More File I/O + Dictionaries

CSE 160 Summer 2026

Questions?

Sli.do #cse160

Announcements

- **Written Check-in 4** due Friday, July 17th at 11:59pm
- **Programming Practice 3** due Sunday, July 19th at 11:59pm
- **Homework Assignment 3** due Monday, July 20th at 11:59pm
- **Resubmission 1** due Wednesday, July 22th at 11:59pm
    - If you do not submit a form, we cannot accept your resubmission
- **Practice Midterms** now available on the course website
    - Midterm on Monday, July 20th during class (9:40AM-10:40AM)

Previously on *CSE 160*…

path to file

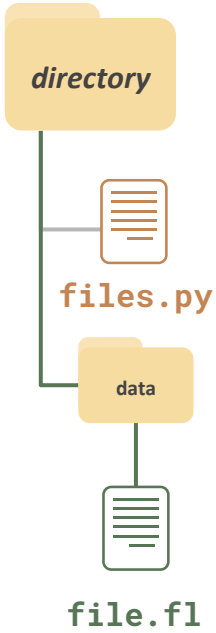
mode argument specifies how the file will be used. default = "r" (read)

```
f = open("data/file.fl", "w")

f.read()
f.write("I'm overwriting the file")

f.close()
```

reads entire file at once

Can only write strings!

Remember to close when you're done!



Today's Roadmap

1. Data Processing
2. Removing Whitespace
3. Dictionaries
4. Dictionary Methods

Jupyter Hub

with Statements

`with` statements simplify file I/O, **automatically closing the file** when finished.

```
with open("data/file.fl", "w") as file:

    file.read()
    file.write("I'm overwriting the file")
```

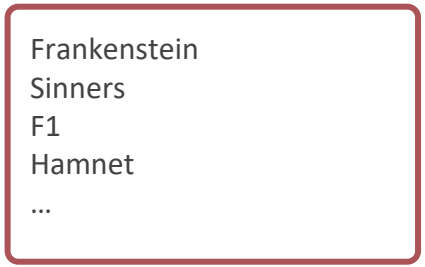
No need to call `file.close()`

Data Processing

Sometimes, just reading the file contents *isn't* enough.

To extract **useful data**, we might need to ***process*** the file contents by:

- Removing **whitespace**
- **Separating** data values



nominees.txt

```
lst = [
      'Frankenstein\n',
      'Sinners\n',
      'F1\n',
      'Hamnet\n',
      'Bugonia\n',
      'Sentimental Value\n',
      'Train Dreams\n',
      'The Secret Agent\n',
      'Marty Supreme\n',
      'One Battle After Another'
]
```

Removing Whitespace

Recall that **newline characters** (**\n**) are **included** when reading from a file.

```
file_text = "This is the first line\nThis is the second"
```

But we often want to remove **whitespace** from our data…

`string.strip(char)`

This **string method** removes any ***leading* or *trailing* characters**. If **char** is not specified, default is **whitespace**.

```
" hello     ".strip() → "hello"
```

```
"22he22llo222".strip('2') → "he22llo"
```

Not removed because it is surrounded on both sides

Separating Data Values

What if we have multiple **data values** on the same line?

```
1, 2, 5, 0
```

This **file structure** is ***comma separated values*** (`.csv`)

How can we leverage the **structure** of the file when processing the **data**?

`string.split(delim)`

This **string method** returns a **list** of strings separated by the given ***delimiter*** (`delim`). If `delim` is not specified, default is **whitespace**.

```
"1, 2, 5, 0".split(',')
→ ["1", " 2", " 5", " 0"]
```

Notice the spaces!

```
"hi CSE 160 \n".split()
→ ["hi", "CSE", "160"]
```

Data Processing Example

Given a partial list of items in **`menu.txt`** and new items and prices in the **`new_items`** and **`prices`** lists, respectively:

```
new_items = ["Chips", "Guac"]
prices = [2.0, 4.75]
```

1. ***Add*** the new menu items to the **`menu.txt`** file with the proper formatting

2. Find the ***total price*** of all menu items combined

Taco - $3.25
Burrito - $7.50

`menu.txt`

Think Pair Share

Write a program that reads in the contents of the file **`temps.dat`** and **creates a nested list**, where each element is a list of temperatures for a given week.

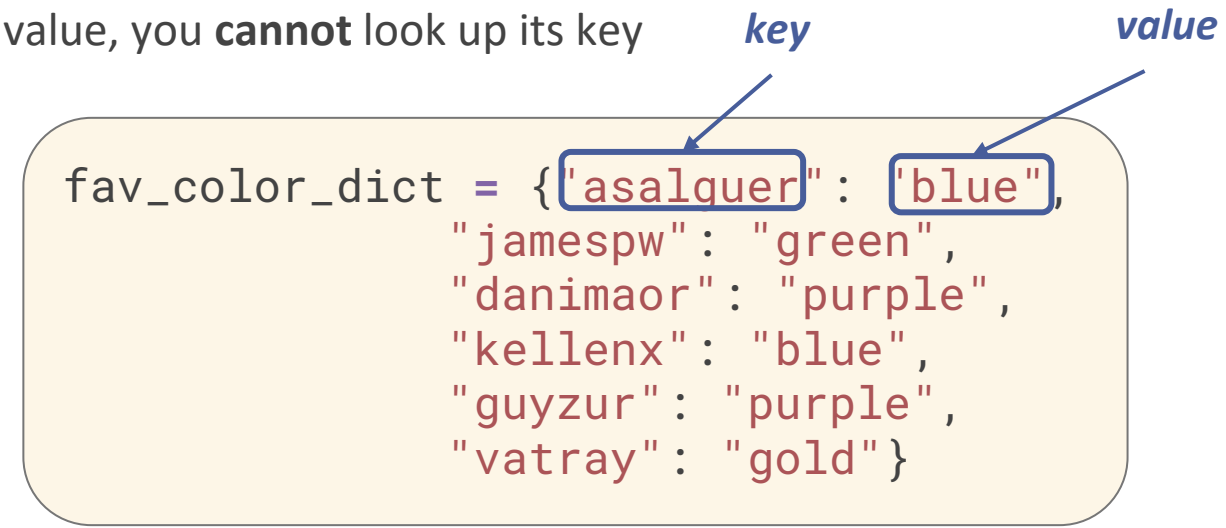
```
41, 45, 50,
46, 43, 43, 39, 41, 48, 48,
46, 52, 54, 48, 46, 46, 46,
45, 45, 45, 43, 39, 41, 39,
41, 45, 46, 48,
```

`temps.dat`

```
[[41, 45, 50],
 [46, 43, 43, 39, 41, 48, 48],
 [46, 52, 54, 48, 46, 46, 46],
 [45, 45, 45, 43, 39, 41, 39],
 [41, 45, 46, 48]]
```

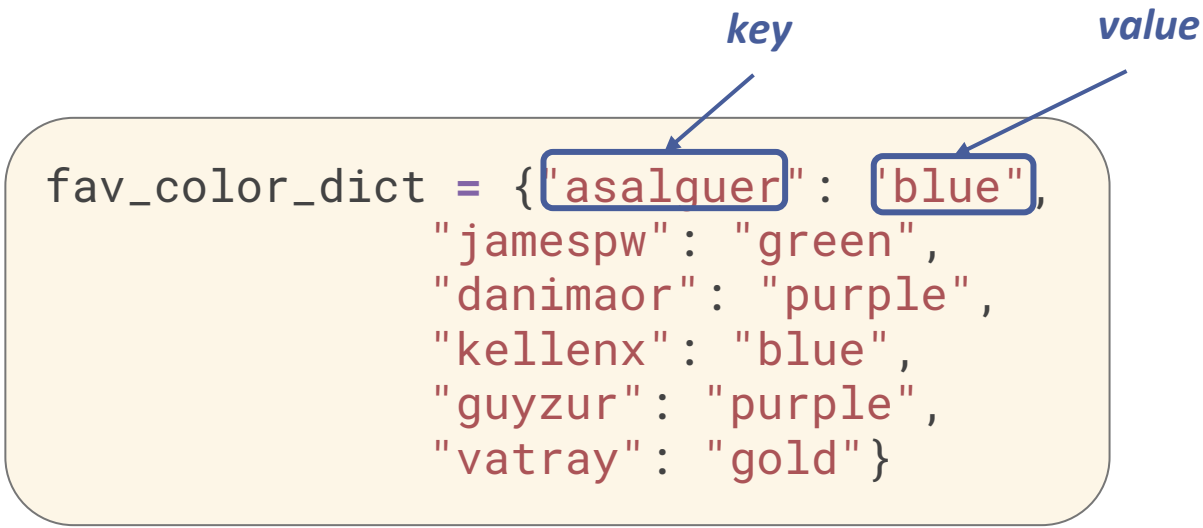
What is a Dictionary?

- A dictionary is a data structure that maps **keys** to **values**
- Unlike lists, order does not matter in dictionaries.
  - You could think of a dictionary as a big bag of key -> value pairs
- Given a key, you can look up its associated value
  - Given a value, you **cannot** look up its key



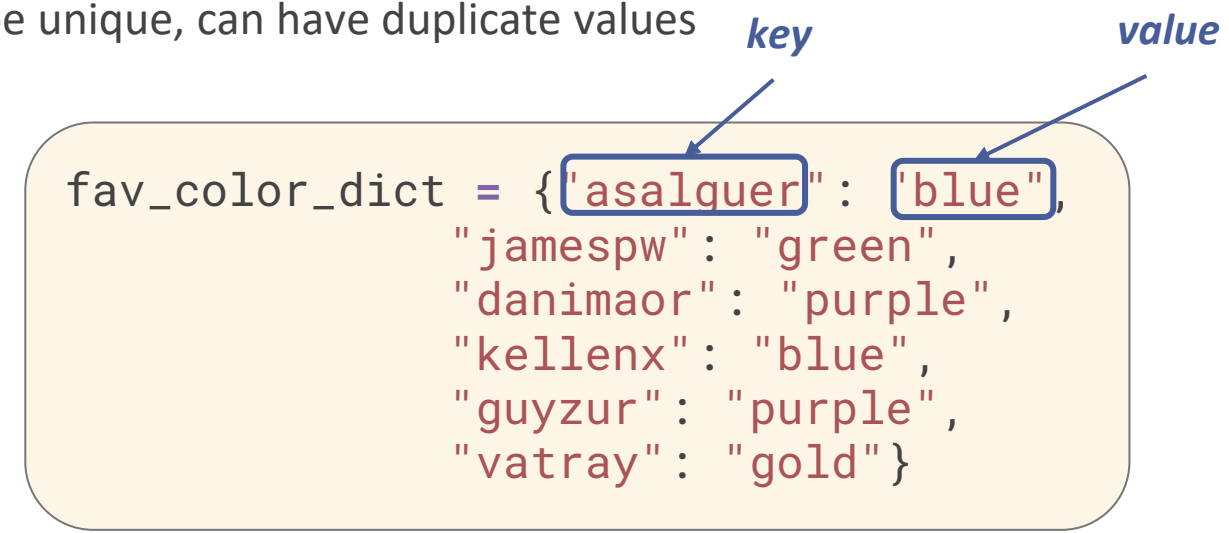
Dictionary Syntax

- **Curly braces** begin and end the dictionary
- **Colon** separates each key from its associated value
- **Comma** between each key: value pair



A Dictionary maps Keys to Values

- **Keys**:
  - Must be unique! No duplicate keys!
  - Must be an *immutable* type (e.g. integer, float, string)
- **Values**:
  - Do not need to be unique, can have duplicate values
  - Can be any type



Creating Dictionaries

Python Tutor

```
fav_color = {"asalguer": "blue", "jamespw": "green",
             "danimaor": "purple", "kellenx": "blue",
             "guyzur": "purple", "vatray": "gold"}

squares = {2: 4, 3: 9, 5: 25, -5: 25, -2: 4, -3: 9}

atomic_number = {"H": 1, "Fe": 26, "Au": 79}

food_price = {"Taco": 3.25, "Burrito": 7.5,
                            "Chips": 2, "Guac": 4.75}
```

Asking for the Value that goes with a Key

- We can look up the **value** that goes with a key, with an expression like this:

  `dict_name[key]`

```
fav_color = {"asalguer": "blue", "jamespw": "green",
             "danimaor": "purple", "kellenx": "blue",
             "guyzur": "purple", "vatray": "gold"}
adrians_fav_color = fav_color["asalguer"]

squares = {2: 4, 3: 9, 5: 25, -5: 25, -2: 4, -3: 9}
x = squares[5] + squares[-5]

atomic_number = {"H": 1, "Fe": 26, "Au": 79}
print(atomic_number["Au"])
```

Python Tutor

What if a Key is not present?

- It is a `KeyError` if a key is not present in the dictionary

```
fav_color = {"asalguer": "blue", "jamespw": "green",
             "danimaor": "purple", "kellenx": "blue",
             "guyzur": "purple", "vatray": "gold"}
adrians_fav_color = fav_color["asalguer"]

print(fav_color["vatray"])  # "gold"
print(fav_color["dubs"])  # KeyError: 'dubs'
```

Python Tutor

Another way to Create a Dictionary

- We can create an empty dictionary using empty curly braces: { }
- We can **add** a mapping to a dictionary with an assignment statement like this:

```
dict_name[key] = value
```

```
fav_color = {}  # Creates an empty dictionary
fav_color["asalguer"] = "blue"
fav_color["jamespw"] = "green"
fav_color["danimaor"] = "purple"
fav_color["kellenx"] = "blue"
fav_color["guyzur"] = "purple"
fav_color["vatray"] = "gold"
```

Python Tutor

Changing the Value that goes with a Key

- We can change the **value** that goes with a key, like this:

```
dict_name[existing_key] = new_value
```

```
fav_color = {"asalguer": "blue", "jamespw": "green",
             "danimaor": "purple", "kellenx": "blue",
             "guyzur": "purple", "vatray": "gold"}

# Adrian decided he likes purple better
fav_color["asalguer"] = "purple"
```

Python Tutor

Removing a Key

- We can remove a **key** (and its value) with a **del** statement:

```
del dict_name[existing_key]
```

```
fav_color = {"asalguer": "blue", "jamespw": "green",
             "danimaor": "purple", "kellenx": "blue",
             "guyzur": "purple", "vatray": "gold"}

# Kellen would like to be removed from this dictionary
del fav_color["kellenx"]
# You can only delete existing keys, not values
del fav_color["purple"]  # KeyError: 'purple'
```

Python Tutor

Think Pair Share

Select **all** of the statements that will result in an error (or indicate that none do):

A. `print(food_price[2])`

B. `food_price["Guac"] = 3`

C. `del food_price["taco"]`

A. `food_price[2] = "Chips"`

B. None of these cause an error

```
food_price = {"Taco": 3.25,
              "Burrito": 7.5,
              "Chips": 2,
              "Guac": 4.75}
```

Sli.do #cse160

Is a Key in a dictionary?

`x in dict_name` is a boolean expression that evaluates to **True** if **`x`** is *one of the keys* in **`dict_name`**.

`x not in dict_name` is a boolean expression that evaluates to **True** if **`x`** is ***not*** *one of the keys* in **`dict_name`**.

```
fav_color = {"asalguer": "blue", "jamespw": "green",
             "danimaor": "purple", "kellenx": "blue",
             "guyzur": "purple", "vatray": "gold"}

print(fav_color["asalguer"])        # "blue"
print("maria" not in fav_color)     # True
print("jamespw" in fav_color)       # True
print("blue" in fav_color)          # False
```

Python Tutor

More Dictionary Methods

- `dict_name.keys()`
  - Retrieve all the keys in the dictionary
  - Returned in the form: `dict_keys([key1, key2,...])`
- `dict_name.values()`
  - Retrieve all the values in the dictionary
  - Return in the form: `dict_values([value1, value2,...])`
- `dict_name.items()`
  - Retrieve all the key-value pairs in the dictionary
  - Returned in the form: `dict_items([(key1, value1), (key2, value2), …])`

These are not lists, but a special dictionary object!
However, you can iterate through them using a for-loop just like a list!

Iterating through a dictionary

- Usually we want to look up what value is a associated with a single key:
  ```
  print("Adrian's fav color is ", fav_color["asalguer"])
  ```
- Sometimes it is useful to iterate over an entire dictionary
  - Note: It is not necessary to iterate over a dictionary to find a key, just look it up!
    ```
    print("The person's fav color is ", fav_color[person])
    ```

```
fav_color = {"asalguer": "blue", "jamespw": "green",
             "danimaor": "purple", "kellenx": "blue",
             "guyzur": "purple", "vatray": "gold"}

for uwnetid in fav_color.keys():
    print(uwnetid, " fav color is ", fav_color[uwnetid])
```

Python Tutor

Usually you should not iterate over a dictionary

- A very common mistake is to iterate over a dictionary with a loop, when all that is needed is to do a look up of a single value
- Iterate over a dictionary with a loop when:
  - You need to **modify** the values associated with every key in the dictionary
  - You need to **print** the values associated with every key in the dictionary
- Do NOT use a loop when you just need to look up the value associated with one key!

```
fav_color = {"asalguer": "blue", "jamespw": "green",
             "danimaor": "purple", "kellenx": "blue",
             "guyzur": "purple", "vatray": "gold"}

for uwnetid in fav_color.keys():
    print(uwnetid, " fav color is ", fav_color[uwnetid])
```

Counting words

Write code that uses a dictionary to count how many times each unique word appears in a phrase.

```
phrase = "how much wood could a woodchuck chuck if a
woodchuck could chuck wood"
```

Python Tutor

Jupyter Hub