CENG 113 – Programming Basics

Lab 12 - Sets, Tuples, Dictionaries, and File Operations

Sets

- A set is an unordered collection of unique elements.

```
some_numbers = set([2, 1, 3, 3])
same_numbers = set([3, 2, 1])
other_numbers = set([4, 3, 1])

print(some_numbers)
        # {1, 2, 3}
print(some_numbers == same_numbers)                    # True
print(len(some_numbers))
        # 3
print(other_numbers.issubset(some_numbers))            # False
print(other_numbers.union(some_numbers))                       # {1, 2,
3, 4}
print(other_numbers.difference(some_numbers))    # {4}
print(5 in some_numbers)
        # False
```

Tuples

- A tuple is a comma-separated, ordered sequence of immutable elements.

```
tup1 = ('physics', 'chemistry', 3, 8, 15.2)
tup2 = (1, 2, 3, 4)
tup3 = 'a', 'b', 'c'
tup4 = ()
tup5 = (50,)

a_list = ["15", "25"]
tup6 = tuple(a_list)                    # Convert a list into a
tuple

(x, y) = (4, 'Maths')                   # x, y = (4, 'Maths')

print(tup2[0])                                  # 1
print(tup2[-2])                                 # 3
print(tup2[1:3])                                # (2,3)

print(len(tup5))                                # 1
print(tup2 + tup3)                      # (1, 2, 3, 4, 'a',
'b', 'c')
print(3 in (1, 2, 3))                   # True
for x in (1, 2):                                # Iteration over
a tuple
```

Dictionaries

- A dictionary is a set of key-value pairs
- Keys must be ***unique*** within a dictionary (No ***duplicate*** keys)
- Keys must be ***immutable***. Which means you can use *strings*, *numbers* or *tuples* as dictionary keys.
- **e.g.** age = {'Alice' : 25, 'Bob' :28}

age['Alice'] is 25 age['Bob'] is 28

- Lists index their entries based on the position in the list
- Dictionaries has no order between items
- So we index the things we put in the dictionary with a "lookup tag"

```
>>> purse = {}
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

Dictionaries (cont.)

- **Displaying Contents:**

```
>>> age = {'Alice' : 25, 'Carol': 'twenty-two'}
>>>list(age.items())
[ ('Alice', 25), ('Carol', 'twenty-two') ]
>>>list(age.keys())
[ 'Alice', 'Carol' ]
>>>list(age.values())
[25, 'twenty-two']
```

- **Updating Dictionaries:**

```
>>> age = {'Alice': 26 , 'Carol' : 22}
>>> age.update({'Bob' : 29})
>>> age
{'Bob': 29, 'Carol': 22, 'Alice': 26}
>>> age.update({'Carol' : 23})
>>> age
{'Bob': 29, 'Carol': 23, 'Alice': 26}
```

- **Removing an item:**

```
>>> age.pop('Carol')
'twenty-two'
>>> age
{'Alice': 25}
```

- **Test for a Key with the `in`**

```
if "Alice" in age:
    print("Age of Alice is",age["Alice"])
else:
    print("There is no Alice in dictionary")
```

Reading Files

- Built-in functions to read files
  - `f.read()` -> file's entire contents as a string
  - `f.readline()` -> next line from file as a string
  - `f.readlines()` -> file's contents as a list of lines

Line by line processing

- Reading a file line-by-line using **a for loop**:

```
for line in f.readlines():
        statements
```

- **Example:**

```
f = open("bankaccount.txt")
count = 0
for line in f.readlines():
        count = count + 1
print("The file contains", count, "lines.")
```

Writing Files

- open file for write (deletes previous contents), or open file for append (new data goes after previous data)
  - `f = open("filename", "w")`
  - `f = open("filename", "a")`

- Writing the given string to the file:
  - `f.write(str)` -> Returns the number of characters written.

- Closing the file
  - `f.close()` - saves file once writing is done

```
>>> out = open("output.txt", "w")
>>> out.write("Hello, world!\n")
>>> out.write("How are you?")
>>> out.close()
>>> open("output.txt").read()
```

Strings

- split breaks a string into tokens that you can loop over.
  - **str**.split() # break by whitespace
  - **str**.split(**delimiter**) # break by delimiter

- join performs the opposite of a split
  - **delimiter**.join(**list of tokens**)

- strip removes both begin and ending whitespace.

```
>>> name = "Brave Sir Robin"
>>> for word in name.split():
…             print(word)
Brave
Sir
Robin
>>>"32,24,25,57".split(",")
['32', '24', '25', '57']
```

```
>>>"  Hello World  ".strip()
Hello World

>>>"LL".join(name.split("r"))
'BLLave SiLL Robin
```

Exercises

Books

```
books = ["ULYSSES","ANIMAL FARM","BRAVE NEW WORLD","ENDER'S GAME"]
```

1. Implement a function called as **`construct_dict(…)`**
   - This function takes one input: list of books.
   - It constructs a dictionary. Its keys should be book names and values of each element should be a **tuple** that contains two elements: **the number of characters** in the corresponding book name, and **the number of unique characters** in that name.
   - This function should return the constructed dictionary.
   - After implementation is finished, call the function by giving predefined list named **“`books`”** as the input and assign its output to a variable called “`book_dict`”.
2. Implement a function called as **`print_dict(…)`**
   - This function takes a dictionary as input, and prints it as follows.

     ```
     key1 -> value1
     key2 -> value2
     …
     ```

   - Call this function for book_dict.

Books (cont.)

```
books = ["ULYSSES","ANIMAL FARM","BRAVE NEW WORLD","ENDER'S GAME"]
```

3. Implement a function called as **`update_dict(…)`**
   - This function takes one input: dictionary.
   - In this function, the dictionary that is given to the function as input should be updated. Its keys should remain the same, but the values of each element should be updated.
   - For each value, you should calculate the average of the two numbers that are already there. After calculation, the **average** should be appended to the value of each key. So the values of each key should be a tuple that contains three elements: the number of characters in the corresponding book name, and the number of unique characters in that name, and the average of the previous two values.
   - This function will not return any value.
   - Call the function with "`book_dict`" as input.
   - Call the function `print_dict` for "`book_dict`".

Sets

```
numbers1 = [2,3,4,20,5,5,15]
numbers2 = [10,20,20,15,30,40]
```

1. Implement a function called `unique_list(...)`
   - This function takes a list and returns another list after removing duplicate elements.
   - Call the function with the given list “numbers1” and assign it back to “numbers1”.
   - Call the function with the given list “numbers2” and assign it back to “numbers2”.
2. Implement a function called as `intersection(…,...)`
   - This function takes two sets.
   - It finds the intersection of two sets and returns it.
   - Call the function with sets that you convert at the first exercise.
3. Implement a function called as `union(…,...)`
   - This function takes two sets.
   - It finds the union of the two sets and returns it.
   - Call the function with sets that you convert at the first exercise.

File Operations

IMDB top 100 movies

You can download the CSV file from this link:

https://drive.google.com/file/d/160pauqS0yEICr4SJ2ZGgMs3-xT4B73SA/view?usp=sharing

Write a program for steps below:

- Implement a function called **search_director()** to search director in your data (you will need this to create data).

- Create a list of dictionaries data for movie director information (check the example in the next slide). Read each line from file and add each movie name, movie year, movie IMDB score, leading actor's name to the director's dictionary. If director exist in your data, append movie information to the movies key, if not create new director information.

File Operations cont.

- DATA TYPE Example:

directors = [[{'name': 'Frank Darabont', 'movies': [{'movie': 'The Shawshank Redemption', 'year': '1994', 'score': '9.3', 'star': 'Tim Robbins'}]}], [{'name': 'David Fincher', 'movies': [{'movie': 'Fight Club', 'year': '1999', 'score': '8.8', 'star': 'Brad Pitt'}, {'movie': 'Se7en', 'year': '1995', 'score': '8.6', 'star': 'Morgan Freeman'}]}]]

- Define a function called count_movie() to return director name and his/her directed movie count.

- Define a function called avr_score() to return director name and his/her movies' average IMDB score.

- Define a function called count_leading_roles() to create and return a dictionary of {'actor name' : star, 'movie count' : movie_count, 'average IMDB score': avr_score}