Welcome to Lecture 17: Tuples and Dictionaries

1) Open a Code Editor- your choice
2) Use Iclicker for attendance
3) Lecture 17 Guide: tinyurl.com/S24CS10L17

Agenda

- Review
- Tuples
- Dictionaries

Lecture 17 Guide: tinyurl.com/S24CS10L17

Announcements

- Midterm Review on Wednesday from 5-7PM Soda 606
- Project 4: Pyturis will be released on Thursday
- Midterm Retake on Friday, 1 to 4PM
    - Same logistics as Midterm

Lecture 17 Guide: tinyurl.com/S24CS10L17

COMPUTATIONAL THINKING + Creative Problem Solving

CS10 is not a course about Snap! What we're learning is computational thinking and Creative Problem Solving

- How to use computational tools to solve problems.
- Using abstraction to manage complexity.
  - Detail removal
  - Generalization

Lecture 17 Guide: tinyurl.com/S24CS10L17

Review: Python List Basics

- We make a list in Python by putting the items in square brackets and separating them with commas

```
some_list = [0, 10, 20, 30, 40, 50]
```

- We can always get the length of the list by calling the *len* function

```
len(some_list) → 6
```

- We can access values in our list by indexing into the list using square brackets

```
some_list[0] → 0
some_list[4] → 40
some_list[len(some_list)-1] → 50
some_list[-1] → 50
```

List Basics: Index and Slicing

- We can slice our list to grab a subset of items in our list by using square brackets with a colon between the start and end indices
    - The left side is inclusive and the right side is exclusive

```
some_list = [0, 10, 20, 30, 40, 50]
some_list[1:5] → [10, 20, 30, 40]
```

-
    - If you leave on of the endpoints off, by default it will go to the end of the list, depending on which side is left off

```
some_list[:3] → [0, 10, 20]
some_list[2:] → [20, 30, 40, 50]
some_list[:] → [0, 10, 20, 30, 40, 50]
```

List Basics: Index and Slicing

- You can change the value of an item in a list by accessing that item using indexing and then re-assigning it to the new value

```
some_list[3] = 35
print(some_list) → [0, 10, 20, 35, 40, 50]
```

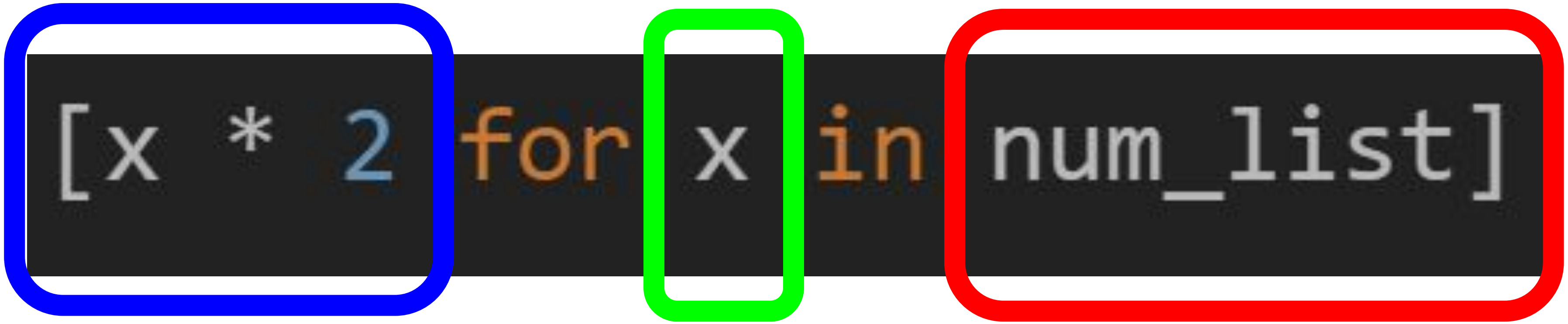
Built-in List Methods

- There are many other built-in functions in python that can be used to change a list
- Most of these functions will actually mutate or change the list itself rather than return a new list

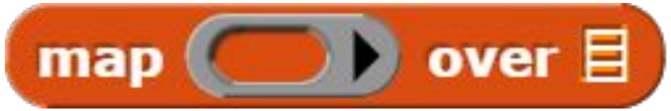
```
some_list.pop()
some_list.append(4)
some_list.remove(3)
```

List Comprehension

Function Variable List

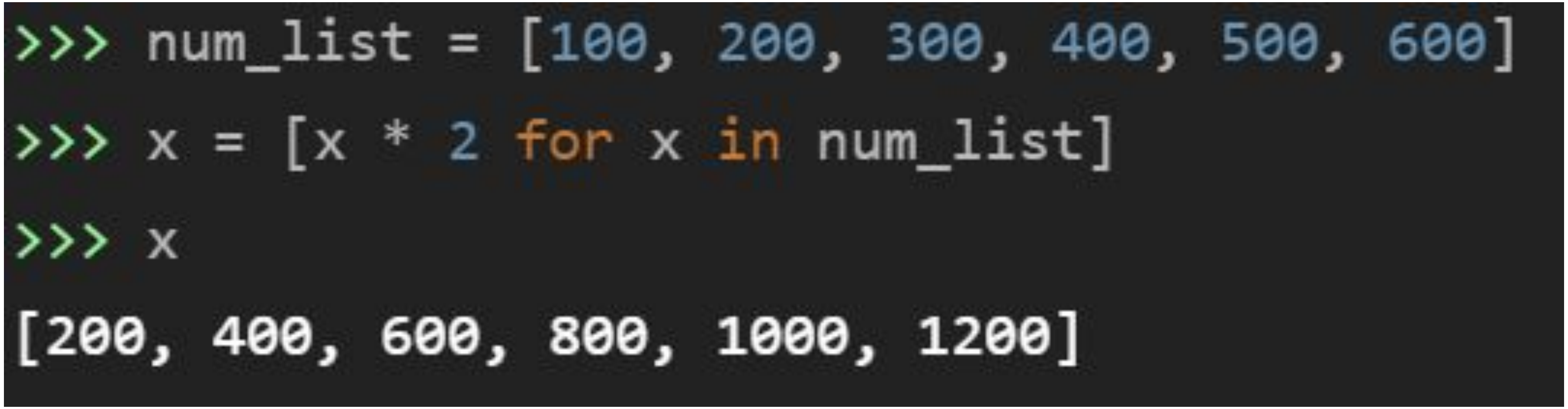


```
[x * 2 for x in num_list]
```



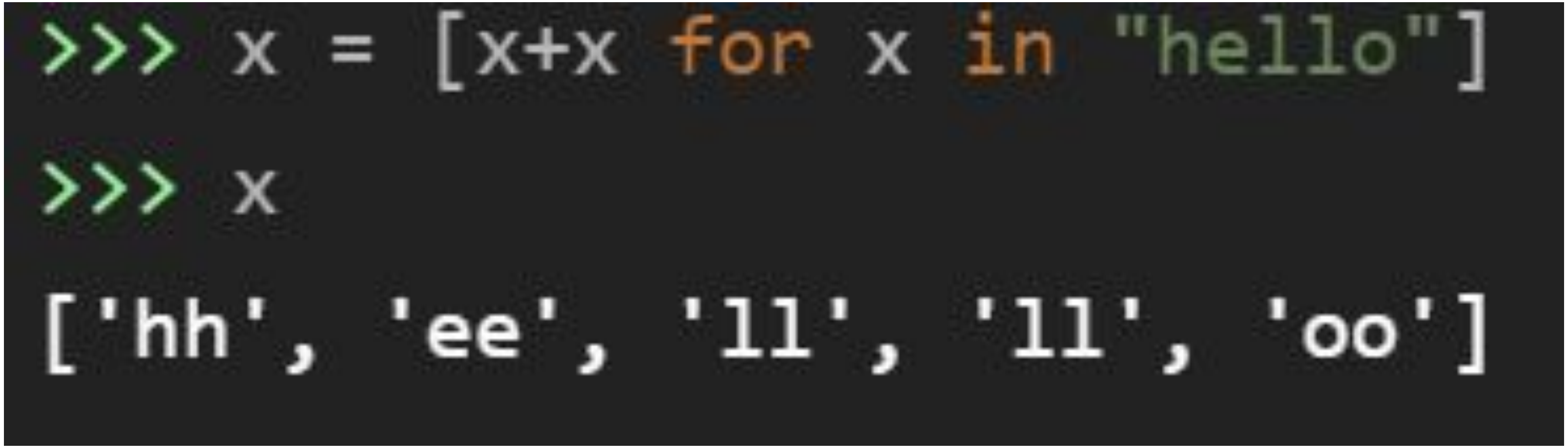
List Comprehension

- Concise ways to iterate over a list or string and perform functions or operations over the items



```
>>> num_list = [100, 200, 300, 400, 500, 600]
>>> x = [x * 2 for x in num_list]
>>> x
[200, 400, 600, 800, 1000, 1200]
```

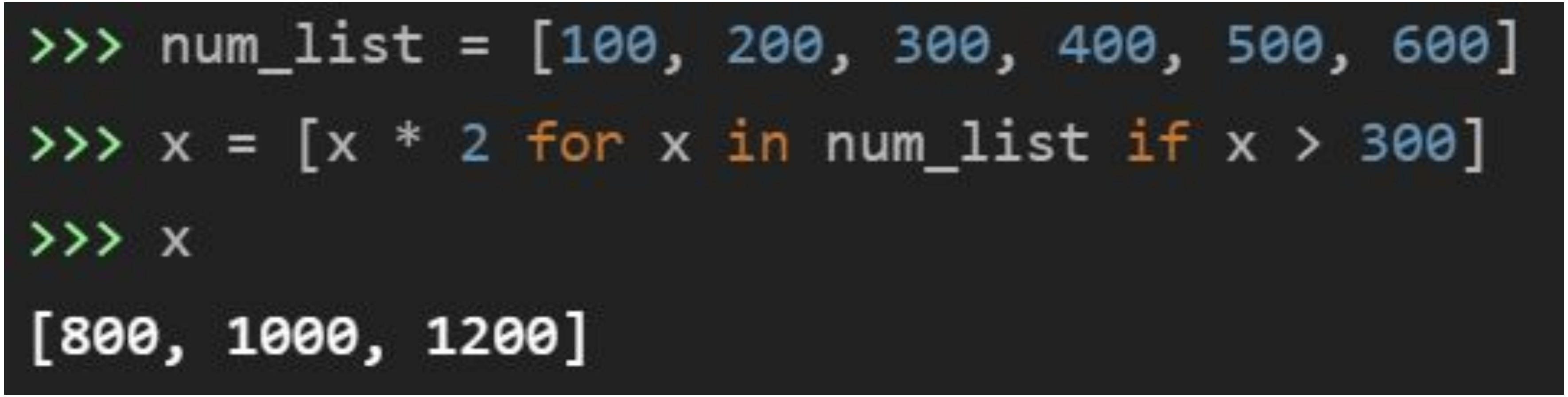
List Comprehension with Strings



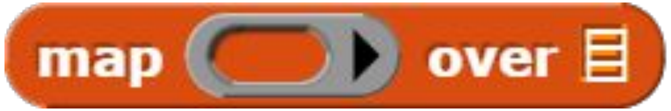
```
>>> x = [x+x for x in "hello"]
>>> x
['hh', 'ee', 'll', 'll', 'oo']
```

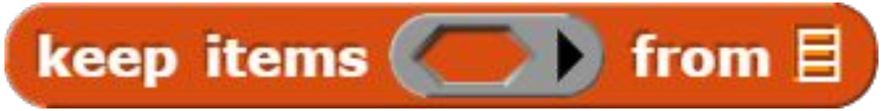
List Comprehension

- We can also use it to filter out items

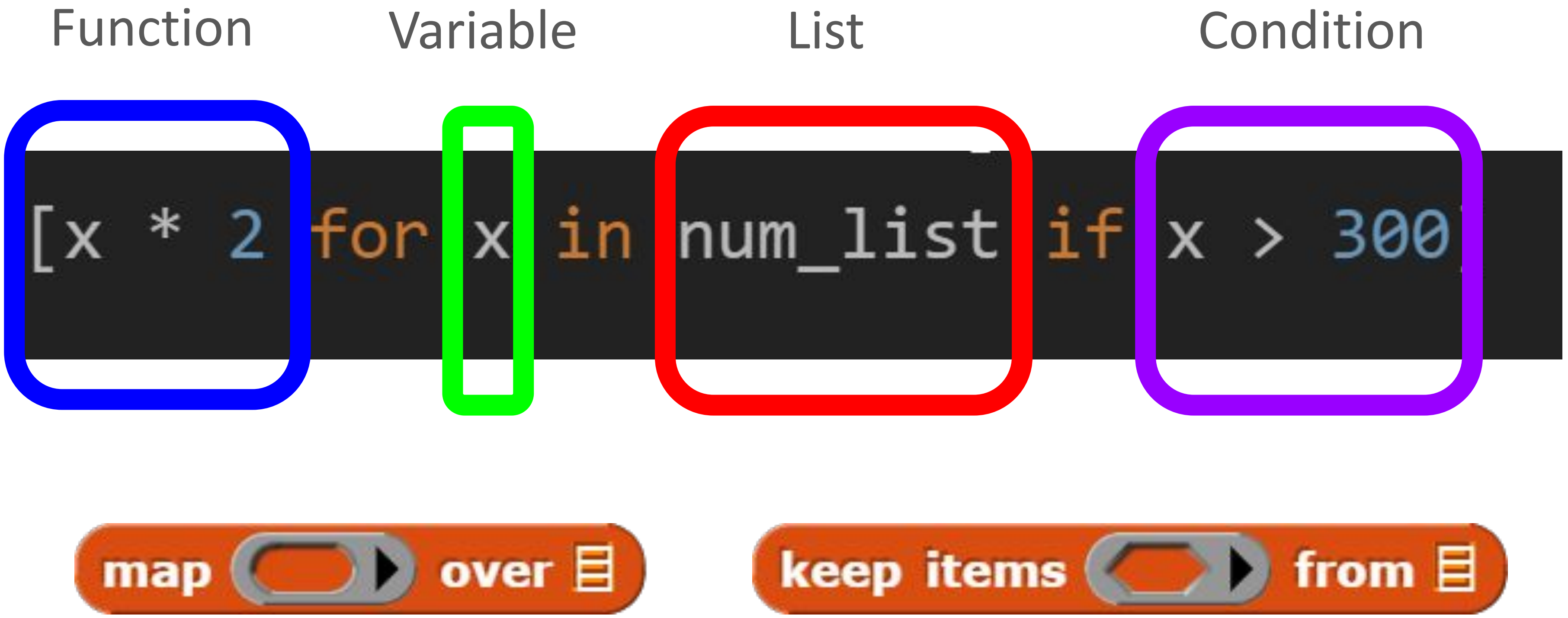


```
>>> num_list = [100, 200, 300, 400, 500, 600]
>>> x = [x * 2 for x in num_list if x > 300]
>>> x
[800, 1000, 1200]
```





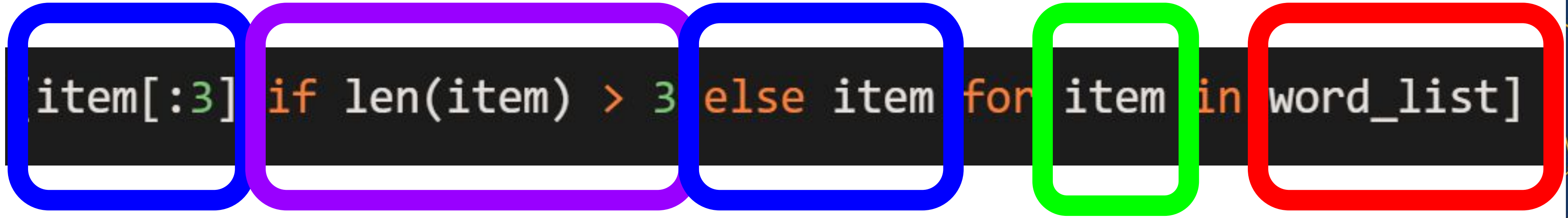
List Comprehension



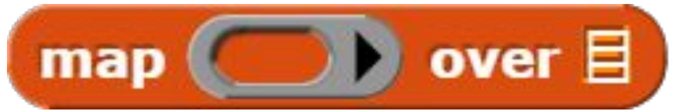
```
[x * 2 for x in num_list if x > 300
```

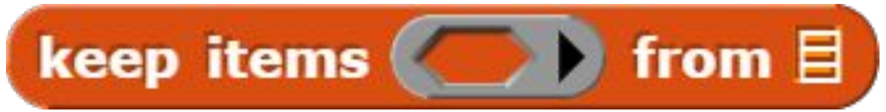
List Comprehension

Function 1 Condition Function 2 Variable List



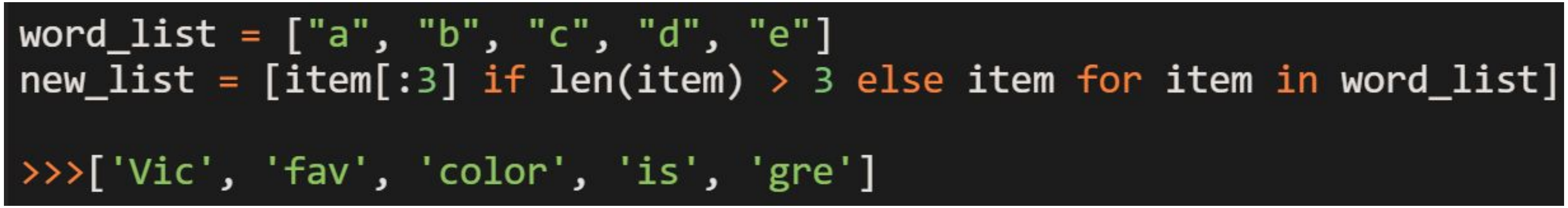
```
[item[:3] if len(item) > 3 else item for item in word_list]
```



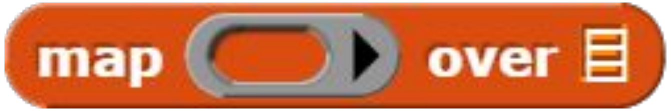


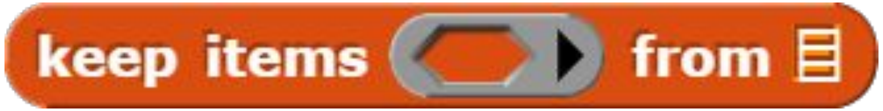
List Comprehension

- Using If and Else in list comprehension



```
word_list = ["a", "b", "c", "d", "e"]
new_list = [item[:3] if len(item) > 3 else item for item in word_list]

>>>['Vic', 'fav', 'color', 'is', 'gre']
```





List Comprehension Summary

map

```
[f(element) for element in iterable]
```

keep

```
[element for element in iterable if cond(element)]
```

map+keep

```
[f(element) for element in iterable if cond(element)]
[f(element) if cond(element) else other for element in iterable]
```

Intro to Tuples

Tuples are similar to lists:

- You create them using comma separated lists inside parentheses rather than square brackets
- You can access values at specific indices with square brackets just like with lists, you just can’t change the values
- Tuples are immutable, lists are mutable

```
some_tuple = (1, 5, 10, 4, 7, 16, 2)

some_list = [1, 5, 10, 4, 7, 16, 2]
```

Why use Tuples

Immutability: Since tuples are immutable, they can be used as keys in dictionaries and elements of sets, which require immutable data types.

Data Integrity: The immutability of tuples ensures that the data cannot be changed, which is useful for fixed collections of items.

Performance: Tuples can be more efficient than lists in terms of memory usage and performance.

Creating a Tuple

```
my_tuple = (1, 'apple', 3.14, True, 'Python')
```

Access Tuple Elements

```
my_tuple = (1, 'apple', 3.14, True, 'Python')

print(my_tuple[0]) # Output: 1
print(my_tuple[2]) # Output: 3.14
print(my_tuple[-1]) # Output: Python
```

Tuple Operations - Concatenate

```
tuple1 = (1, 2, 3)

tuple2 = ('a', 'b', 'c')

result = tuple1 + tuple2

print(result) # Output: (1, 2, 3, 'a', 'b', 'c')
```

Tuple Operations - Repeat

```
my_tuple = ('repeat',)

result = my_tuple * 3

print(result)

# Output: ('repeat', 'repeat', 'repeat')
```

Slicing Tuples

```
my_tuple = (1, 2, 3, 4, 5)

print(my_tuple[1:4])

# Output: (2, 3, 4)
```

Tuple Unpacking - Unpack elements into separate variables

```
person = ('Alice', 30, 'Engineer')

name, age, profession = person



print(name)  # Output: Alice

print(age)  # Output: 30

print(profession)  # Output: Engineer
```

Iterating Over Tuples

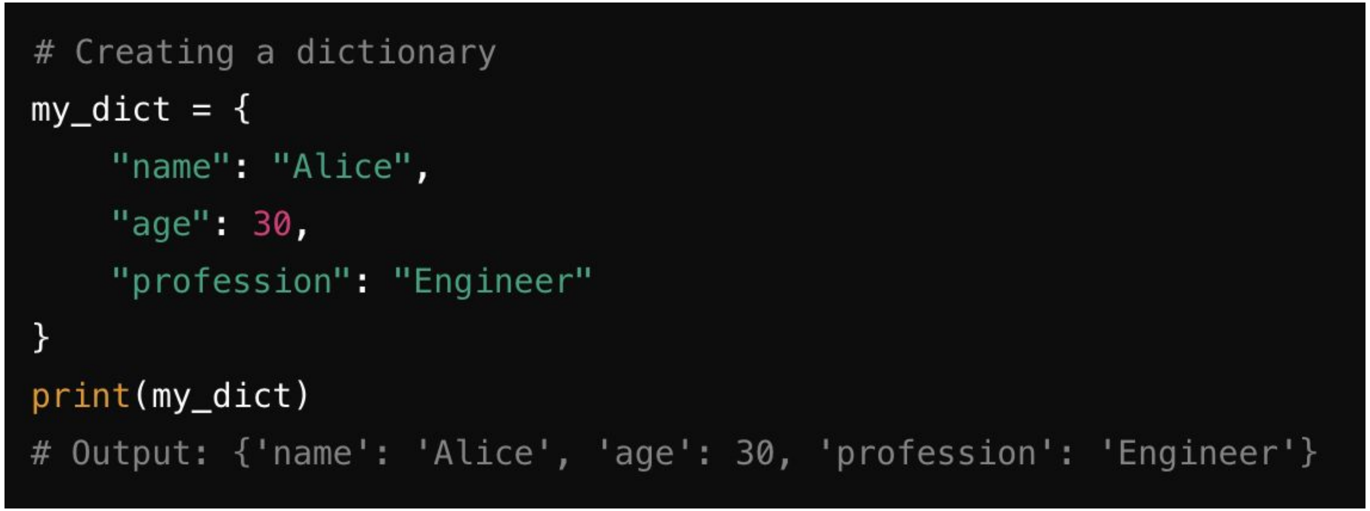
```
my_tuple = (1, 'apple', 3.14, True, 'Python')

for item in my_tuple:

	print(item)
```

Task 1

- 1) make a list of integers [1,2,3,4,5]
- 2) write a function that
  - that takes in a list
  - returns the minimum and maximum as a tuple
  - Hint:
    - Create 2 variables -> Min and Max
    - Return a Tuple with Min and Max as the values

Intro to Python Dictionaries

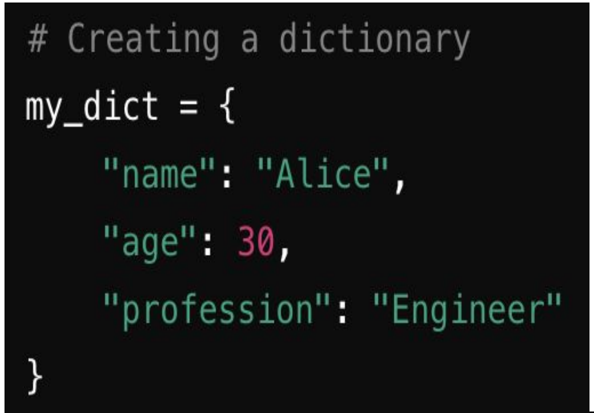
- A dictionary in Python is an unordered collection of key-value pairs.
- Each key is unique and maps to a value.
- Dictionaries are mutable, meaning you can change their content after creation



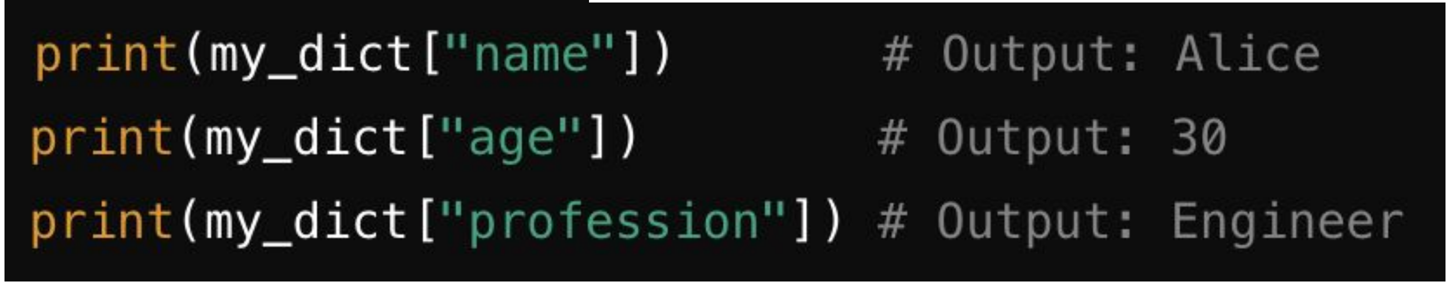
```python
# Creating a dictionary
my_dict = {
    "name": "Alice",
    "age": 30,
    "profession": "Engineer"
}
print(my_dict)
# Output: {'name': 'Alice', 'age': 30, 'profession': 'Engineer'}
```

Accessing Values

You can access values in a dictionary by using their corresponding keys.



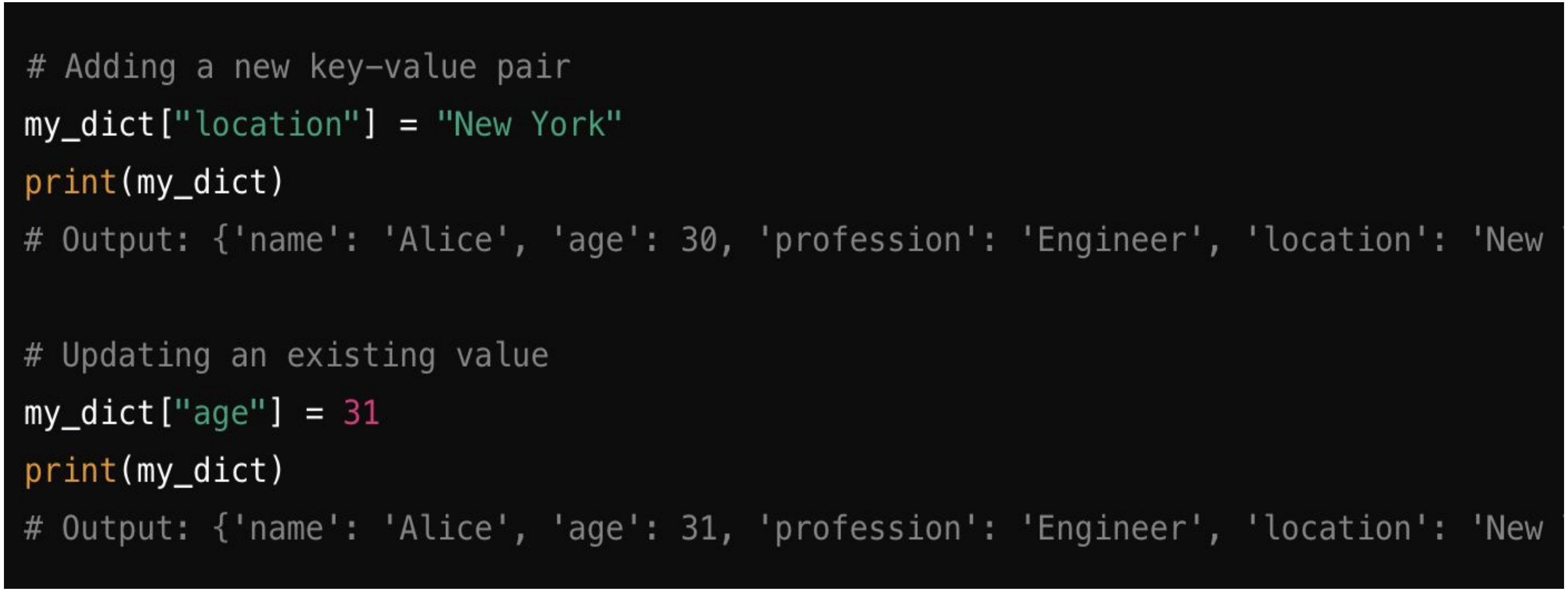
```python
# Creating a dictionary
my_dict = {
    "name": "Alice",
    "age": 30,
    "profession": "Engineer"
}
```



```python
print(my_dict["name"])       # Output: Alice
print(my_dict["age"])        # Output: 30
print(my_dict["profession"]) # Output: Engineer
```

Adding and Updating Entries

You can add new key-value pairs or update existing ones.



```
# Adding a new key-value pair
my_dict["location"] = "New York"
print(my_dict)
# Output: {'name': 'Alice', 'age': 30, 'profession': 'Engineer', 'location': 'New

# Updating an existing value
my_dict["age"] = 31
print(my_dict)
# Output: {'name': 'Alice', 'age': 31, 'profession': 'Engineer', 'location': 'New
```

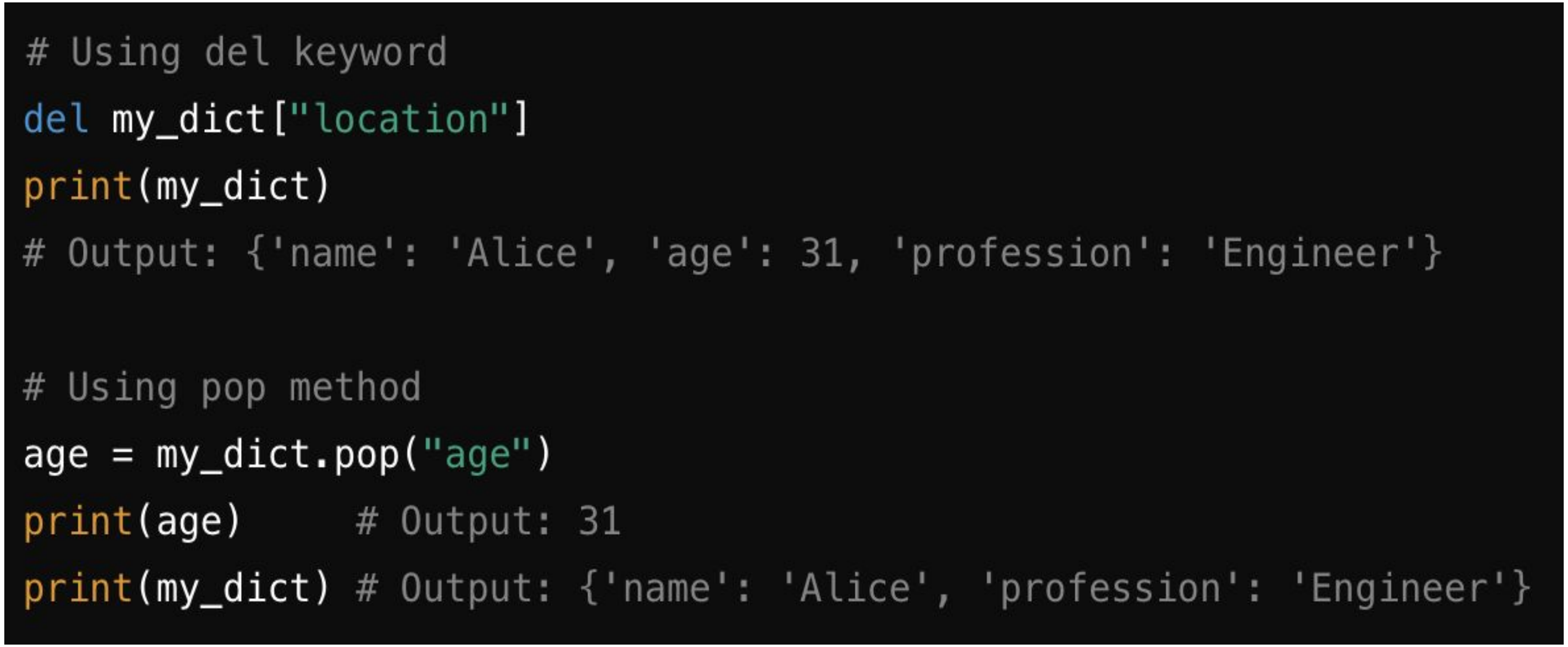
Intro to Dictionaries

- Unlike lists, dictionaries do not have any pre-defined order
- A collection of un-ordered **key-value pairs**
    - The key points us to the value
- We can have repeated values but not repeated keys
    - Values can be mutable but keys must be immutable

```
phonebook = {}
phonebook["Alonzo"] = "713-474-2731"
phonebook["Oznola"] = "713-474-3750"
phonebook["Tom Bates"] = "510-981-7100"

print(phonebook["Tom Bates"])
```

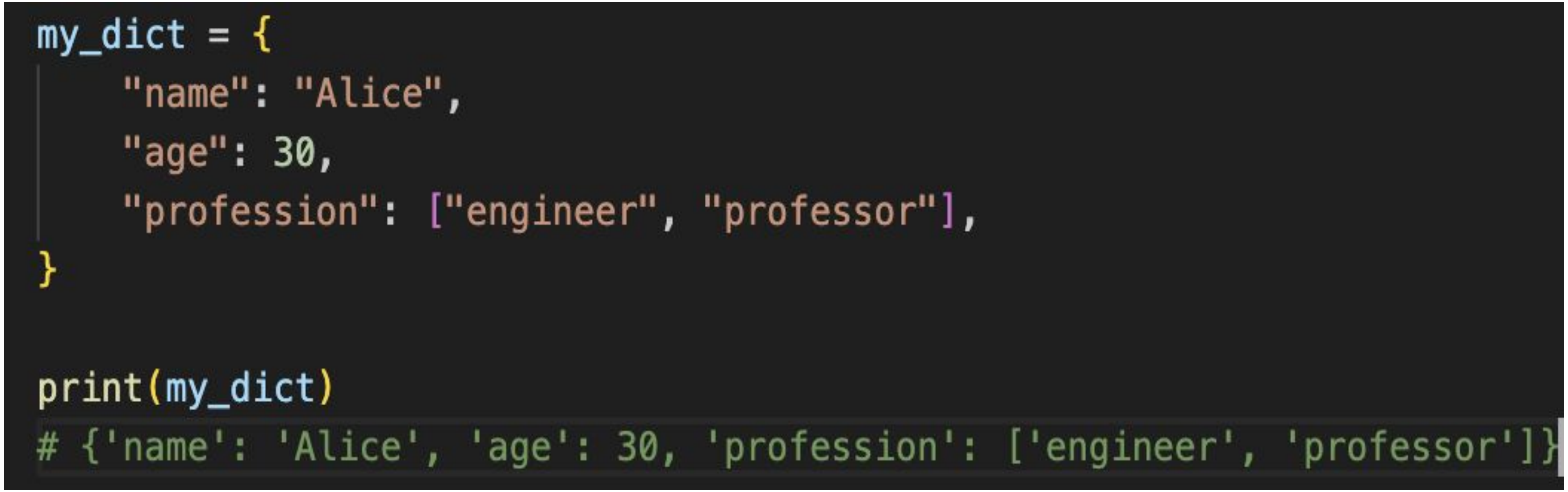
Removing Entries

You can remove key-value pairs using the `del` keyword or the `pop` method.



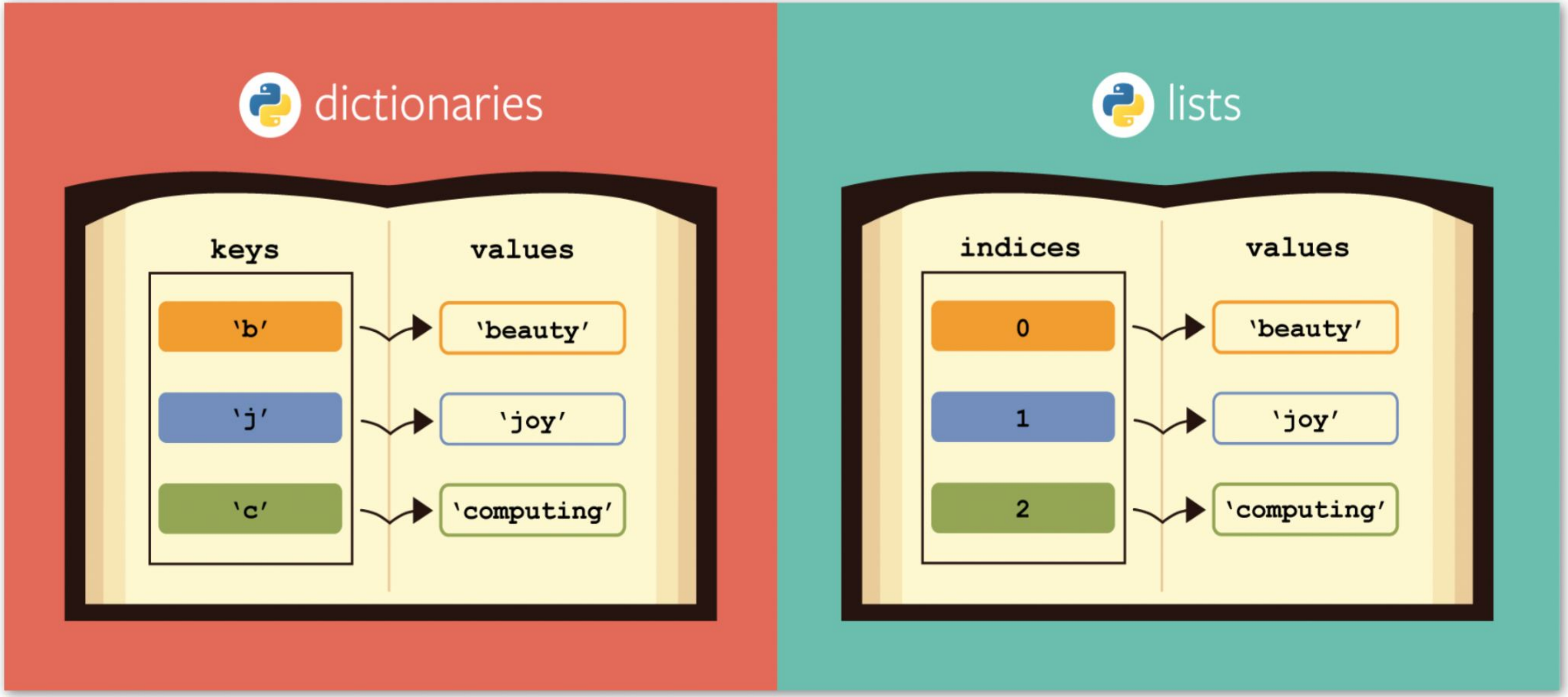
```python
# Using del keyword
del my_dict["location"]
print(my_dict)
# Output: {'name': 'Alice', 'age': 31, 'profession': 'Engineer'}

# Using pop method
age = my_dict.pop("age")
print(age)     # Output: 31
print(my_dict) # Output: {'name': 'Alice', 'profession': 'Engineer'}
```

You can have more than 1 value for each key!



```
my_dict = {
    "name": "Alice",
    "age": 30,
    "profession": ["engineer", "professor"],
}

print(my_dict)
# {'name': 'Alice', 'age': 30, 'profession': ['engineer', 'professor']}
```

Dictionaries vs Lists



Why Dictionaries

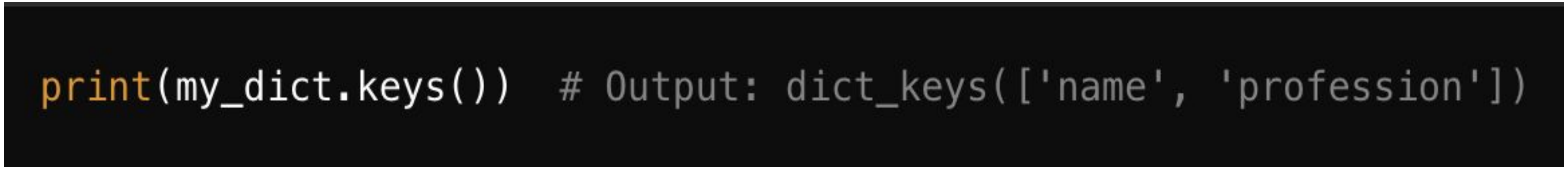
Fast Lookups/Search: Dictionaries provide fast lookups for retrieving values using keys.

Unique Keys: Each key in a dictionary is unique, preventing duplicate entries.

Flexible Data Structures: Dictionaries can store various types of data, including other dictionaries.

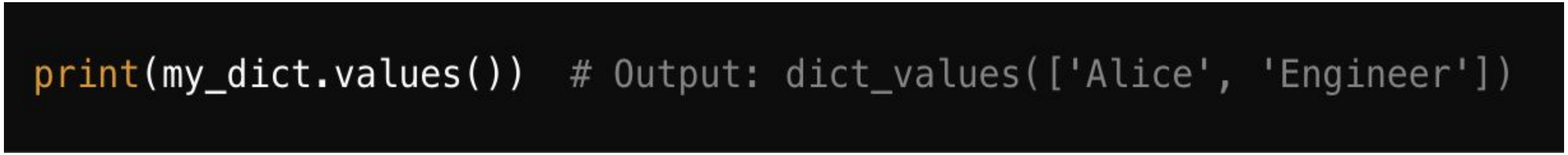
Dictionary Methods

keys(): Returns a view object of all the keys in the dictionary.



```
print(my_dict.keys())  # Output: dict_keys(['name', 'profession'])
```

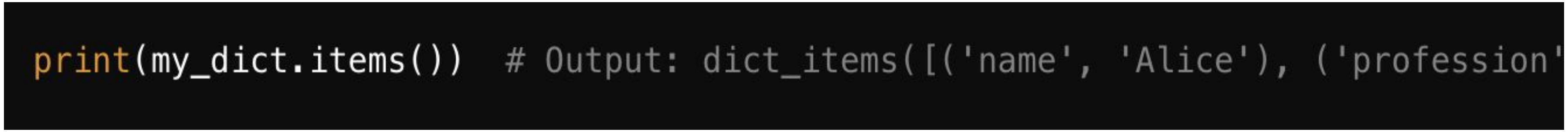
values(): Returns a view object of all the values in the dictionary.



```
print(my_dict.values())  # Output: dict_values(['Alice', 'Engineer'])
```

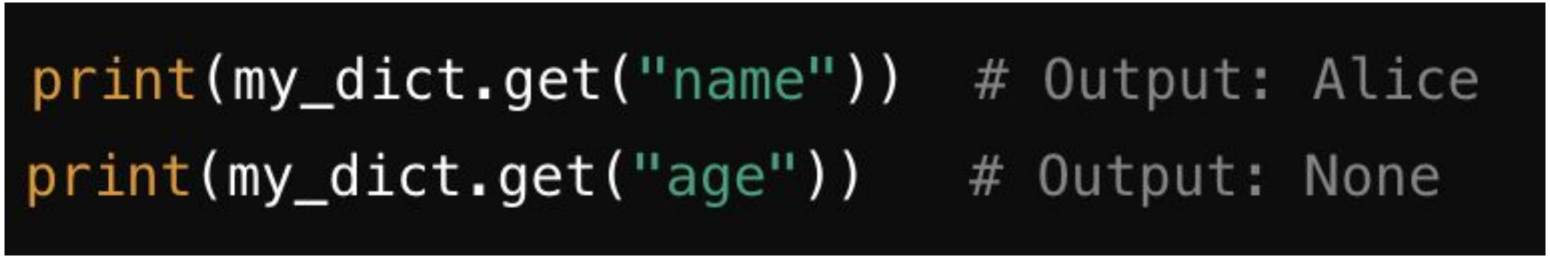
Dictionary Methods

items(): Returns a view object of all the key-value pairs in the dictionary.



```
print(my_dict.items())  # Output: dict_items([('name', 'Alice'), ('profession'
```

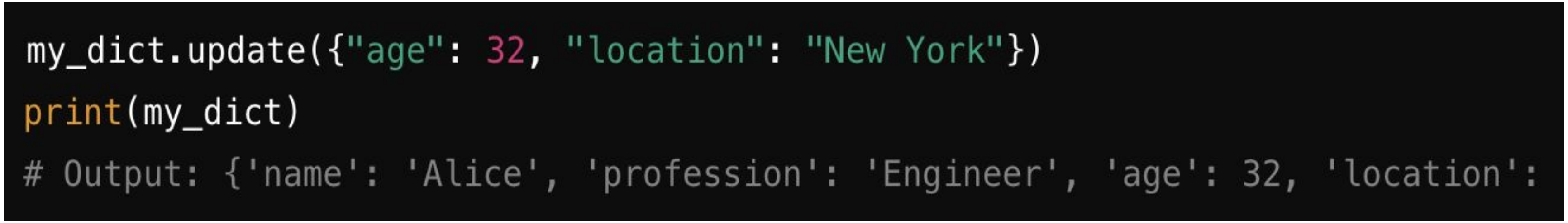
get(): Returns the value for a specified key if the key is in the dictionary.



```
print(my_dict.get("name"))  # Output: Alice
print(my_dict.get("age"))   # Output: None
```

Dictionary Methods

update(): Updates the dictionary with elements from another dictionary object or from an iterable of key-value pairs.



```
my_dict.update({"age": 32, "location": "New York"})
print(my_dict)
# Output: {'name': 'Alice', 'profession': 'Engineer', 'age': 32, 'location':
```

Iterating through a Dictionary

- Iterate through **keys**
```
for k in my_dictionary.keys():
for k in my_dictionary:
```

- Iterate through **values**
```
for v in my_dictionary.values():
```

- Iterate through **keys and values**
```
for k,v in my_dictionary.items():
```

- Check **if k is a key** in dictionary
```
k in my_dictionary.keys()
k in my_dictionary
```

- Check **if v is a value** in dictionary
```
v in my_dictionary.values()
```

Task 2: How many A’s?

1) Make a dictionary:
    a) Test_scores = { "a": 90, "b": 70, "c": 100, "d": 60 }

2) Write a function that takes in a dictionary and returns the number of scores 90 or above

Task 3: Sum Values

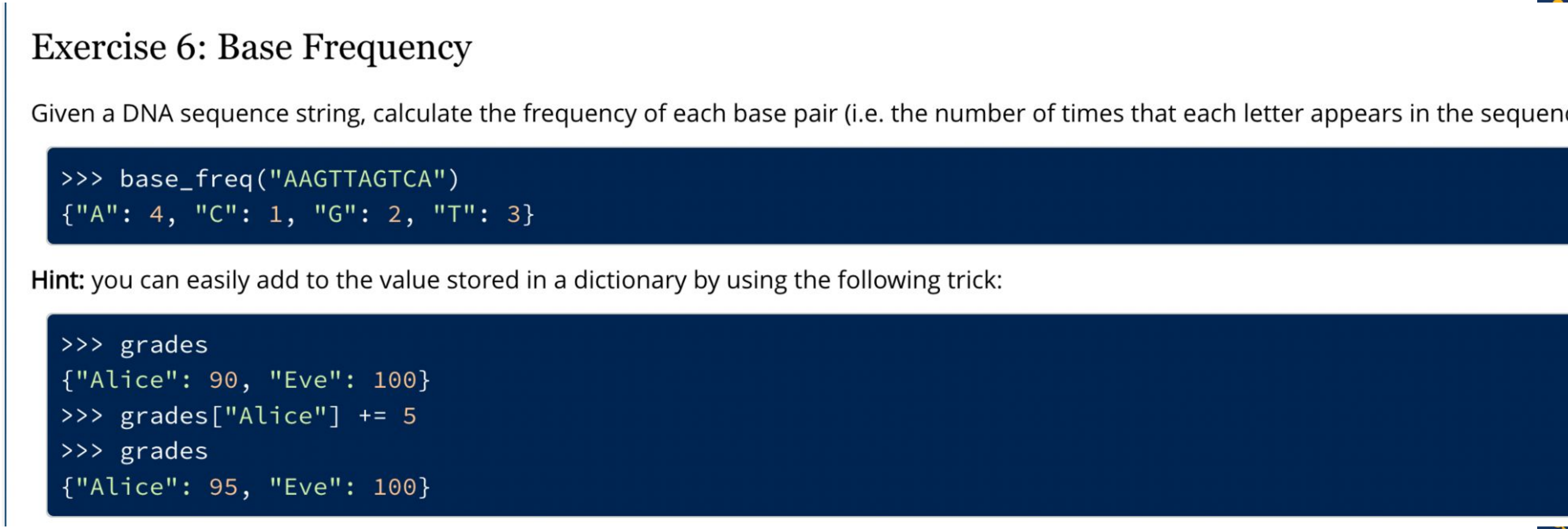
1) Make a dictionary:
    a) my_dict = { "a": 10, "b": 20, "c": 30, "d": 40 }

2) Write a function that takes in a dictionary and returns the sum of the values

Challenge:

-how could you modify this challenge to work with a dictionary that has multiple data types as values?

1) Go back through Labs and see what students need to know…build basic activities from that



Exercise 6: Base Frequency

Given a DNA sequence string, calculate the frequency of each base pair (i.e. the number of times that each letter appears in the sequenc

```
>>> base_freq("AAGTTAGTCA")
{"A": 4, "C": 1, "G": 2, "T": 3}
```

Hint: you can easily add to the value stored in a dictionary by using the following trick:

```
>>> grades
{"Alice": 90, "Eve": 100}
>>> grades["Alice"] += 5
>>> grades
{"Alice": 95, "Eve": 100}
```

Exercise 7.1: Substitute Characters

Write a function `substitute_chars` that takes as input a string and a dictionary. The dictionary will have characters as keys and values, which represent what to replace certain characters with (as shown above). This function should return a string with each character substituted with that characters's value from the dictionary. If a character doesn't exist as a key in the dictionary, it should be left alone.

```
>>> replacements = {"S":"Z", "E":"U", "T":"P", "A":"M"}
>>> substitute_chars("SECRET MESSAGE", replacements)
"ZUCRUP MUZZMGU"
```

Exercise 7.2: Invert Dictionary

Write a function `invert_dict` that takes as input a dictionary. The output should be a new dictionary that has each key, value pair reversed. The input dictionary should be left unchanged.

```
>>> original = {"A":"X", "B":"Y", "C":"Z"}
>>> invert_dict(original)
{"X":"A", "Z":"C", "Y":"B"}
>>> original
{"A":"X", "B":"Y", "C":"Z"}
```

Question: What does this function do when some of the values in the dictionary are identical? Can a dictionary have identical keys? Talk with your partner, then try using your function on one such dictionary, like this: `invert_dict({"A":"X", "B":"Z", "C":"Z"})` and see if you were right.