Remember this?
Bouncing Balls
Learning Goals

1. Practice with classes
2. See how to trace memory with classes
Guiding question for today:
what does it take to go from what you know to writing big-scale software?
Some *large* programs are in Python
<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Album</th>
<th>Date</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Innerbloom</td>
<td>RÜFÜS DU SOL</td>
<td>Bloom</td>
<td>2019-09-27</td>
<td>9:38</td>
</tr>
<tr>
<td>Nevermind</td>
<td>Dennis Lloyd</td>
<td>Nevermind</td>
<td>2019-09-27</td>
<td>2:37</td>
</tr>
<tr>
<td>Obiero</td>
<td>Ayub Ogada</td>
<td>En Mana Huyo (Real World Gold)</td>
<td>2019-09-27</td>
<td>5:40</td>
</tr>
<tr>
<td>Memories</td>
<td>Petit Biscuit</td>
<td>Memories</td>
<td>2019-10-07</td>
<td>3:35</td>
</tr>
<tr>
<td>The Sun</td>
<td>Pared Stelan, Graham Candy</td>
<td>The Sun (Klingande Remix)</td>
<td>2019-10-07</td>
<td>2:55</td>
</tr>
<tr>
<td>Receiver</td>
<td>Tycho</td>
<td>Gourmet</td>
<td>2019-10-18</td>
<td>4:15</td>
</tr>
<tr>
<td>Flow</td>
<td>Crooked Colours</td>
<td>Vera</td>
<td>2019-10-19</td>
<td>4:50</td>
</tr>
<tr>
<td>Antofagasta de la Sierra - El Buho's Nocturnal Remix</td>
<td>Lagartijo, El Buho</td>
<td>Antofagasta de la Sierra</td>
<td>2019-10-29</td>
<td>5:25</td>
</tr>
<tr>
<td>Moonwalk Away</td>
<td>GoldFish</td>
<td>Three Second Memory</td>
<td>2019-11-11</td>
<td>3:58</td>
</tr>
<tr>
<td>Hang Outback</td>
<td>Hang In Balatine, Martin Cradick</td>
<td>Lisa</td>
<td>2019-11-11</td>
<td>3:20</td>
</tr>
<tr>
<td>I Keep Ticking On</td>
<td>The Harmaleighs</td>
<td>Pretty Picture, Dirty Brush</td>
<td>2019-11-11</td>
<td>2:30</td>
</tr>
<tr>
<td>Far From Home</td>
<td>Orfatri, Mougleta</td>
<td>Far From Home</td>
<td>2019-11-12</td>
<td>2:46</td>
</tr>
<tr>
<td>Baby</td>
<td>Bäkermat</td>
<td>Baby</td>
<td>2019-01-06</td>
<td>2:40</td>
</tr>
<tr>
<td>Ween El-Kalam</td>
<td>Haydan</td>
<td>Ya Bay</td>
<td>2019-01-06</td>
<td>5:36</td>
</tr>
<tr>
<td>Coming Home</td>
<td>Adon, Nicolas Haelg, Sam Halabi</td>
<td>Coming Home</td>
<td>2019-01-22</td>
<td>2:43</td>
</tr>
<tr>
<td>Flumintrüdenen</td>
<td>Emil Bedirser</td>
<td>Flumintrüdenen</td>
<td>2019-02-21</td>
<td>5:12</td>
</tr>
<tr>
<td>Bum Bum Tam Tam</td>
<td>MC Slim, Future, J Bakir, Steffon Dan, J</td>
<td>Bum Bum Tam Tam</td>
<td>2019-04-06</td>
<td>3:34</td>
</tr>
<tr>
<td>Fassinated - Instrumental Mix</td>
<td>Mr Blue Sky, Lloyd Chapman, Angell Turn</td>
<td>Fassinated</td>
<td>2019-04-08</td>
<td>6:38</td>
</tr>
<tr>
<td>Nunchaku Club</td>
<td>Gnu Vol, 980</td>
<td>Nunchaku Club</td>
<td>2019-04-20</td>
<td>7:04</td>
</tr>
</tbody>
</table>
How?
Define New Variable Types

- Song
- Playlist
- User

- Song Player
- Song Retriever
You Have Been *Using* Variable Types

SimpleImage  Canvas  Karel

String  int

What would it take to define your own?
type
Classes define new variable types
Classes decompose your program across files
Classes are like blueprints

class: A template for a new type of variable.

A blueprint is a helpful analogy
You must define three things

1. What **variables** does each instance store?

2. What **methods** can you call on an instance?

3. What happens when you make a **new** one?

*details on how to define these three things coming soon*
.__dict__
```python
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1

def main():
    jupiter = Dog()
    juno = Dog()

    jupiter.bark()
    juno.bark()
    jupiter.bark()

    print(jupiter.__dict__)
    print(juno.__dict__)
```

```python
# Dog.py

# life.py
```
1. What **variables** does each instance store?
2. What **methods** can you call on an instance?
def main():
    jupiter = Dog()
    juno = Dog()
    jupiter.bark()
    juno.bark()
    jupiter.bark()
    print(jupiter.__dict__)
    print(juno.__dict__)

3. What happens when you make a new one?
Classes Review

Did I mention that a class is like a fancy dictionary?

Dog.py

class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1

life.py

def main():
    jupiter = Dog()
    juno = Dog()

    jupiter.bark()
    juno.bark()
    jupiter.bark()

    print(jupiter.__dict__)
    print(juno.__dict__)
What is a class?
A class defines a new variable type
How many variables for the ball?

1. oval
2. change_x
3. change_y
Bouncing Balls
# 1: Store a list of dictionaries

# 2: Store a list of Balls
Next step in writing large programs: Better understand memory
You are now ready...
What does this do?

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

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

stack

heap

overhead

value

main

x

4563589904

5
What does this do?

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

stack

main

x

4563589904

heap

int

1

5
What does this do?

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

Stack:
- `main`
- `x` with type `int` and value `5`

Heap:
- `int` type
- `1` for ref count
- `5` for value
def main():
    x = 5
    print(id(x))
    x += 1
    print(id(x))

What does this do?
What does this do?

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

stack

heap

main

x

4563589904

int

value

4563589904

1

5

value

ref count

type
What does this do?

def main():
    x = 5
    print(id(x))
    x = x + 1  # Blue box highlights this line.
    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?
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>int</th>
<th>type</th>
<th>ref count</th>
</tr>
</thead>
<tbody>
<tr>
<td>4563589904</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>type</th>
<th>ref count</th>
</tr>
</thead>
<tbody>
<tr>
<td>4563589936</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<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.
def main():
    x = 5
    x = x + 1
What does this do?

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

When a variable is “assigned” 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
    x = x + 1

What does this do?

When a variable is “used” you are accessing its value

You know a variable is being used to if it is not on the left hand side of an = sign
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)

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)
What does this do?

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

def binky(y):
    pinky(y)

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

Stack:
- main
- x 5563936
- binky
- y 9563936

```
5563936
---
int
1
5

9563936
---
int
1
9
```
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)
What does this do?

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

def binky(y):
    pinky(y)

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

Stack:
- main
  - x: 5
  - binky
    - y: 9
  - pinky
    - z: 9
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)

What does this do?
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)
What is... the matrix?
The matrix origins

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

def main():
    x = ['a', 'b', 'c']
    update(x)

def update(x):
    for v in x:
        print(type(v), v)
        v = v + '!
        print(v)

if __name__ == '__main__':
    main()
What is *self*?
class Dog:
    def __init__(self, name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    print(first)
    print(type(first))
    print(id(id(first))
    print(first.__dict__)
```python
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
```

Stack:

- main
- first
- second
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
```python
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
```

Stack:
- main
- first
- second
- Dog.__init__
  - self
  - new_name: 'jupiter'
- Dog
  - 1

---

What does this do?
```python
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
```

What does this do?

class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
What does this do?

```python
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
```

Stack:
- main
- first
- second
- Dog.__init__
  - self: 42
  - new_name: 'jupiter'

42

Dog
- 1
- name: 'jupiter'
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
What does this do?

class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...

def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
What does this do?

class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
What does this do?

class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...

def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)

Stack

main

first 42

second
```python
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
```

What does this do?

class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
What does this do?

class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
What does this do?

class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
What does this do?

```python
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)  # Stack
```

```
1
Dog
name
jupiter'
42
```

```
1
Dog
name
juno'
48
```

```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')

    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
What does this do?

class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

    # put in another file...
    def main():
        first = Dog('jupiter')
        second = Dog('juno')

        print(first)
        print(type(first))
        print(id(first))
        print(first.__dict__)
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('jupiter')
    second = Dog('juno')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1

def main():
    jupiter = Dog()
    juno = Dog()

    jupiter.bark()
    juno.bark()
    jupiter.bark()

    print(jupiter.__dict__)
    print(juno.__dict__)
Learning Goals

1. Practice with classes
2. See how to trace memory with classes