]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
new_list = []

for n in nums:
    new_n = n * -1
    new_list.append(new_n)

new_list
```

6.4.1.2. Looping Through Other Collections

We can use a `for` loop on any collection we've learned about so far, including dictionaries. Remember that Python treats strings as a sequence of characters, and dictionaries require use of the `.items()` function.

In [ ]:

```
my_string = '14:14:06.337121'

for item in my_string:
    print(item)
```

In [ ]:

```
my_tuple = (2018, 11, 1, 14, 15, 7, 833493)

for item in my_tuple:
    print(item)
```

In [ ]:

```
my_set = {(38.94, -77.03), (38.89, -76.99), (38.95, -76.95), (38.91, -76.98), (38.96, -76
for coord in my_set:
    print('coordinate:', coord)

    for item in coord:
        print('   degree: ', item)

    print('- ' * 15)
```

6.4.1.3. Looping through dictionaries with `.items()`

Looping through dictionaries is less straightforward than looping through other collections because dictionary items are not just singular data points. Each item in a dictionary consists of a key and value. There are a few different ways to loop through dictionaries, but the most helpful way is to use `.items()`. The `.items()` method works a lot like the `.keys()` and `.values()` methods we learned about in Lesson 3, but provides both the keys and the values. Using `.items()` allows us to store two temporary variables at the same time, one for the key and one for the value.

Syntax	Description
`for key, val in my_dict.items():`	Allows you to loop through `my_dict`, storing each entry's key in `key` and value in `val`

In [ ]:

```
my_dictionary = {'66828': 'Jackalyn E.', '89339': 'Sunil K.', '59887': 'Colin O.',
                 '94383': 'Marianna I.', '62172': 'Huong N.', '69578': 'Agrippa O.',
                 '36563': 'Zlatko J.', '35851': 'Kumari V.'}
```

In [ ]:

```
for k, v in my_dictionary.items():
    print('Name:', v)
    print('Employee ID:', k)
    print('-'*20)
```

Exercise: Loop through the dictionary below and print the keys and values where the value is greater than `5`.

In [ ]:

```
practice_dict = {'Z': 0, 'Y': 1, 'X': 2, 'W': 3, 'V': 4, 'U': 5,
                 'T': 6, 'S': 7, 'R': 8, 'Q': 9, 'P': 10, 'O': 11}
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
for let, num in practice_dict.items():
    if num > 5:
        print(let, '--', num)
```

6.4.1.4. Counting with a Dictionary

You can use `for` loops and dictionaries count up all of the items in a collection at once. Say you have a list of coordinates like the one below. We first need to initialize a new dictionary to hold our counts. Then, the logic put simply is: for each item in the list, 1) if the item is NOT in the dictionary yet, set the item as a key in a new dictionary entry with the value `1`; 2) if the item is already in the dictionary, add `1` to its current value.

In [ ]:

```
coords = [(17.06, 18.07), (17.01, 18.08), (17.04, 18.01), (17.05, 18.09), (17.09, 18.05),
          (17.03, 18.03), (17.05, 18.07), (17.04, 18.05), (17.02, 18.02), (17.08, 18.05),
          (17.03, 18.07), (17.04, 18.03), (17.04, 18.06), (17.01, 18.09), (17.03, 18.07),
          (17.03, 18.09), (17.08, 18.02), (17.08, 18.06), (17.08, 18.06), (17.02, 18.02),
          (17.05, 18.05), (17.07, 18.05), (17.01, 18.08), (17.05, 18.02), (17.09, 18.06),
          (17.07, 18.04), (17.09, 18.09), (17.05, 18.03), (17.02, 18.08), (17.03, 18.07),
          (17.06, 18.04), (17.08, 18.06), (17.04, 18.05), (17.02, 18.09), (17.03, 18.06),
          (17.08, 18.06), (17.04, 18.04), (17.08, 18.02), (17.05, 18.07), (17.07, 18.01),
          (17.04, 18.06), (17.06, 18.02), (17.04, 18.03), (17.01, 18.04), (17.04, 18.02),
          (17.03, 18.01), (17.06, 18.01), (17.04, 18.01), (17.06, 18.06), (17.09, 18.07)]
```

In [ ]:

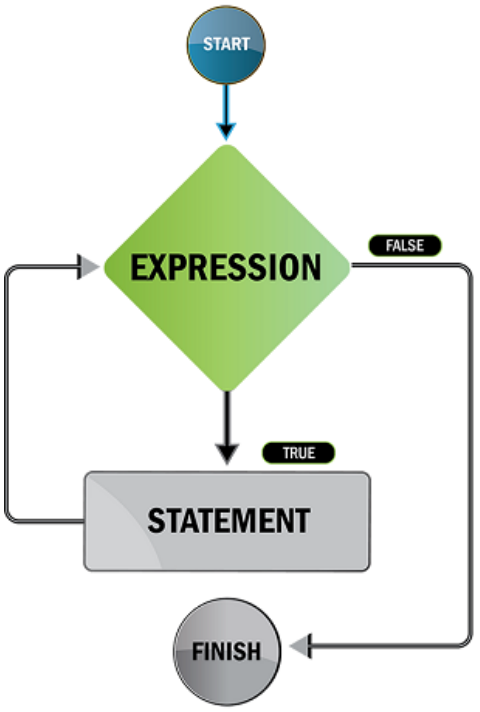
```
counts = {}

for coord in coords:
    if coord not in counts:
        counts[coord] = 1
    else:
        counts[coord] += 1

counts
```

6.4.2. While Loops

References:

- Python: The while statement (https://docs.python.org/3.4/reference/compound_stmts.html#while)
- TutorialsPoint: Python - Loops (https://www.tutorialspoint.com/python/python_loops.htm)

The `while` loop repeats while a given Boolean test expression is `True`. The `while` loop tests the Boolean test expression before executing the loop body.



The code below shows an example `while` loop. Once the Boolean expression becomes `False`, the loop is terminated. The `while` loop is different from the `for` loop in that it will keep looping until its condition evaluates to `False`, so it can run many times and is not limited by the number of items in a collection.

In [ ]:

```
count = 0
```

In [ ]:

```
while count < 10:
    print(count)
    count += 1
```

Just like in a `for` loop, we can put any code we want inside a `while` loop, including conditionals and other loops. For example:

In [ ]:

```
file_size = 0

while file_size < 10000000:
    file_size += 1

    if file_size == 5000000:
        print('file is 50% full...')

    if file_size == 7500000:
        print('file is 75% full...')


print('done writing.')
```

Above, we set `file_size` equal to zero. The `while` statement checks if the value of `file_size` is greater than `1000`. Until this statement becomes `False`, `file_size` increases by one. When `file_size` is equal to `5000000` and `7500000`, we print a message to the screen.

6.4.2.1. Infinite Loops

Use caution when writing `while` loops. Since a `while` loop will continue until its Boolean expression evaluates to `False`, it is possible for a loop to continue forever. Infinite loops usually break your script, and should be avoided. The `while` loop example below will execute infinitely, because its condition will always evaluate to `True`. We didn't put this in a code cell because running it could cause your notebook to freeze!

```
while 1 > 0:
    print('I am running!')
```

If you end up running an infinite loop in your notebook, you'll need to manually interrupt the process. The "Interrupt" option in the notebook's "Kernel" menu will attempt to interrupt any ongoing Python operations. If this option does not work, try using "Restart." This will restart the kernel and halt the execution of your loop.

> NOTE: Restarting your kernel wipes all variables from memory, so you will need to rerun any cells you need starting from the top of the notebook.

Exercise: Write a `while` loop. Be careful to avoid infinite loops!

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

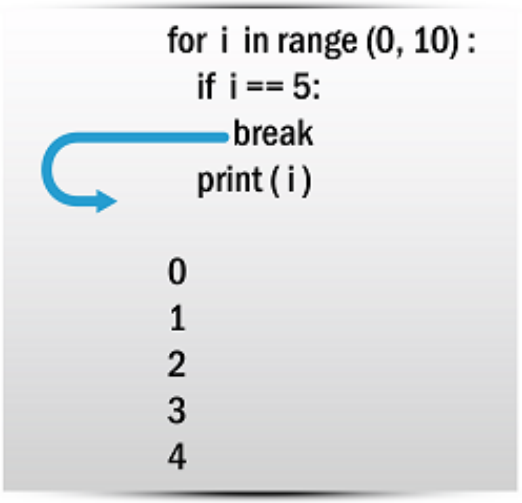
```
## INSTRUCTION SOLUTION(S) ##
number = 20

while number % 7 != 0:
    print(number)
    number -= 1
```

6.4.3. Break and Continue

References:

- Python: break and continue Statements, and else Clauses on Loops (https://docs.python.org/3.4/tutorial/controlflow.html#break-and-continue-statements-and-else-clauses-on-loops)
- TutorialsPoint: Python break, continue and pass Statements (http://www.tutorialspoint.com/python/python_loop_control.htm)

The `break` statement terminates the first `for` or `while` loop it is nested under. Programmers sometimes put a `break` statement in `while` loops they are developing to make sure they don't run infinitely. But `break` statements are useful in other cases too, such as terminating a `for` loop if a certain condition is met. The graphic below illustrates how the `break` statement affects the flow of your code.

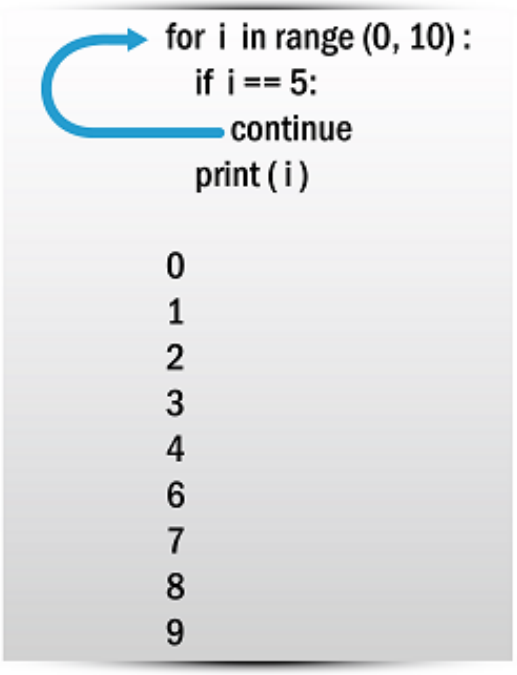


Break is effective when you are only looking for a certain number of things, and you don't know how long it will take to find them. The `for` loop below will hit the `break` statement after it finds 2 items that meet the criteria, breaking it out of the loop and skipping the rest of the IDs.

In [ ]:

```
all_ids = ['96193', '77954', '45162', '82362', '94513', '34251',
           '66835', '65384', '95875', '63816', '27472', '98119']

target_ids = []

for i in all_ids:

    if i[0] == '9':
        target_ids.append(i)

    if len(target_ids) >= 2:
        break

target_ids
```

Another flow control statement is `continue`. It skips any code following it in the `for` or `while` loop that contains it, and begins the next loop iteration. This is similar to `break`, but instead of terminating the entire loop it just terminates the current iteration of the loop and the loop continues to execute. The graphic below illustrates how the `continue` statement affects the flow of your code.



The `continue` statement, like `break`, can only affect `while` and `for` loops. The `continue` statement is useful when you want to skip items in a collection that meet certain criteria.

In [ ]:

```
all_ids = ['96193', '77954', '45162', '82362', '94513', '34251',
           '66835', '65384', '95875', '63816', '27472', '98119']

for i in all_ids:

    if i[0] == '9':
        continue

    print(i)
```

Instructor Guidance: Refer back to Lesson 1 and relate the four steps of problem-solving using Computational Thinking (Decomposition, Pattern Recognition, Abstraction, & Algorithm Design) as appropriate throughout these exercises.

6.5. Guided Exercise: Credit Card Data

We have been given data about credit card users in a dictionary. Each key is a user's name, and linked to that is another dictionary containing their credit card type and the number of transactions using their card. We want to accomplish the following:

1. Determine what types of cards are in the data set.
2. Create a dictionary that maps the card types to the number of people in the data set who use that card.
3. Find the card type in the data set with the most users.

In [ ]:

```
card_data = {'Albert B.': {'transactions': 58, 'type': 'Mastercard'}, 'Andrea M.': {'tran
             'Barbara S.': {'transactions': 69, 'type': 'Discover'}, 'Carolyn R.': {'tran
             'Cheryl S.': {'transactions': 30, 'type': 'Discover'}, 'Christopher R.': {'t
             'Connie G.': {'transactions': 90, 'type': 'Visa'}, 'Craig F.': {'transaction
             'Daniel G.': {'transactions': 84, 'type': 'Visa'}, 'Dawn D.': {'transactions
             'Dolores C.': {'transactions': 90, 'type': 'Mastercard'}, 'Douglas H.': {'tr
             'Edna W.': {'transactions': 162, 'type': 'Mastercard'}, 'Emma D.': {'transac
             'Fran W.': {'transactions': 49, 'type': 'Discover'}, 'Heath M.': {'transacti
             'Iona H.': {'transactions': 145, 'type': 'Discover'}, 'Janice P.': {'transac
             'Jenette P.': {'transactions': 68, 'type': 'Mastercard'}, 'Jennifer W.': {'t
             'John H.': {'transactions': 96, 'type': 'Discover'}, 'John L.': {'transactio
             'Joshua H.': {'transactions': 89, 'type': 'Visa'}, 'Kenneth L.': {'transacti
             'Leah E.': {'transactions': 167, 'type': 'Mastercard'}, 'Leona D.': {'transa
             'Linda B.': {'transactions': 62, 'type': 'Visa'}, 'Luz G.': {'transactions':
             'Mabel C.': {'transactions': 80, 'type': 'Mastercard'}, 'Marina L.': {'trans
             'Paul B.': {'transactions': 188, 'type': 'Discover'}, 'Randy E.': {'transact
             'Reynaldo M.': {'transactions': 197, 'type': 'Visa'}, 'Rose C.': {'transacti
             'Samuel P.': {'transactions': 28, 'type': 'Mastercard'}, 'Tammi J.': {'trans
             'Tammy A.': {'transactions': 93, 'type': 'Visa'}, 'Therese S.': {'transactio
             'Whitney D.': {'transactions': 140, 'type': 'Visa'}, 'William D.': {'transac
```

Problem 1: Determine what types of cards are in the data set.

Pseudocode:

1. Loop over the `card_data` dictionary (`.items()`)
2. For each card user:
    A. access the card type
    B. store the card type in a set

In [ ]:

```
card_types = set()

for name, user in card_data.items():
    card = user['type']
    card_types.add(card)

card_types
```

Problem 2: Create a dictionary that maps the card types to the number of people in the data set who use that card.

Pseudocode:

1. Create a `card_count` dictionary where each card type starts with a count of zero
    A. Loop over the `card_types` set
    B. Store each card type in a dictionary with the value of `0`

2. Loop over the `card_data` dictionary (`.items()`)
3. For each user:
    A. Access the card type
    B. Increment the value stored at that card type in `card_count` by one

In [ ]:

```
card_count = {}

for card in card_types:
    card_count[card] = 0

card_count
```

In [ ]:

```
for name, user in card_data.items():
    card = user['type']
    card_count[card] += 1

card_count
```

Problem 3: Find the card type in the data set with the most users.

Pseudocode:

1. Loop over the `card_count` dictionary (`.items()`)
2. For each card-count pair:
    A. If the count is the biggest one we've seen:
        a. Store the card
        b. Store the count as the new highest count we've seen

In [ ]:

```
biggest_count = -1

for card, count in card_count.items():
    if count > biggest_count:
        most_users = card
        biggest_count = count

most_users
```

Instructor Guidance: Refer back to Lesson 1 and relate the four steps of problem-solving using Computational Thinking (Decomposition, Pattern Recognition, Abstraction, & Algorithm Design) as appropriate throughout these exercises.

Instructor Guidance: The practical exercises deemed most important due to content and/or a cumulative result, which should be completed first in the interest of maximum training value in relation to time are Practical Exercises 1-3. Ensure you go over the exercise solutions and (as necessary) the processes to arrive at the solutions with the students.

Instructor Guidance: Follow-up questions are designed to be asked by the facilitators individually as each student completes the task and has it looked at by a facilitator.

6.6. Practical Exercises

6.6.1. Practical Exercise 1: Iterating through a List

Given a list of the integers 1 through 100, iterate through the list and print only those integers divisible by 3 and 5 (e.g. `15`, `30`, `45` ...). HINT: A number `N` is divisible by a number `M` if `N % M == 0` (refer back to Lesson 2's *Numbers and Arithmetic Operators* section for more help).

In [ ]:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
           21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40
           41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60
           61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80
           81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 10
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
for num in numbers:
    if (num % 3 == 0) and (num % 5 == 0):
        print(num)
```

6.6.2. Practical Exercise 2: Latitude or Longitude

Given the following list of geocoordinates in DMS format. Count how many latitudes and how many longitudes are in the list. HINT: You can determine whether the coordinate is a latitude or a longitude by its length or its direction character (`'N'` / `'S'` for latitude, and `'E'` / `'W'` for longitude)

Answer: 42 latitudes, 58 longitudes

In [ ]:

```
coordinates = ['165111S', '772936N', '1540314W', '524920N', '1382226W', '1494046W', '1273
               '1425335E', '743414N', '0105626E', '0403851E', '1000553W', '1441608W', '11
               '1254845E', '0202309W', '214335S', '0672053E', '0350757W', '1500421E', '02
               '384150N', '1411905E', '0893952E', '141245S', '842323S', '0613531E', '7149
               '144353S', '053704S', '783821S', '320622S', '1505342W', '450616N', '163481
               '1671010W', '0183120W', '0180436E', '1235710E', '1230421W', '1074359W', '1
               '1690909E', '615032S', '1623158W', '0423401E', '281842S', '471328N', '0293
               '1585044W', '112310N', '083548S', '332630N', '0060556E', '1615349E', '0510
               '1071918E', '0294522W', '644533S', '213823N']
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
lats = 0
lons = 0

for coord in coordinates:
    if ('N' in coord) or ('S' in coord):
        lats += 1

    elif ('E' in coord) or ('W' in coord):
        lons += 1

print('lats:', lats)
print('lons:', lons)
```

6.6.3. Practical Exercise 3: Lists in a List

Problem 1: Using indexing, access the value `'here'` in the list of lists below.

In [ ]:

```
numbers = [[5, 'here', 2], [4, 2, 5, 7], [4, 2, 8, 0, 1, 2]]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
numbers[0][1]
```

Problem 2: Using indexing, access the value `'here'` in the list of lists below.

In [ ]:

```
numbers = [[5, 4, 2], [4, 2, 'here', 7], [4, 2, 8, 0, 1, 2]]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
numbers[1][2]
```

Problem 3: Using indexing, access the value `'here'` in the list of lists below.

In [ ]:

```
numbers = [[5, 4, 2], [4, 2, 5, 7], [4, 2, 8, 'here', 1, 2]]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
numbers[2][3]
```

Problem 4: Using a nested `for` loop, access each number in the list below. For each number, if it's a `4`, print it out; otherwise do nothing.

In [ ]:

```
numbers = [[5, 4, 2], [4, 2, 5, 7], [4, 2, 8, 0, 1, 2]]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
for sublist in numbers:
    for num in sublist:
        if num == 4:
            print(num)
```

Problem 5: Using a nested `for` loop, calculate the sum of all the numbers in the list below. HINT: The answer is 76.

In [ ]:

```
numbers = [[5, 4, 2], [4, 2, 5, 7], [4, 2, 8, 0, 1, 2], [6, 8, 6, 8, 2]]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
total = 0
for sublist in numbers:
    for num in sublist:
        total += num

total
```

6.6.4. Practical Exercise 4: Sets in a List

You are given a list of sets below. Create the union of every set that has exactly 1 or exactly 2 elements. Note that your answer may display in different order than the answer below.

Bonus: How would you check whether your answer matches ours using Python?

Answer: `{'q', 198, 'T', 'V', 784, 'N', 404, 342, 'g', 282, 155, 'b', 92, 411, 481, 'e', 551, 684, 438, 247, 827, 'F'}`

In [ ]:

```
setlist = [{718, 176, 'A', 'N', 'y'}, {358, 71, 332, 'D', 'G'}, {25, 'P', 'D'}, {'I', 'j'
           {'J', 131, 8, 914, 693}, {737, 'f', 941, 926}, {961, 'o', 'J'}, {233, 410, 405
           {'d', 411, 'm', 511}, {865, 'N', 'I'}, {'T', 'e'}, {233, 395, 'l'}, {481, 'N'}
           {312, 1, 645}, {'C', 901, 'T', 'p', 316}, {'a', 'C', 'S'}, {720, 'u', 'S', 822
           {198, 551}, {438, 'g'}, {'r', 'B', 'D', 'k', 'f'}, {323, 668, 965, 739}, {'a',
           {695, 'S', 711}, {784}, {'T', 282}, {342}, {827, 404}, {707, 'p', 787, 'b', 66
           {467, 645, 'd', 'l'}, {'V'}, {'b'}, {420, 'j', 'F'}, {684, 92}, {'c', 840, 'j'
           {'T', 'u', 564, 'B'}, {411}, {'Z', 'e', 't', 'V'}]
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
my_set = set()
for s in setlist:
    if len(s) in {1, 2}:
        my_set = my_set.union(s)

print(my_set)
my_set == {'q', 198, 'T', 'V', 784, 'N', 404, 342, 'g', 282, 155, 'b', 92, 411, 481, 'e',
```

6.6.5. Practical Exercise 5: Crime Data

Problem 1: Loop over the dictionary below. Print out each date along with how many crimes were reported on that date.

Bonus: Create a new dictionary to store this information.

In [ ]:

```
crime_dates = {'07-11-2016': ['110716AT001', '110716AT010', '110716DN011', '110716DN014',
                              '110716NA039', '110716NA049', '110716NA051', '110716NA054',
               '08-11-2016': ['110816AT003', '110816AT006', '110816AT014', '110816DN004',
                              '110816NA028', '110816NA039', '110816NA040', '110816NA049',
               '09-11-2016': ['110916AT000', '110916DN009', '110916DN019', '110916DN021',
                              '110916NA001', '110916NA036', '110916NA046', '110916RB004',
               '10-11-2016': ['111016DN013', '111016DN015', '111016DN022', '111016DN036',
                              '111016NA029', '111016NA040', '111016RB010', '111016RB014',
               '11-11-2016': ['111116AT004', '111116DN008', '111116DN015', '111116DN030',
                              '111116NA031', '111116RB005', '111116RB012', '111116TF004',
              }
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
for date, crimes in crime_dates.items():
    print(date, ':', len(crimes))

## BONUS SOLUTION ##
num_crimes = {}
for date, crimes in crime_dates.items():
    num_crimes[date] = len(crimes)
num_crimes
```

Problem 2: Loop over the dictionary below. Count the number of times each coordinate appears in the data set. Store this information in a dictionary where the coordinate is the key and the appearance count is the value. HINT: See this Lesson's Appendix for help on counting using a dictionary.

In [ ]:

```
crime_coords = {'110716AT001': (39.29, -76.65), '110716AT010': (39.29, -76.64), '110716DN
                '110716NA003': (39.29, -76.68), '110716NA023': (39.29, -76.68), '110716NA
                '110716RB004': (39.28, -76.65), '110716RB008': (39.30, -76.66), '110716TF
                '110816AT014': (39.29, -76.68), '110816DN004': (39.28, -76.64), '110816DN
                '110816NA028': (39.28, -76.65), '110816NA039': (39.29, -76.66), '110816NA
                '110816TF015': (39.29, -76.65), '110816TF032': (39.30, -76.67), '110816TF
                '110916DN019': (39.30, -76.64), '110916DN021': (39.30, -76.67), '110916DN
                '110916NA001': (39.30, -76.65), '110916NA036': (39.30, -76.65), '110916NA
                '110916TF013': (39.30, -76.64), '110916TF017': (39.29, -76.65), '110916TF
                '111016DN022': (39.29, -76.66), '111016DN036': (39.29, -76.64), '111016DN
                '111016NA029': (39.30, -76.65), '111016NA040': (39.28, -76.65), '111016RB
                '111016TF015': (39.30, -76.65), '111016TF024': (39.28, -76.64), '111016TF
                '111116DN030': (39.30, -76.65), '111116DN033': (39.28, -76.65), '111116DN
                '111116RB005': (39.28, -76.67), '111116RB012': (39.30, -76.67), '111116TF
               }
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
coord_count = {}
for crime_id, coord in crime_coords.items():
    if coord not in coord_count:
        coord_count[coord] = 1
    else:
        coord_count[coord] += 1

coord_count
```

Problem 3: Using the dictionary you just created, programmatically find the coordinate that appeared most in the data set. HINT: The answer is `(39.29, -76.65)` .

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
biggest_count = -1

for coord, count in coord_count.items():
    if count > biggest_count:
        biggest_count = count
        target_coord = coord

print(target_coord)

## ALTERNATE SOLUTION ##
biggest_count = max(coord_count.values())
for coord, count in coord_count.items():
    if count == biggest_count:
        print(coord)
        break
```

6.6.6. Practical Exercise 6: Shortest and Longest Words

Problem 1: Use a loop to find the length of the shortest word in the set below. HINT: The answer is 2.

In [ ]:

```
words = {'Rick', 'Omar', 'William', 'Brenda', 'Susan', 'Marilyn', 'Fritz', 'Jesse', 'June
         'Gary', 'Ryan', 'Larry', 'David', 'Hertha', 'Earl', 'Jared', 'Robert', 'Cheryl',
         'Jose', 'Chad', 'Irwin', 'Sammy', 'Karl', 'Eric', 'Coleen', 'Kaitlin', 'Lee', 'J
         'Jill', 'Clay', 'Troy', 'Gerald', 'Richard', 'Heather', 'Kim', 'James', 'Lindsay
         'Mildred', 'Valentina', 'Paula', 'Rachel', 'Clinton', 'Joseph', 'Quintin', 'Devi
         'Lorena', 'Jennefer', 'Bernadine', 'Kyle', 'Beverly', 'Laurel', 'Leonard', 'Ferm
         'Amos', 'Cornelius', 'John', 'Karen', 'Howard', 'Angela', 'Linda', 'Gwendolyn',
         'Harold', 'Darnell', 'Mary', 'Gordon', 'Ada', 'Isidro', 'Jack', 'Lynn', 'Sana',
         'Antoinette', 'Alejandro'}
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
min_length = len(words.pop())

for word in words:
    if len(word) < min_length:
        min_length = len(word)

min_length
```

Problem 2: Use a loop to find the length of the longest word in the set below.

In [ ]:

```
words = {'Rick', 'Omar', 'William', 'Brenda', 'Susan', 'Marilyn', 'Fritz', 'Jesse', 'June
         'Gary', 'Ryan', 'Larry', 'David', 'Hertha', 'Earl', 'Jared', 'Robert', 'Cheryl',
         'Jose', 'Chad', 'Irwin', 'Sammy', 'Karl', 'Eric', 'Coleen', 'Kaitlin', 'Lee', 'J
         'Jill', 'Clay', 'Troy', 'Gerald', 'Richard', 'Heather', 'Kim', 'James', 'Lindsay
         'Mildred', 'Valentina', 'Paula', 'Rachel', 'Clinton', 'Joseph', 'Quintin', 'Devi
         'Lorena', 'Jennefer', 'Bernadine', 'Kyle', 'Beverly', 'Laurel', 'Leonard', 'Ferm
         'Amos', 'Cornelius', 'John', 'Karen', 'Howard', 'Angela', 'Linda', 'Gwendolyn',
         'Harold', 'Darnell', 'Mary', 'Gordon', 'Ada', 'Isidro', 'Jack', 'Lynn', 'Sana',
         'Antoinette', 'Alejandro', 'Christophe'}
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
max_length = 0

for word in words:
    if len(word) > max_length:
        max_length = len(word)

max_length
```

Problem 3: Loop through the set above again. This time, append any word that is the same length as the longest word to a new list. HINT: Your final list should have two words in it.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
big_words = []
for word in words:
    if len(word) == max_length:
        big_words.append(word)

big_words
```

6.6.7. [Challenge] Practical Exercise 7: Digit Degree

Let's define the digit degree of some positive integer as the number of times we need to replace this number with the sum of its digits until we get to a one-digit number. Write a script to find the digit degree of any positive integer.

Example Input	Expected Output
5	0
100	1
91	2
289	3

Example: Digit degree of 289

```
289 ⟹ 2 + 8 + 9 ⟹ 19
19 ⟹ 1 + 9 ⟹ 10
10 ⟹ 1 + 0 ⟹ 1 (degree = 3) (we've reached a one-digit number, so we stop)
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
n = 289
degree = 0

while len(str(n)) > 1:

    digit_sum = 0
    for digit in str(n):
        digit_sum += int(digit)

    n = digit_sum
    degree += 1

degree
```

6.6.8. [Challenge] Practical Exercise 8: Common Character Count

Given two strings find the total number of common characters between them (case sensitive). For example, if the two strings are `'abda'` and `'ada'` the common character count should be `3` (the strings have three characters in common: two `'a'` s and one `'d'`).

Example Input	Expected Output
`s1 = 'aaa'` `s2 = 'aa'`	`2`
`s1 = 'aabcc'` `s2 = 'adcaa'`	`3`
`s1 = 'abcabc'` `s2 = 'xyzmtf'`	`0`
`s1 = 'zzzzZa'` `s2 = 'zzzzzzzaaa'`	`5`

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
s1 = 'zzzzZa'
s2 = 'zzzzzzzaaa'

result = 0
for letter in set(s1):
    if letter in s2:
        count1 = s1.count(letter)
        count2 = s2.count(letter)
        result += min(count1, count2)

result
```

6.6.9. [Challenge] Practical Exercise 9: Guessing Game

References:

- Python: The Random Library's `.randint()` function (https://docs.python.org/3/library/random.html#random.randint)

Create a game to guess the number that the computer randomly generates.

Problem 1: Using the code given below, generate a random number between 1 and 9 inclusive. For more information, see the documentation (https://docs.python.org/3/library/random.html#random.randint). Store your own guess in the `guess` variable, then write an `if` block to print whether you were correct or not.

In [ ]:

```
import random

number = random.randint(1, 9)
guess = 8
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
if guess < number:
    print('Too low!')

elif guess > number:
    print('Too high!')

else:
    print('You got it!')
```

Problem 2: You can use the `input()` function to capture a guess from the player using the code below. Copy your code from above into the cell below and try it out.

In [ ]:

```
number = random.randint(1, 9)
guess = int(input('Enter a number 1 through 9: '))
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
if guess < number:
    print('Too low!')

elif guess > number:
    print('Too high!')

else:
    print('You got it!')
```

Problem 3: Take all the code from the problem above and nest it under a `while` loop. The idea is to keep asking for guesses until the player gets it right.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

In [ ]:

```
## INSTRUCTION SOLUTION(S) ##
number = random.randint(1, 9)
guess = int(input('Enter a number 1 through 9: '))

while guess != number:
    if guess < number:
        print('Too low!')

    elif guess > number:
        print('Too high!')

    guess = int(input('Enter a number 1 through 9: '))

print('You got it!')
```

6.7. Appendix

Looping Keywords

Keyword	Description
`for`	Initializes a `for` loop, which continues through every item in a collection
`while`	Initializes a `while` loop, which continues until a stop condition is met
`break`	Exits the nearest `for` or `while` loop it is nested under
`continue`	Exits the current iteration of the nearest `for` or `while` loop it is nested under

Looping Through a Dictionary

Syntax	Description
`for key, val in my_dict.items():`	Allows you to loop through `my_dict`, storing each entry's key in `key` and value in `val`