Lecture 5:

- Quiz 2
- Python Basics continued
- Structured Data

Announcements:

- HW-1 due
- HW-2 out, due Tues Sept. 26
Python Classes

Python classes are similar to other languages, with some differences:

- How initialization works, including member variables ("attributes")
- No notion of non-public visibility (except through naming conventions)
- No notion of member function ("method") overloading
- Supports special methods, which are similar to operator overloading
- Special methods surrounded by 2 underscores (e.g., `__init__()`)
- Does support single and multiple inheritance (which we won’t need)

Classes take the general form:

```python
class ClassName:
    class_var = value  # class variable
    ...
    # other statements
    def __init__(self, p1, ...):  # constructor
        self.inst_var = value  # instance variable
        ...
        # more statements ...
    def another_method(self, p1, ...):
        ...
        # more statements ...
    ...
```

Creating instances (objects) of classes:

```
x = ClassName()  # if no required parameters
x = ClassName(v1, ...)  # if required parameters
```

- Object creation results in a call to `__init__()`
- Object attributes and methods called in normal way: `x.a` or `x.f()`
An example: Simple class for representing 2-dimensional points

```python
class Point:
    def __init__(self, x_coord=0, y_coord=0):
        self.x = x_coord
        self.y = y_coord

    def __repr__(self):
        return f'({self.x}, {self.y})'

    def scale(self, m):
        self.x *= m
        self.y *= m

    def translate(self, a, b):
        self.x += a
        self.y += b

Using our Point class:

# create a simple point
p1 = Point()  # x=0, y=0
print(p1)  # prints (0, 0) from __repr__
print(type(p1))  # prints "<class 'Point'>"
p1.translate(1, 2)  # x=1, y=2
p1.scale(3)  # x=3, y=6
print(p1)  # prints (3, 6)

# create two similar points
p2 = Point(10, 20)  # x=20, y=30
print(p2)  # prints (20, 30)
p3 = Point(10, 20)  # x=20, y=30
print(p3)  # prints (20, 30)
print(p2 == p3)  # False (id comparison, see: __eq__())

# note: x and y aren't private!
p2.x = 15  # changes p2's x coordinate!
print(p2)  # prints (15, 30)
```
“Private” member variables and functions:

- by convention, prefix private attributes and methods with “__”
- e.g.: `self.__x = x_coord`
- performs “name mangling” (e.g., can access via `p2._Point__x`)

Some useful **special functions**:

- `__eq__(self, other)` for `==` ... similarly `le, ge, lt, gt, ne`
- `__add__(self, other)` for `+` ... similarly for other operators
- `__getitem__(self, key)` for `[key]` ... similarly set item, delete item
- `__iter__(self)__(and __next__(self)__)` ... for iteration
- and so on

**Static (class) methods** in Python

- Just a method that doesn’t have `self` as (first) argument
- Good practice to annotate with `@staticmethod`

*Note: We’ll go over more Python features/constructs as we go ...*
**Getting Started: Datasets**

Our focus is **Tabular Data** ... aka **Relational** or **Structured**

- Data is organized into tables (rows and columns)

<table>
<thead>
<tr>
<th>Age</th>
<th>Impressions</th>
<th>Clicks</th>
<th>SignedIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>61</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- Each row is an **instance**
  ... aka example, record, or object

- Each column is an **attribute** (of the instance)
  ... aka fields or sometimes variables or features

- A **dataset** is a (sample) collection of instances
  ... from the **universe of objects** (universe of instances)

This is a sample of (simulated) daily website click stream data¹

- Each row contains attribute values (features) for one user
- User’s age, ads shown, ads clicked, and if logged in (0=no, 1=yes)

---

¹Adopted from “Doing Data Science”, Schutt and O’Neil
Datasets often have attributes identifying the **objects** vs the **features**

- where a feature represents a property of the object
- in previous example, no attributes identified the objects (users) from features

Modified example with attribute to identify the objects:

<table>
<thead>
<tr>
<th>UserId</th>
<th>Age</th>
<th>Impressions</th>
<th>Clicks</th>
<th>SignedIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>59</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>19</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>44</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>71</td>
<td>28</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>61</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- here, each *UserId* value identifies (or distinguishes) the user
- the rest of the attributes describe properties of the object

Multiple attributes could be used to identify the objects ... or one or none!

- e.g., movie-name and director (since movie names might not be unique)

**Q:** What are the object attributes and what are the feature attributes?

<table>
<thead>
<tr>
<th>CarName</th>
<th>ModelYear</th>
<th>MSRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ford pinto</td>
<td>75</td>
<td>2769</td>
</tr>
<tr>
<td>toyota corolla</td>
<td>75</td>
<td>2711</td>
</tr>
<tr>
<td>ford pinto</td>
<td>76</td>
<td>3025</td>
</tr>
<tr>
<td>toyota corolla</td>
<td>76</td>
<td>2789</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Note:** object and feature attributes can depend on how dataset being used!