# predator-prey simulations

1. Hopping Frogs
   - an object oriented model of a frog
   - animating frogs with threads
   - a GUI for hopping frogs

2. Flying Birds
   - an object oriented model of a bird
   - defining a pond of frogs
   - giving birds access to the swamp

MCS 260 Lecture 38
Introduction to Computer Science
Jan Verschelde, 26 July 2023

## predator-prey simulations

1. Hopping Frogs
   - an object oriented model of a frog
   - animating frogs with threads
   - a GUI for hopping frogs

2. Flying Birds
   - an object oriented model of a bird
   - defining a pond of frogs
   - giving birds access to the swamp

# Modeling Frogs

## in object oriented fashion

Imagine frogs happily hopping around in a pond ...

Functional attributes in the class `Frog`:

- `hop` : wait a bit and hop to next spot,
- `run` : do some fixed number of hops.

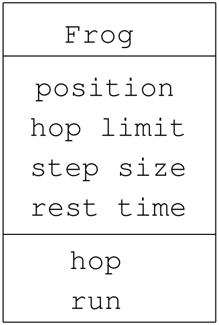
Data attributes in the class `Frog`:

- `position` : coordinates $(x, y)$ for its location,
- `hop limit` : number of hops done by the frog,
- `step size` : largest step size a frog can do,
- `rest time` : time while frog rests between steps.

Adding some randomness makes it more interesting.

## The Class Diagram

In the Unified Modeling Language (UML), we draw



## predator-prey simulations

1. Hopping Frogs
   - an object oriented model of a frog
   - animating frogs with threads
   - a GUI for hopping frogs

2. Flying Birds
   - an object oriented model of a bird
   - defining a pond of frogs
   - giving birds access to the swamp

## multithreading in Python

### many frogs hop independently

Most processors have multiple cores, allowing for parallel execution in real time, speeding up calculations.

Even on one core, the operating system runs many threads.

Python provides the Thread class in the threading module.

In addition to the `__init__` we override the definition of `run` when defining a class, inheriting from `Thread`.

For `t`, an instance of the inherited Thread class:
`t.start()` executes the defined `run` method.

$\rightarrow$ multiple threads run simultaneously

# predator-prey simulations

1. Hopping Frogs
   - an object oriented model of a frog
   - animating frogs with threads
   - a GUI for hopping frogs

2. Flying Birds
   - an object oriented model of a bird
   - defining a pond of frogs
   - giving birds access to the swamp

## Frogs on Canvas

### a GUI for hopping frogs

In our GUI, the canvas will be the pond.

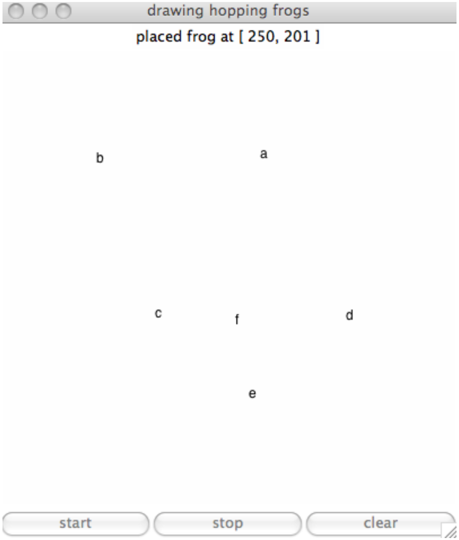
The user can place frogs in the pond with the mouse.

Three buttons:

- `start` : to start animating the hopping of frogs,
- `stop` : to stop the animation for adding frogs,
- `clear` : to clear the canvas after killing frogs.

The names of the frogs are letters `'a'`, `'b'`, `'c'`, ...
used as text to mark the frogs on canvas.

## running the GUI


placed frog at [ 250, 201 ]

# predator-prey simulations

1. Hopping Frogs
   - an object oriented model of a frog
   - animating frogs with threads
   - a GUI for hopping frogs

2. Flying Birds
   - an object oriented model of a bird
   - defining a pond of frogs
   - giving birds access to the swamp

## Modeling Birds

in object oriented fashion

Birds fly over the pond, preying on frogs.

Functional attributes for an object of the class `Bird`:

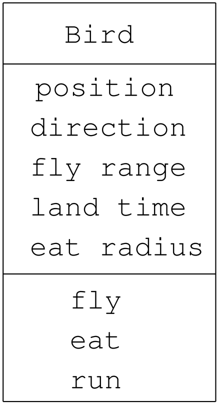
- `fly` : fly in some direction and then land,
- `eat` : eat closest frog within a certain radius,
- `run` : do `fly()` and `eat()` a fixed number of times.

Data attributes in the class `Bird`:

- `position` : coordinates $(x, y)$ for its location,
- `direction` : tuple $(d_x, d_y)$ defines flying direction,
- `fly range` : how many times a bird flies and lands,
- `land time` : time spent on land between air trips,
- `eat radius` : distance for closest frog to be eaten.

## The Class Diagram

In the Unified Modeling Language (UML), we draw



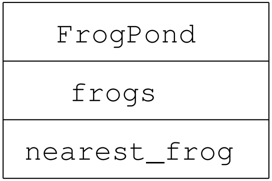
## predator-prey simulations

1. Hopping Frogs
   - an object oriented model of a frog
   - animating frogs with threads
   - a GUI for hopping frogs

2. Flying Birds
   - an object oriented model of a bird
   - defining a pond of frogs
   - giving birds access to the swamp

## finding the closest frog

To define the `eat()` method of the class `Bird`, we need to compute the closest frog.



## predator-prey simulations

1. Hopping Frogs
    - an object oriented model of a frog
    - animating frogs with threads
    - a GUI for hopping frogs

2. Flying Birds
    - an object oriented model of a bird
    - defining a pond of frogs
    - giving birds access to the swamp

## at start of `flying_birds.py`

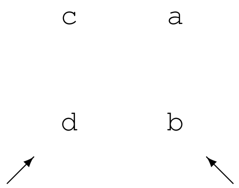
```
from class_threadfrog import ThreadFrog
from frog_pond import FrogPond

print('swamp with four frogs')
ANN = ThreadFrog('ann', +10, +10, 2, 5, 20)
BOB = ThreadFrog('bob', +10, -10, 2, 5, 20)
CINDY = ThreadFrog('cindy', -10, +10, 2, 5, 20)
DAVE = ThreadFrog('dave', -10, -10, 2, 5, 20)
SWAMP = FrogPond([ANN, BOB, CINDY, DAVE])

class Bird(Thread):
    """
    Exports flying birds as threads.
    """
```

## testing the simulation

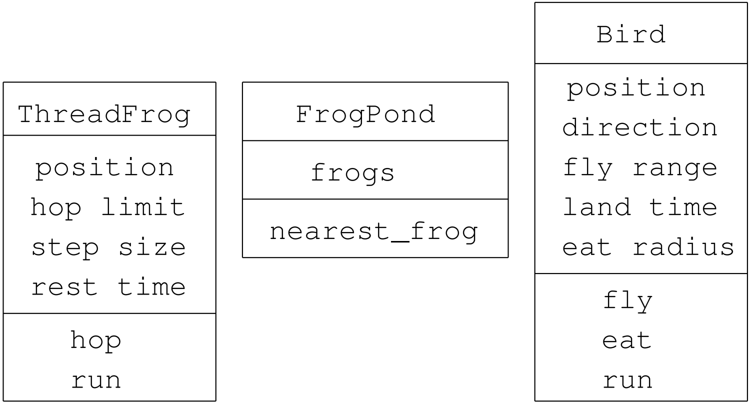
We have a swamp with four frogs,
initially placed as below:



and two birds flying into the pond,
with directions drawn by arrows above.

As they lie straight in the path of the birds,
frogs `b` and `d` are easy preys,
while `a` and `c` may hop away, escaping the birds.

## Three Classes



The class `FrogPond` depends on the class `ThreadFrog`.

The `eat()` method in `Bird` depends on `ThreadFrog` and on an instance of `FrogPond`.

## Exercises

1. Extend the `Frog` class with a method `breed()`. When two frogs are within a certain distance from each other, new frogs are born.
2. Adjust the direction of the bird in the `eat()` method so that the bird flies towards the closest frog.
3. Change the class `Bird` so the simulation runs from a script not in the same file as the definition of the class.
4. Add flying birds to the GUI to place frogs. Represent the birds by circles with radius equal to their eat radius so the user can observe when a frog is gobbled up.