Classes + Memory
CS106A, Stanford University
Housekeeping

- Survey about Embedded EthiCS
  - [https://forms.gle/GB9LzheQaUTfETMaA](https://forms.gle/GB9LzheQaUTfETMaA)
  - Or, use QR Code

*Results will be reported to & used to improve the Embedded Ethics program*
And now… a song about tuples!
Learning Goals

1. More practice with classes
2. See how to trace memory
Review:
classes and objects
Classes are like blueprints

class: A template for a new type of variable

A blueprint is a helpful analogy

When defining a new variable type you make a blueprint

Class Student
(Blueprint for student)

Student instances
(Objects)
Classes define new variable types
Classes help decompose your program across files
Classes Can Include Three Things

• Constructor
  – Method (function) called when a new object is being created

• Methods
  – Functions that you can call on an instance (object) of that class

• Instance variables
  – Variables inside each object of that class
  – Referred to using `self. variable_name`
class Dog:
    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')
    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)
def __init__(self, breed):
    print('A new dog is born')
    self.times_barked = 0
    self.breed = breed

def bark(self):
    if self.breed == 'pomeranian':
        print('yip')
    else:
        print('woof')
    self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)

• Constructor is called each time we create a new object
class Dog:

def __init__(self, breed):
    print('A new dog is born')
    self.times_barked = 0
    self.breed = breed

def bark(self):
    if self.breed == 'pomeranian':
        print('yip')
    else:
        print('woof')
    self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)

• Instance variables are stored inside each object
• Each object has its own version of the instance variables
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)

• Methods are functions that can be called on a particular object
class Dog:
    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)

• When authoring a class, self means:
"the instance (aka object) I am currently working with"
class Dog:
    
    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)
class Dog:
    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed
    
    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')
    simba.bark()
    juno.bark()
    simba.bark()
    print('simba', simba.__dict__)
    print('juno', juno.__dict__)
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')
    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)
class Dog:
    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed
    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)

A new dog is born
A new dog is born
yip
woof
yip
• Instance variables in an object are stored in internal \texttt{.__dict__} variable
class Dog:
    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)

A new dog is born
A new dog is born
yip
woof
yip
simba {'times_barked': 2, 'breed': 'pomeranian'}
class Dog:
    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

    simba.bark()
    juno.bark()
    simba.bark()

    print('simba', simba.__dict__)
    print('juno', juno.__dict__)

A new dog is born
A new dog is born
yip
woof
yip
simba {'times_barked': 2, 'breed': 'pomeranian'}
juno {'times_barked': 1, 'breed': 'great dane'}
Recall Functions?

Coder: **Function Author**
- Writes helper functions others can use

Coder: **Function Caller**
- Uses helper functions
Classes also split up the work!

Coder: **Class Author**

Writes the class (often in its own file), thus defining a new variable type

Coder: **Class Client**

Uses the new variable type to solve problems (often from main).

Because they are classy
Class Author: Writes the class, thus defining a new variable type (often in its own file)

Class Client: Uses the new variable type to solve problems (often from main).
Next step in writing large programs: Better understand memory
You are now ready...
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
What does this do?

```python
def main():
x = 5
print(id(x))
x = x + 1
print(id(x))
```
```python
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
```
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))

What does this do?
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))

What does this do?
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))

What does this do?
Each time a function is called, a new frame of memory is created.

Each frame has space for all the local variables declared in the function, and parameters.

Each variable has a reference which is like a URL.

When a function returns, its frame is destroyed.
The heap

<table>
<thead>
<tr>
<th>ID</th>
<th>value</th>
<th>type</th>
<th>ref count</th>
</tr>
</thead>
<tbody>
<tr>
<td>4563589904</td>
<td>0</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4563589936</td>
<td>1</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Where values are stored
- Every value has an address (like a URL address)
- Values don’t go away when functions return
- Memory is recycled when its no longer used.
What does this do?

def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))

When a variable is “used” you are accessing its **value**
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))

When a variable is “assigned” via *binding* you are changing its *reference*

You know a variable is being assigned to if it is on the left hand side of an *=`* sign
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
\textbf{What does this do?}

```python
def main():
    x = 5
    \textcolor{blue}{\textbf{binky}(9)}

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```

Stack:

- **main**
- **x**: 5563936

- **int**
  - 1
  - 5
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
def main():
    x = 5
    binky(9)

def binky(y):  
    pinky(y)

def pinky(z):  
    print(z)

What does this do?
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
What does this do?

def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)

What does this do?
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)

console
9
What does this do?

def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)

console

9
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)

console
9
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)

console

9

Stack

main

x

5563936

int

1

5

0

9
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)

What does this do?
What does this do?

```python
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```

console

```console
9
```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)

console
9
```python
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```

console

9
Whoa!
def main():
    x = ['a', 'b', 'c']
    print(x)
    update_in_list(x)
    print(x)
    update_list(x)
    print(x)

def update_in_list(x):
    x[0] = 'z'

def update_list(x):
    x = ['m', 'n', 'o']

if __name__ == '__main__':
    main()

http://www.pythontutor.com/visualize.html
Learning Goals

1. More practice with classes
2. See how to trace memory