Sets and Dictionaries

Welcome back to CS 2100!

Prof. Rasika Bhalerao

Sets

A set is very similar to a list: it is a collection of items.

```
from typing import Set

words: Set[str] = {'hi', 'hi', 'hello', 'hi', 'howdy', 'hi'}

print(words)  # {'hi', 'hello', 'howdy'}
```

Differences between a set and a list:

- A set is unordered
- A set can only hold each item (at most) once -- no duplicates

Exercise

Let's write a function that takes a `str` and counts the number of unique (distinct) words in it.

Solution

```
def count_unique_words(text: str) -> int:
    return len(set(text.split()))

print(count_unique_words('hello hi hi hello howdy hi'))  # 3
```

Some set syntax

Creating a set:

```
words: Set[str] = {'hi', 'hi', 'hello', 'hi', 'howdy', 'hi'}

numbers: Set[int] = set(range(5))
print(numbers)  # {0, 1, 2, 3, 4}

list_of_floats: List[float] = [3.4, 3.2, 2.9, 3.4, 3.0]
measurements: Set[float] = set(list_of_floats)
print(measurements)  # {3.2, 3.0, 2.9, 3.4}
```

Some set syntax

Adding and removing items, iterating over a set, and getting its size:

```
nums: Set[float] = set()

for i in range(100):
    random_float = round(random(), 2) # random float rounded to nearest hundredth
    nums.add(random_float)

print(len(nums))

numbers: Set[int] = set(range(5))
numbers.remove(3)
print(numbers)  # {0, 1, 2, 4}
```

Exercise

Let's write a function that checks if any two people in this room have the same birthday. It should have a loop that iterates (up to) 40 times.

Each iteration, it should:

- Ask the user to input their birthday via two separate `int`s: the month and the day (ask twice to get the two `int`s)
- Store their birthday as a tuple
- If that birthday is already in the set, return `True`
- If not, add it to the set

After the loop (which it should only reach if no two people have the same birthday), it should return `False`.

Solution

```
num_students: int = 80

def any_same_birthdays() -> bool:
    birthdays: Set[Tuple[int, int]] = set()

    for _ in range(num_students):
        month: int = int(input('Please enter the month as a number between 1 and 12: '))
        day: int = int(input('Please enter the day as a number between 1 and 31: '))
        date: Tuple[int, int] = (month, day)
        if date in birthdays:
            return True
        else:
            birthdays.add(date)

    return False
```

Some set syntax

Binary set operations:

- Union (`a | b`): a set that has all elements that are in either set `a` or set `b`
- Intersection (`a & b`): a set that has all elements that are in both set `a` and set `b`
- Subset (`a <= b`): `True` if all elements in `a` are also in `b`, and `False` otherwise
  - Strict subset (`a < b`): `True` if `a <= b` **and `a` is not equal to `b`**, and `False` otherwise
- Subtraction (`a - b`): a set that has all elements in `a` that are not in `b`

```
nums_a: Set[int] = set(range(1, 5))
nums_b: Set[int] = set(range(3, 9))

print(nums_a | nums_b)  # {1, 2, 3, 4, 5, 6, 7, 8}
print(nums_a & nums_b)  # {3, 4}
print(nums_a <= nums_b) # False
```

Poll: Why is there no binary "Addition" operation for sets? (There is Subtraction.)

1. Because it would be the same as the Intersection operation
2. Because it would be the same as the Union operation
3. Because it would be the same as the Subtraction operation
4. There is an Addition operation

Dictionaries

We use curly brackets (`{` and `}`) to represent sets. But we also use them to represent dictionaries:

```
print(type({'hello'}))  # <class 'set'>
print(type({}))         # <class 'dict'>
```

Curly brackets, when empty (or non-empty, but formatted a specific way), denote a dictionary.

A dictionary is also known as an "associative array".
It's like a list, but the indices are not required to be contiguous ints -- the indices can be of any type

A dictionary maps key --> value

Each key can appear at most once (the keys are a set)

Here are two examples which map each animal (`str`) to their age (`int`):

```
ages: Dict[str, int] = {'elephant': 12, 'cat': 10}
print(ages)  # {'elephant': 12, 'cat': 10}

also_ages: Dict[str, int] = dict([('elephant', 12), ('cat', 10)])
print(also_ages)  # {'elephant': 12, 'cat': 10}  (same as before)
```

Common difficulty in HW1: testing the keys instead of the values

The tests need to work on *any* implementation, not just the one with your specific keys

```
def test_ask_additional_questions(self) -> None:
        """Test ask_additional_questions with all 'y' responses."""

        with patch('builtins.input', side_effect=['y', 'y', 'y', 'y', 'y']):

            result = self.question_asker.ask_additional_questions()

            self.assertTrue(True in result.values())
```

Exercise

Let's write a function that takes a `str` and returns a dictionary that maps from each unique word in the `str` to the number of times it appears.

Solution

```
def word_counter(text: str) -> Dict[str, int]:
    word_counts: Dict[str, int] = dict()
    for word in text.split():
        word_counts[word] = word_counts.get(word, 0) + 1
    return word_counts

print(word_counter('hello hi hi hello howdy hi'))  # {'hello': 2, 'hi': 3, 'howdy': 1}
```

Some dictionary syntax

Access a value given a key:

- brackets (`[key]`)
- `get(key)` method
    - handles the case if the `key` is not in the `dict`

```
ages: Dict[str, int] = {'elephant': 12, 'cat': 10}

print(ages['cat'])  # 10
print(ages.get('cat'))  # 10
print(ages.get('dog'))  # None
print(ages.get('dog'), 3)  # 3
print(ages['dog'])  # raises KeyError
```

Some dictionary syntax

If we add the same `key` twice, it overwrites the original `value` with the second `value`.

```
ages: Dict[str, int] = {'cat': 10}

ages['elephant'] = 12
print(ages)  # {'cat': 10, 'elephant': 12}

ages.update([('elephant', 13)])
print(ages)  # {'cat': 10, 'elephant': 13}

ages['elephant'] = 14
print(ages)  # {'cat': 10, 'elephant': 14}

ages.update([('dog', 3)])
print(ages)  # {'cat': 10, 'elephant': 14, 'dog': 3}
```

Exercise

Let's write a function that helps us with Scrabble.

- A very common situation: We are playing Scrabble. We see we have 3 'O's. What can we do?
- The plan: get a map that gives us options based on a letter
- Let's write a function that takes a letter as a parameter and returns a dictionary where:
  - The keys are all possible frequencies of that letter (except zero)
  - The values are the sets of words in the dictionary with that many of that letter
- Here's a list of english words if you need one (the official Scrabble list is harder to get as a text file)

Solution

```
def scrabble_helper(letter: str) -> Dict[int, Set[str]]:
    result: Dict[int, Set[str]] = dict()
    with open('/path/to/dictionary.txt', 'r', encoding='utf-8') as english_dict:
        for word in english_dict.readlines():
            if letter in word:
                word = word.strip()
                letter_count = word.count(letter)
                if letter_count in result:
                    result[letter_count].add(word)
                else:
                    result[letter_count] = {word}
    return result

result: Dict[int, Set[str]] = scrabble_helper('r')

for key, value in result.items():
    if key > 2:
        print(f'{key}: {value}')
```

Some dictionary syntax

Iterate over a `dict`:

```
ages: Dict[str, int] = {'cat': 10, 'elephant': 14, 'dog': 3}
```

- over its `key`s

```
for key in ages:
    print(f"{key}'s age is {ages.get(key)}")
```

- over its `key-value` pairs

```
for key, value in ages.items():
    print(f"{key}'s age is {value}")
```

JSON (JavaScript Object Notation)

- Popular format for storing data
- Very common for APIs to send us data in JSON format
  - E.g., https://openweathermap.org/api/one-call-3
- Read as a dictionary

```
import json, pprint

with open('example_json_data.json', 'r', encoding='utf-8') as f:
    data = json.load(f)
    pprint.pp(data)
```

Notes to run the previous example on your own later

We took the example API response from the Weather API and stored it in a file called `example_json_data.json`.

- Removed the lines with ellipses (`...`)
- Removed the commas on the lines before them
- Added an ending bracket (`}`).

`pprint` (https://docs.python.org/3/library/pprint.html) is a library for printing data in a readable format.

Poll: Which data structure is best suited for this task?

Creating a product that works differently on different operating systems, and we want to know which operating systems we need to support

1. List
2. Tuple
3. Set
4. Dictionary

Poll: Which data structure is best suited for this task?

Storing the order in which young children should stand in line

1. List
2. Tuple
3. Set
4. Dictionary

Poll: Which data structure is best suited for this task?

Storing the 7 days of the week (Sunday, Monday, Tuesday, ..., Saturday)

1. List
2. Tuple
3. Set
4. Dictionary

Poll: Which data structure is best suited for this task?

Keeping track of each student's favorite color

1. List
2. Tuple
3. Set
4. Dictionary

Poll:

1. What is your main takeaway from today?

2. What would you like to revisit next time?