CS65: Introduction to Computer Science

Dictionary
Tuples

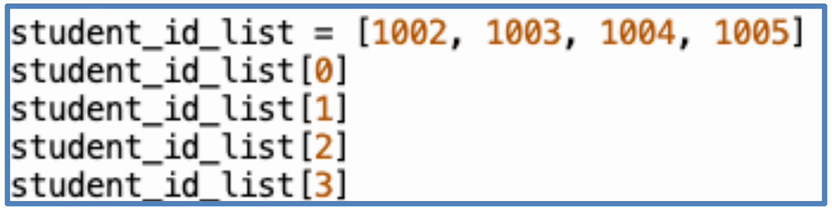
Md Alimoor Reza
Assistant Professor of Computer Science

Topic

- Dictionary:
  - What is it and why do we need it?
- Dictionary creation
- Dictionary manipulation
  - Adding
  - Updating
  - Removing
- Iterating over a dictionary
- Tuple

Dictionary

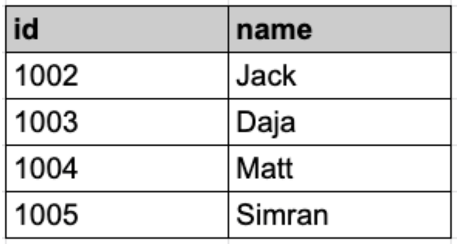
- **List**: Access a position using **only numeric index**



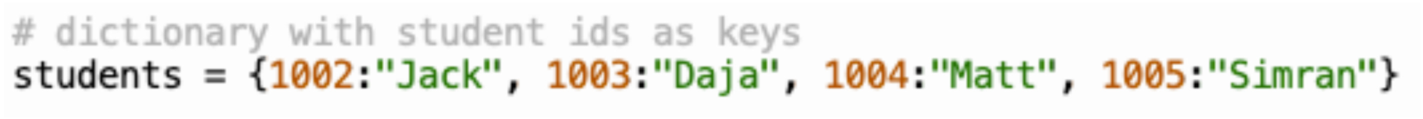
```
student_id_list = [1002, 1003, 1004, 1005]
student_id_list[0]
student_id_list[1]
student_id_list[2]
student_id_list[3]
```

- **Dictionary**: Access an element using (eg number, string, character) as index (keys) to associate to something else (values)

- Dictionary is an object that stores a collection of data
  - collection of key-value pairs
  - variable_name = { key$_1$: value$_1$, key$_2$: value$_2$, …, key$_N$: value$_N$ }



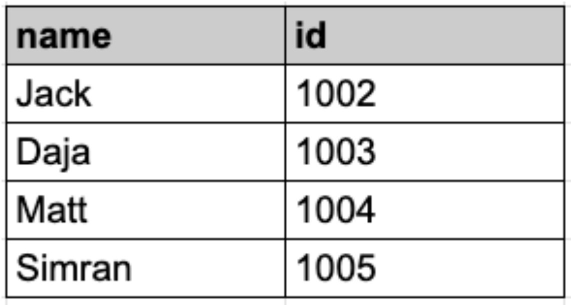
id	name
1002	Jack
1003	Daja
1004	Matt
1005	Simran



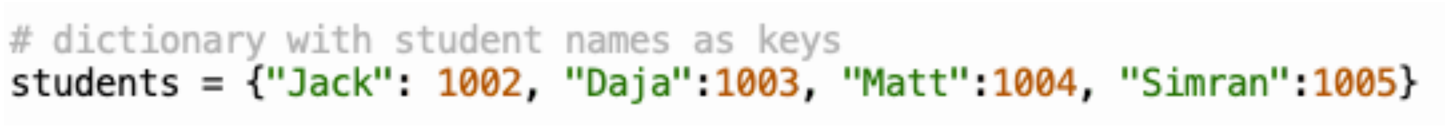
```
# dictionary with student ids as keys
students = {1002:"Jack", 1003:"Daja", 1004:"Matt", 1005:"Simran"}
```

Dictionary

- Dictionary is collection of key-value pairs or mapping between them
- variable_name = { key_1 : $value_1$, key_2 : $value_2$, …, key_N : $value_N$ }
- The keys **must be unique**
  - Unlike previous example, keys below are **names**



name	id
Jack	1002
Daja	1003
Matt	1004
Simran	1005



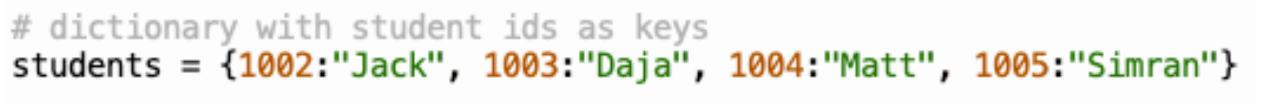
```
# dictionary with student names as keys
students = {"Jack": 1002, "Daja":1003, "Matt":1004, "Simran":1005}
```

Retrieving a Value from a Dictionary

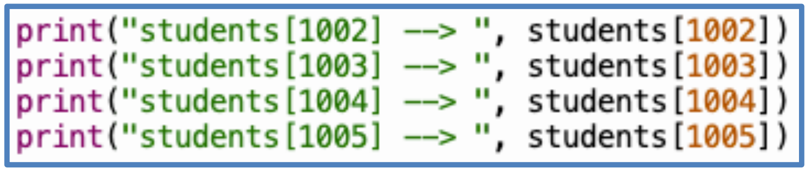
- Given *variable_name* = { key$_1$: value$_1$, key$_2$: value$_2$, …, key$_N$: value$_N$ }

- Use the syntax: *variable_name*[key]

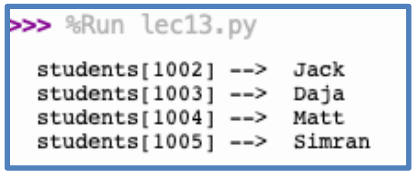
- Since keys are **unique**, accessing via a key will return a specific value



```
# dictionary with student ids as keys
students = {1002:"Jack", 1003:"Daja", 1004:"Matt", 1005:"Simran"}
```



```
print("students[1002] --> ", students[1002])
print("students[1003] --> ", students[1003])
print("students[1004] --> ", students[1004])
print("students[1005] --> ", students[1005])
```



```
>>> %Run lec13.py

 students[1002] -->  Jack
 students[1003] -->  Daja
 students[1004] -->  Matt
 students[1005] -->  Simran
```

- Do you find any similarity with List?

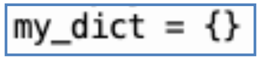
Topic

- Dictionary:
  - What is it and why do we need it?
- Dictionary creation
- Dictionary manipulation
  - Adding
  - Updating
  - Removing
- Iterating over a dictionary
- Tuple

Creating a Dictionary

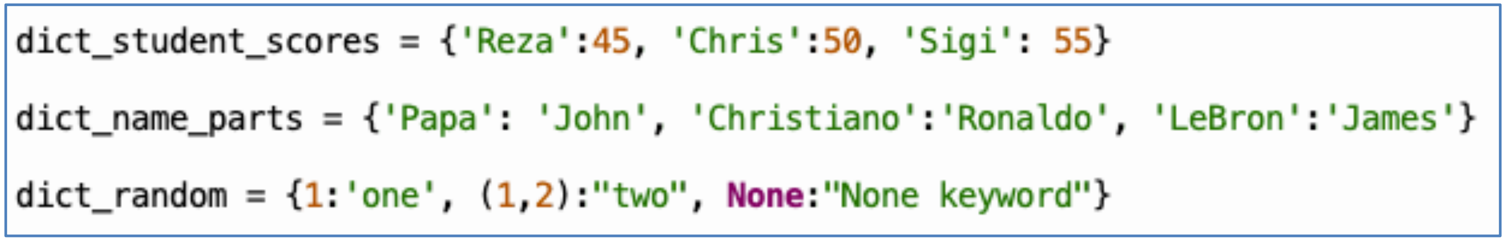
- Several ways to create a dictionary:

- **Approach 1:** an empty dictionary



```
my_dict = {}
```

- **Approach 2:** with predefined entries

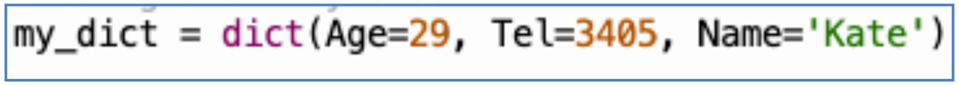


```
dict_student_scores = {'Reza':45, 'Chris':50, 'Sigi': 55}

dict_name_parts = {'Papa': 'John', 'Christiano':'Ronaldo', 'LeBron':'James'}

dict_random = {1:'one', (1,2):"two", None:"None keyword"}
```

Creating a Dictionary

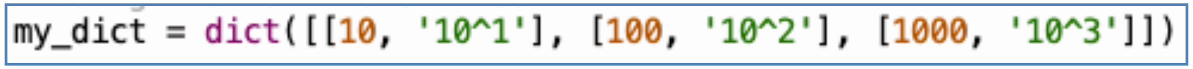
- Several ways to create a dictionary:

- **Approach 3:** with **dict()** with keyword args, <u>unquoted-strings</u>



```
my_dict = dict(Age=29, Tel=3405, Name='Kate')
```

- **Approach 4:** with **dict**() and list of two entries



```
my_dict = dict([[10, '10^1'], [100, '10^2'], [1000, '10^3']])
```

Exercise

- **Exercise 1:** Create a dictionary that can save names of different exams and their scores as key-value pairs. More specifically, your dictionary should store information about the following three tests:
    - ‘***Quiz 1***’ and its ***score***
    - ‘***Quiz 2***’ and its ***score***
    - ‘***Midterm exam***' and its ***score***

- First, decide — what would be your **keys** and what would be **values**
- Second, pick a reasonable variable name for the dictionary
- Finally, create your new dictionary

Exercise

- **<u>Exercise 2:</u>** Create a dictionary that can save names of 3 different students and a list of <u>5 scores</u> as ***key:value*** pairs.

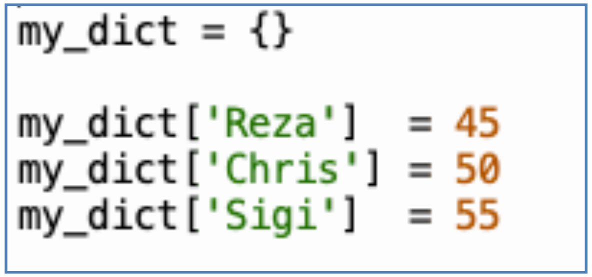
- More specifically, your dictionary should store information:
    - Student_1 name and a list of scores for student 1
    - Student_2 name and a list of scores for student 2
    - Student_3 name and a list of scores for student 4

Topic

- Dictionary:
  - What is it and why do we need it?
- Dictionary creation
- Dictionary manipulation
  - Adding
  - Updating
  - Removing
- Iterating over a dictionary
- Tuple

Adding or updating key-value pairs to a Dictionary

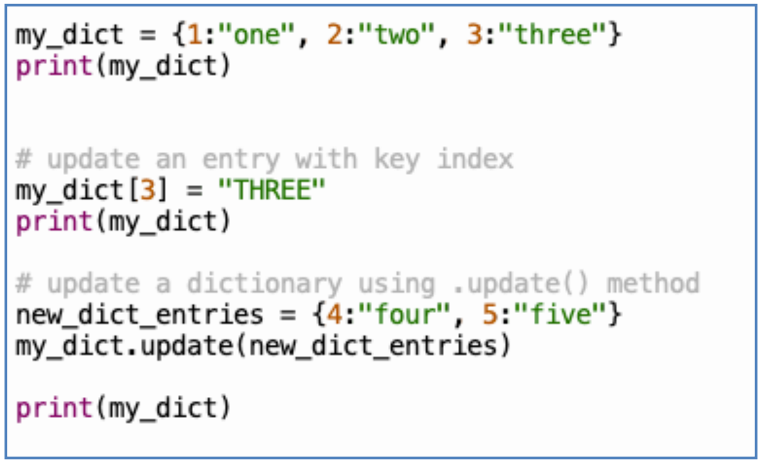
- Use the following syntax to add a key-value pair to a dictionary
- *variable_name*[key] = value



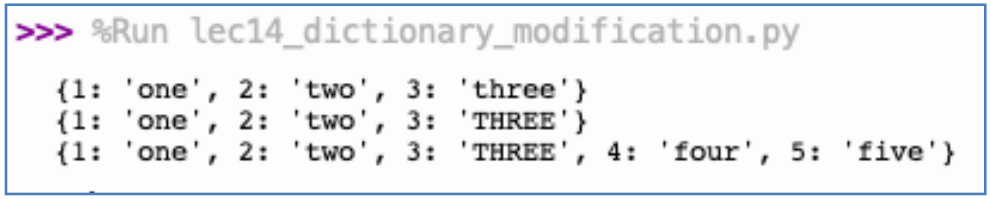
```
my_dict = {}

my_dict['Reza']  = 45
my_dict['Chris'] = 50
my_dict['Sigi']  = 55
```

Updating Dictionary Item

- It is also possible to update a dictionary using .update() method



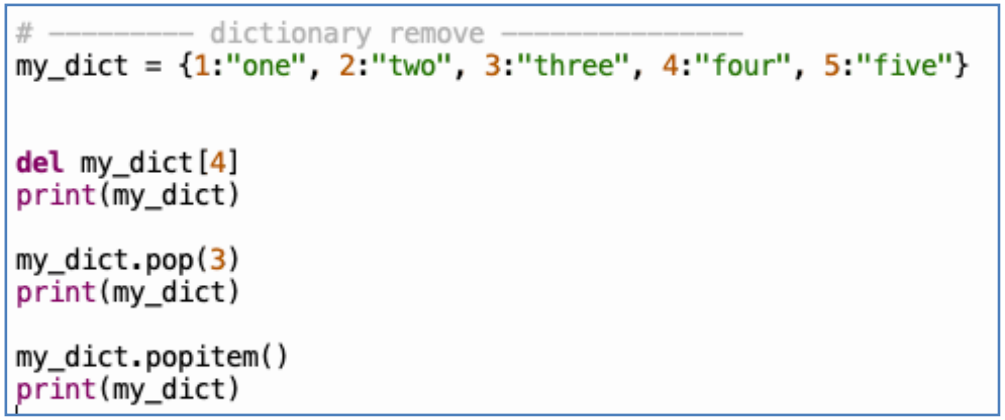
```python
my_dict = {1:"one", 2:"two", 3:"three"}
print(my_dict)


# update an entry with key index
my_dict[3] = "THREE"
print(my_dict)

# update a dictionary using .update() method
new_dict_entries = {4:"four", 5:"five"}
my_dict.update(new_dict_entries)

print(my_dict)
```



```
>>> %Run lec14_dictionary_modification.py

  {1: 'one', 2: 'two', 3: 'three'}
  {1: 'one', 2: 'two', 3: 'THREE'}
  {1: 'one', 2: 'two', 3: 'THREE', 4: 'four', 5: 'five'}
```

Removing Dictionary Item

- pop() method: remove an item by a given key
- del keyword: similar to list removal
- popitem() method: pop an arbitrary item



```
# --------- dictionary remove ---------------
my_dict = {1:"one", 2:"two", 3:"three", 4:"four", 5:"five"}


del my_dict[4]
print(my_dict)

my_dict.pop(3)
print(my_dict)

my_dict.popitem()
print(my_dict)
```

```
{1: 'one', 2: 'two', 3: 'three', 5: 'five'}
{1: 'one', 2: 'two', 5: 'five'}
{1: 'one', 2: 'two'}
```

Topic

- Dictionary:
  - What is it and why do we need it?
- Dictionary creation
- Dictionary manipulation
  - Adding
  - Updating
  - Removing
- Iterating over a dictionary
- Tuple

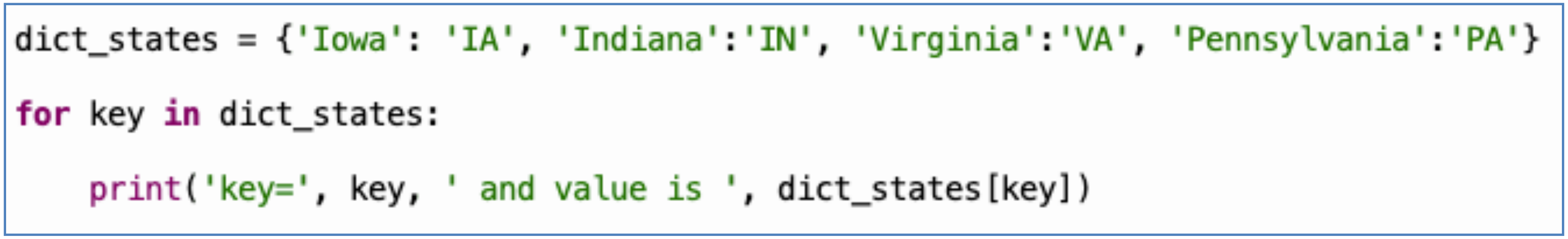
Iterating through a Dictionary: Approach 1

- Use the following syntax to iterate through a dictionary (more like accessing a list using value for loop)

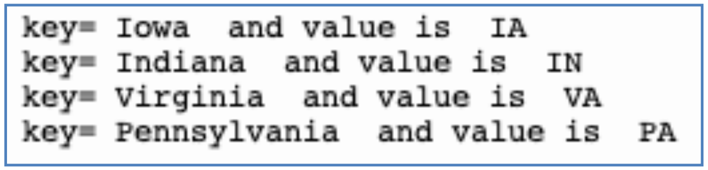
-
```
for key_var in dictionary_variable_name:
        print(dictionary_variable_name[key_var])
```

- It will process every key_var in the dictionary, the following line will access corresponding value associated with the key_var

Example: Iterating through a Dictionary



```
dict_states = {'Iowa': 'IA', 'Indiana':'IN', 'Virginia':'VA', 'Pennsylvania':'PA'}

for key in dict_states:

    print('key=', key, ' and value is ', dict_states[key])
```



```
key= Iowa  and value is  IA
key= Indiana  and value is  IN
key= Virginia  and value is  VA
key= Pennsylvania  and value is  PA
```

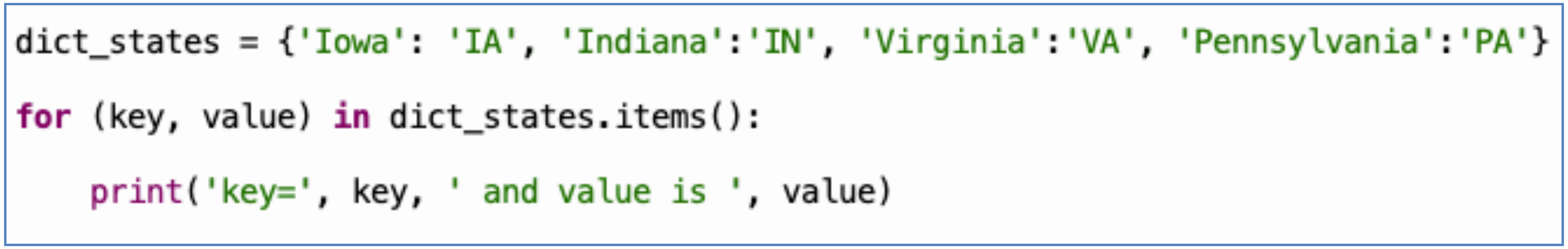
Iterating through a Dictionary: Approach 2

- Another way to iterate through a dictionary is to use following syntax

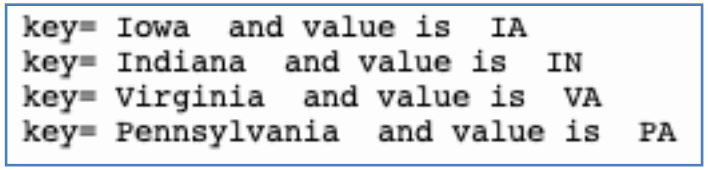
- for (key, value) in *dictionary_variable_name*.items():
  print('key = ', key, and 'value = ', value)

- It will process every key and value pair in the dictionary

Example: Iterating through a Dictionary

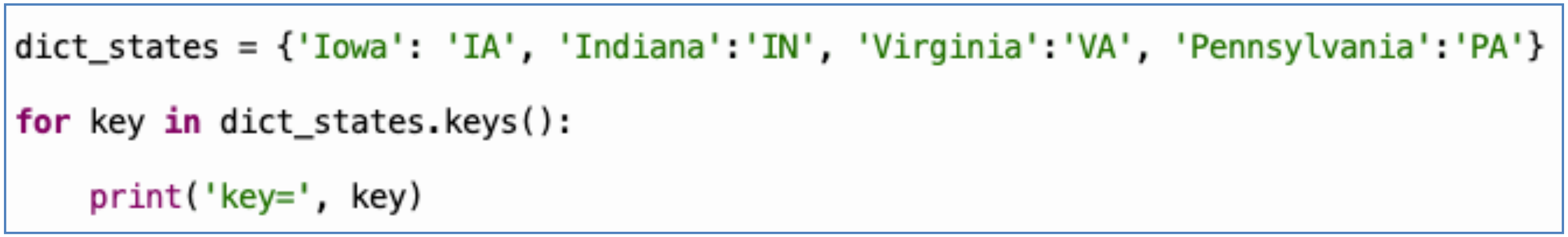


```python
dict_states = {'Iowa': 'IA', 'Indiana':'IN', 'Virginia':'VA', 'Pennsylvania':'PA'}

for (key, value) in dict_states.items():

    print('key=', key, ' and value is ', value)
```

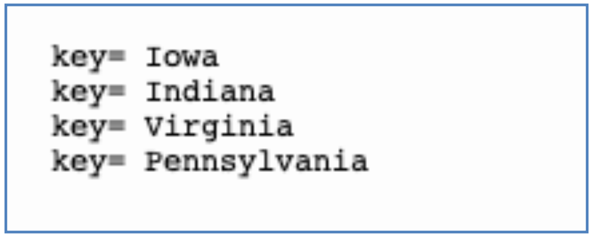


```
key= Iowa  and value is  IA
key= Indiana  and value is  IN
key= Virginia  and value is  VA
key= Pennsylvania  and value is  PA
```

Example: Iterating through keys

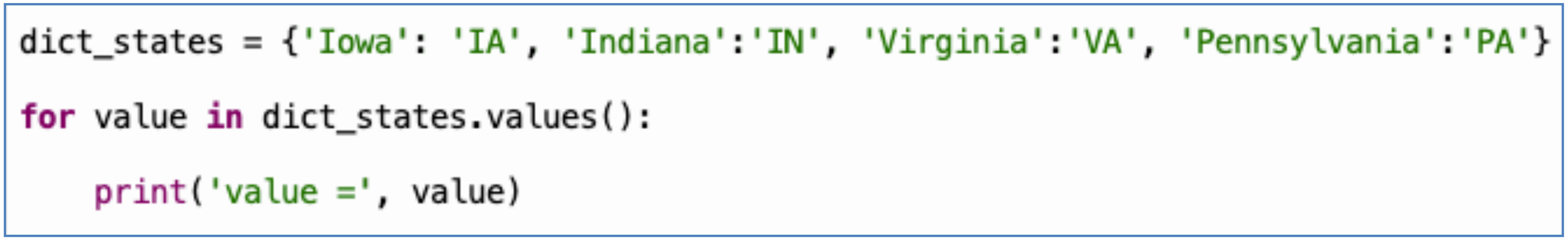


```python
dict_states = {'Iowa': 'IA', 'Indiana':'IN', 'Virginia':'VA', 'Pennsylvania':'PA'}

for key in dict_states.keys():

    print('key=', key)
```

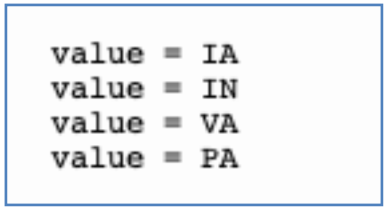


```
key= Iowa
key= Indiana
key= Virginia
key= Pennsylvania
```

Example: Iterating through values



```python
dict_states = {'Iowa': 'IA', 'Indiana':'IN', 'Virginia':'VA', 'Pennsylvania':'PA'}

for value in dict_states.values():

    print('value =', value)
```



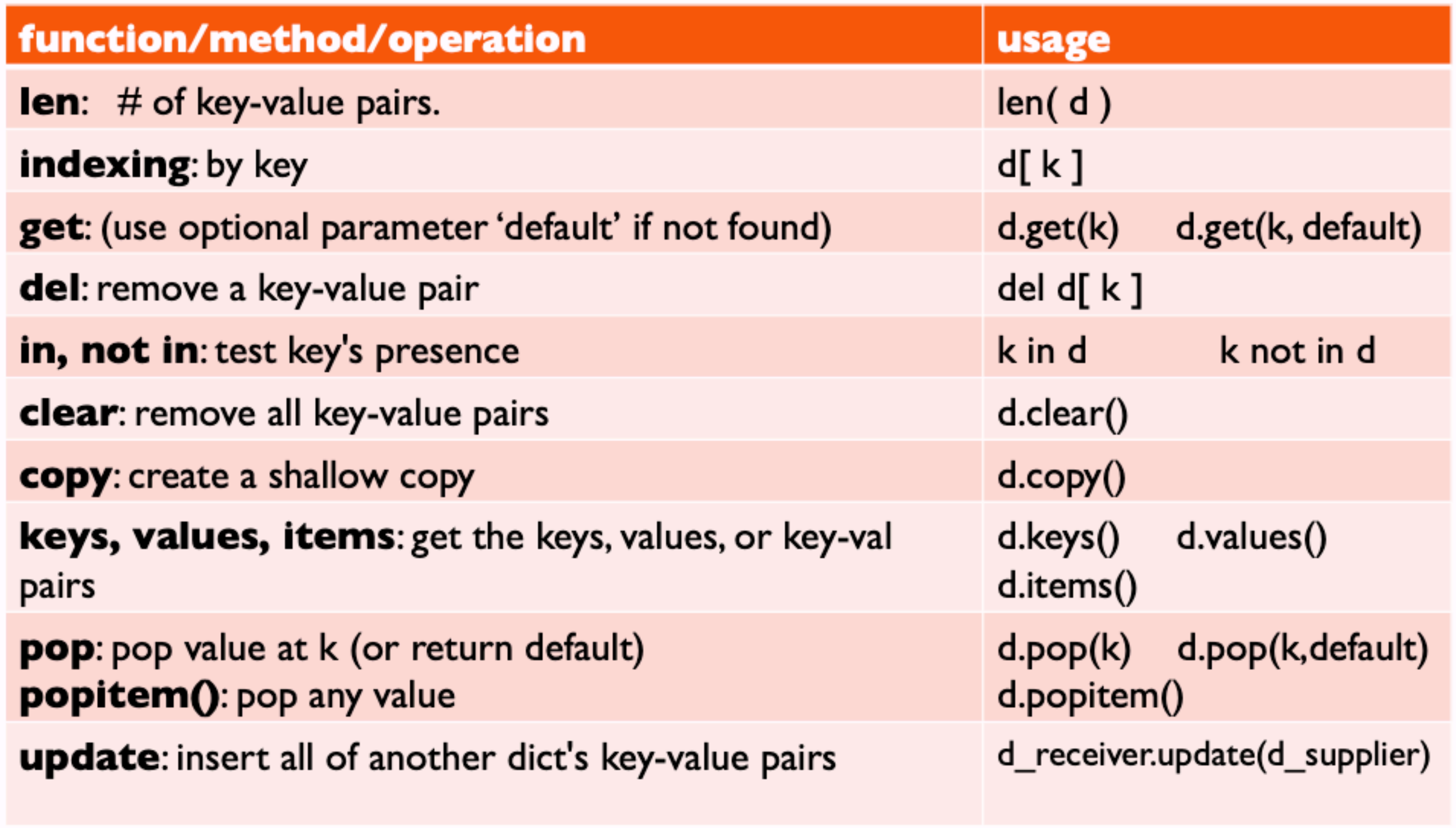
```
value = IA
value = IN
value = VA
value = PA
```

Exercise

- **Exercise 1:** Now you try out these four different ways of iterating through the dictionary

- **Exercise 2:** Iterate through the dictionary you have created in an earlier exercise eg exam score

Dictionary Operations



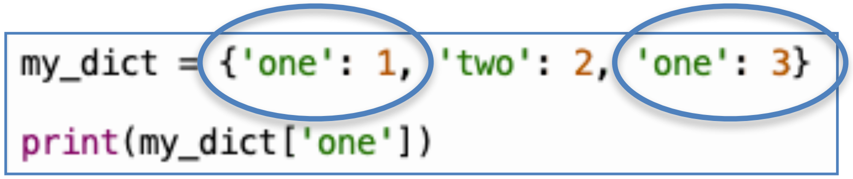
function/method/operation	usage
len: # of key-value pairs.	len(d)
indexing: by key	d[k]
get: (use optional parameter ‘default’ if not found)	d.get(k) d.get(k, default)
del: remove a key-value pair	del d[k]
in, not in: test key's presence	k in d k not in d
clear: remove all key-value pairs	d.clear()
copy: create a shallow copy	d.copy()
keys, values, items: get the keys, values, or key-val pairs	d.keys() d.values() d.items()

pop: pop value at k (or return default)
popitem(): pop any value
	d.pop(k) d.pop(k,default)
	d.popitem()

update: insert all of another dict's key-value pairs	d_receiver.update(d_supplier)

Dictionary: Important Notes

- Only one entry per key is allowed! When there is a duplicate, the last entry wins



```
my_dict = {'one': 1, 'two': 2, 'one': 3}

print(my_dict['one'])
```

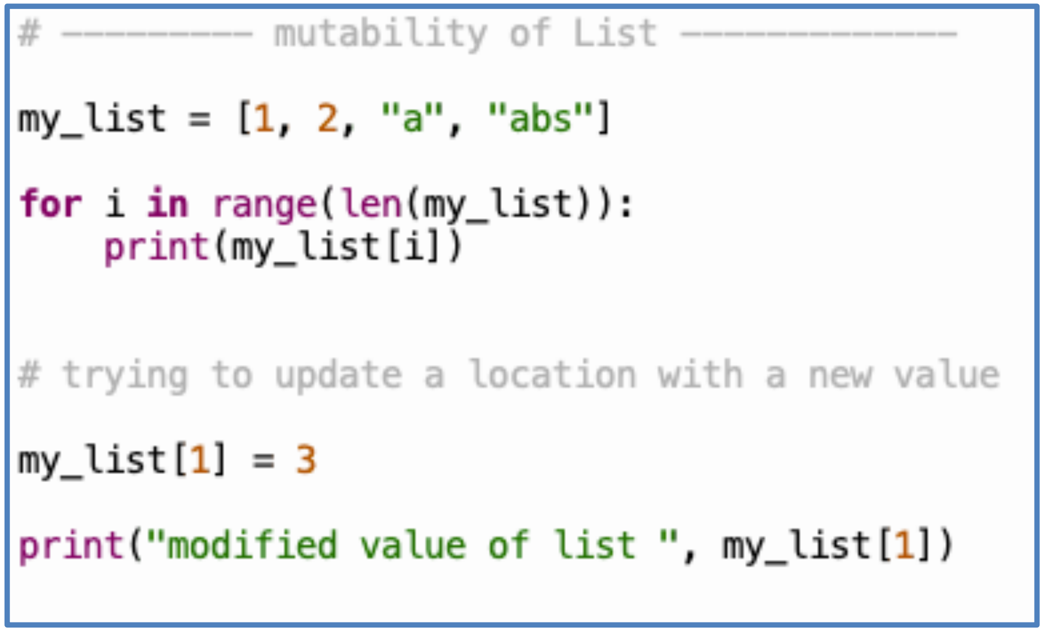
- Lists are not allowed as ***keys***
- No restrictions on ***values***
- Dictionaries do not keep order
- Keys must be **unique** and **immutable**

Tuple: another type of a sequence

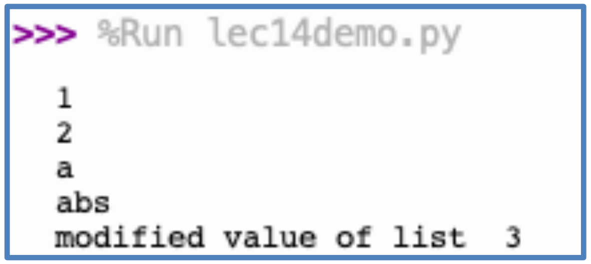
- <u>We can't change/modify its items after creation (**immutability**)</u>
- Items are accessed by index (similar to other two sequences List and String)

Sequence	Example	Syntax	Accessing
String	my_str = "My name is walle"	within enclosing quotation marks, ie, " " or ' '	my_str[0] my_str[1]
List	my_list = [1, 2, "a", "abs"]	within enclosing brackets [ ] and separated by commas	my_list[0] my_list[1]
Tuple	my_tuple = (1, 2, "a", "abs")	Within enclosing parenthesis ( ) and separated by commas	my_tuple[0] my_tuple[1]

Mutable Property of List

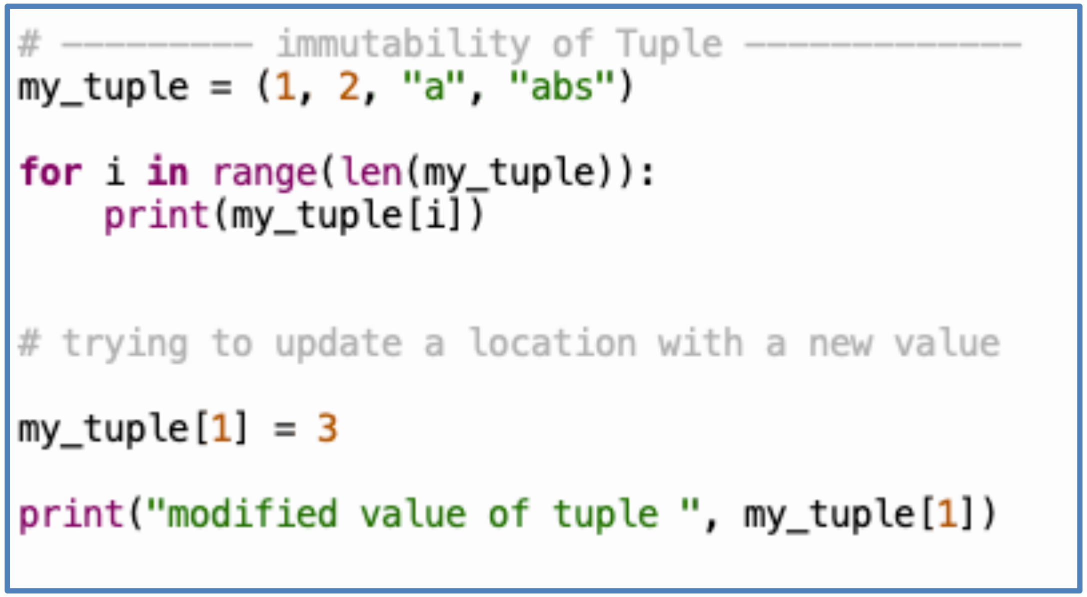


```
# --------- mutability of List -------------

my_list = [1, 2, "a", "abs"]

for i in range(len(my_list)):
    print(my_list[i])


# trying to update a location with a new value

my_list[1] = 3

print("modified value of list ", my_list[1])
```

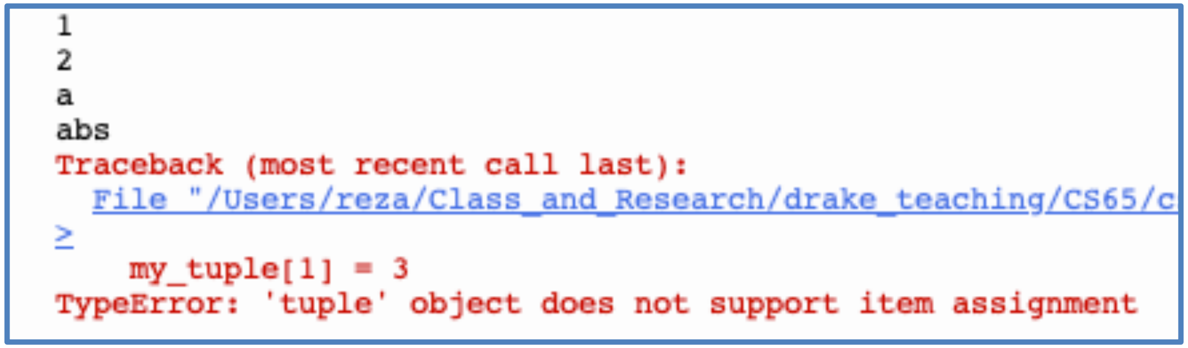


```
>>> %Run lec14demo.py

  1
  2
  a
  abs
  modified value of list  3
```

Immutable Property of Tuple



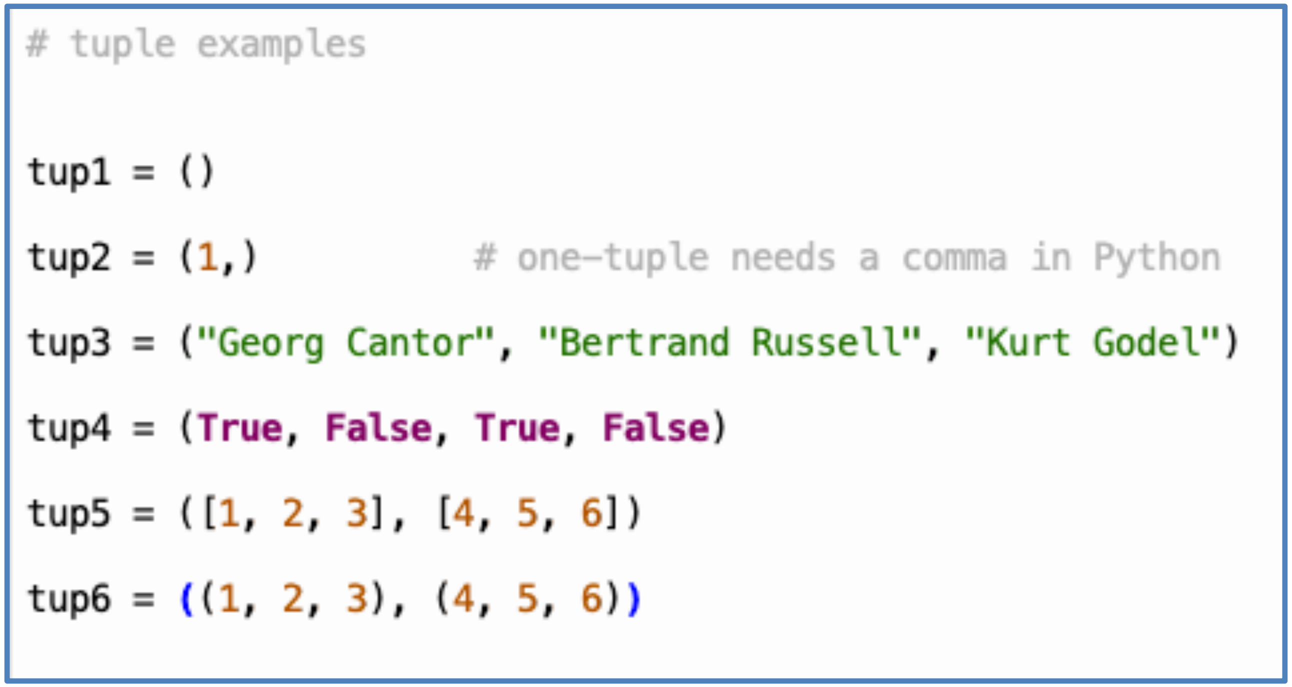
```
# ---------- immutability of Tuple --------------
my_tuple = (1, 2, "a", "abs")

for i in range(len(my_tuple)):
    print(my_tuple[i])


# trying to update a location with a new value

my_tuple[1] = 3

print("modified value of tuple ", my_tuple[1])
```



```
1
2
a
abs
Traceback (most recent call last):
  File "/Users/reza/Class_and_Research/drake_teaching/CS65/c
>
    my_tuple[1] = 3
TypeError: 'tuple' object does not support item assignment
```

Tuple

- **Tuple examples**



```python
# tuple examples

tup1 = ()
tup2 = (1,)            # one-tuple needs a comma in Python
tup3 = ("Georg Cantor", "Bertrand Russell", "Kurt Godel")
tup4 = (True, False, True, False)
tup5 = ([1, 2, 3], [4, 5, 6])
tup6 = ((1, 2, 3), (4, 5, 6))
```

- Exercise: Try the examples above in Thonny. Find the items you can modify and which the ones you cannot