CDS 230
Modeling and Simulation I

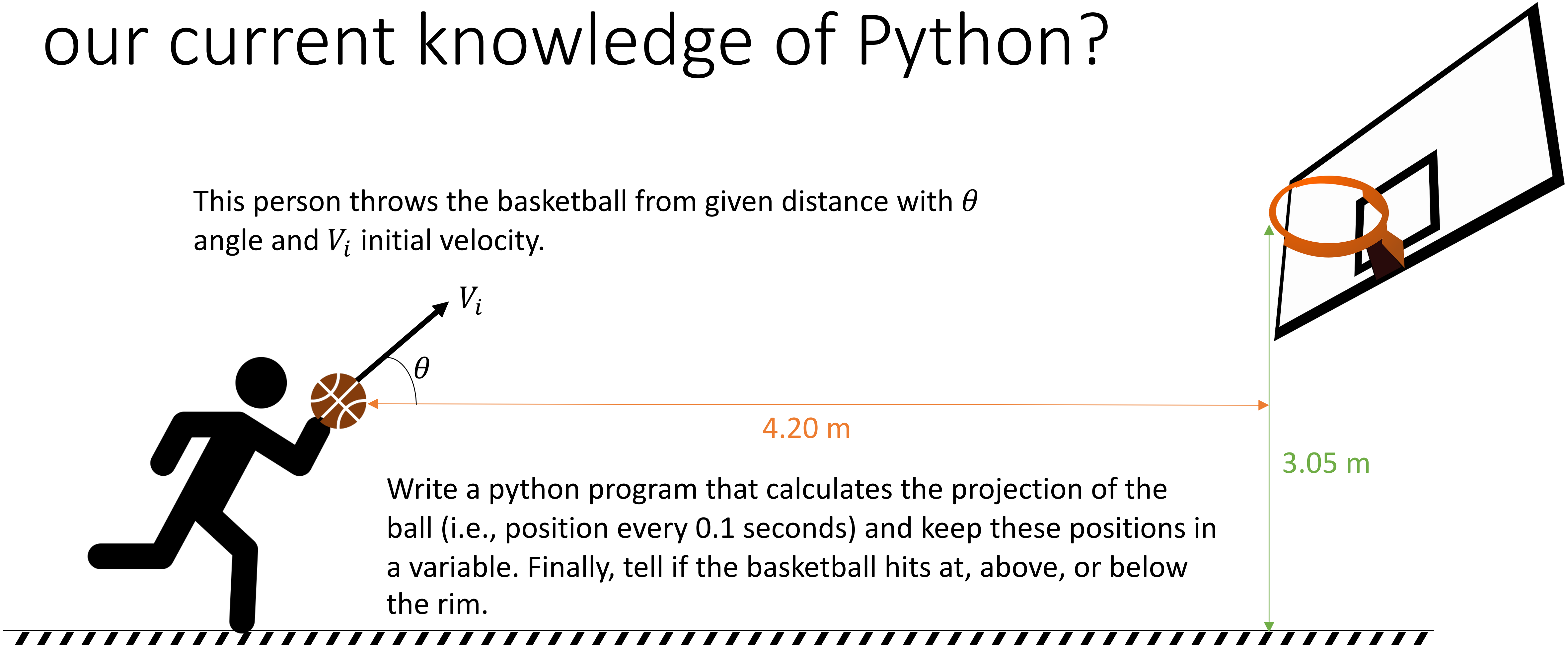
Module 5

Lists, Tuples, Dictionaries, and Sets

Dr. Hamdi Kavak
http://www.hamdikavak.com
hkavak@gmu.edu

Can you solve this question efficiently based on our current knowledge of Python?

This person throws the basketball from given distance with θ angle and V_i initial velocity.



Write a python program that calculates the projection of the ball (i.e., position every 0.1 seconds) and keep these positions in a variable. Finally, tell if the basketball hits at, above, or below the rim.

What concepts do we need to learn?

- An object type that can hold more than one value. E.g., positions = [(0,0), (0.1, 0.5), (0.3, 0.9),… (3.2,0)]

Today's Lecture

- A mechanism to generate many equally spaced values at once. E.g., time = [0.0, 0.1, 0.2, …., 1.5]

Wed Lecture

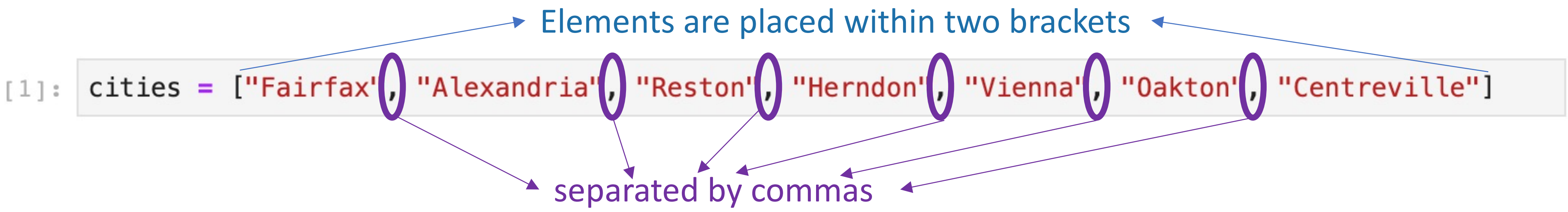
Once you learn these concepts, you can do very useful things like this: https://www.youtube.com/watch?v=MHTizZ_XcUM

Collections

- Can hold multiple values in a variable
- Four main types
  - Lists
  - Tuples
  - Dictionary
  - Sets
- Collections are **must-know** concepts for Python programming
- We will cover all collection types in this lecture
- Next lecture: iteration

Lists

- Used for keeping an **ordered** list of objects, similar to arrays in some programming languages.

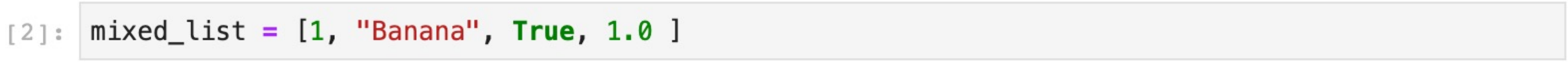


Elements are placed within two brackets

```
[1]: cities = ["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"]
```

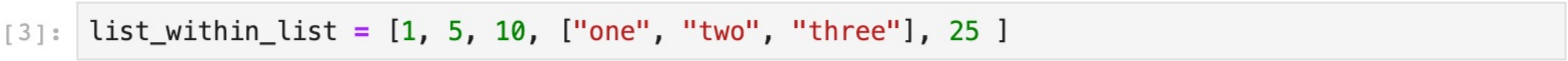
separated by commas

- Can hold different types of objects.



```
[2]: mixed_list = [1, "Banana", True, 1.0 ]
```

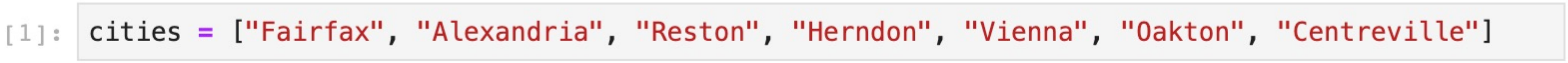
- Can define list within another list



```
[3]: list_within_list = [1, 5, 10, ["one", "two", "three"], 25 ]
```

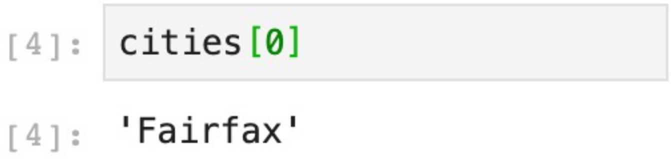
Lists – accessing elements

- Can access using index values

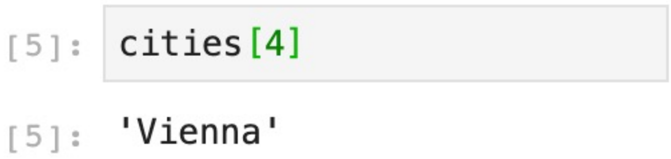


```
[1]: cities = ["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"]
```

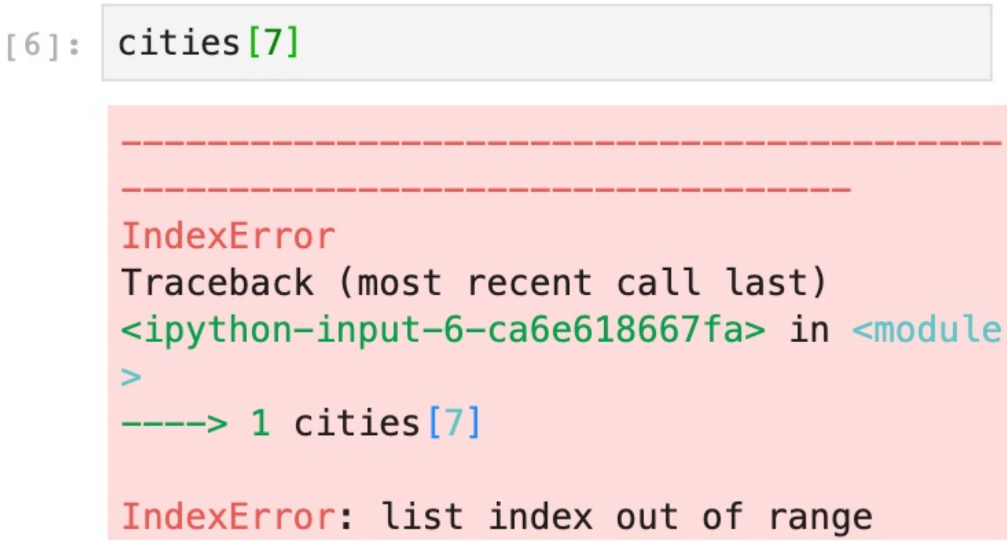
Index =>   0   1   2   3   4   5   6



```
[4]: cities[0]

[4]: 'Fairfax'
```



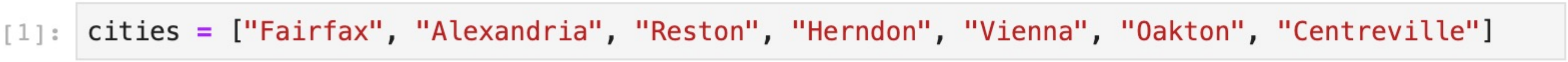
```
[5]: cities[4]

[5]: 'Vienna'
```



```
[6]: cities[7]

-----------------------------------------
-----------------------------------
IndexError
Traceback (most recent call last)
<ipython-input-6-ca6e618667fa> in <module
>
----> 1 cities[7]

IndexError: list index out of range
```

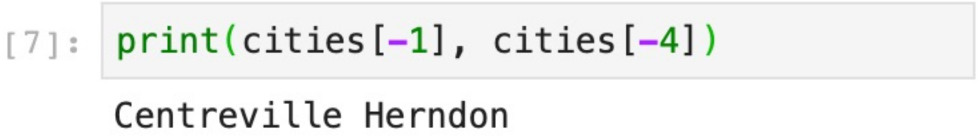
?

Lists – accessing elements

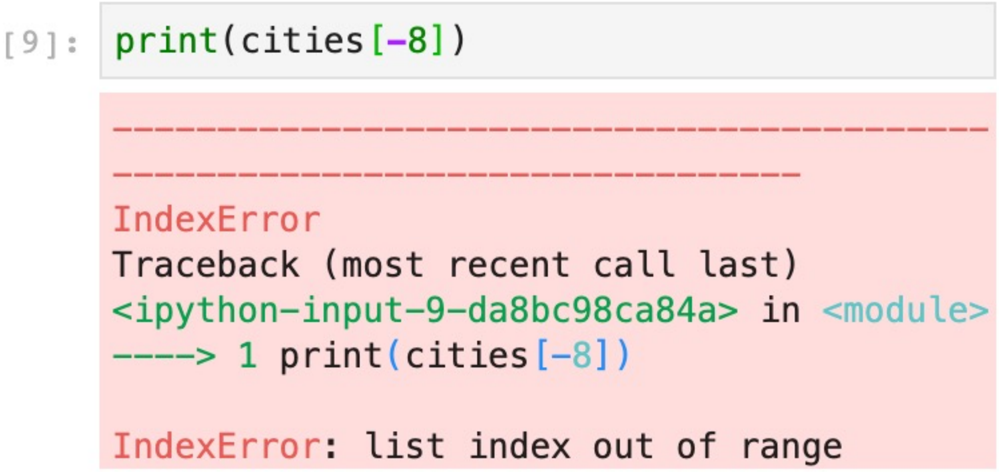


```
[1]: cities = ["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"]
```

- Negative index starts from the last item

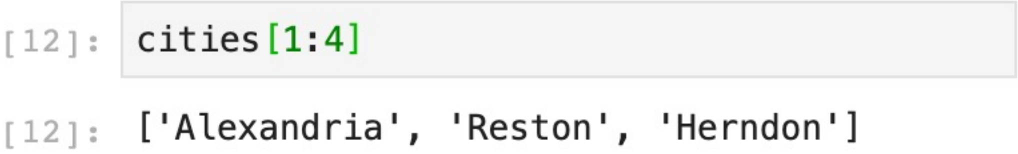


```
[7]: print(cities[-1], cities[-4])

     Centreville Herndon
```



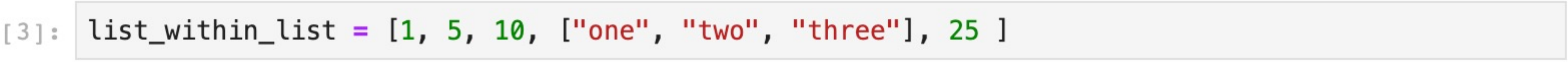
```
[9]: print(cities[-8])

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-9-da8bc98ca84a> in <module>
----> 1 print(cities[-8])

IndexError: list index out of range
```

- You can use slicing `[x:y]` x (inclusive) y (exclusive)

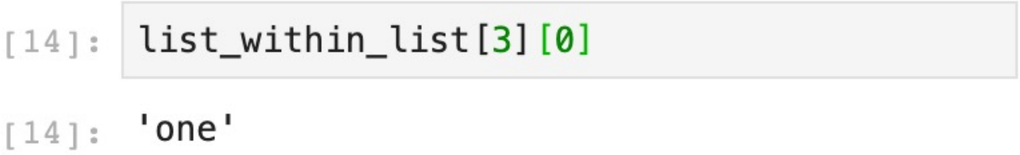


```
[12]: cities[1:4]

[12]: ['Alexandria', 'Reston', 'Herndon']
```

- Accessing list within another list elements



```
[3]: list_within_list = [1, 5, 10, ["one", "two", "three"], 25 ]
```

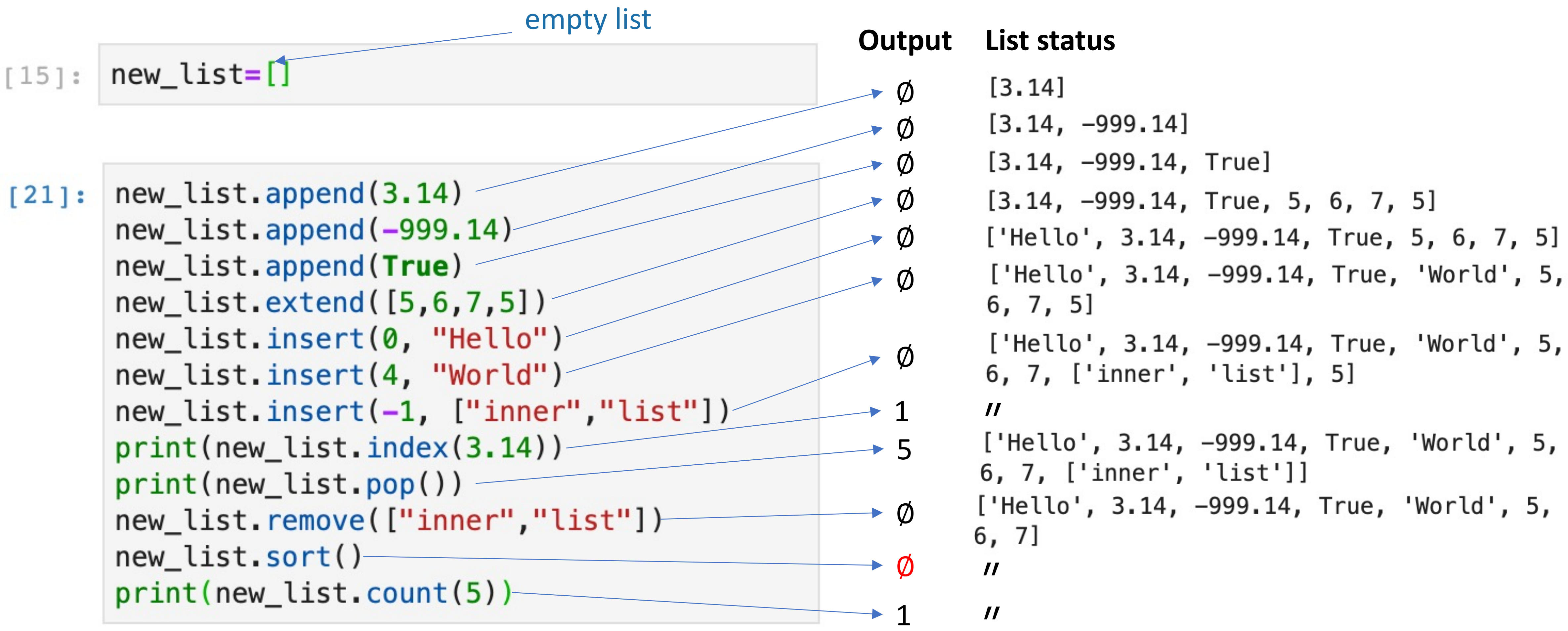


```
[14]: list_within_list[3][0]

[14]: 'one'
```

List functions

- `append` adds an element to the end of the list
- `extend` adds a list of elements to the end of the list
- `index` returns the lowest index of the element equals to given object
- `insert` adds an element to the specified location of the list
- `pop` remove and return the last element of the list
- `reverse` reverses the order of the list
- `remove` removes the first occurrence of an element from the list
- `sort` sort the order of the list in place
- `copy` return a copy of the list
- `count` returns the number of elements equal to the given element

Lists (code)



```
[15]: new_list=[]
```

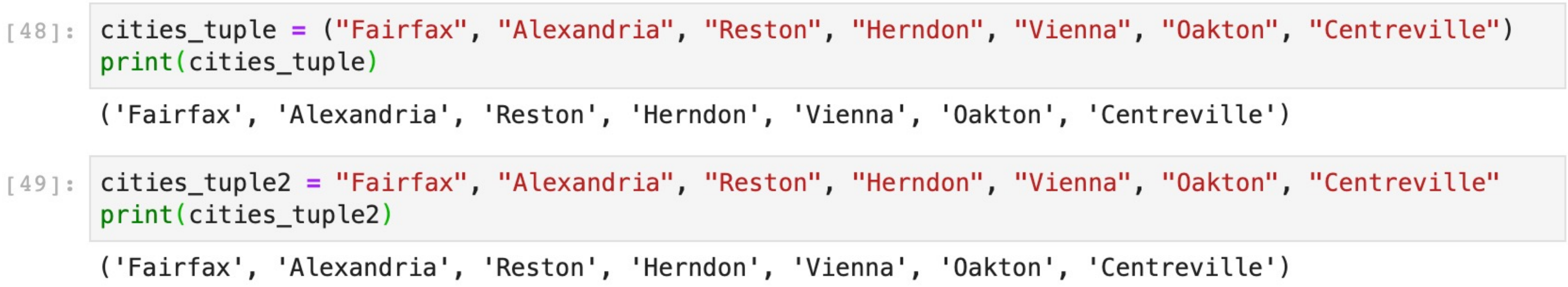
```
[21]: new_list.append(3.14)
      new_list.append(-999.14)
      new_list.append(True)
      new_list.extend([5,6,7,5])
      new_list.insert(0, "Hello")
      new_list.insert(4, "World")
      new_list.insert(-1, ["inner","list"])
      print(new_list.index(3.14))
      print(new_list.pop())
      new_list.remove(["inner","list"])
      new_list.sort()
      print(new_list.count(5))
```

```
Output   List status
∅        [3.14]
∅        [3.14, -999.14]
∅        [3.14, -999.14, True]
∅        [3.14, -999.14, True, 5, 6, 7, 5]
∅        ['Hello', 3.14, -999.14, True, 5, 6, 7, 5]
∅        ['Hello', 3.14, -999.14, True, 'World', 5,
         6, 7, 5]
∅        ['Hello', 3.14, -999.14, True, 'World', 5,
         6, 7, ['inner', 'list'], 5]
1        〃
5        ['Hello', 3.14, -999.14, True, 'World', 5,
         6, 7, ['inner', 'list']]
∅        ['Hello', 3.14, -999.14, True, 'World', 5,
         6, 7]
∅        〃
1        〃
```

∅ means nothing, 〃 means same as above

Tuples

- Similar to `list` but cannot be changed (i.e., immutable).
- Constructed within an optional parenthesis.

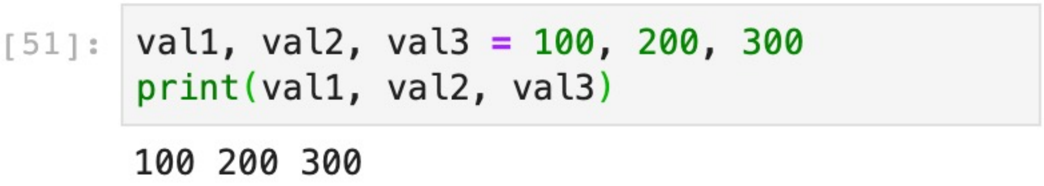


```
[48]: cities_tuple = ("Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville")
      print(cities_tuple)

      ('Fairfax', 'Alexandria', 'Reston', 'Herndon', 'Vienna', 'Oakton', 'Centreville')

[49]: cities_tuple2 = "Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"
      print(cities_tuple2)

      ('Fairfax', 'Alexandria', 'Reston', 'Herndon', 'Vienna', 'Oakton', 'Centreville')
```

- Things you can do: index and slice.
- Things you **cannot** do: append, extend, or remove elements.

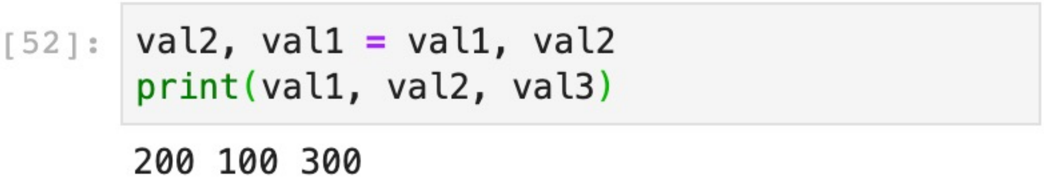
Why would you use tuples?

- They are faster than lists (because immutable)
- Tuple packing



```
[51]: val1, val2, val3 = 100, 200, 300
      print(val1, val2, val3)

      100 200 300
```

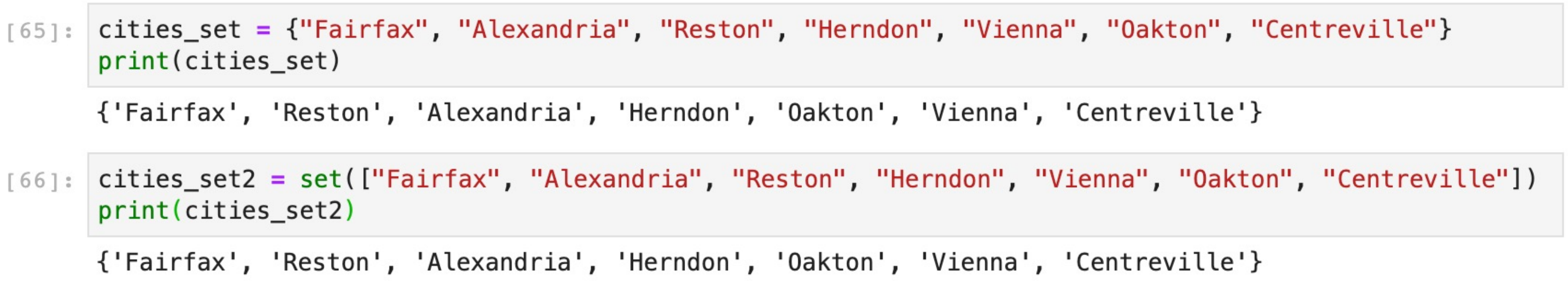
- Value swapping without temporary assignment



```
[52]: val2, val1 = val1, val2
      print(val1, val2, val3)

      200 100 300
```

Sets

- Similar to `list` but only holds **unique** and **unordered** values.
- Constructed within curly braces `{}` or using `set` function.

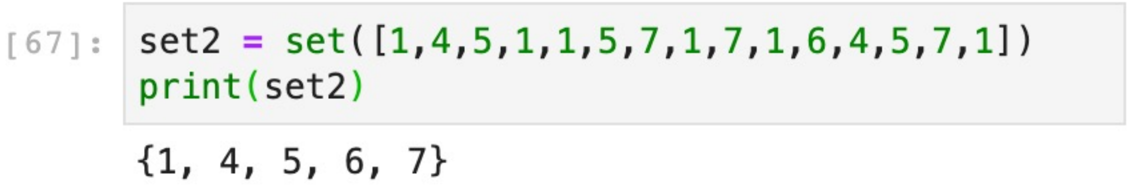


```
[65]: cities_set = {"Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"}
      print(cities_set)

      {'Fairfax', 'Reston', 'Alexandria', 'Herndon', 'Oakton', 'Vienna', 'Centreville'}

[66]: cities_set2 = set(["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"])
      print(cities_set2)

      {'Fairfax', 'Reston', 'Alexandria', 'Herndon', 'Oakton', 'Vienna', 'Centreville'}
```

- Things you can do: add element, remove element, and set operations
- Things you **cannot** do: index and slice.

Why would you use `sets`?

- Removing duplicates from a list



```
[67]: set2 = set([1,4,5,1,1,5,7,1,7,1,6,4,5,7,1])
      print(set2)

      {1, 4, 5, 6, 7}
```

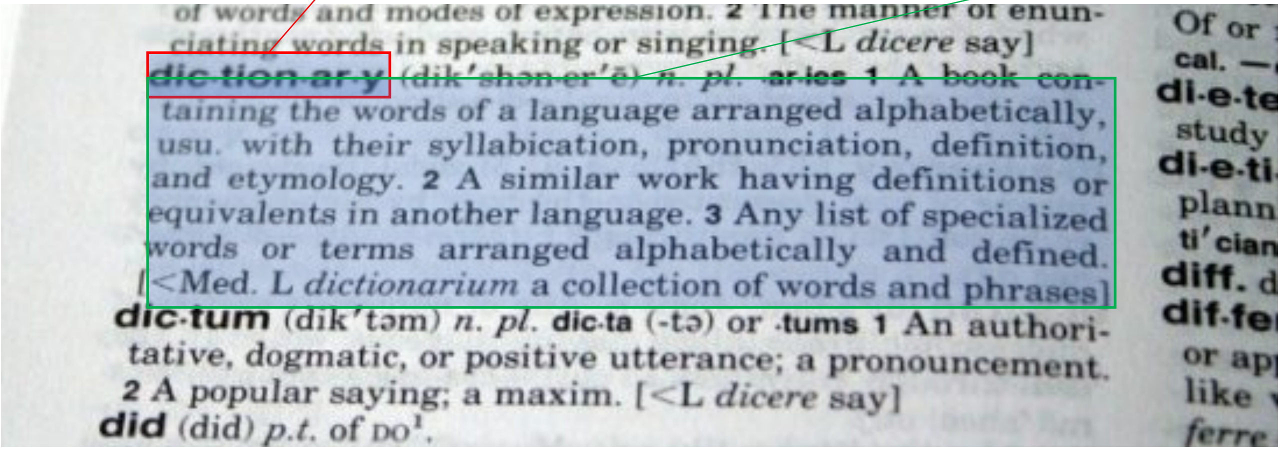
- Applying many set theory operations
  - Cardinality
  - Equality
  - Subset
  - Superset
  - Union
  - Intersection
  - Jointness…

Dictionary

- Quite different from `lists` with holding values as key-data or key-value pairs.

The Key is like the entry word

The Data is everything else.
In fact, it could be a collection of items.



of words and modes of expression. **2** The manner of enunciating words in speaking or singing. [<L *dicere* say]

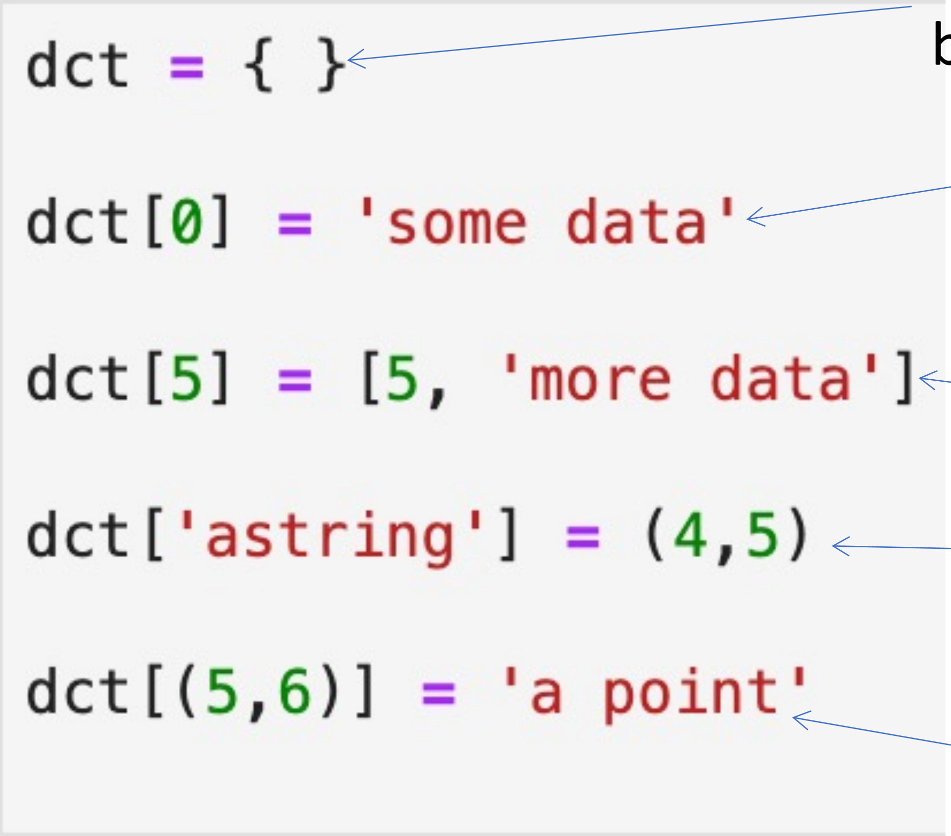
dic·tion·ar·y (dik'shən·er'ē) *n. pl.* **·ar·ies 1** A book containing the words of a language arranged alphabetically, usu. with their syllabication, pronunciation, definition, and etymology. **2** A similar work having definitions or equivalents in another language. **3** Any list of specialized words or terms arranged alphabetically and defined. [<Med. L *dictionarium* a collection of words and phrases]

dic·tum (dik'təm) *n. pl.* **dic·ta** (-tə) or **·tums 1** An authoritative, dogmatic, or positive utterance; a pronouncement. **2** A popular saying; a maxim. [<L *dicere* say]

did (did) *p.t.* of DO¹.

Source: https://www.protagonistsoccer.com/features/war-of-words

Python dictionary



```
dct = { }

dct[0] = 'some data'

dct[5] = [5, 'more data']

dct['astring'] = (4,5)

dct[(5,6)] = 'a point'
```

A Python dictionary is defined by curly braces.

In this case, the Key is 0 and the Data is a string: ‘some data’

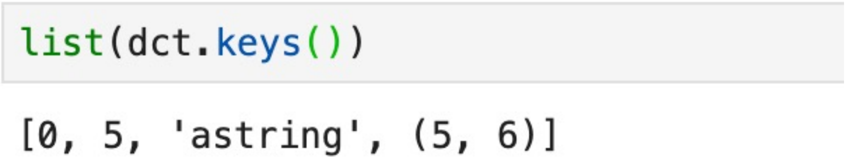
Here the Key is 5 and the Data is a list.

The key can be a string.

The Key can be a tuple, but not a list because the contents of a list can change.

Some dictionary functions

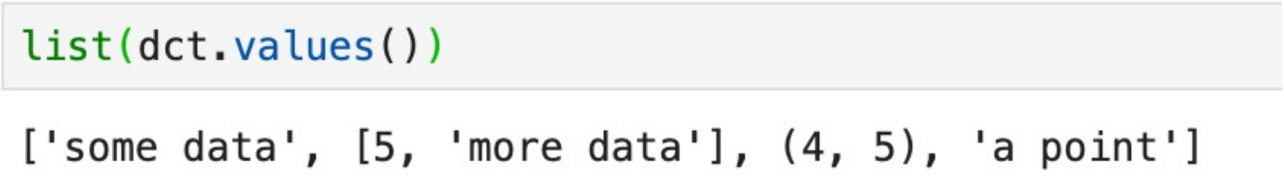
- Dictionary keys can be extracted



```
list(dct.keys())
```

```
[0, 5, 'astring', (5, 6)]
```

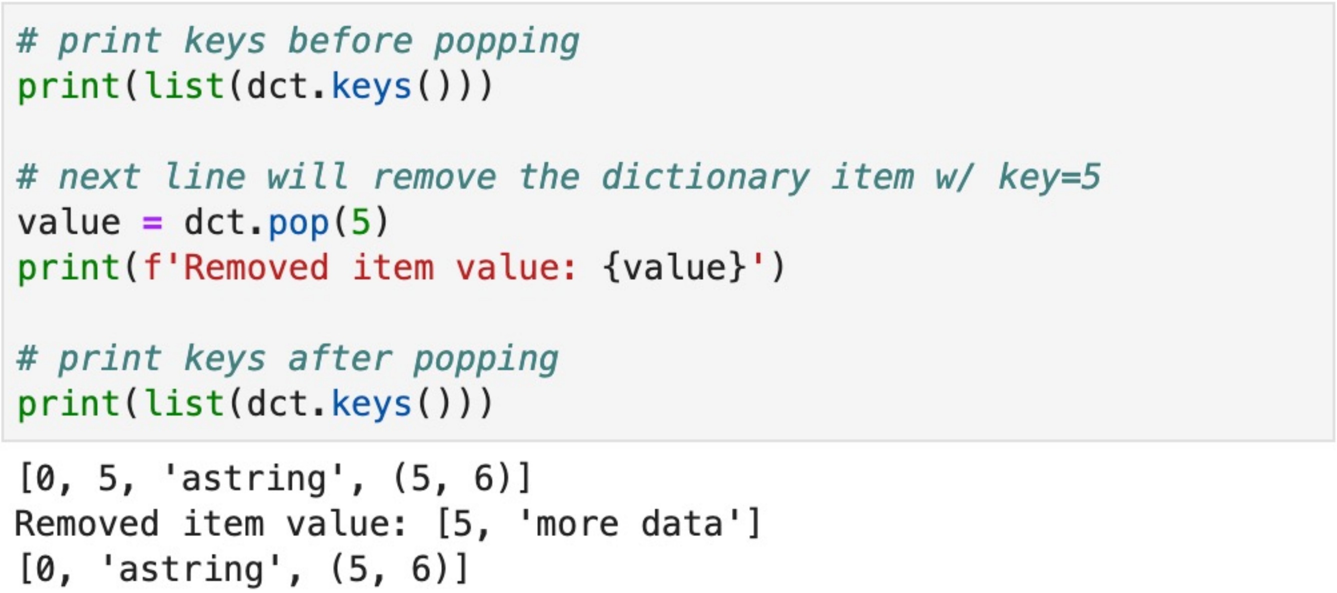
- So as the values



```
list(dct.values())
```

```
['some data', [5, 'more data'], (4, 5), 'a point']
```

- You can individually pop (i.e., retrieve and remove) a dictionary item using its key

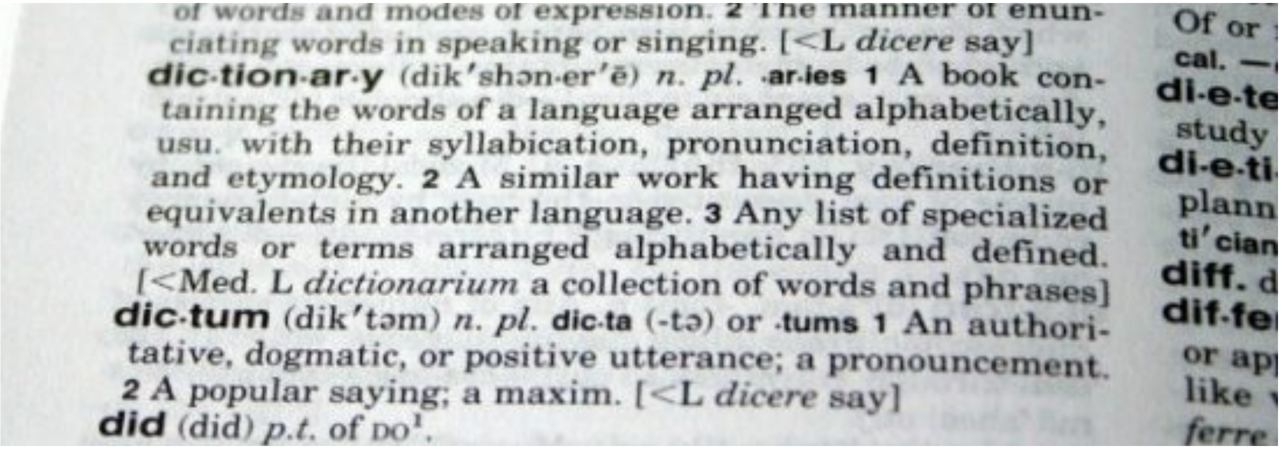


```
# print keys before popping
print(list(dct.keys()))

# next line will remove the dictionary item w/ key=5
value = dct.pop(5)
print(f'Removed item value: {value}')

# print keys after popping
print(list(dct.keys()))
```

```
[0, 5, 'astring', (5, 6)]
Removed item value: [5, 'more data']
[0, 'astring', (5, 6)]
```

Why would you use `dictionaries`?

- Searching
  - Just like a regular word dictionary – the Python dictionary searches only on the KEY.
  - We can look up the word dictionary, But we can’t directly look it up by its definition.



of words and modes of expression. 2 The manner of enunciating words in speaking or singing. [<L *dicere* say]

dic·tion·ar·y (dik′shən·er′ē) *n. pl.* **·ar·ies** 1 A book containing the words of a language arranged alphabetically, usu. with their syllabication, pronunciation, definition, and etymology. 2 A similar work having definitions or equivalents in another language. 3 Any list of specialized words or terms arranged alphabetically and defined. [<Med. L *dictionarium* a collection of words and phrases]

dic·tum (dik′təm) *n. pl.* **dic·ta** (-tə) or **·tums** 1 An authoritative, dogmatic, or positive utterance; a pronouncement. 2 A popular saying; a maxim. [<L *dicere* say]

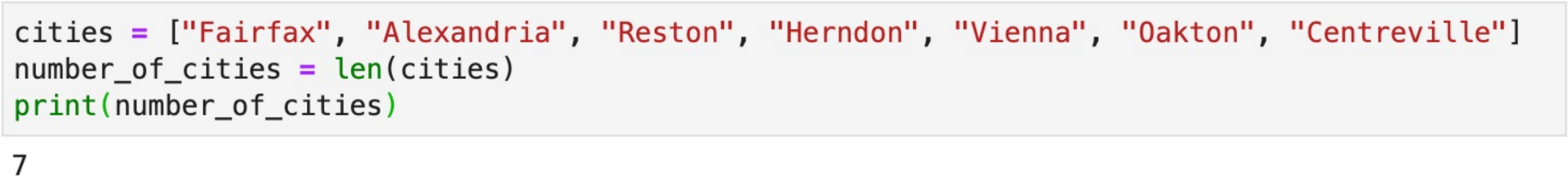
did (did) *p.t.* of DO^1.

Of or
cal. —
di·e·te
study
di·e·ti
plann
ti′cian
diff. d
dif·fe
or ap
like
ferre

- Hashing makes it is super fast

Useful functions

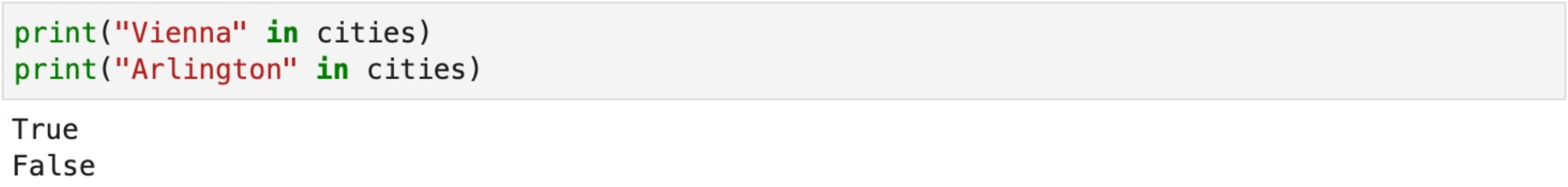
- `len` returns the size of the collection



```
cities = ["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"]
number_of_cities = len(cities)
print(number_of_cities)
```

```
7
```

- `in` checks the existence of an element in the collection



```
print("Vienna" in cities)
print("Arlington" in cities)
```

```
True
False
```