

Inheritance

- When we make a new class - we can reuse an existing class and **inherit** all the capabilities of an existing class and then add our own little bit to make our new class
- Another form of store and reuse
- Write once - reuse many times
- The new class (child) has all the capabilities of the old class (parent) - and then some more

Terminology: Inheritance



‘Subclasses’ are more specialized versions of a class, which **inherit** attributes and behaviors from their parent classes, and can introduce their own.

http://en.wikipedia.org/wiki/Object-oriented_programming

```
class PartyAnimal:

    def __init__(self, nam):
        self.x = 0
        self.name = nam
        print(self.name, "constructed")

    def party(self) :
        self.x = self.x + 1
        print(self.name, "party count", self.x)

class FootballFan(PartyAnimal) :

    def __init__(self, nam) :
        super().__init__(nam)
        self.points = 0

    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "points", self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

FootballFan is a class which extends PartyAnimal. It has all the capabilities of PartyAnimal and more.

party7.py

```
class PartyAnimal:

    def __init__(self, nam):
        self.x = 0
        self.name = nam
        print(self.name, "constructed")

    def party(self) :
        self.x = self.x + 1
        print(self.name, "party count", self.x)
```

```
class FootballFan(PartyAnimal):

    def __init__(self, nam) :
        super().__init__(nam)
        self.points = 0

    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "points", self.points)
```

→ `s = PartyAnimal("Sally")`
`s.party()`

`j = FootballFan("Jim")`
`j.party()`
`j.touchdown()`

S

x: 0

name: Sally

```
class PartyAnimal:

    def __init__(self, nam):
        self.x = 0
        self.name = nam
        print(self.name, "constructed")

    def party(self) :
        self.x = self.x + 1
        print(self.name, "party count", self.x)

class FootballFan(PartyAnimal):

    def __init__(self, nam) :
        super().__init__(nam)
        self.points = 0

    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "points", self.points)
```

→

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

S

x: 1

name: Sally

```
class PartyAnimal:

    def __init__(self, nam):
        self.x = 0
        self.name = nam
        print(self.name, "constructed")

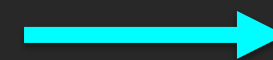
    def party(self) :
        self.x = self.x + 1
        print(self.name, "party count", self.x)

class FootballFan(PartyAnimal):

    def __init__(self, nam) :
        super().__init__(nam)
        self.points = 0

    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "points", self.points)
```

```
s = PartyAnimal("Sally")
s.party()
```



```
j = FootballFan("Jim")
j.party()
j.touchdown()
```

j

x: 0

name: Jim

points: 0

```
class PartyAnimal:

    def __init__(self, nam):
        self.x = 0
        self.name = nam
        print(self.name, "constructed")

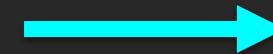
    def party(self) :
        self.x = self.x + 1
        print(self.name, "party count", self.x)

class FootballFan(PartyAnimal):

    def __init__(self, nam) :
        super().__init__(nam)
        self.points = 0

    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "points", self.points)
```

```
s = PartyAnimal("Sally")
s.party()
```



```
j = FootballFan("Jim")
j.party()
j.touchdown()
```

j

x: 1

name: Jim

points: 0

```
class PartyAnimal:

    def __init__(self, nam):
        self.x = 0
        self.name = nam
        print(self.name, "constructed")

    def party(self) :
        self.x = self.x + 1
        print(self.name, "party count", self.x)

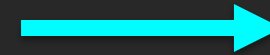
class FootballFan(PartyAnimal):

    def __init__(self, nam) :
        super().__init__(nam)
        self.points = 0

    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "points", self.points)
```

```
s = PartyAnimal("Sally")
s.party()
```

```
j = FootballFan("Jim")
j.party()
j.touchdown()
```



j

x: 1

name: Jim

points: 7

Definitions

Class - a template

Attribute – A variable within a class

Method - A function within a class

Object - A particular instance of a class

Constructor – Code that runs when an object is created

Inheritance - The ability to extend a class to make a new class.



Summary

- Object Oriented programming is a very structured approach to code reuse.
- We can group data and functionality together and create many independent instances of a class



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors here

Additional Source Information

- "Snowman Cookie Cutter" by Didriks is licensed under CC BY

<https://www.flickr.com/photos/dinnerseries/23570475099>

- Photo from the television program *Lassie*. Lassie watches as Jeff (Tommy Rettig) works on his bike is Public Domain

https://en.wikipedia.org/wiki/Lassie#/media/File:Lassie_and_Tommy_Rettig_1956.JPG