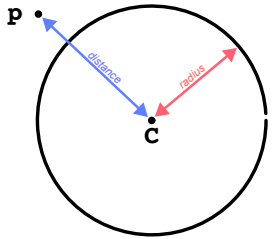
Lecture 23: Object Oriented Programming
Inheritance

Comp 102

Forman Christian University

Recap

Circle class



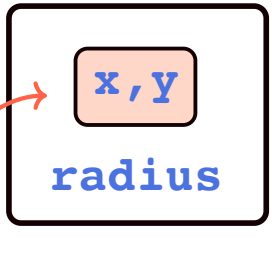

```
1 class Point:
2     ...
3 class Circle:
4     def __init__(self,x,y,radius):
5         self.x = x
6         self.y = y
7         self.radius = radius
```

Abstraction violation

could've used 'Point' class

Circle class

```
1 class Point:
2     ...
3 class Circle:
4     def __init__(self,center,radius):
5         self.center = center
6         self.radius = radius
```



Python is NOT Great at Encapsulation

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5 p = Person('Zain')
6
7
8
```

Python is NOT Great at Encapsulation

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5 p = Person('Zain')
6 print(p.name)    # access data from outside
7
8
```

Python is NOT Great at Encapsulation

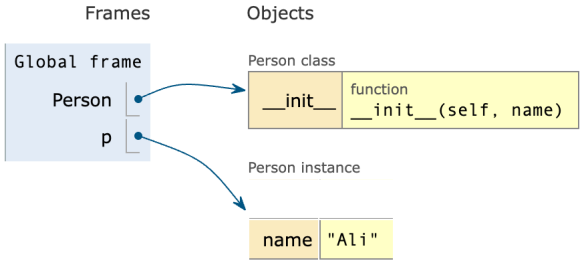
```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5 p = Person('Zain')
6 print(p.name)    # access data from outside
7 p.name = 'Ali'  # change data from outside
8
```

Python is NOT Great at Encapsulation

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5 p = Person('Zain')
6 print(p.name)    # access data from outside
7 p.name = 'Ali'  # change data from outside
8 p.age = 20      # add new data from outside
```

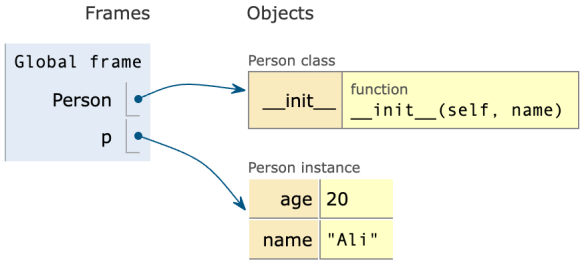
Python is NOT Great at Encapsulation

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5 p = Person('Zain')
6 print(p.name)
7 p.name = 'Ali'
8
```



Python is NOT Great at Encapsulation

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5 p = Person('Zain')
6 print(p.name)
7 p.name = 'Ali'
8 p.age = 20
```



Storing Many Objects

Consider the following Animal class

```
1  class Animal(object):
2      def __init__(self, age):
3          self.age = age
4          self.name = None
5      def __str__(self):
6          return f'animal:{self.name}-{self.age}'
7
8  cat = Animal(2)
9  cat.name = "Kitty"  # Warning: direct access
10 print(cat)
```

Using our new Animal class

```
1  def animal_dict(L):
2      """ Input: L is a list
3      Return: a dict, d, mappping an int to an Animal obj. A key in
           d is all non-negative ints n in L. A value corresponding
          to a key is an Animal object with n as its age. """
4      d = {}
5      for n in L:
6          if type(n) == int and n >= 0:
7              d[n] = Animal(n)
8      return d
9
10 L = [2,5,'a',-5,0]
11 animals = animal_dict(L)
12 print(animals)              # Python can't call print recursively
```

Using our new Animal class

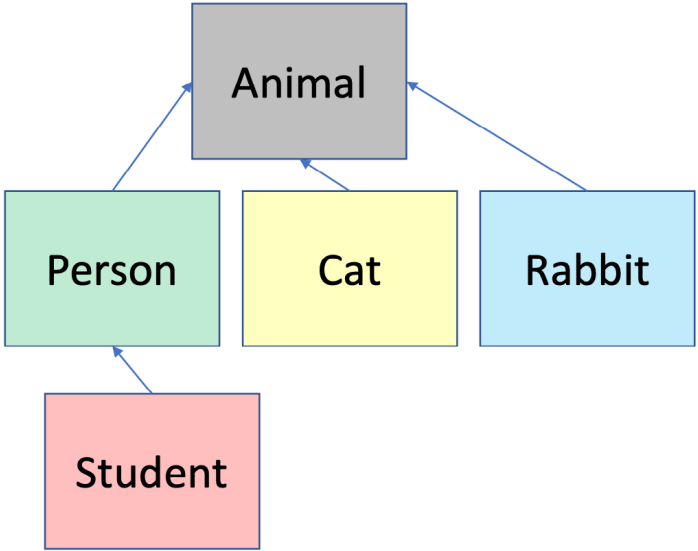
```
1  def animal_dict(L):
2      """ Input: L is a list
3      Return: a dict, d, mappping an int to an Animal obj. A key in
           d is all non-negative ints n in L. A value corresponding
          to a key is an Animal object with n as its age. """
4      d = {}
5      for n in L:
6          if type(n) == int and n >= 0:
7              d[n] = Animal(n)
8      return d
9
10 L = [2,5,'a',-5,0]
11 animals = animal_dict(L)
12 for n,a in animals.items():
13     print(f'key {n} with val {a}') # automatically calls __str__
```

You Try!

```
1  def make_animals(L1, L2):
2      """ Input: L1 is a list of ints and L2 is a list of str L1
           and L2 have the same length
3      Return: Creates a list of Animals the same length as L1 and
           L2. An animal object at index i has the age and name
           corresponding to the same index in L1 and L2, respectively
            """
4
5  #For example:
6  L1 = [2,5,1]
7  L2 = ["blobfish", "crazyant", "parafox"]
8  animals = make_animals(L1, L2)
9  print(animals) # note this prints a list of animal objects
10 for i in animals: # this loop prints the individual animals
11     print(i)
```

Hierarchies

- **Parent class**
  (superclass)
- **Child class**
  (subclass)
  - **Inherits** all data and behaviors of parent class
  - **Add** more **info**
  - **Add** more **behavior**
  - **Override** behavior



Inheritance: Parent Class

```
1  class Animal(object):
2      def __init__(self, age):
3          self.age = age
4          self.name = None
5      def get_age(self):
6          return self.age
7      def get_name(self):
8          return self.name
9      def set_age(self, newage):
10         self.age = newage
11     def set_name(self, newname=""):
12         self.name = newname
13     def __str__(self):
14         return f'animal:{self.name}-{self.age}'
```

Inheritance: Parent Class

```
1  class Animal(object):
2      def __init__(self, age):
3          self.age = age
4          self.name = None
5      def get_age(self):
6          return self.age
7      def get_name(self):
8          return self.name
9      def set_age(self, newage):
10         self.age = newage
11     def set_name(self, newname=""):
12         self.name = newname
13     def __str__(self):
14         return f'animal:{self.name}-{self.age}'
```

`object` is the parent class of all classes in Python

Subclass Cat

Subclass Cat

```
class Cat(Animal):
    def speak(self):
        print("meow")
    def __str__(self):
        return f'cat:{self.name}-{self.age}'

c = Cat(2)
c.set_name('simba')
print(c)
```

Subclass Cat

```
1 class Cat(Animal):
2     def speak(self):
3         print("meow")
4     def __str__(self):
5         return f'cat:{self.name}-{self.age}'
6
7 c = Cat(2)
8 c.set_name('simba')
9 print(c)
```

Inherits all attributes and methods from the Animal class

Subclass Cat

```
1 class Cat(Animal):
2     def speak(self):
3         print("meow")
4     def __str__(self):
5         return f'cat:{self.name}-{self.age}'
6
7 c = Cat(2)
8 c.set_name('simba')
9 print(c)
```

Add new functionality. Not present in the parent class

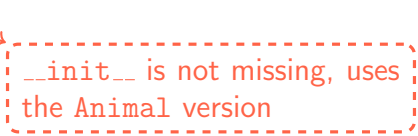
Subclass Cat

```
1  class Cat(Animal):
2      def speak(self):
3          print("meow")
4      def __str__(self):
5          return f'cat:{self.name}-{self.age}'
6
7  c = Cat(2)
8  c.set_name('simba')
9  print(c)
```

"**Override**" `__str__`, replacing parent's method

Subclass Cat

```
1 class Cat(Animal):
2     def speak(self):
3         print("meow")
4     def __str__(self):
5         return f'cat:{self.name}-{self.age}'
6
7 c = Cat(2)
8 c.set_name('simba')
9 print(c)
```



Big Idea

Override a method: Create a **new method** in the child class but with **same name** as in the parent class.

Can't use child class Methods

```
1 a = Animal(1)
2 c = Cat(2)   # Child CAN use parent's methods:
3              #   (__init__)
4 c.speak()    # meow
5 a.speak()    # ERROR: parent can NOT use
6              #   child's methods or attributes
```

Which Method to Use?

- Subclass can have **methods with same name** as superclass *(method* **override***)*

Which Method to Use?

- Subclass can have **methods with same name** as superclass *(method* **override***)*
- For an instance of a class, look for a method name in **current class definition**
- If not found, look for method name **up the hierarchy** *(in parent, then grandparent, and so on)*

Which Method to Use?

- Subclass can have **methods with same name** as superclass *(method* **override***)*
- For an instance of a class, look for a method name in **current class definition**
- If not found, look for method name **up the hierarchy** *(in parent, then grandparent, and so on)*
- Use first method up the hierarchy that you found with that method name

Subclass Person

Subclass Person

```
1  class Person(Animal):
2      def __init__(self, name, age):
3          Animal.__init__(self, age)
4          self.set_name(name)
5          self.friends = []
6      def get_friends(self):
7          return self.friends.copy()
8      def add_friend(self, fname):
9          if fname not in self.friends:
10             self.friends.append(fname)
11     def speak(self):
12         print("hello")
13     def age_diff(self, other):
14         diff = self.age - other.age
15         print(abs(diff), "year difference")
16     def __str__(self):
17         return f'person:{self.name}-{self.age}'
```

Subclass Person

```
1  class Person(Animal):
2      def __init__(self, name, age):
3          Animal.__init__(self, age)
4          self.set_name(name)
5          self.friends = []
6      def get_friends(self):
7          return self.friends.copy()
8      def add_friend(self, fname):
9          if fname not in self.friends:
10             self.friends.append(fname)
11     def speak(self):
12         print("hello")
13     def age_diff(self, other):
14         diff = self.age - other.age
15         print(abs(diff), "year difference")
16     def __str__(self):
17         return f'person:{self.name}-{self.age}'
```

Parent is the Animal class

Subclass Person

```
1  class Person(Animal):
2      def __init__(self, name, age):
3          Animal.__init__(self, age)
4          self.set_name(name)
5          self.friends = []
6      def get_friends(self):
7          return self.friends.copy()
8      def add_friend(self, fname):
9          if fname not in self.friends:
10             self.friends.append(fname)
11     def speak(self):
12         print("hello")
13     def age_diff(self, other):
14         diff = self.age - other.age
15         print(abs(diff), "year difference")
16     def __str__(self):
17         return f'person:{self.name}-{self.age}'
```

Note: Person class **overrides** `__init__` method

Subclass Person

```
1  class Person(Animal):
2      def __init__(self, name, age):
3          Animal.__init__(self, age)
4          self.set_name(name)
5          self.friends = []
6      def get_friends(self):
7          return self.friends.copy()
8      def add_friend(self, fname):
9          if fname not in self.friends:
10             self.friends.append(fname)
11     def speak(self):
12         print("hello")
13     def age_diff(self, other):
14         diff = self.age - other.age
15         print(abs(diff), "year difference")
16     def __str__(self):
17         return f'person:{self.name}-{self.age}'
```

Due to overriding, Animal's `__init__` method will **not** be called automatically

Subclass Person

```
1  class Person(Animal):
2      def __init__(self, name, age):
3          Animal.__init__(self, age)
4          self.set_name(name)
5          self.friends = []
6      def get_friends(self):
7          return self.friends.copy()
8      def add_friend(self, fname):
9          if fname not in self.friends:
10             self.friends.append(fname)
11     def speak(self):
12         print("hello")
13     def age_diff(self, other):
14         diff = self.age - other.age
15         print(abs(diff), "year difference")
16     def __str__(self):
17         return f'person:{self.name}-{self.age}'
```

Person class has **additional** attributes

Subclass Person

```
1  class Person(Animal):
2      def __init__(self, name, age):
3          Animal.__init__(self, age)
4          self.set_name(name)
5          self.friends = []
6      def get_friends(self):
7          return self.friends.copy()
8      def add_friend(self, fname):
9          if fname not in self.friends:
10             self.friends.append(fname)
11     def speak(self):
12         print("hello")
13     def age_diff(self, other):
14         diff = self.age - other.age
15         print(abs(diff), "year difference")
16     def __str__(self):
17         return f'person:{self.name}-{self.age}'
```

Person class has **additional** methods

Subclass Person

```
1  class Person(Animal):
2      def __init__(self, name, age):
3          Animal.__init__(self, age)
4          self.set_name(name)
5          self.friends = []
6      def get_friends(self):
7          return self.friends.copy()
8      def add_friend(self, fname):
9          if fname not in self.friends:
10             self.friends.append(fname)
11     def speak(self):
12         print("hello")
13     def age_diff(self, other):
14         diff = self.age - other.age
15         print(abs(diff), "year difference")
16     def __str__(self):
17         return f'person:{self.name}-{self.age}'
```

Person class **overrides** __str__ method

You Try!

Write a function according to the following specification:

```
1  def make_pets(d):
2      ''' Input: d is a dict mapping a Person obj to a Cat obj
3      Prints: on each line, the name of a person, a colon, and the
4      name of that person's cat
5      Output: None '''
6      pass
7
8  p1 = Person("zaid", 54)
9  p2 = Person("ahmed", 38)
10 c1 = Cat(1)
11 c1.set_name("simba")
12 c2 = Cat(1)
13 c2.set_name("tom")
14 d = {p1:c1, p2:c2}
15 make_pets(d) # prints zaid:simba
16             #        ahmed:tom
```

Big Idea

A subclass can **use** a parent's attributes, **override** a parent's attributes, or **define new** attributes.

Attributes are either data or methods.

Subclass Student

```python
import random
class Student(Person):
    def __init__(self, name, age, major=None):
        Person.__init__(self, name, age)
        self.major = major
    def change_major(self, major):
        self.major = major
    def speak(self):
        r = random.random()
        if r < 0.25:
            print("i have homework")
        elif 0.25 <= r < 0.5:
            print("i need sleep")
        elif 0.5 <= r < 0.75:
            print("i should eat")
        else:
            print("i'm still zooming")
    def __str__(self):
        return f'person:{self.name}-{self.age}-{self.major}'
```

```python
import random
class Student(Person):
    def __init__(self, name, age, major=None):
        Person.__init__(self, name, age)
        self.major = major
    def change_major(self, major):
        self.major = major
    def speak(self):
        r = random.random()
        if r < 0.25:
            print("i have homework")
        elif 0.25 <= r < 0.5:
            print("i need sleep")
        elif 0.5 <= r < 0.75:
            print("i should eat")
        else:
            print("i'm still zooming")
    def __str__(self):
        return f'person:{self.name}-{self.age}-{self.major}'
```

Student inherits both **Person** and **Animal** attributes

```python
import random
class Student(Person):
    def __init__(self, name, age, major=None):
        Person.__init__(self, name, age)
        self.major = major
    def change_major(self, major):
        self.major = major
    def speak(self):
        r = random.random()
        if r < 0.25:
            print("i have homework")
        elif 0.25 <= r < 0.5:
            print("i need sleep")
        elif 0.5 <= r < 0.75:
            print("i should eat")
        else:
            print("i'm still zooming")
    def __str__(self):
        return f'person:{self.name}-{self.age}-{self.major}'
```

Person's __init__ creates it's own attributes as well as **Animal's** attributes

```python
import random
class Student(Person):
    def __init__(self, name, age, major=None):
        Person.__init__(self, name, age)
        self.major = major
    def change_major(self, major):
        self.major = major
    def speak(self):
        r = random.random()
        if r < 0.25:
            print("i have homework")
        elif 0.25 <= r < 0.5:
            print("i need sleep")
        elif 0.5 <= r < 0.75:
            print("i should eat")
        else:
            print("i'm still zooming")
    def __str__(self):
        return f'person:{self.name}-{self.age}-{self.major}'
```

Student class creates **additional** attributes

```
1  import random
2  class Student(Person):
3      def __init__(self, name, age, major=None):
4          Person.__init__(self, name, age)
5          self.major = major
6      def change_major(self, major):
7          self.major = major
8      def speak(self):
9          r = random.random()
10         if r < 0.25:
11             print("i have homework")
12         elif 0.25 <= r < 0.5:
13             print("i need sleep")
14         elif 0.5 <= r < 0.75:
15             print("i should eat")
16         else:
17             print("i'm still zooming")
18     def __str__(self):
19         return f'person:{self.name}-{self.age}-{self.major}'
```

Student speaks differently than **Person** *(behavior override)*

Class Variables and the **Subclass Rabbit**

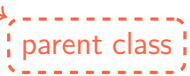
Class Variables and the Rabbit Subclass

- **Class variables** and their values are shared between all instances of a class

```
1  class Rabbit(Animal):
2      tag = 1
3      def __init__(self,age,parent1=None,parent2=None):
4          Animal.__init__(self, age)
5          self.parent1 = parent1
6          self.parent2 = parent2
7          self.rid = Rabbit.tag
8          Rabbit.tag += 1
```

Class Variables and the Rabbit Subclass

- **Class variables** and their values are shared between all instances of a class




```
1  class Rabbit(Animal):
2      tag = 1
3      def __init__(self,age,parent1=None,parent2=None):
4          Animal.__init__(self, age)
5          self.parent1 = parent1
6          self.parent2 = parent2
7          self.rid = Rabbit.tag
8          Rabbit.tag += 1
```

Class Variables and the Rabbit Subclass

- **Class variables** and their values are shared between all instances of a class


```
1  class Rabbit(Animal):
2      tag = 1
3      def __init__(self,age,parent1=None,parent2=None):
4          Animal.__init__(self, age)
5          self.parent1 = parent1
6          self.parent2 = parent2
7          self.rid = Rabbit.tag
8          Rabbit.tag += 1
```

Class Variables and the Rabbit Subclass

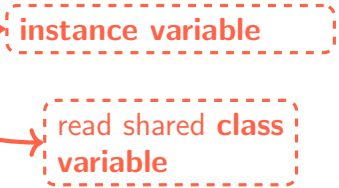
- **Class variables** and their values are shared between all instances of a class

```
1  class Rabbit(Animal):
2      tag = 1
3      def __init__(self,age,parent1=None,parent2=None):
4          Animal.__init__(self, age)
5          self.parent1 = parent1
6          self.parent2 = parent2
7          self.rid = Rabbit.tag
8          Rabbit.tag += 1
```

Class Variables and the Rabbit Subclass

- **Class variables** and their values are shared between all instances of a class

```
1  class Rabbit(Animal):
2      tag = 1
3      def __init__(self,age,parent1=None,parent2=None):
4          Animal.__init__(self, age)
5          self.parent1 = parent1
6          self.parent2 = parent2
7          self.rid = Rabbit.tag
8          Rabbit.tag += 1
```



Class Variables and the Rabbit Subclass

- **Class variables** and their values are shared between all instances of a class

```
1  class Rabbit(Animal):
2      tag = 1
3      def __init__(self,age,parent1=None,parent2=None):
4          Animal.__init__(self, age)
5          self.parent1 = parent1
6          self.parent2 = parent2
7          self.rid = Rabbit.tag
8          Rabbit.tag += 1
```

Modifying class variable changes it for **all** instances that may reference it

Class Variables and the Rabbit Subclass

- **Class variables** and their values are shared between all instances of a class

```
1 class Rabbit(Animal):
2     tag = 1
3     def __init__(self,age,parent1=None,parent2=None):
4         Animal.__init__(self, age)
5         self.parent1 = parent1
6         self.parent2 = parent2
7         self.rid = Rabbit.tag
8         Rabbit.tag += 1
```

- `tag` used to give **unique id** to each new rabbit instance

```
def __init__(self,age,parent1=None,
             parent2=None):
    Animal.__init__(self, age)
    self.parent1 = parent1
    self.parent2 = parent2
    self.rid = Rabbit.tag
    Rabbit.tag += 1
```

Rabbit.tag 1

```
def __init__(self,age,parent1=None,
             parent2=None):
    Animal.__init__(self, age)
    self.parent1 = parent1
    self.parent2 = parent2
    self.rid = Rabbit.tag
    Rabbit.tag += 1
```

Rabbit.tag 2

r1

Age: 8
Parent1: None
Parent2: None
Rid: 1

```
r1 = Rabbit(8)
```

```
def __init__(self,age,parent1=None,
             parent2=None):
    Animal.__init__(self, age)
    self.parent1 = parent1
    self.parent2 = parent2
    self.rid = Rabbit.tag
    Rabbit.tag += 1
```

```
r1 = Rabbit(8)
r2 = Rabbit(6)
```

Rabbit.tag 3

r1

Age: 8
Parent1: None
Parent2: None
Rid: 1

r2

Age: 6
Parent1: None
Parent2: None
Rid: 2

```
def __init__(self,age,parent1=None,
             parent2=None):
    Animal.__init__(self, age)
    self.parent1 = parent1
    self.parent2 = parent2
    self.rid = Rabbit.tag
    Rabbit.tag += 1
```

```
r1 = Rabbit(8)
r2 = Rabbit(6)
r3 = Rabbit(10)
```

Rabbit.tag 4

r1

Age: 8
Parent1: None
Parent2: None
Rid: 1

r2

Age: 6
Parent1: None
Parent2: None
Rid: 2

r3

Age: 10
Parent1: None
Parent2: None
Rid: 3

Rabbit Getter Methods

```
1  class Rabbit(Animal):
2      tag = 1
3      def __init__(self, age, parent1=None, parent2=None):
4          Animal.__init__(self, age)
5          self.parent1 = parent1
6          self.parent2 = parent2
7          self.rid = Rabbit.tag
8          Rabbit.tag += 1
9      #-------------------------------------#
10     def get_rid(self):                    #
11         return str(self.rid).zfill(5)     #  Getter Methods
12     def get_parent1(self):                #    specific to the
13         return self.parent1               #    Rabbit class
14     def get_parent2(self):                #
15         return self.parent2               #
16     #-------------------------------------#
```

Working with Your Own Types

```
1 def __add__(self, other):
2     # returning object of same type as this class
3     return Rabbit(0, self, other)
```

- Define + **operator** between two Rabbit instances
  - For example:

    `r4 = r1 + r2`

    `r1` and `r2` are `Rabbit` instances, combine to create `r4`

Working with Your Own Types

```
1 def __add__(self, other):
2     # returning object of same type as this class
3     return Rabbit(0, self, other)
```

- Define + **operator** between two Rabbit instances
  - For example:

    `r4 = r1 + r2`

    `r1` and `r2` are `Rabbit` instances, combine to create `r4`
- `r4` is a new `Rabbit` instance with age 0

Working with Your Own Types

```
1 def __add__(self, other):
2     # returning object of same type as this class
3     return Rabbit(0, self, other)
```

- Define **+ operator** between two Rabbit instances
  - For example:

    `r4 = r1 + r2`

    `r1` and `r2` are `Rabbit` instances, combine to create `r4`
- `r4` is a new `Rabbit` instance with age 0
- `r4` has `self` as one parent and `other` as the other parent

Working with Your Own Types

```
1 def __add__(self, other):
2     # returning object of same type as this class
3     return Rabbit(0, self, other)
```

- Define **+ operator** between two Rabbit instances
  - For example:
    `r4 = r1 + r2`
    `r1` and `r2` are `Rabbit` instances, combine to create `r4`
- `r4` is a new `Rabbit` instance with age 0
- `r4` has `self` as one parent and `other` as the other parent
- In `__init__`, `parent1` and `parent2` are of type `Rabbit`

Special Method to Compare Two Rabbits

- Decide that two rabbits are equal if they have the **same two parents**

```
1 def __eq__(self, other):
2     parents_same = (self.p1.rid == other.p1.rid and
3                     self.p2.rid == other.p2.rid)
4     parents_opp  = (self.p2.rid == other.p1.rid and
5                     self.p1.rid == other.p2.rid)
6     return parents_same or parents_opp
```

Special Method to Compare Two Rabbits

- Decide that two rabbits are equal if they have the **same two parents**

```
1 def __eq__(self, other):
2     parents_same = (self.p1.rid == other.p1.rid and
3                     self.p2.rid == other.p2.rid)
4     parents_opp  = (self.p2.rid == other.p1.rid and
5                     self.p1.rid == other.p2.rid)
6     return parents_same or parents_opp
```

Booleans checking
r1 + r2 or
r2 + r1

Special Method to Compare Two Rabbits

- Decide that two rabbits are equal if they have the **same two parents**

```
def __eq__(self, other):
    parents_same = (self.p1.rid == other.p1.rid and
                    self.p2.rid == other.p2.rid)
    parents_opp  = (self.p2.rid == other.p1.rid and
                    self.p1.rid == other.p2.rid)
    return parents_same or parents_opp
```

- Compare ids of parents since **ids are unique** *(due to class var)*

Special Method to Compare Two Rabbits

- Decide that two rabbits are equal if they have the **same two parents**

```
def __eq__(self, other):
    parents_same = (self.p1.rid == other.p1.rid and
                    self.p2.rid == other.p2.rid)
    parents_opp  = (self.p2.rid == other.p1.rid and
                    self.p1.rid == other.p2.rid)
    return parents_same or parents_opp
```

- Compare ids of parents since **ids are unique** *(due to class var)*
- **Note:** you **CAN'T** compare objects directly *(recursive if `__eq__`)*
  Also, can't call on `None` *(`AttributeError` when `None.parent1`)*

Big Idea

Class Variables are **shared** between all instances

If one instance changes it, it's changed for every instance.

You Try! — Predict the Output

```
r1 = Rabbit(3)
r1.set_name("flopsy")
r2 = Rabbit(5)
r2.set_name("mopsy")
print(r1.get_rid())
print(r2.get_rid())

r3 = r1 + r2
print(r3.age)
print(r3.get_rid())
print(r3.get_parent1().get_name())
```

You Try! — Predict the Output

```
1  r1 = Rabbit(3)
2  r1.set_name("flopsy")
3  r2 = Rabbit(5)
4  r2.set_name("mopsy")
5  print(r1.get_rid())    00001
6  print(r2.get_rid())
7
8  r3 = r1 + r2
9  print(r3.age)
10 print(r3.get_rid())
11 print(r3.get_parent1().get_name())
```

tag starts at 1, r1.rid = 1
get_rid() zero-fills to 5 digits

You Try! — Predict the Output

```
1  r1 = Rabbit(3)
2  r1.set_name("flopsy")
3  r2 = Rabbit(5)
4  r2.set_name("mopsy")
5  print(r1.get_rid())    00001
6  print(r2.get_rid())    00002
7
8  r3 = r1 + r2
9  print(r3.age)
10 print(r3.get_rid())
11 print(r3.get_parent1().get_name())
```

You Try! — Predict the Output

```
1  r1 = Rabbit(3)
2  r1.set_name("flopsy")
3  r2 = Rabbit(5)
4  r2.set_name("mopsy")
5  print(r1.get_rid())    00001
6  print(r2.get_rid())    00002
7
8  r3 = r1 + r2
9  print(r3.age)           0
10 print(r3.get_rid())
11 print(r3.get_parent1().get_name())
```

r1 + r2 calls Rabbit(0, r1, r2)
new rabbit has age 0

You Try! — Predict the Output

```
1  r1 = Rabbit(3)
2  r1.set_name("flopsy")
3  r2 = Rabbit(5)
4  r2.set_name("mopsy")
5  print(r1.get_rid())    00001
6  print(r2.get_rid())    00002
7
8  r3 = r1 + r2
9  print(r3.age)           0
10 print(r3.get_rid())    00003
11 print(r3.get_parent1().get_name())
```

You Try! — Predict the Output

```
1  r1 = Rabbit(3)
2  r1.set_name("flopsy")
3  r2 = Rabbit(5)
4  r2.set_name("mopsy")
5  print(r1.get_rid())    00001
6  print(r2.get_rid())    00002
7
8  r3 = r1 + r2
9  print(r3.age)           0
10 print(r3.get_rid())    00003
11 print(r3.get_parent1().get_name()) flopsy
```

parent1 is r1, whose name was set to "flopsy"

You Try! — Are They Equal?

```
r1 = Rabbit(3)
r2 = Rabbit(5)
r3 = Rabbit(2)

r4 = r1 + r2
r5 = r2 + r1
r6 = r1 + r3

print(r4 == r5)
print(r4 == r6)
```

You Try! — Are They Equal?

```
r1 = Rabbit(3)
r2 = Rabbit(5)
r3 = Rabbit(2)

r4 = r1 + r2    # parent1=r1, parent2=r2
r5 = r2 + r1    # parent1=r2, parent2=r1
r6 = r1 + r3    # parent1=r1, parent2=r3

print(r4 == r5)
print(r4 == r6)
```

You Try! — Are They Equal?

```
1  r1 = Rabbit(3)
2  r2 = Rabbit(5)
3  r3 = Rabbit(2)
4
5  r4 = r1 + r2    # parent1=r1, parent2=r2
6  r5 = r2 + r1    # parent1=r2, parent2=r1
7  r6 = r1 + r3    # parent1=r1, parent2=r3
8
9  print(r4 == r5) True
10 print(r4 == r6)
```

Same parents, opposite order
parents_opp is True

You Try! — Are They Equal?

```
1  r1 = Rabbit(3)
2  r2 = Rabbit(5)
3  r3 = Rabbit(2)
4
5  r4 = r1 + r2    # parent1=r1, parent2=r2
6  r5 = r2 + r1    # parent1=r2, parent2=r1
7  r6 = r1 + r3    # parent1=r1, parent2=r3
8
9  print(r4 == r5) True
10 print(r4 == r6) False
```

r4: parents r1, r2
r6: parents r1, r3
Different parents → False

Summary

Questions?