Review for Quiz 2

Welcome back to CS 2100!

Prof. Rasika Bhalerao

Recommeded Review Topics

- (Revisited) Classes
  - `__str__()`
  - Constructors, methods, attributes
- Properties
  - Attributes named using `_` / `__`
  - `@property` and `*.setter`
- Stakeholder-value matrices
  - Selecting stakeholders
  - Selecting values
- Sets and dictionaries
  - How to iterate through a dictionary
  - Rules about duplication
  - Set operators (`|` , `-` , etc.)
- Lists: sorting, mapping, filtering
  - Syntax for list comprehension
  - Syntax for sorting lists
  - Syntax for filtering a list
- Correlation
  - Definition of correlation
  - Pearson's correlation coefficient

Attribute visibility

It is impossible to block an attribute from being accessed externally.

- `self.size` (attribute with no underscores): anyone can access
- `self._contents` (single underscore): nicely ask others to avoid using it
  - Externally accessible (`dataframe._contents`)
- `self.__password` (two underscores): even stronger suggestion to keep away
  - External name is mangled (`my_diary._Diary__password`)

Poll: Which are true?

```python
class BankAccount:
    def __init__(self, account_id: int, pin: str):
        self.balance = 0
        self._account_id = account_id
        self.__pin = pin
```

1. All three attributes are truly private and cannot be accessed from outside the class
2. `balance` and `_account_id` are publicly accessible
3. `_account_id` and `__pin` are not accessible from outside the class
4. Accessing `_account_id` from outside `BankAccount` is avoided by convention
5. Externally, `__pin` is name-mangled to `_BankAccount__pin`

Properties: `@property` and `*.setter`

- Create a property by putting the `@property` decorator above a method with the name for the property
  - Returns the value of the property (likely using `_` or `__` attributes)
- Give the property a "setter" using another method with the same name, with the decorator `@age.setter`
  - Takes the property's new value as an arg
  - Updates any internal attributes

```python
class Person:
    def __init__(self, age: int):
        self._age = age

    @property
    def age(self) -> int:
        return self._age

    @age.setter
    def age(self, new_age: int) -> None:
        if new_age >= 0:
            self._age = new_age


mini: Person = Person(10)
mini.age = 11
print(mini.age) # 11
```

Poll: Which code snippet will work?

```
class Temperature:
    def __init__(self, celsius: float):
        self._celsius = celsius

    @property
    def fahrenheit(self) -> float:
        return self._celsius * 9/5 + 32

    @fahrenheit.setter
    def fahrenheit(
            self, value: float) -> None:
        self._celsius = (value - 32) * 5/9
```

1.

```
temp = Temperature(0)
print(temp.fahrenheit(0))
temp.fahrenheit(32)
```

2.

```
temp = Temperature(0)
print(temp.fahrenheit)
temp.fahrenheit = 32
```

3.

```
temp = Temperature(0)
print(temp.get_fahrenheit())
temp.set_fahrenheit(32)
```

4.

```
temp = Temperature(0)
print(temp._fahrenheit)
temp._fahrenheit = 32
```

Dictionaries

- Maps `key` --> `value`
- Can have the same `value` twice, but not the same `key` twice (`key` s are a set)
- If we map a `key` to a `value` , and then later on, map the same `key` to a different `value` , it overwrites the old `value` with the new one
- `dictionary['cat']` vs `dictionary.get('cat', 4)` :
  - get method will return `4` if `'cat'` isn't in the dictionary (or `None` if we didn't specify the `4`)
  - brackets will raise `KeyError` if `'cat'` isn't in the dictionary

Poll: What happens?

```
student_grades = {"Binnie": 85, "Ginnie": 92, "Mini": 78}

result1 = student_grades.get("Spleenie")
result2 = student_grades.get("Spleenie", 0)
result3 = student_grades["Spleenie"]
```

1. All three results will be None
2. `result1` is `None`, `result2` is 0, and the third result line raises a `KeyError`
3. The first result line raises a `KeyError`, `result2` is 0, and `result3` is `None`
4. All three result lines raise a `KeyError` because 'Spleenie' is not in the dictionary

Dictionary syntax: iteration

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

- iterate over its `key` s

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

- iterate over its `key-value` pairs

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

Poll: Which correctly prints each student's name and grade?

```
student_grades = {"Binnie": 85, "Ginnie": 92, "Mini": 78}
```

1.

```
for student in grades:
    print(student, grades[student])
```

2.

```
for student, grade in grades.items():
    print(student, grade)
```

3.

```
for pair in grades:
    print(pair[0], pair[1])
```

4.

```
for pair in grades:
    print(pair[0], pair[1])
```

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`

Poll: What is in `result1` and `result2`?

```
students_in_math = {'Mini', 'Binnie', 'Ginnie'}
students_in_cs = {'Mini', 'Mega', 'Micro'}

result1 = students_in_math & students_in_cs
result2 = students_in_math | students_in_cs
```

1.

```
result1 = {'Mini'}
result2 = {'Mini', 'Binnie',
    'Ginnie', 'Mega', 'Micro'}
```

2.

```
result1 = {'Mini', 'Binnie',
    'Ginnie', 'Mega', 'Micro'}
result2 = {'Mini'}
```

3.

```
result1 = {'Mini'}
result2 = {'Mini'}
```

4.

```
result1 = {'Mini', 'Binnie',
    'Ginnie', 'Mega', 'Micro'}
result2 = {'Mini', 'Binnie',
    'Ginnie', 'Mega', 'Micro'}
```

List comprehension

Move the body of a `for loop` to right before the `for` (after the opening bracket `[`)

```
my_nums: List[int] = [6, 7, 8, 9]

increased_nums: List[int] = [i + 1 for i in my_nums] # list comprehension

print(increased_nums)  # [7, 8, 9, 10]
```

If we want the resulting list to **filter elements**, we add the `if` clause after the `for` clause.

```
def positive_copy(nums: List[int]) -> List[int]:
    return [i for i in nums if i >= 0]
```

Poll: Which list comprehension correctly creates a list of only the even numbers from 1 to 10?

1. `[x if x % 2 == 0 for x in range(1, 11)]`
2. `[x for x in range(1, 11) where x % 2 == 0]`
3. `[x for x in range(1, 11) if x % 2 == 0]`
4. `[x % 2 == 0 for x in range(1, 11)]`

Sorting

```
words: List[str] = 'never gonna give you up'.split()

sorted_alphabetically: List[str] = sorted(words)
print(' '.join(sorted_alphabetically)) # give gonna never up you

sorted_by_length: List[str] = sorted(words, key = lambda word: len(word))
print(' '.join(sorted_by_length))  # up you give gonna never
```

The `sorted()` function has an optional argument `key`, which is a function that is applied to each element when determining the sorted order.

Poll: Given this list of tuples representing students and their scores:

```
[('Mini', 85), ('Binnie', 92), ('Ginnie', 78), ('Spleenie', 92)]
```

Which code will sort the students by score in descending order (highest to lowest)?

1. `sorted(students, key=lambda x: -x[1], reverse=True)`
2. `sorted(students, key=lambda x: x[1], reverse=True)`
3. `sorted(students, key=lambda x: -x[1])`
4. `sorted(students, reverse=True)`

Correlation

Correlation measures the strength of a linear relationship between two variables.

Correlation does not imply that an increase in one variable *causes* the other to increase (or decrease). We cannot know whether one of the variables is the cause.

Pearson's Correlation Coefficient

Value	Meaning	Relationship between variables
1	Strong positive correlation	As one increases, so does the other
-1	Strong negative correlation	As one increases, the other decreases
0	No correlation	Change in one variable says nothing about the other

Stakeholder-Value Matrices

1. **Stakeholders**: people who are affected by the software in any way
2. **Values**: values at stake for those stakeholders when considering the software
3. **Cells in the stakeholder-value matrix**: (columns are values, rows are stakeholders), cell contains how each stakeholder's value relates to the software
4. **Analyze conflicts**: e.g., one cell says to increase `x` and another says to decrease `x`

Let's go through Practice Quiz 2!

Poll:

1. What is your main takeaway from today?

2. What would you like to revisit next time?