CIS 1100

Dictionaries

Python
Fall 2024
University of Pennsylvania

Reminder about Autograder Output

- Gradescope has automatic style deductions
- Take a look sometime today to make sure that you have no automatic style errors!
- Other things that are **your responsibility to check for:**
  - submitting all files
  - runtime issues
- Not an autograder thing, but also remember:
  - When we say pennkey, we mean e.g. `sharry` or `tqmcgaha`
  - In the readme, you have to copy the collaboration statement exactly

If you see 0/40, that's not something you should ignore!

Exam Reminders

- Plan to take your exam at this time in this place on Monday, March 3
- Take a practice exam once you're done with *Restaurant Recommender* (HW4).
  - coming out early tomorrow
- All students who require SDS accommodation to take the exam should schedule their exam through the Weingarten Testing Center ASAP.
  - Any time on March 3rd is acceptable.
- The exam only covers material up until sets, dictionaries, and unit tests (this week.)

Set Operations

Name	Meaning	Method	Operator
Union	Create a new set with all elements from both	`s.union(t)`	`s | t`
Intersection	Create a new set with only elements that appear in both sets	`s.intersection(t)`	`s & t`
Difference	Create a new set with only elements in `s` that don't appear in `t`	`s.difference(t)`	`s - t`
Symmetric Difference	Create a new set with elements that appear in only one set *but not both*	`s.symmetric_difference(t)`	`s ^ t`

Set Operations Practice

Put both of these in (C14)

Implement both an intersection function and a union function without using the built-in intesection or union operators or functions.

```
def set_union(s1, s2):
    # given two sets, return a new set that has all elements of both input sets
    # set_union({"hi","ho"}, {"bad","hi"}) -> {"hi", "ho", "bad"}
```

```
def set_intersection(s1, s2):
    # given two sets, return a new set that only has the elements that are in both input sets
    # set_intersection({"hi","ho"}, {"bad","hi"}) -> {"hi"}
```

Dictionaries

Dictionaries (also called "dicts") are the much more commonly used **unordered** collection

- Associates **keys** to **values**
- Allow for looking up some information associated with a search key
- Keys must be unique, values do not need to be unique

What is a Mapping?

Any association from **keys** (things you can search by) to **values** (information you might want to know.)

The Penn Directory, for example:

```
Name : Email
Harry Smith : sharry@seas
Travis McGaha : tqmcgaha@seas
...
```

Here, the names are keys and the emails are values.

Dict Syntax

Dict literals are defined with curly braces (`{}`) and separate keys and values with a colon.

- `{3, 10, 15}`
  - is a **set** with three elements
- `{"Harry" : "sharry", "Travis" : "tqmcgaha"}`
  - is a **dict** with two elements (key-value pairs)
- `{}` is an empty dict
  - writing just `dict()` gets the same result

Dictionary Practice: Reading

Given the following dictionaries, which ones are legal dictionaries? (Legal / Illegal)

(S8)

```
speak = {
  "dog": "woof",
  "cat": "meow",
  "seal": "arf",
  "fox": "woof"
}
```

(S9)

```
last_year_movies = {"Anora", "Conclave", "A Real Pain"}
```

(S10)

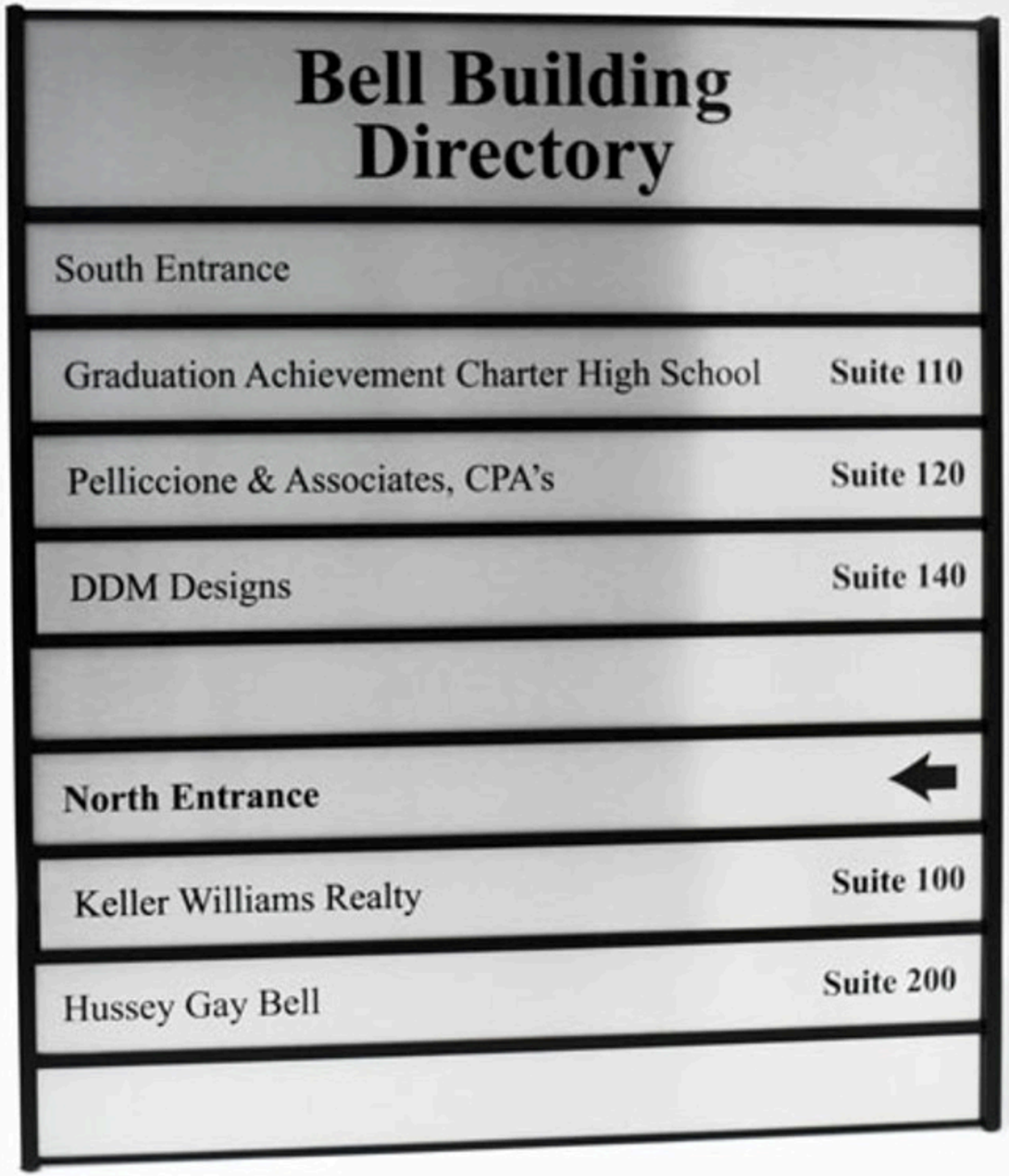
```
recs = {201:["Zwe", "Hannah"], 203: ["Adi", "Sofia"],
        204: ["Jana", "Hannah"], 201: ["Hannah", "Zwe"]}
```

Mapping

Dictionaries associate keys with values.

When you're trying to find your doctor in an office using a building directory, the names are _____ and the rooms are _____ (M1)

- A: keys, values
- B: keys, keys
- C: values, values
- D: values, keys



Describe the Issue:

We want to write a function that turns a list of names into a dictionary where the *keys* are the first initials and the *values* are lists of names that start with that letter.

```
["Susan", "Sally", "Paul", "Nico"]
⬇️
{"S" : ["Susan", "Sally"],
 "P" : ["Paul"],
 "N" : ["Nico"]
}
```

What's wrong with this implementation? (L11; describe, don't fix)

```python
def group_names_by_letter(names: list[str]) -> dict[str, list[str]]:
    names_by_letter = dict()
    for name in names:
        first_letter = name[0]
        names_by_letter[first_letter].append(name)
    return names_by_letter
```

Worked Example: Books!

I'm vain, so we're going to use my personal data: my collection of books read from Goodreads. (follow me?)

We'll use this data to build a recommender system

- "I heard about this author, can you recommend me her best book?"
- "What's the best book from last year?"

Understanding the Data

What's going on here? What do we have to work with?

```
Voices in the Evening
Natalia Ginzburg
1952 170 3.76
The Dry Heart
Natalia Ginzburg
1947 88 3.99
Childhood / Youth / Dependency (The Copenhagen Trilogy, #1-3)
Tove Ditlevsen
1967 371 4.36
In the Eye of the Wild
Nastassja Martin
2019 128 3.96
Kudos
Rachel Cusk
2018 236 3.91
Jack (Gilead, #4)
Marilynne Robinson
2020 309 3.86
```

Understanding the Data

For each data point (Book), we have:

- title
- author
- year, page count, rating

How many lines of the file do we need to read to process all the data for a single book? (S7)

```
Voices in the Evening
Natalia Ginzburg
1952 170 3.76
The Dry Heart
Natalia Ginzburg
1947 88 3.99
Childhood / Youth / Dependency (The Copenhagen Trilogy, #1-3)
Tove Ditlevsen
1967 371 4.36
In the Eye of the Wild
Nastassja Martin
2019 128 3.96
Kudos
Rachel Cusk
2018 236 3.91
Jack (Gilead, #4)
Marilynne Robinson
2020 309 3.86
```

Understanding the Data

How many lines of the file do we need to read to process all the data for a single book? (S7)

What are the types for each of these properties? (S8)

```
Voices in the Evening
Natalia Ginzburg
1952 170 3.76
The Dry Heart
Natalia Ginzburg
1947 88 3.99
Childhood / Youth / Dependency (The Copenhagen Trilogy, #1-3)
Tove Ditlevsen
1967 371 4.36
In the Eye of the Wild
Nastassja Martin
2019 128 3.96
Kudos
Rachel Cusk
2018 236 3.91
Jack (Gilead, #4)
Marilynne Robinson
2020 309 3.86
```

Understanding the Data

For each data point (Book), we have:

- title
  - `str`
- author
  - `str`
- year, page count, rating
  - `int`, `int`, `float`

Process Books

```
def process_book_file(filename : str) -> dict[str, tuple]:
    """Returns a dictionary d mapping
    book titles to tuples of book information.
    """
    ...
    return d
```

What is the type of `d`? (S9)

Process Books

```
def process_book_file(filename : str) -> dict[str, tuple]:
    """Returns a dictionary d mapping
    book titles to tuples of book information.
    """
    ...
    return d
```

What are the types of `d`'s keys? What are the types of its values? **Be as detailed as possible.** (S10)

Process Books

```
def process_book_file(filename : str) -> dict[str, tuple]:
    """Returns a dictionary d mapping
    book titles to tuples of book information.
    """
    ...
    return d
```

Implement `process_book_file` in **(C12)**

Helpful: `file.readlines()` returns a list of all lines in the file that can easily be processed.

Together: Print all books by author

```
def find_by_author(d: dict[str, tuple], author: str) -> dict[str, tuple]:
    """Return a dictionary containing
    just those books by the given author.
    """
    ...
```

You: Find Best Book by Author

```
def best_by_author(d: dict[str, tuple], author: str) -> str:
    """Return the name of the book that
    has the highest rating by that author.
    """
    ...
```

(C14)