

Intro to Python - Day 3

joseph.rejive

November 2018

So far, the main data structure we've talked about is a list. In this lecture, we'll talk about sets and dictionaries.

1 Sets

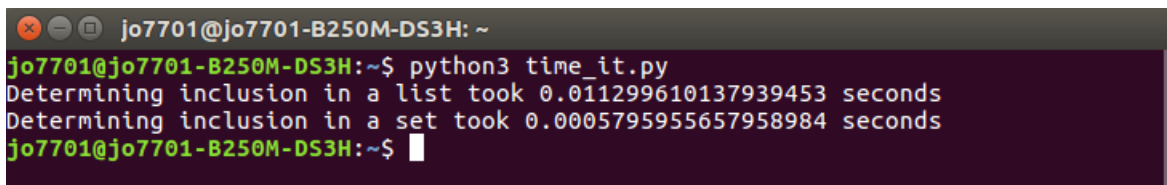
A set is an unordered collection of data where every element is unique. The advantages of a set is that it can quickly determine whether an element is present in the set. Lets look at the following code to verify this:

```
import time

my_list = [4]*100+[5] #creates a list where the first 100 numbers are 4 and the last number is 5
start = time.time()
for x in range(10000):
    a = 5 in my_list
print("Determining inclusion in a list took", time.time() - start, "seconds")

my_set = {*range(101)} #this creates a set that contains the numbers 0 to 100
start = time.time()
for x in range(10000):
    a = 5 in my_set
print("Determining inclusion in a set took", time.time() - start, "seconds")
```

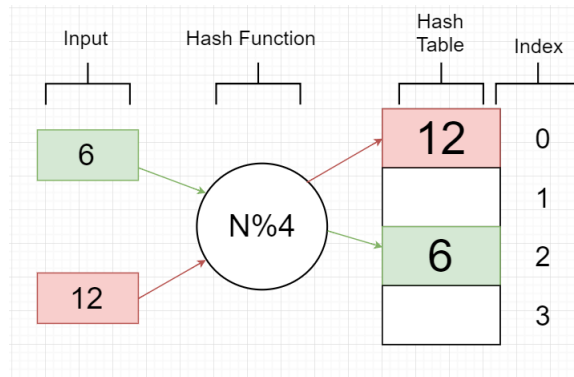
Below is the output:



```
jo7701@jo7701-B250M-DS3H: ~
jo7701@jo7701-B250M-DS3H:~$ python3 time_it.py
Determining inclusion in a list took 0.011299610137939453 seconds
Determining inclusion in a set took 0.0005795955657958984 seconds
jo7701@jo7701-B250M-DS3H:~$
```

As we can see from the output, the code took 0.11 seconds to check 10000 times if 5 was present in the list. On the other hand, it took 0.0006 seconds to check 10000 times if 5 was present in the set. When we check for inclusion in a list, Python loops through each element in the list until it finds the right one. As you might imagine, if the element we want to search for is at the end of a very long list, this operation will take a long time. On the other hand, when data is added to a set, it is hashed. A “hash” is a mathematical function which takes an object and represents it as an integer.

The following picture shows how sets store their data using hashes:



The input goes through a hash function and the output of the hash is the index where the element is stored. In the example above, our hash function is $N\%4$. When the input is 6, $6\%4 = 2$, so 6 will be placed at index 2. Hashing allows for a quick lookup, as all we need to do to check if an element is present in the set is to hash it and check that index.

1.1 Unique Elements

```
my_set = set()
my_set.add(5)
my_set.add(4)
my_set.add(2)
my_set.add(4)

print(len(my_set))
print(my_set)
```

The takeaway from this simple example is that sets only store unique elements. Even though we added “4” twice, it only showed up once in the list. Let’s take a look at a more interesting example:

```
a = [2,4,2,1,2,5,6,7,2,3,5]
unique_elems = set(a)

print(unique_elems)
```

In this example, we find all the unique numbers in list “a”. We do so by simply converting (or “casting”) the list into a set so that all duplicate items are removed. The next example shows operations you can do with multiple sets.

1.2 Set Operations

```
a = {3,6,2,9}
b = {3,9,5,0}

union = a|b
intersection = a&b
difference = a-b

print(union)
print(intersection)
print(difference)
```

In this example, we perform three different operations on sets “a” and “b”. The “—” operator takes the union of both sets. In other words, we simply add the two sets together. The “&” operator takes the intersection of both sets. This operation returns a set that contains elements that are present in both set “a” and “b”. The “-” operator takes the difference of the two sets. It returns a set that contains elements in set “a” that are not present in set “b”.

1.3 Loops

Just like in lists and strings, we can loop through sets as well. However, we are limited to “for each loops”, as sets cannot be indexed. Here’s an example of iterating through sets.

```
a = {2,4,7,3}
for number in a:
    print(number*2)
```

1.4 Lab

Create a function which determines if a string contains no duplicate characters.

Input: “Banana”
Output: False
Input: “Cars”
Output: True

2 Dictionaries

A dictionary in Python is a lookup table which maps unique keys to values. This concept is based off an English dictionary. In the Webster’s dictionary, each word is mapped to its definition, and no two word appears twice. This is exactly how Python dictionaries work. Here’s an example:

```
name_to_age = {} #the empty curly braces create an empty dictionary

name_to_age["Ryan"] = 17
name_to_age["Richard"] = 21

print(name_to_age)

print("Ryan's age is", name_to_age["Ryan"])
```

In the above example, they “keys” of the dictionary are names while the “values” are ages. Dictionaries can be indexed by using brackets (“[]”) and putting they key name inside the brackets.

2.1 Searching for Inclusion

Dictionaries are similar to sets in that they hash their keys. This means that checking to see if a certain element is a key in the dictionary is very fast.

```
my_dict = {2:4, 5:10, 15:30, 100:200}

for number in range(200):
    if number in my_dict:
        print(number, "times 2 equals", my_dict[number])
```

In this example, we loop through numbers from 0 to 199 (inclusive) and check to see if that number is in the dictionary. If it is, we access the value, which happens to be that number multiplied by 2.

Now let's look at a more practical example.

```
my_dict = {}
my_string = "how much wood would a wood chuck chuck if a wood chuck could chuck wood"
my_list = my_string.split(" ")

for word in my_list:
    if word in my_dict:
        my_dict[word] += 1
    else:
        my_dict[word] = 1

print(my_dict)
```

This code creates a dictionary which maps each word in “my_string” to its frequency (how many times its present in that string). We loop through each word and check if it's already in the dictionary. If it is, this means we have seen it before and we want to increase the frequency by one. If it isn't in the dictionary, we want to add that word to the dictionary and set the frequency to 1.

2.2 Iterating through a Dictionary

Using a for each loop, we can loop through the keys of the dictionary:

```
my_dict = {"Tim":15, "Chris":29, "Tom":12}

for name in my_dict:
    print(name, "is", my_dict[name], "years old")
```

2.3 Lab

Use the input function to have the user type a sentence. Print out the word that appears the most.

Input: I scream you scream we all scream for ice cream!

Output: scream