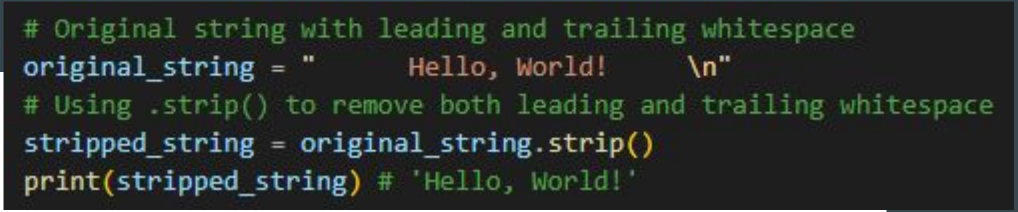
COSI 10A Recitation #9

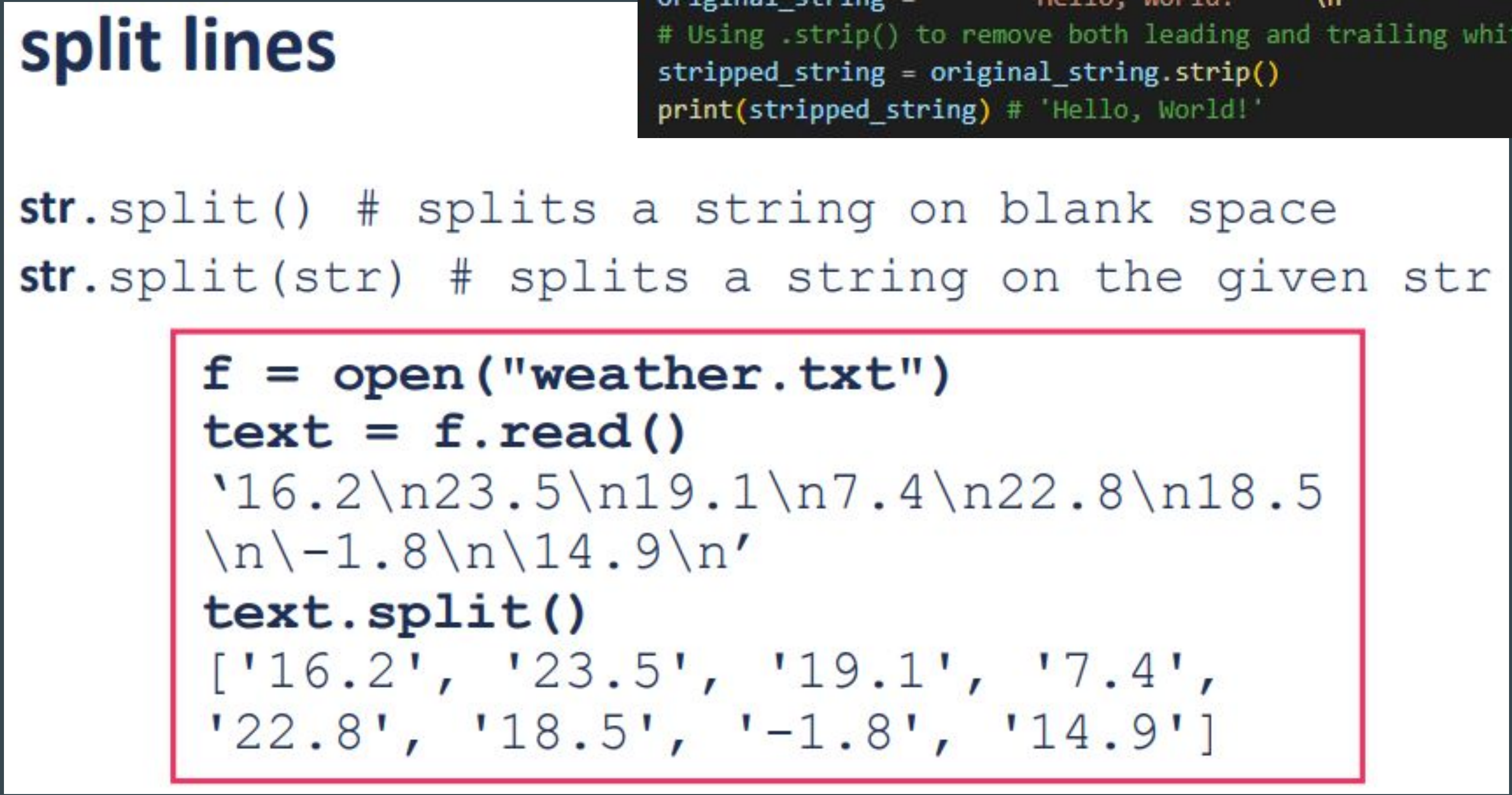
Agenda: File Processing, Sets, Dictionaries

Review: File Processing

.strip() is also helpful to get rid of leading and trailing spaces including “\n” or “\t”



```
# Original string with leading and trailing whitespace
original_string = "      Hello, World!     \n"
# Using .strip() to remove both leading and trailing whitespace
stripped_string = original_string.strip()
print(stripped_string) # 'Hello, World!'
```

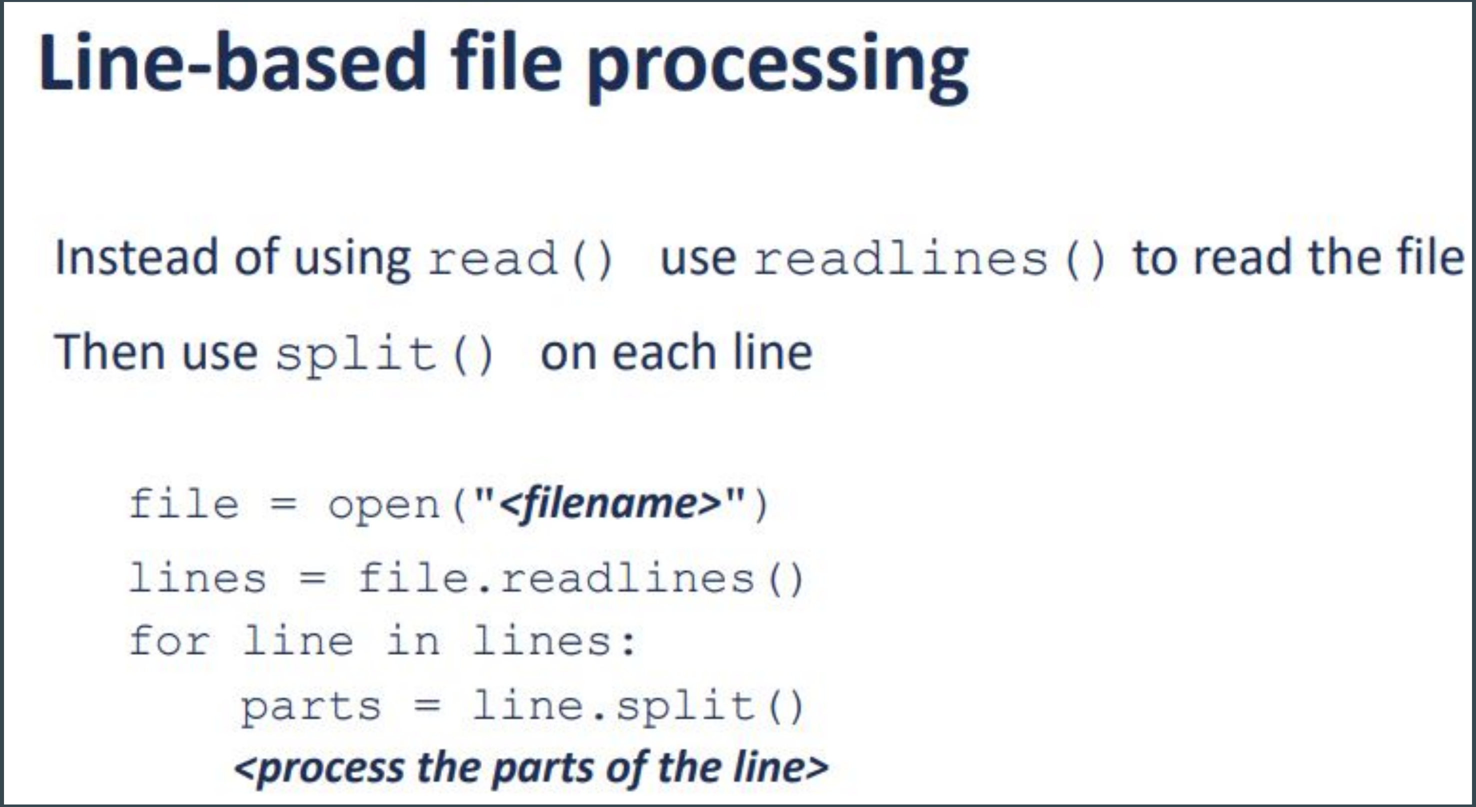


split lines

```
str.split() # splits a string on blank space
str.split(str) # splits a string on the given str
```

```
f = open("weather.txt")
text = f.read()
‘16.2\n23.5\n19.1\n7.4\n22.8\n18.5
\n\-1.8\n\14.9\n’
text.split()
['16.2', '23.5', '19.1', '7.4',
'22.8', '18.5', '-1.8', '14.9']
```

Review: File Processing



Line-based file processing

Instead of using `read()` use `readlines()` to read the file

Then use `split()` on each line

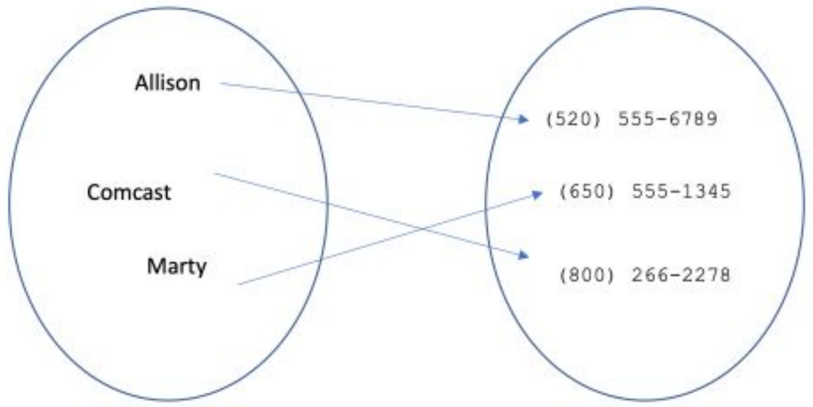
```
file = open("<filename>")
lines = file.readlines()
for line in lines:
    parts = line.split()
    <process the parts of the line>
```

Review: Dictionaries

Creating dictionaries

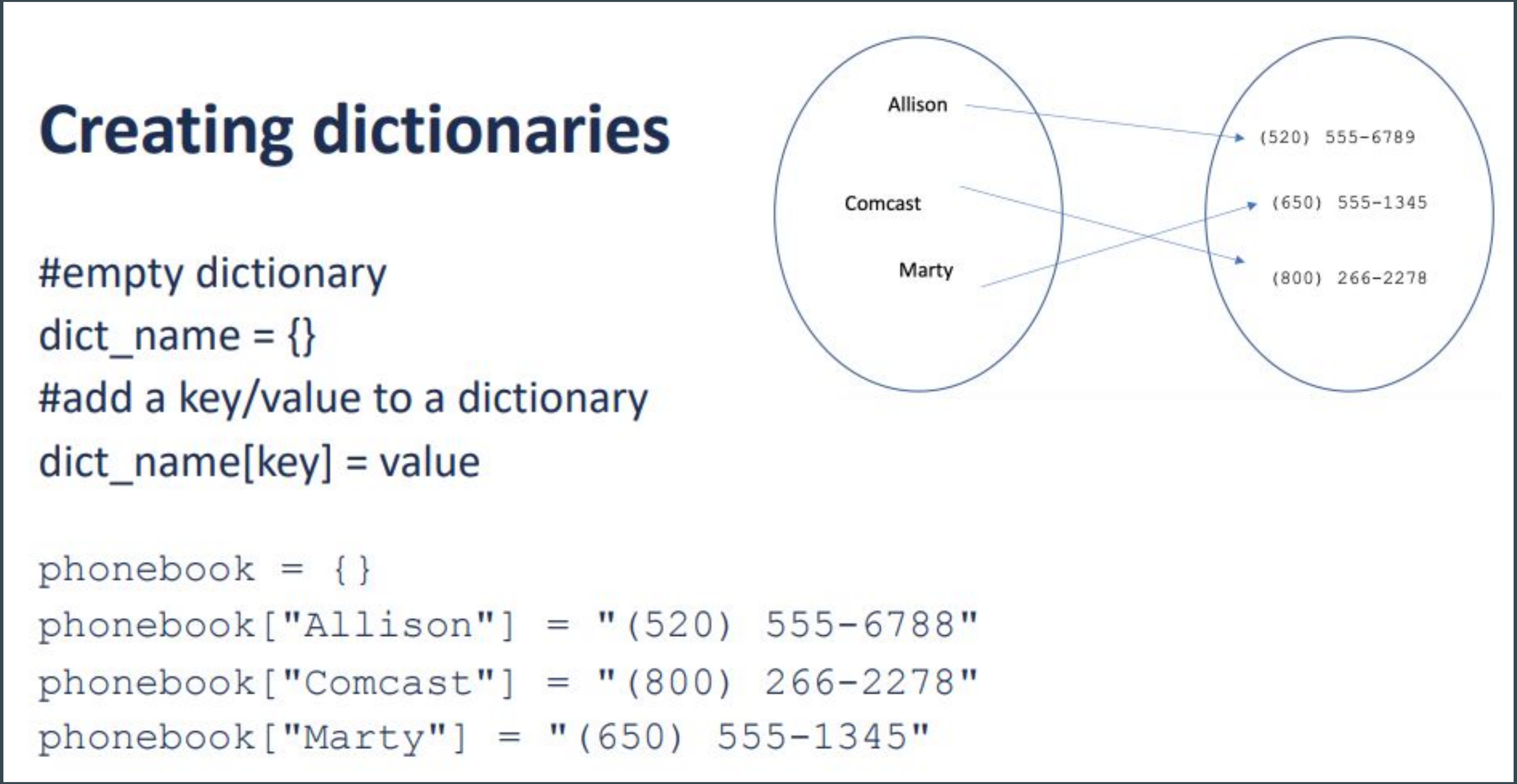
Creating a dictionary

{key : value, ..., key : value}

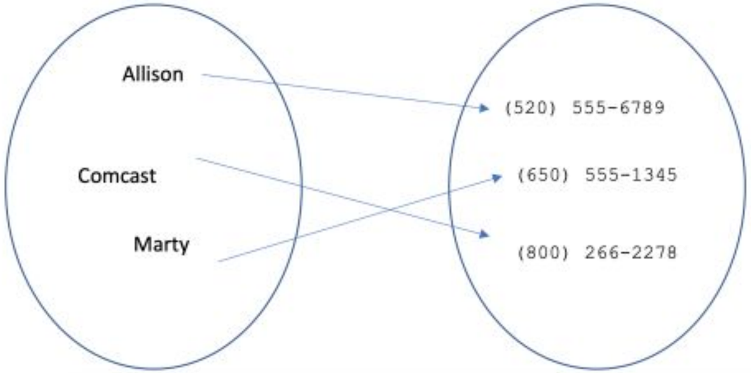


```
phonebook = {"Allison": "(520) 555-6788",
             "Comcast":"(800) 266-2278",
             "Marty" : "(650) 555-1345" }
```

Review: Dictionaries



Creating dictionaries

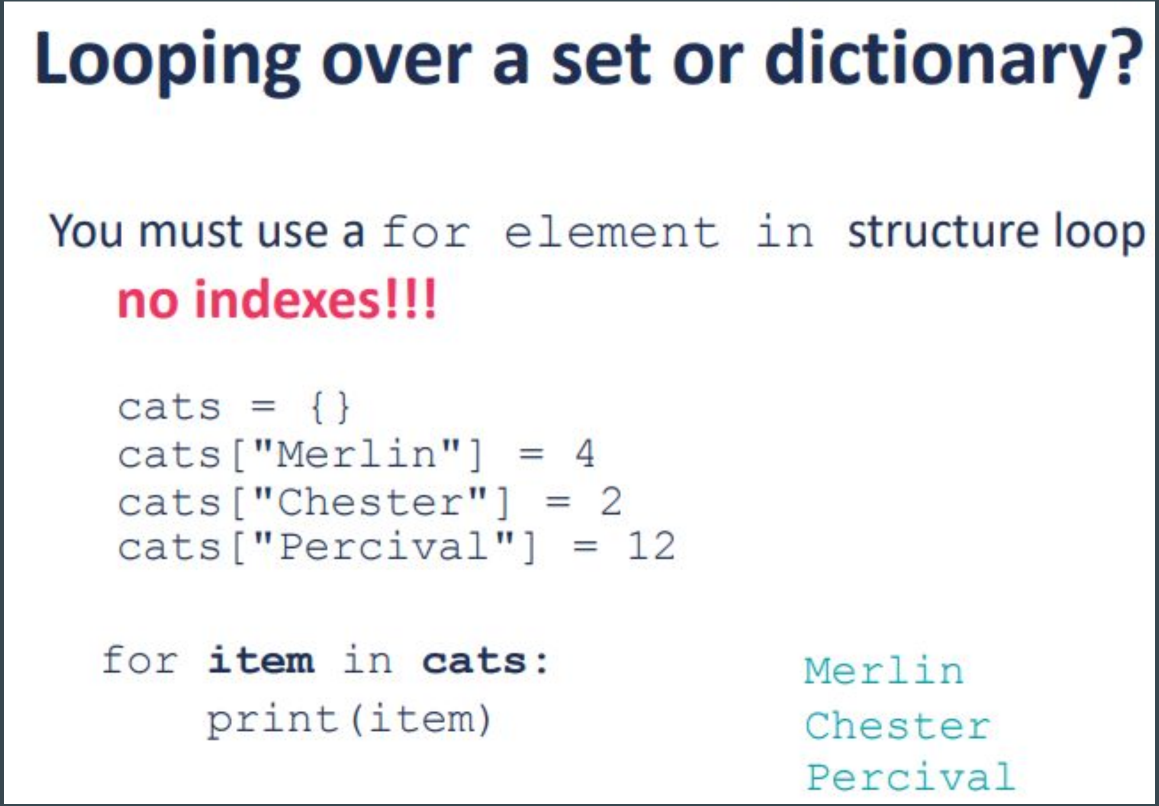


#empty dictionary
dict_name = {}
#add a key/value to a dictionary
dict_name[key] = value

```
phonebook = {}
phonebook["Allison"] = "(520) 555-6788"
phonebook["Comcast"] = "(800) 266-2278"
phonebook["Marty"] = "(650) 555-1345"
```

Review: Dictionaries

Note: Dictionaries are ordered in that looping through them will give you the keys in order of insertion



Looping over a set or dictionary?

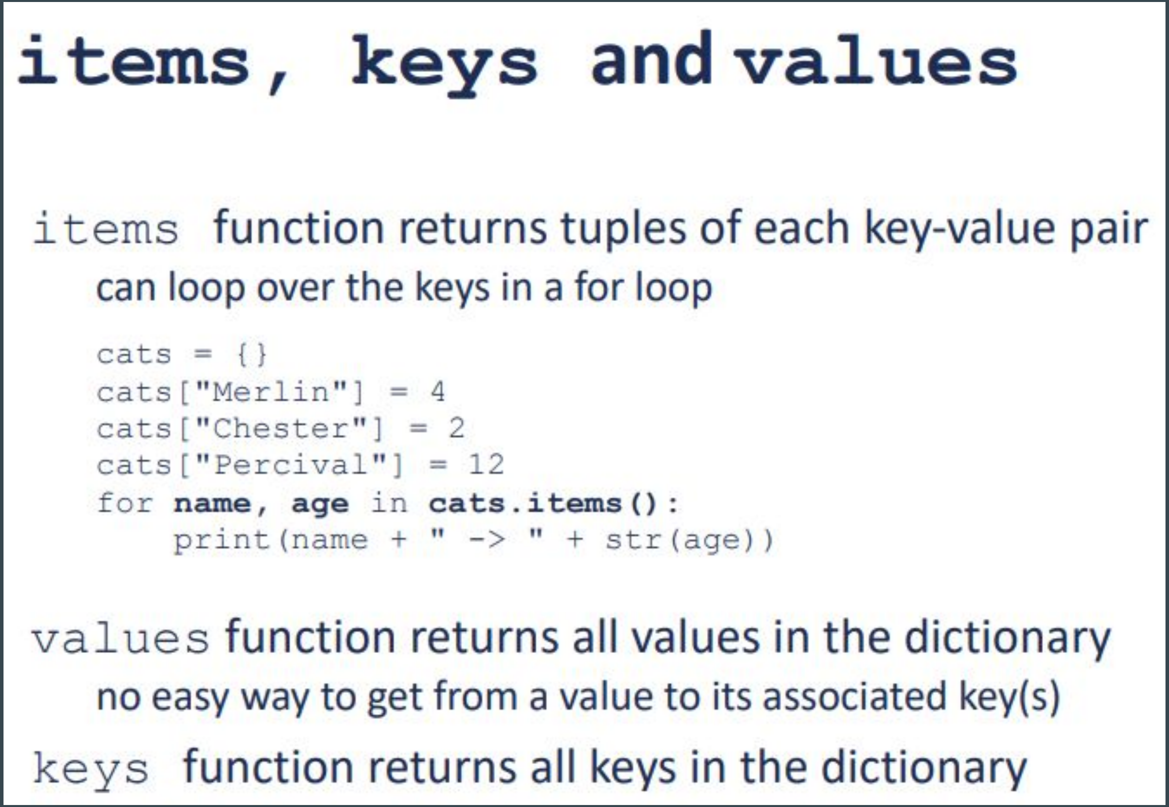
You must use a `for element in` structure loop

no indexes!!!

```
cats = {}
cats["Merlin"] = 4
cats["Chester"] = 2
cats["Percival"] = 12

for item in cats:
    print(item)
```

```
Merlin
Chester
Percival
```

Review: Dictionaries



items, keys and values

`items` function returns tuples of each key-value pair

can loop over the keys in a for loop

```
cats = {}
cats["Merlin"] = 4
cats["Chester"] = 2
cats["Percival"] = 12
for name, age in cats.items():
    print(name + " -> " + str(age))
```

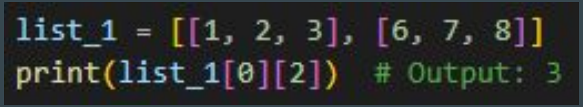
`values` function returns all values in the dictionary

no easy way to get from a value to its associated key(s)

`keys` function returns all keys in the dictionary

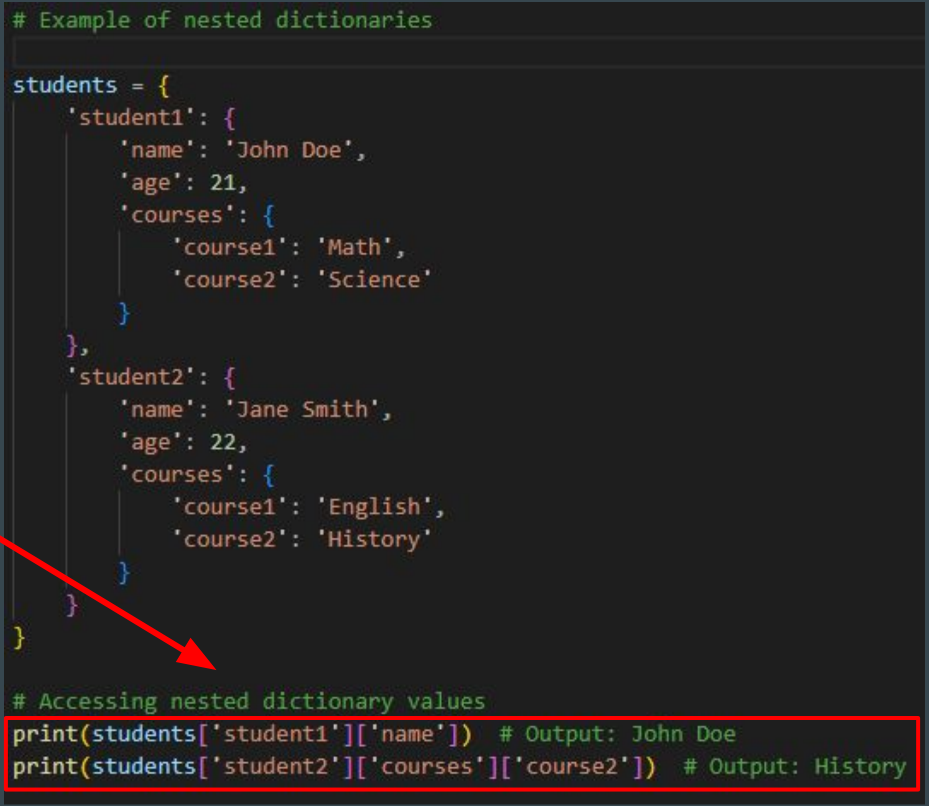
Review: Dictionaries

Similar to nested lists:



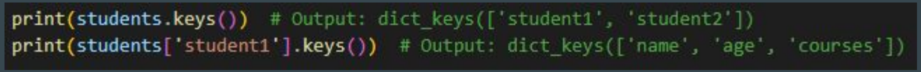
```python
list_1 = [[1, 2, 3], [6, 7, 8]]
print(list_1[0][2])  # Output: 3
```

Dictionaries can also be nested just like lists!



```python
# Example of nested dictionaries

students = {
    'student1': {
        'name': 'John Doe',
        'age': 21,
        'courses': {
            'course1': 'Math',
            'course2': 'Science'
        }
    },
    'student2': {
        'name': 'Jane Smith',
        'age': 22,
        'courses': {
            'course1': 'English',
            'course2': 'History'
        }
    }
}

# Accessing nested dictionary values
print(students['student1']['name'])  # Output: John Doe
print(students['student2']['courses']['course2'])  # Output: History
```

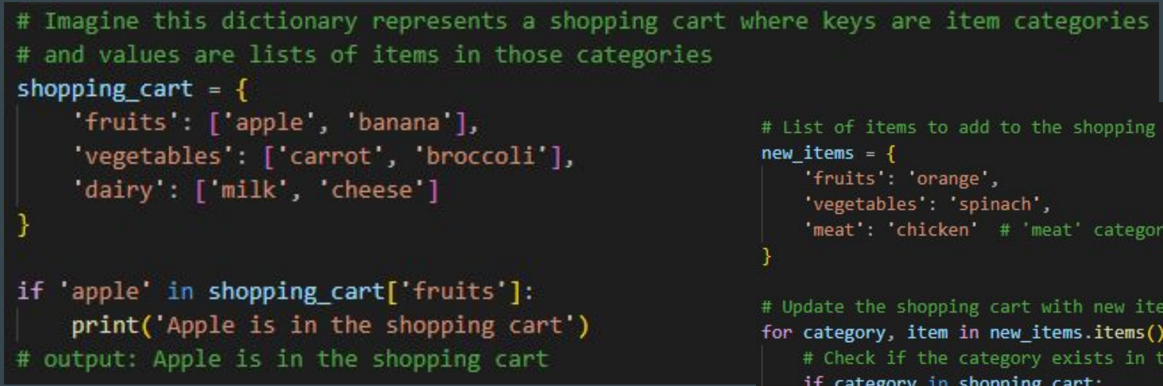
Key Structure:



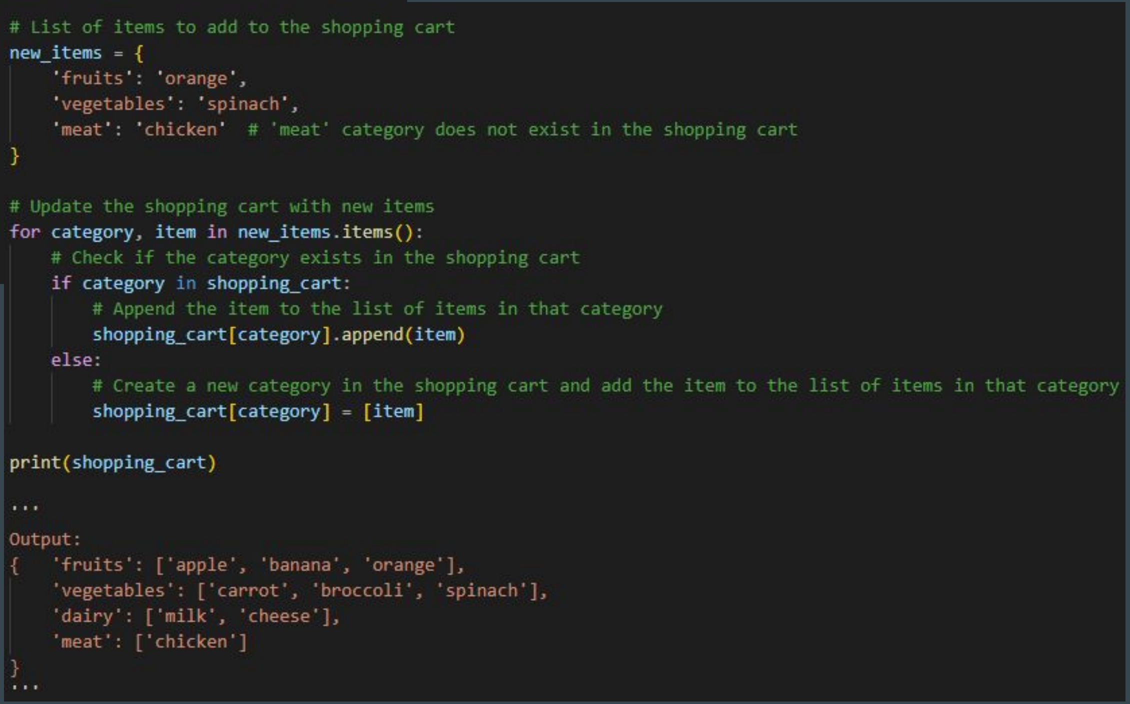
```python
print(students.keys())  # Output: dict_keys(['student1', 'student2'])
print(students['student1'].keys())  # Output: dict_keys(['name', 'age', 'courses'])
```

Review: Dictionaries

Checking for elements in dictionaries are just like lists, use a conditional!



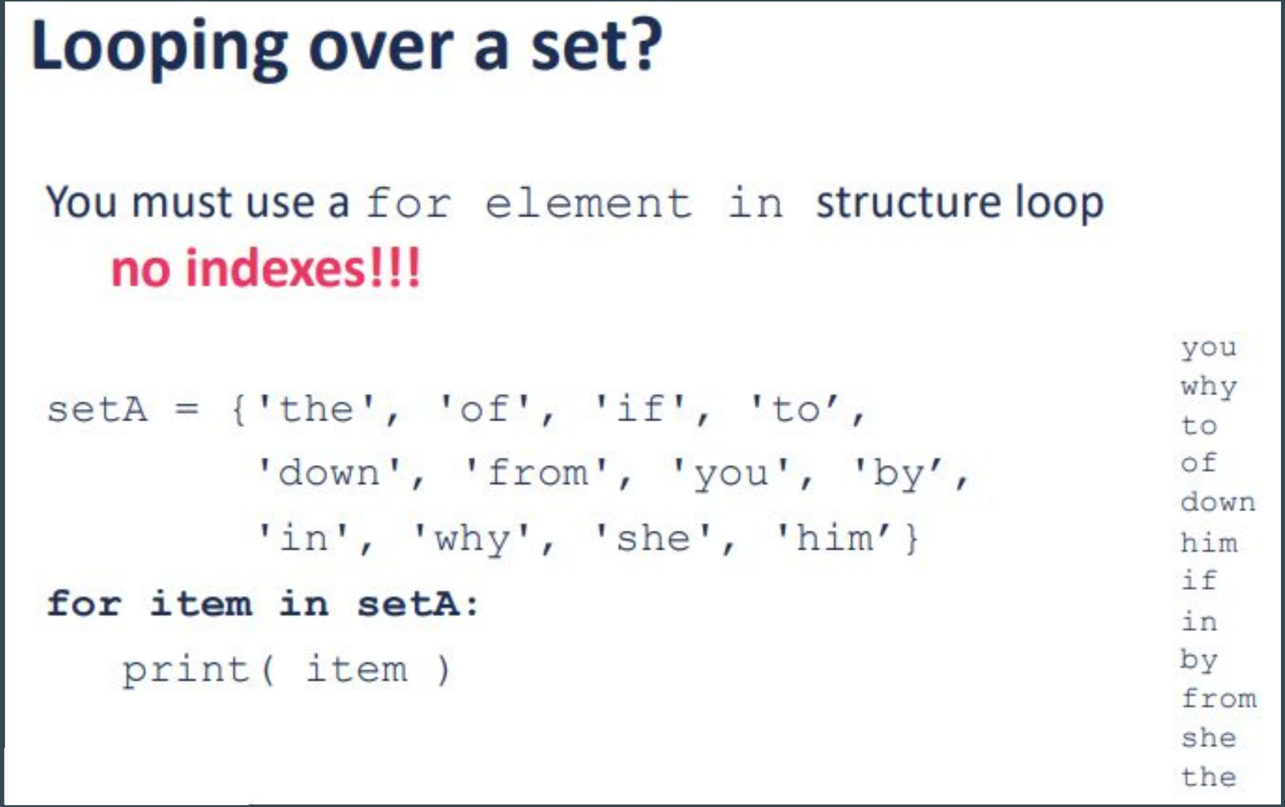
```python
# Imagine this dictionary represents a shopping cart where keys are item categories
# and values are lists of items in those categories
shopping_cart = {
    'fruits': ['apple', 'banana'],
    'vegetables': ['carrot', 'broccoli'],
    'dairy': ['milk', 'cheese']
}

if 'apple' in shopping_cart['fruits']:
    print('Apple is in the shopping cart')
# output: Apple is in the shopping cart
```



```python
# List of items to add to the shopping cart
new_items = {
    'fruits': 'orange',
    'vegetables': 'spinach',
    'meat': 'chicken'  # 'meat' category does not exist in the shopping cart
}

# Update the shopping cart with new items
for category, item in new_items.items():
    # Check if the category exists in the shopping cart
    if category in shopping_cart:
        # Append the item to the list of items in that category
        shopping_cart[category].append(item)
    else:
        # Create a new category in the shopping cart and add the item to the list of items in that category
        shopping_cart[category] = [item]

print(shopping_cart)

...
Output:
{   'fruits': ['apple', 'banana', 'orange'],
    'vegetables': ['carrot', 'broccoli', 'spinach'],
    'dairy': ['milk', 'cheese'],
    'meat': ['chicken']
}
...
```

Review: Sets

Note: Sets are unordered, so you cannot be sure in which order the items will appear.



Looping over a set?

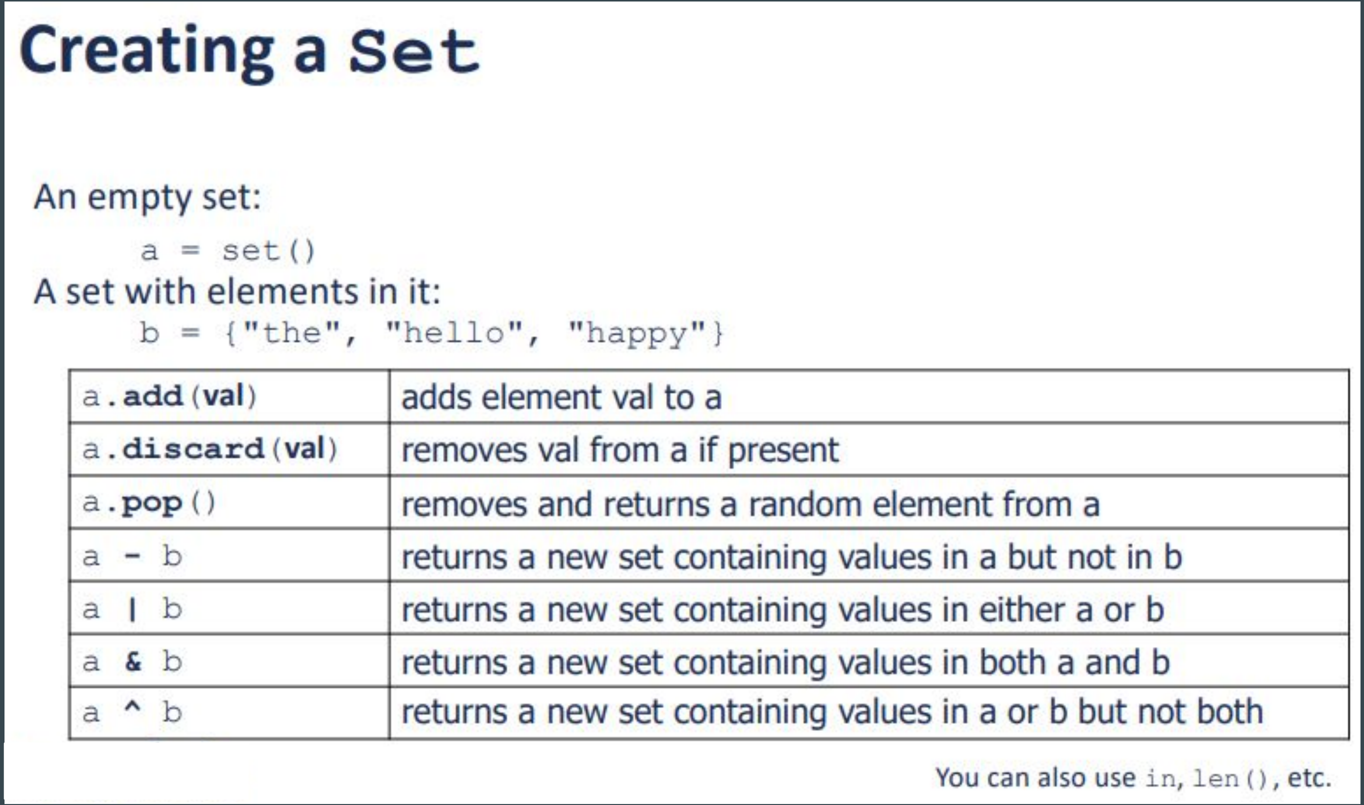
You must use a `for element in` structure loop

no indexes!!!

```
setA = {'the', 'of', 'if', 'to’,
        'down', 'from', 'you', 'by’,
        'in', 'why', 'she', 'him’}
for item in setA:
    print( item )
```

```
you
why
to
of
down
him
if
in
by
from
she
the
```

Review: Sets



Creating a `Set`

An empty set:

```
    a = set()
```

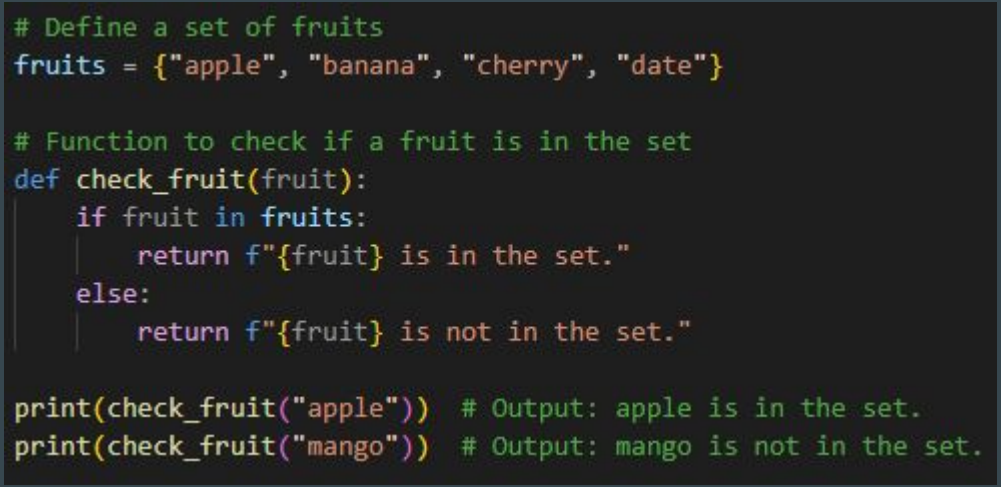
A set with elements in it:

```
    b = {"the", "hello", "happy"}
```

`a.`**`add`**`(`**val**`)`	adds element val to a
`a.`**`discard`**`(`**val**`)`	removes val from a if present
`a.`**`pop`**`()`	removes and returns a random element from a
`a - b`	returns a new set containing values in a but not in b
`a | b`	returns a new set containing values in either a or b
`a & b`	returns a new set containing values in both a and b
`a ^ b`	returns a new set containing values in a or b but not both

You can also use `in`, `len()`, etc.

Review: Sets

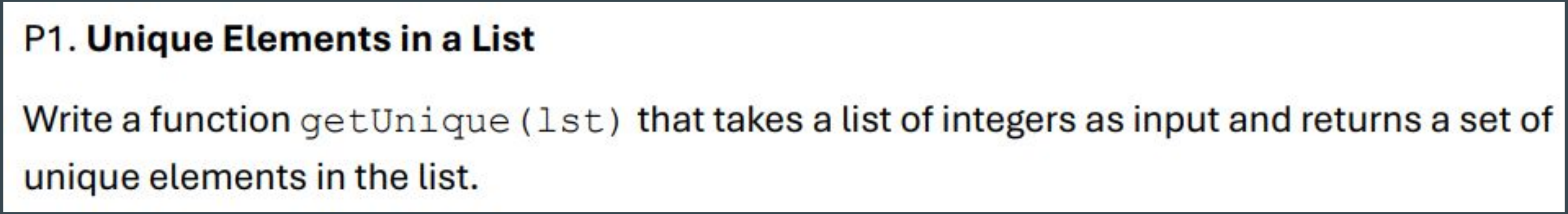


```
# Define a set of fruits
fruits = {"apple", "banana", "cherry", "date"}

# Function to check if a fruit is in the set
def check_fruit(fruit):
    if fruit in fruits:
        return f"{fruit} is in the set."
    else:
        return f"{fruit} is not in the set."

print(check_fruit("apple"))  # Output: apple is in the set.
print(check_fruit("mango"))  # Output: mango is not in the set.
```

Just like with dictionaries and sets, you can also check for items within sets!

If you simply need to keep track of already added items, using a set can be much faster than lists as checking for an item in a set is constant time compared to a list having to iterate through all elements of the list to search for the matching item

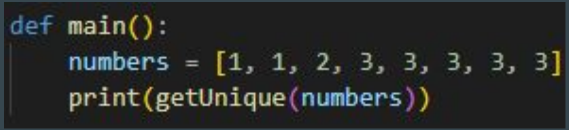
Problem 1



P1. **Unique Elements in a List**

Write a function `getUnique(lst)` that takes a list of integers as input and returns a set of unique elements in the list.

Example Input:



```
def main():
    numbers = [1, 1, 2, 3, 3, 3, 3, 3]
    print(getUnique(numbers))
```

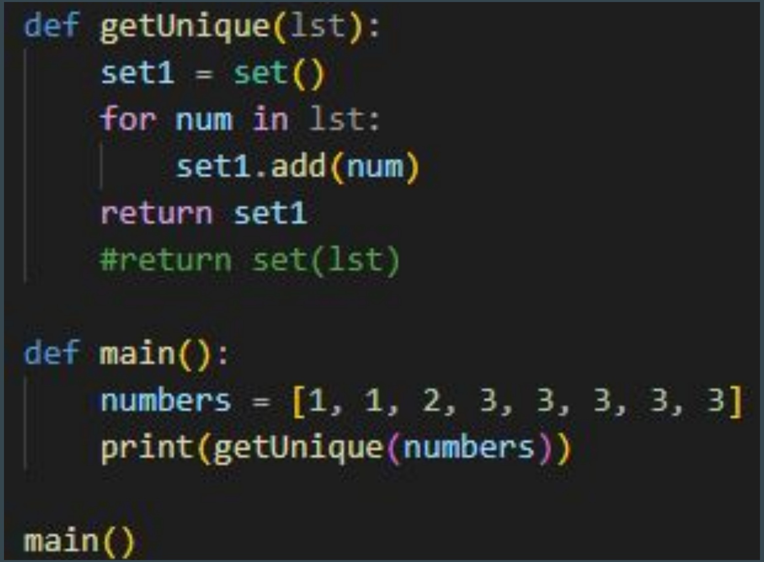
Example Output:

{1, 2, 3}

codeshare.io/NKNKrr

P1. **Unique Elements in a List**

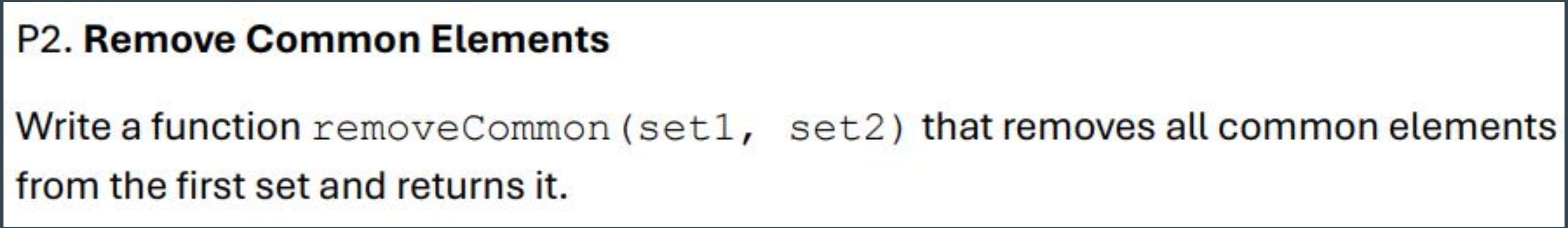
Write a function `getUnique(lst)` that takes a list of integers as input and returns a set of unique elements in the list.



```python
def getUnique(lst):
    set1 = set()
    for num in lst:
        set1.add(num)
    return set1
    #return set(lst)

def main():
    numbers = [1, 1, 2, 3, 3, 3, 3, 3]
    print(getUnique(numbers))

main()
```

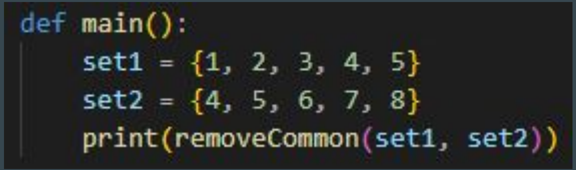
`a.`**add**(**val**)	adds element val to a

Problem 2


P2. **Remove Common Elements**

Write a function `removeCommon(set1, set2)` that removes all common elements from the first set and returns it.

Example Input:


```
def main():
    set1 = {1, 2, 3, 4, 5}
    set2 = {4, 5, 6, 7, 8}
    print(removeCommon(set1, set2))
```

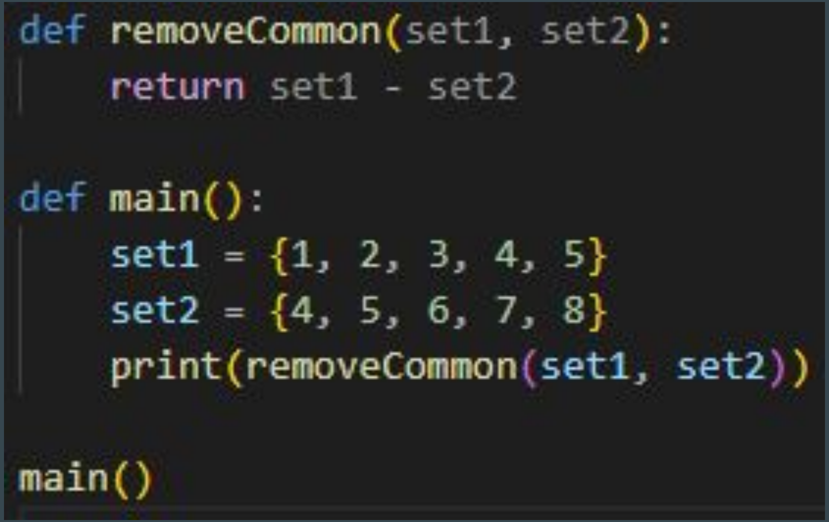
Example Output:

{1, 2, 3}

codeshare.io/NKNKrr

P2. **Remove Common Elements**

Write a function `removeCommon(set1, set2)` that removes all common elements from the first set and returns it.



```python
def removeCommon(set1, set2):
    return set1 - set2

def main():
    set1 = {1, 2, 3, 4, 5}
    set2 = {4, 5, 6, 7, 8}
    print(removeCommon(set1, set2))

main()
```

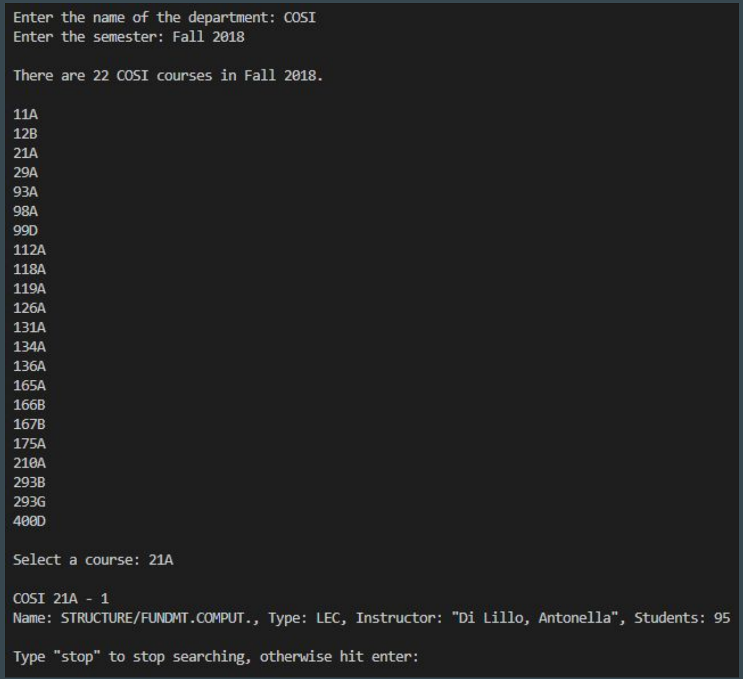
`a - b`	returns a new set containing values in a but not in b

Problem 3

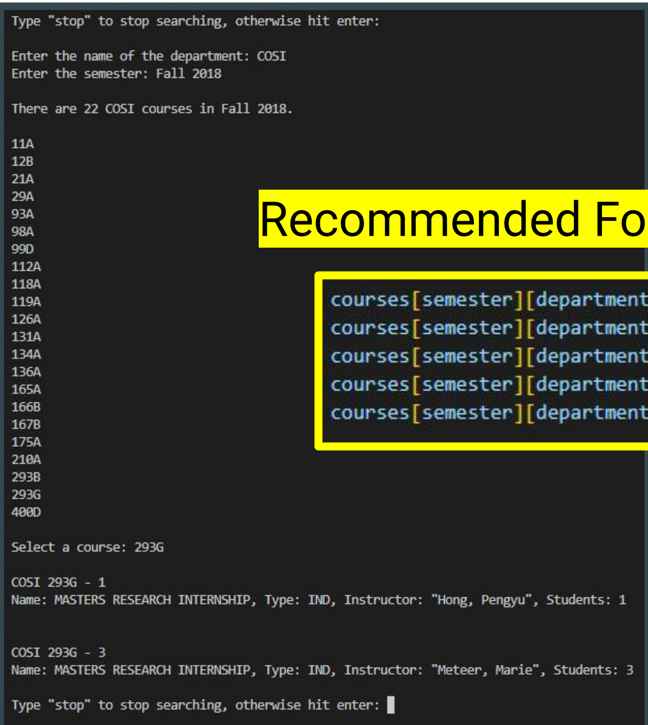
For this problem we will use `R9_courses_2014-19.tsv` which is in tab-separated-values format. Each line has the following information department, coursenum, section, title, format, instructor, semester, code, number of enrolled students.

Write a Python program which reads this into a list of dictionaries.

It should then have a loop where it asks the user for department and semester and then provide the available courses in the given semester.

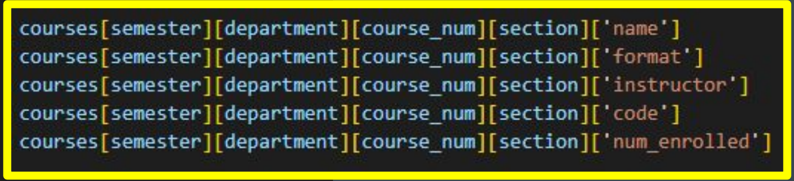


```
Enter the name of the department: COSI
Enter the semester: Fall 2018

There are 22 COSI courses in Fall 2018.

11A
12B
21A
29A
93A
98A
99D
112A
118A
119A
126A
131A
134A
136A
165A
166B
167B
175A
210A
293B
293G
400D

Select a course: 21A

COSI 21A - 1
Name: STRUCTURE/FUNDMT.COMPUT., Type: LEC, Instructor: "Di Lillo, Antonella", Students: 95

Type "stop" to stop searching, otherwise hit enter:
```



```
Type "stop" to stop searching, otherwise hit enter:

Enter the name of the department: COSI
Enter the semester: Fall 2018

There are 22 COSI courses in Fall 2018.

11A
12B
21A
29A
93A
98A
99D
112A
118A
119A
126A
131A
134A
136A
165A
166B
167B
175A
210A
293B
293G
400D

Select a course: 293G

COSI 293G - 1
Name: MASTERS RESEARCH INTERNSHIP, Type: IND, Instructor: "Hong, Pengyu", Students: 1


COSI 293G - 3
Name: MASTERS RESEARCH INTERNSHIP, Type: IND, Instructor: "Meteer, Marie", Students: 3

Type "stop" to stop searching, otherwise hit enter: 
```

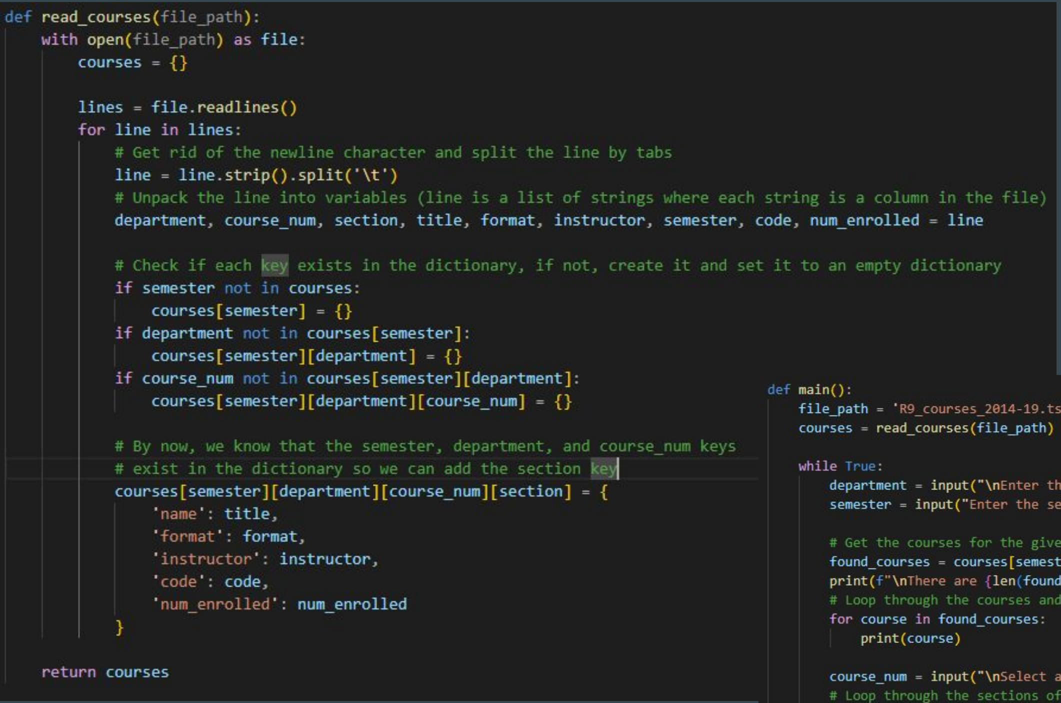
Recommended Format of your dictionaries



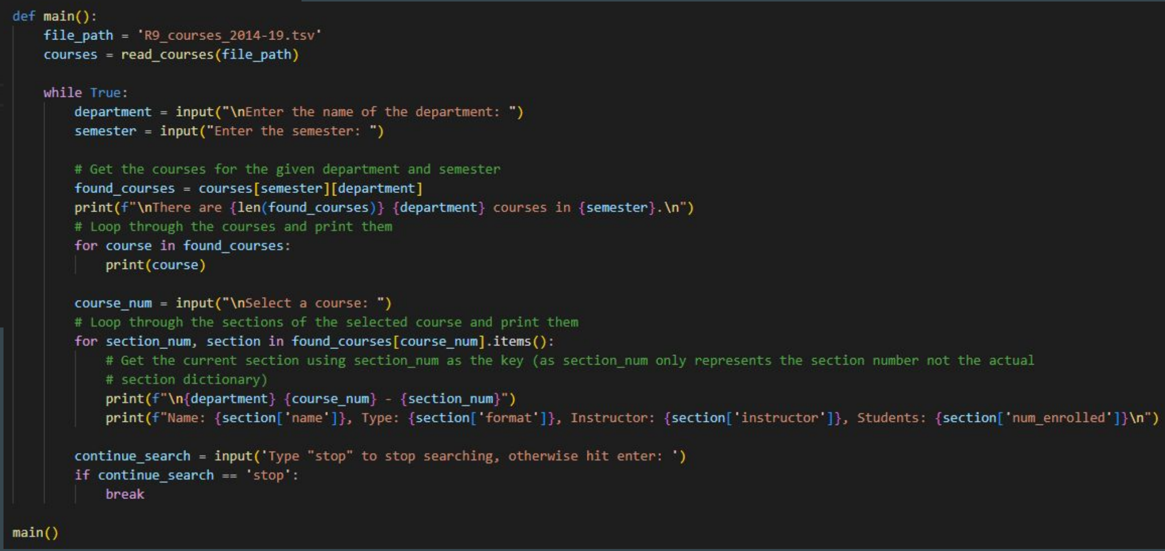
```
courses[semester][department][course_num][section]['name']
courses[semester][department][course_num][section]['format']
courses[semester][department][course_num][section]['instructor']
courses[semester][department][course_num][section]['code']
courses[semester][department][course_num][section]['num_enrolled']
```

codeshare.io/NKNKrr

Problem 3



```python
def read_courses(file_path):
    with open(file_path) as file:
        courses = {}

        lines = file.readlines()
        for line in lines:
            # Get rid of the newline character and split the line by tabs
            line = line.strip().split('\t')
            # Unpack the line into variables (line is a list of strings where each string is a column in the file)
            department, course_num, section, title, format, instructor, semester, code, num_enrolled = line

            # Check if each key exists in the dictionary, if not, create it and set it to an empty dictionary
            if semester not in courses:
                courses[semester] = {}
            if department not in courses[semester]:
                courses[semester][department] = {}
            if course_num not in courses[semester][department]:
                courses[semester][department][course_num] = {}

            # By now, we know that the semester, department, and course_num keys
            # exist in the dictionary so we can add the section key
            courses[semester][department][course_num][section] = {
                'name': title,
                'format': format,
                'instructor': instructor,
                'code': code,
                'num_enrolled': num_enrolled
            }

    return courses
```



```python
def main():
    file_path = 'R9_courses_2014-19.tsv'
    courses = read_courses(file_path)

    while True:
        department = input("\nEnter the name of the department: ")
        semester = input("Enter the semester: ")

        # Get the courses for the given department and semester
        found_courses = courses[semester][department]
        print(f"\nThere are {len(found_courses)} {department} courses in {semester}.\n")
        # Loop through the courses and print them
        for course in found_courses:
            print(course)

        course_num = input("\nSelect a course: ")
        # Loop through the sections of the selected course and print them
        for section_num, section in found_courses[course_num].items():
            # Get the current section using section_num as the key (as section_num only represents the section number not the actual
            # section dictionary)
            print(f"\n{department} {course_num} - {section_num}")
            print(f"Name: {section['name']}, Type: {section['format']}, Instructor: {section['instructor']}, Students: {section['num_enrolled']}\n")

        continue_search = input('Type "stop" to stop searching, otherwise hit enter: ')
        if continue_search == 'stop':
            break

main()
```

Questions?