COMP 204
Object Oriented Programming (OOP) - Examples

Yue Li
based on material from Mathieu Blanchette
<table>
<thead>
<tr>
<th>SCIENTISTS PRESENT WILL BE:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dr. Tony Mittermaier:</strong></td>
<td>Chemical Biology and Chemical Physics - Protein structural mobility</td>
</tr>
<tr>
<td><strong>Dr. Jackie Vogel:</strong></td>
<td>Cellular Biology and Biophysics - Spindle alignment in cell division of budding yeast</td>
</tr>
<tr>
<td><strong>Dr. Arnold Hayer:</strong></td>
<td>Cellular Mechanisms - Coordination and signalling among migrating cells and cytoskeletal dynamics during collective cell migration</td>
</tr>
<tr>
<td><strong>Dr. Mathieu Blanchette:</strong></td>
<td>Bioinformatics - Computational tools to study genomic evolution and gene expression regulation</td>
</tr>
<tr>
<td><strong>Dr. Justin Marleau:</strong></td>
<td>Postdoctorate student at Dr. Guichard’s lab, Biomathematics and Ecological Dynamics</td>
</tr>
</tbody>
</table>
COMP204 Midterm grade

Midterm Exam Class Statistics

Number of submitted grades: 65 / 68

- Minimum: 16.7%
- Maximum: 96%
- Average: 68.5%
- Mode: 93%
- Median: 72%
- Standard Deviation: 17.9%

Grade Distribution

Number of Users (%)

Grade Received (%)
Outline

Object-Oriented Programming Vocabulary (recap)

Bus simulation object-oriented program (recap)

Lecture Quiz 24

Medical diagnostic program (similar but not equivalent to A3)

An ecosystem simulation program (A4 preview)
Object-Oriented Programming Vocabulary (recap)


- **class**: A user-defined compound type. A class can also be thought of as a template for the objects that are instances of it.
- **attribute**: One of the named data items that makes up an instance.
- **method**: A function that is defined inside a class definition and is invoked on instances of that class.
- **initializer (or constructor) method**: A special method in Python (called `__init__`) that is invoked automatically to set a newly-created object’s attributes to their initial state.
Object-Oriented Programming Vocabulary (recap)


- **object**: A compound data type that is often used to model a thing or concept in the real world. It bundles together the data and the operations that are relevant for that kind of data. Instance and object are used interchangeably.

- **instance**: An object whose type is of some class. Instance and object are used interchangeably.

- **to instantiate**: To create an instance of a class, and to run its initializer.

- **object-oriented programming**: A powerful style of programming in which data and the operations that manipulate it are organized into classes and methods.

- **object-oriented language**: A language that provides features, such as user-defined classes and inheritance, that facilitate object-oriented programming.
Outline

Object-Oriented Programming Vocabulary (recap)

Bus simulation object-oriented program (recap)

Lecture Quiz 24

Medical diagnostic program (similar but not equivalent to A3)

An ecosystem simulation program (A4 preview)
To make use of a class, we need to create objects of that class. An **object** is an instantiation of a class that contains all the data for a particular example of that class.

**Class**

- **Bus**
  - Attributes: station, capacity, passengers, terminus
  - Methods: move(), unload(), load()
class Bus:
    def __init__(self):
        self.station = 0  # the position of the bus
        self.capacity = 5  # the capacity of the bus
        self.passengers = []  # the content of the bus
        self.terminus = 5  # The last station

    def move(self):
        if self.station < self.terminus:
            self.station += 1

    def unload(self):
        unloaded = [dest for dest in self.passengers if dest == self.station]
        self.passengers = [dest for dest in self.passengers if dest != self.station]
        return len(unloaded)

    def load(self, waiting_line):
        nb_board = min(len(waiting_line), self.capacity - len(self.passengers))
        people_boarding = waiting_line[0:nb_board]
        self.passengers.extend(people_boarding)
        return nb_board
The `__str__(self)` method

It is often useful to define how an object of given class should be converted to a string (e.g. for the `print` function). This is achieved by defining the method `__str__(self)`:

class Bus:
    def __init__(self):
        self.station = 0
        self.passengers = []

    def __str__(self):
        """
        Args: self
        Returns: String describing bus
        """
        return "Bus at station "+str(self.station) + \\
        " contains passengers " + \\n        \" str(self.passengers)

my_bus = Bus()
print(my_bus)  # will execute `__str__()` on my_bus to get a String, which then gets printed.
The OOP design makes the program a more readable
All the code that pertains to the bus behavior is in the Bus class.
See busSim_object_oriented.py

```python
for time in range(0, simulation_duration):
    # how many people are still waiting?
    for station, waiting in waiting_at_stop.items():
        nb_waiting_over_time[station][time] = len(waiting)

    # move the buses up by one station
    for bus in buses:
        bus.move()

    # bring new bus to station 0 at start_frequency
    if time % start_frequency == 0:
        new_bus = Bus()
        buses.append(new_bus)

    # let people disembark if they are at their station
    for bus in buses:
        nb_disembarked = bus.unload()
        nb_arrivals_over_time[bus.station][time] = nb_disembarked

    # let people embark, until the bus is full
    for bus in buses:
        nb_boarded = bus.load(waiting_at_stop[bus.station])
        del waiting_at_stop[bus.station][0:nb_boarded]
```
Outline

Object-Oriented Programming Vocabulary (recap)

Bus simulation object-oriented program (recap)

Lecture Quiz 24

Medical diagnostic program (similar but not equivalent to A3)

An ecosystem simulation program (A4 preview)
Outline

Object-Oriented Programming Vocabulary (recap)

Bus simulation object-oriented program (recap)

Lecture Quiz 24

Medical diagnostic program (similar but not equivalent to A3)

An ecosystem simulation program (A4 preview)
An OOP diagnostic program
(similar but not equivalent to our A3)

- **Encapsulation**: Define separate classes for separate concepts:
  - Symptoms
  - Patient
  - Probabilistic_diagnostics

- Each class will be stored in a different Python file (also called a module):
  - symptoms.py
  - patient.py
  - probabilistic_diagnostic.py.

- A module can import code (classes, functions, etc.) from another module.

- This allows big programs to be broken down into smaller, more digestible chunks.

- Makes easier understanding, developing, and debugging large programs
Patient Class

**Attributes:**
- ID # int
- symptoms # Symptom object
- diagnostic # String

**Methods:**
- `__init__(self, my_patient_ID, my_symptoms, my_diagnostic)`
- `most_similar_patients(self, all_patients, n_top=10)`
- `diagnostics_from_symptoms(self, all_patients, n_top=10)`
- `recommend_symptom_to_test(self, all_patients, n_top=10)`

Symptoms Class

**Attributes:**
- present
- absent

**Methods:**
- `__init__(self, pres, ab)`
- `symptom_similarity(self, other)`
- `__str__(self)`

Probabilistic_diagnostic Class

**Attributes:**
- prob # dict key: symp; value: prob

**Methods:**
- `__init__(self)`
- `count_diagnostics(self,patient_set)`
- `pretty_print_diagnostics(self)`
- `diagnostic_clarity(self)`
Symptoms class

- **Attributes:**
  - `present`: Set of symptoms (Strings) that are present
  - `absent`: Set of symptoms (Strings) that are absent

- **Methods:**
  - `__init__(self, pres, abs)`
  - `symptom_similarity(self, other)`
  - `__str__(self)`

See symptoms.py
Patient class

- **Attributes:**
  - ID: Integer
  - symptoms: Object of class Symptoms
  - diagnostic: String

- **Methods:**
  - `_init_`(self, my_patient_ID, my_symptoms, my_diagnostic)
  - most_similar_patients(self, all_patients, n_top=10)
  - diagnostics_from_symptoms(self, all_patients, n_top=10)
  - recommend_symptom_to_test(self, all_patients, n_top=10)
  - `__str__`(self)

Note: The Patient class needs to know about the Symptoms and Probabilistic_diagnostic classes. See patient.py
Probabilistic_diagnostic class

▶ Attributes:
  ▶ prob: Dictionary of diagnostic probabilities
  ▶ symptoms: Object of class Symptoms
  ▶ diagnostic: String

▶ Methods:
  ▶ __init__(self)
  ▶ count_diagnostics(self, patient_set):
  ▶ pretty_print_diagnostics(self):
  ▶ diagnostic_clarity(self):

See probabilistic_diagnostic.py
The test code that puts everything together is in a separate file: medical_diagnostic_tester.py.

It needs to import the three other modules:

```python
1 from symptoms import Symptoms
2 from patient import Patient
3 from probabilistic_diagnostic import Probabilistic_diagnostic
```
Outline

Object-Oriented Programming Vocabulary (recap)

Bus simulation object-oriented program (recap)

Lecture Quiz 24

Medical diagnostic program (similar but not equivalent to A3)

An ecosystem simulation program (A4 preview)
An OOP simulation program for ecosystem (A4 preview)

See the movie ecosim.mp4 file
Ecosim classes

- **Animal**
  - **Attributes**: id, age, age_max, age_spawn_min, age_spawn_max, spawn_waiting, spawn_waiting_time, hunger, hunger_max, visual_range, position
  - **Methods**: `__init__`, starve, eat, grow, die, will_spawn, inspect, move

- **Plant**
  - **Attributes**: id, available, regenerate_time, regenerate_countdown, position
  - **Methods**: `__init__`, consumed, regenerate

- **Position**
  - **Attributes**: x, y

- **Terrain**
  - **Attributes**: width, height, plants, animals
  - **Methods**: `__init__`, update_terrain, update_stats, `__str__`, simulate
Ecosim OOP overall design

**Animal**

- Attributes:
  - ID
  - age
  - age_max
  - age_spawn_min, age_spawn_max
  - age_spawn_waiting,
  - age_spawn_waiting_time
  - hunger, hunger_max
  - visual_range
  - position # Position object

- Methods:
  - __init__(self, terrain, id, position)
  - starve(self)
  - eat(self)
  - grow(self)
  - die(self)
  - will_spawn(self, terrain)
  - get_spawn_location(self, terrain)
  - inspect(self, terrain)
  - move(self, terrain)

**Plant**

- Attributes:
  - ID
  - available
  - regenerate_time
  - regenerate_countdown
  - position # Position object

- Methods:
  - __init__(self, terrain, id, position, regenerate_time)
  - consumed(self)
  - regenerate(self)

**Terrain**

- Attributes:
  - x, y

**Position**

- Attributes:
  - x, y

- Methods:
  - __init__(self, terrain, id, position)
  - starve(self)
  - eat(self)
  - grow(self)
  - die(self)
  - will_spawn(self, terrain)
  - get_spawn_location(self, terrain)
  - inspect(self, terrain)
  - move(self, terrain)

- Attributes:
  - plants # dict of Plant objects
  - animals # dict of Animal objects

- Methods:
  - __init__(self, width, height, nb_animals, nb_plants, ...)
  - update_terrain(self)
  - update_stats(self)
  - __str__(self) # print terrain map
  - simulate # run one step of the simulation
How can we make the pray and predator behave differently while sharing other attributes and methods under the Animal class?

Next lecture: Class Inheritance