Python Loops and Data

Introduction

Welcome to Day 6 of QuickPass! Today, we'll explore loops, a fundamental concept in programming, and dive into a bit of data manipulation (moving data around, presenting different ways, and changing types). Loops allow us to perform repetitive tasks, and allow us to methodically work with data without a lot of code. Let's get started!

Table of Contents

Loops

- Python: 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.

A basic `for` loop will have this structure:

```
for item in collection:  # item is a temporary variable that is assigned on each iteration of the loop
    do_something()
```

Loops are used to execute a block of code repeatedly. That is what the `do_something()` represents in the example above.

Here are examples of different types of `for` loops in Python along with explanations for their use cases:

`for i in x` (for element in iterable)

In [ ]:
```
urls = ["www.nike.com", "www.reddit.com", "www.cnn.com", "www.dell.com", "www.apple.com", "www.facebook.com"]

for url in urls:
    if "apple" in url or "dell" in url:
        print(f"Computer site: {url}")
    elif "reddit" in url or "facebook" in url:
        print(f"Social Media site: {url}")
    else:
        print(f"Other site: {url}")
```

```
Other site: www.nike.com
Social Media site: www.reddit.com
Other site: www.cnn.com
Computer site: www.dell.com
Computer site: www.apple.com
Social Media site: www.facebook.com
```

Use Case: This loop iterates through the elements of a list or any iterable. It's suitable when you want to perform the same operation for each item in the iterable. In this case, it's using conditional logic to print each website URL and category of site.

`for i in range(start, stop, step)`

This type of loop is useful when you want to contol the number of iterations (and even *how* its iterating). The `range()` method is quite customizable to loop in various ways.

Arguemnt	Description
start (optional)	integer starting from which the sequence of integers is to be returned. Defaults to `0`
stop	integer before which the sequence of integers is to be returned. The range of integers end at stop – 1.
step (optional)	integer value which determines the increment between each integer in the . Defaults to `1`

In [ ]:
```
for i in range(1, 20, 2):
    print(i)
```

```
1
3
5
7
9
11
13
15
17
19
```

Use Case: The `for i in range` loop is used when you need to iterate a specific number of times, often with a loop counter (`i`) that increments by a constant value (`step`). It's useful for tasks where you need to control the number of iterations and when you don't necessarily need to access elements in a collection.

`for idx, item in enumerate(iterable)`

In [ ]:
```python
fruits = ["Bananas", "Apples", "Cherries"]
for index, fruit in enumerate(fruits):
    print(f"Index {index}: {fruit}")
```

```
Index 0: Bananas
Index 1: Apples
Index 2: Cherries
```

Use Case: The `for idx, item in enumerate` loop allows you to access both the elements and their indices within an iterable. This is helpful when you need to keep track of the position of each element in the collection.

for key, value in dict.items()

In [ ]:
```python
phones = {"Francisco": "555-555-5555", "Alex": "222-222-2222", "Allison": "111-111-1111"}
for name, number in phones.items():
    print(f"{name}'s phone number is {number}")
```

```
Francisco's phone number is 555-555-5555
Alex's phone number is 222-222-2222
Allison's phone number is 111-111-1111
```

Use Case: The `for key, value in dict.items()` loop is used to iterate through the key-value pairs in a dictionary. It's beneficial when you need to work with both keys and values from a dictionary simultaneously.

When to Choose

Syntax	Use Case
`for i in x`	when you want to iterate through the elements of a collection
`for i in range`	when you need a loop with a specific number of iterations
`for idx, item in enumerate`	when you need to access both elements and their indices within an iterable
`for key, value in dict.items()`	when working with key-value pairs in a dictionary

Choose the appropriate `for` loop based on your specific task and what you need to achieve in your code. Each type of loop serves a different purpose and offers advantages depending on the situation.

`While` Loop

The `while` loop repeatedly executes a block of code as long the condition being evaluated at each iteration is `True`.

Be careful with while loops. You can easily start an infinite loop with a typo

In [ ]:
```python
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.')
```

```
file is 50% full...
file is 75% full...
done writing.
```

Concerning the risk for infinite loops, what would happen if we accidentally decremented `file_size`?

`Break` and `Continue`

- TutorialsPoint: Python break, continue and pass Statements

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.

In [ ]:
```python
cookies = 9

students = ['Aaron', "Nick", "Rod", "Chad", "Francisco", "Adam", "Tish", "Tom", "Guillermo", "Kevin"]

for student in students:
    if cookies > 0:
        print(f"{student} gets a cookie.")
        cookies -= 1
    else:
        print("Out of cookies!  Better luck next time!")
        break
```

```
Aaron gets a cookie.
Nick gets a cookie.
Rod gets a cookie.
Chad gets a cookie.
Francisco gets a cookie.
Adam gets a cookie.
Tish gets a cookie.
Tom gets a cookie.
Guillermo gets a cookie.
Out of cookies!  Better luck next time!
```

The `continue` statement is primarily used to skip items in a collection when a certain condition is met.

In [ ]:
```python
all_ids = ['961093', '7794554', '451762', '8236202', '94345513', '3426551',
           '686835', '6532384', '9587523', '6381456', '27234472', '9886119']

for i in all_ids:

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

    print(i)
```

```
7794554
451762
8236202
3426551
686835
6532384
6381456
27234472
```

Loops with Data Structures

Let's explore how loops can be used with various data structures.

Strings

Strings are an interable data structure. You can loop through characters in a string using a `for-in` loop.

In [ ]:
```python
text = "Hello, World!"
for char in text:
    print(char)
```

```
H
e
l
l
o
,

W
o
r
l
d
!
```

Lists

Loops are often used to iterate through the elements of a list.

In [ ]:
```python
auto_brands = ["Chevrolet", "Tesla", "Jeep", "Rivian", "Ford", "Porsche"]
for brand in auto_brands:
    print(brand)
```

```
Chevrolet
Tesla
Jeep
Rivian
Ford
Porsche
```

Tuples

Similar to lists, you can use loops to iterate through elements in a tuple.

In [ ]:
```python
coordinates = (3, 4)
for coordinate in coordinates:
    print(coordinate)
```

```
3
4
```

Dictionaries

Dictionaries are iterable, and you can loop through keys, values, or key-value pairs.

In [ ]:
```python
person = {"name": "Alice", "age": 25, "occupation": "Software Engineer", "salary": 95000, "division": "R&D"}
for key in person:
    print(f"This is the key: {key.upper()} - and this is the value: {str(person[key]).upper()}")
```

```
This is the key: NAME - and this is the value: ALICE
This is the key: AGE - and this is the value: 25
This is the key: OCCUPATION - and this is the value: SOFTWARE ENGINEER
This is the key: SALARY - and this is the value: 95000
This is the key: DIVISION - and this is the value: R&D
```

In [ ]:
```python
for k, v in person.items():
    print(k, v)
```

```
name Alice
age 25
occupation Software Engineer
salary 95000
division R&D
```

Nesting Loops

SNesting loops is running a loop inside of another loop. Typically, nested loops are used to iterate through nested data collections, such as lists inside of lists, or lists inside of dictionaries. Below is what a nested loop would look like:

In [ ]:
```
nested_list = [[1,2,3],[4,5,6]]

for row in nested_list:
    for item in row:
        print(item)
```

```
1
2
3
4
5
6
```

Data Manipulation

So far, you've learned about the basic foundational concepts of Python, to include the basic data types, data collections, functions, and loops. Now we are going to start combining some of those tools to manipulate data and provide insights (and good practice!)

Data Presentation

Using some of the basic tools we learned about, we can use Python to take a data structure and present the data it contains in a visually appealling way. We can, for example, use loops to creatively iterate over a data structure, perform an operation on the element, and add some way to logically divide the data so it prints out neatly. Check out this example:

In [ ]:
```
my_set = {(34.944556, -172.035656), (37.8948456, -45.9933284), (36.9569793, -77.9512123), (32.9189643, 123.9879538), (42.96127474, -23.993473123)}

for coord in my_set:
    print('coordinate:', coord)

    for item in coord:
        item = round(item, 2)
        print('   rounded degree: ', item)

    print('- ' * 15)
```

```
coordinate: (36.9569793, -77.9512123)
   rounded degree:  36.96
   rounded degree:  -77.95
- - - - - - - - - - - - - - -
coordinate: (37.8948456, -45.9933284)
   rounded degree:  37.89
   rounded degree:  -45.99
- - - - - - - - - - - - - - -
coordinate: (34.944556, -172.035656)
   rounded degree:  34.94
   rounded degree:  -172.04
- - - - - - - - - - - - - - -
coordinate: (42.96127474, -23.993473123)
   rounded degree:  42.96
   rounded degree:  -23.99
- - - - - - - - - - - - - - -
coordinate: (32.9189643, 123.9879538)
   rounded degree:  32.92
   rounded degree:  123.99
- - - - - - - - - - - - - - -
```

Data Manipulation Exercises

Exercise 1: Given this list of coordinates, format them with a decimal point. In this listing, latitude coordinates (denoted with `N` or `S` at the end) have 2 digits with 4 decimal degrees, and longitude has 3 digits with 4 decimal degrees. Coordinates in your result should look like this: `36.4256N` or `120.5726W`. Save the new coordinates in a new list and print them out.

In [ ]:
```
coordinates = ['001251S', '1458145W', '751609S', '0702701W', '169501N', '1618989W', '558547S', '0214097W', '835864N',
               '1268796E', '832255N', '0327724E', '564026S', '0010394W', '095331N', '0331801E', '268950N', '1494330E',
               '807760S', '0205861E  ', '738919N   ', '0664699W', '605174S  ', '0166332W', '219956S   ', '0925232E', '183798S',
               '0912672W', '728902N', '1511569E', '882030N', '1771585E', '777760N', '0886417W', '872853S', '1013176E',
               '381111N', '0790259E', '408154N', '1678905W', '184926S', '0466500E', '534746S', '1439038E', '722829S',
               '  0956044E   ', '158121S', '1355671W', '726763S', '1233750W', '047979N', '0746807W', '692209N', '0244100E',
               '396767S', '1130952W', '577572N', '1075059E', '083091S', '0502460W', '208663N', '0512649E', '446746N',
               '0943990W', '748701S', '1670451E', '564214N', '0838180E', '891697N', '1355671W', '726763S', '0173152E',
               '415852N', '1770667E', '712605S', '0808857E', '005686S', '0675283W', '023440S', '0565808W', '123331N',
               '0169811E', '884606S', '1699401E', '728619S', '0153910E', '493564S', '1381804E', '040780S', '0381086W',
               '   294066S', '  0768541W', '874393S', '1279228E', '640463S', '1423572W', '346219N', '1248809E', '316762S',
               '1545964W', '343856S', '1177982W', '798425S  ', '0253498E', '460875N', '0486881W  ', '811978S', '0317535W',
               '  445757N', '0163294E', '712272S', '0926236E', '556458N  ', '1134517W', '223318S', '0909808W', '725175N',
               '1482888W', '810648S  ', '0983814W', '128615N', '0918245W', '648537N', '  1660775E  ', '099212S', '1613558W',
               ' 169083N', '0069521E', '494381S', '0821162E', '323002S', '  0895635E  ', '405559N', '1364162E', '781623N   ',
               '  1155485W', '294612S', '0846210E', '279074N', '0617201W', '767002S', '0568345E', '598954S', '1166391E',
               '556746N', '1193564W', '865104N', '0741456E', '770912S', '0018400W', '643547N', '0000984E', '721042N',
               '1140391W', '282575N', '1429943W', '009296S', '0122358W  ', '021803S', '1537587W', '817594S', '1304475E   ',
               '462865N', '1066145W  ', '554805S', '1129959W', '217818S', '1137339E', '338721N', '1781291W', '081507N',
               '1020942W', '578799S', '1755547E', '356625N', '1075063E', '330831N', '0408577W', '413240N', '0352458W',
               '   421590S', '0611427W', '808923S  ', '1677702E', '437655S', '1610795W', '747199S', '0260411E', '426133N',
               '1739195E', '181808N', '0479451E', '565707N', '0552191E', '199290S', '1706089W', '185739N', '1291445E',
               '102829S', '  1049376E', '090498N', '0553653E', '491786N   ', '0123907E', '426234S', '0112585W', '891007N',
               '1149042W', ' 410566N', '0231040E', '125391N', '   1158478E  ', '155634S', '0133022W', '466321S', '1571240E',
               '444866S  ', '1669398E', '312817N', '1328573W', '679456S', '0223912E', '018167S', '0322769E', '359070S',
               '0144414W', '  308863S  ', '0040646E', '320359S', '1581091E', '634580S', '1559339E', '610337N', '0081672E',
               '233164S', '0401213E', '787008N', '1684622W', '160913S  ', '0519766W', '767041N', '0891031W', '412772N',
               '1070008W  ', '454291S', '1535091E', '897581N', '1158477W', '846255S', '0101984E', '498480S', '1558721W',
               '286832N', '0178005W', '828046S', '1681195E', '817740S', '0831152E', '321680S', '0560485W', '108417N',
               '0817694W', '569108N', '0859276W', '869007S', '1200099W', '688774N', '0752857W', '590307N', '0390206E',
               '     146336N', '1755119E', '284822S', '1530525E', '441923S', '0179598W', '218161N', '1290153E', '869916S',
               '0380798W', '454267S', '1179553E', '  282461S  ', '1133703W', '373709S', '1134497W', '877272S', '0276842W',
               '419035S', '1610104W', '031060S', '0996233E', '444315S', '0283348W', '346425S', '0915321W', '631689N',
               '0222680E', '773830S', '0659182E', '502604N', '1650731W', '043356N', '0904925E', '068210S', '0027693W',
               '310147S   ', '  1597000W  ', '624352N', '0802110W', '678573S', '1185367W', '673065S', '1545794W', '074848N',
               '   0565096W', '601615S', '1111452W', '230833S', '0412381W', '080455N', '0063638W', '052513N', '1124680E',
               '871211S', '0766947E', '324174S', '0024447E', '283259N', '0887907E', '288460S', '1236934E', '304827S',
               '1147140E', '098914N', '0893373E', '239367S', '0880325E', '014455S', '0310027E', '654170S', '0146584E',
               '211451N', '1620227W', '051343S', '0406677E', '286805N   ', '1328876E', '225997S', '0428105E', '473867S',
               '0779615E', '758910S', '0587889E', '126936S', '1402894W', '159310N', '1296442W', '494509N', '0034254E',
               '661676N', '1345096E', '528086S', '0166528W', '792654S', '0986073E', '357699S', '0822735W', '444064S',
               '1300579W', ' 825756S  ', '1062958W', '745455S', '1212493E', '358371N', '1355671W', '726763S', '0478974E',
               '146651N', '0785372E', '215280N', '1329565W', '362277S', '0954376E', '188458N', '0562158E', '188938S',
               '    0485582W  ', '230565N', '0870492E', '636377S', '1018174E', '671545N', '0160212W', '237370N', '0870965E',
               '640517N', '1692813W', '1355671W', '726763S']
```

Exercise 2: Using the coordinate results, combine the adjacent latitudes and longitudes into tuple coordinate pairs. The result should look like this `[('36.4256N', '120.5726W'), ('12.6523S', 024.3075E), ...]`. Save the resulting list into a new list.

In [ ]:
```
### Your code goes here
```

Exercise 3: Using the list of tuple coordinate pairs, save each one into a dictionary with keys formatted like so: `{"Location 1": ('36.4256N', '120.5726W'), 'Location 2': ('12.6523S', 024.3075E), ...}`

In [ ]:
```
## Your code goes here
```

Stretch Goal: There are a few repeat coordinate pairs. Remove them from the dictionary.

In [ ]:
```
## Your code goes here
```

Conclusion

Today, we've delved into loops, an essential tool for performing repetitive tasks, and functions, which help us organize and reuse code. You've learned about different types of loops and how to use them with various data structures.

Keep practicing loops and functions as they are fundamental building blocks for creating more complex Python programs.

Assignments

- Loops and Loops
- Space X countdown
- FizzBuzz
- Camel Case