# CISS450: Artificial Intelligence
# Lecture 18: Class Inheritance

Yihsiang Liow

# Agenda

- Study class inheritance

# Example

```
class C:
    def __init__(self, x):
        self.__x = x
    def get_x(self):
        return self.__x

class D(C):
    def __init__(self, a, x):
        C.__init__(self, x)
        self.__a = a
    def get_a(self):
        return self.__a

d = D(1,2)
print(d.get_a())
print(d.get_x())
```

Calling superclass constructor

D objects inherits get_x method

# Why?

- Inheritance is useful. For instance if you have two classes sharing similar attributes or methods, then you can create a super class containing the common parts, remove the common parts from the subclasses
- This improves maintainability of code
- Here's a simple example ...

# Why?

```
class Employee:
    def __init__(self, firstname, lastname):
        self.__firstname = firstname
        self.__lastname = lastname
    def salary(self, time):
        return time * 100

class Manager(Employee):
    def __init__(self, firstname, lastname, extra_pay):
        Employee.__init__(self, firstname, lastname)
        self.__extra_pay = extra_pay
    def salary(self, time):
        return Employee.salary(self, time) + self.__extra_pay

x = Manager("John","Doe",1500)
print(x.salary(40))
```

Calling superclass method

# Hiding Superclass Parts

```
class C:
    def __init__(self, x):
        self.__x = x
    def get_x(self):
        return self.__x
    def get_a(self):
        return None

class D(C):
    def __init__(self, a, x):
        C.__init__(self, x)
        self.__a = a
    def get_a(self):
        return self.__a

d = D(1,2)
print(d.get_a())
print(d.get_x())
```

Same name!
Not a syntax error.
But how does **d** invokes **get_a** of **C**??

# Multiple Inheritance

- Python supports multiple inheritance, i.e., a class can have more than one superclass.

```
class C0:
    def __init__(self,a): self.__a = a
    def get_a(self): return self.__a

class C1:
    def __init__(self,b): self.__b = b
    def get_b(self): return self.__b

class D(C0,C1):
    def __init__(self,a,b,c):
        C0.__init__(self,a)
        C1.__init__(self,b)
        self.__c = c
    def get_c(self): return self.__c

d = D(0,1,2)
print(d.get_a(), d.get_b(), d.get_c())
```

Inherits from C0 and C1

# Multiple Inheritance

- Does anyone see an ambiguity problem with multiple inheritance?

# super()

- Recall that you can call a method in the parent.
- You have a mention the parent class explicitly.
- See next slide ...

# super()

```
class C:
    def __init__(self, x):
        self.x = x
    def f(self):
        return "f %s" % self.x
class D(C):
    def __init__(self, x, y):
        C.__init__(self, x)
        self.y = y
    def f(self):
        s = C.f(self)
        return "%s ... g %s" % (s, self.y)

d = D(2,'b')
print(d.f())
```

# super

- You can use super so that you do not have to mention the parent class.
- See next slide ...

# super

```
class C:
    def __init__(self, x):
        self.x = x
    def f(self):
        return "f %s" % self.x
class D(C):
    def __init__(self, x, y):
        super(D, self).__init__(x)
        self.y = y
    def f(self):
        s = super(D, self).f()
        return "%s ... g %s" % (s, self.y)

d = D(2,'b')
print(d.f())
```