University of Tripoli
Faculty of Information Technology

Department of Software Engineering

ITSE305 مواضيع مختارة
Python Programming
S2025

Lecture (6): Python Classes and Objects

Python Classes/Objects

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.

Class

- To **create** a class, use the keyword class
- class definitions cannot be empty.
- To have a class definition with no content, put in the pass statement to avoid getting an error.

Class

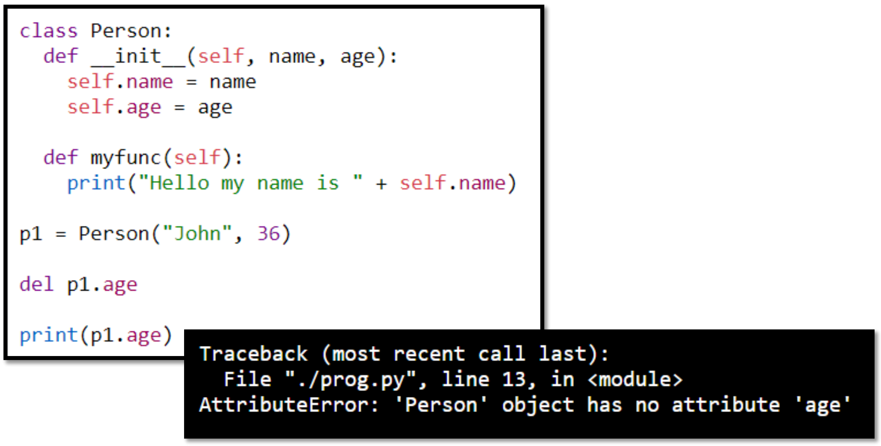
- **Modify** Object Properties

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

p1.age = 40

print(p1.age)
```

```
40
```

Class

- **<u>Delete</u>** Object Properties by using the del keyword

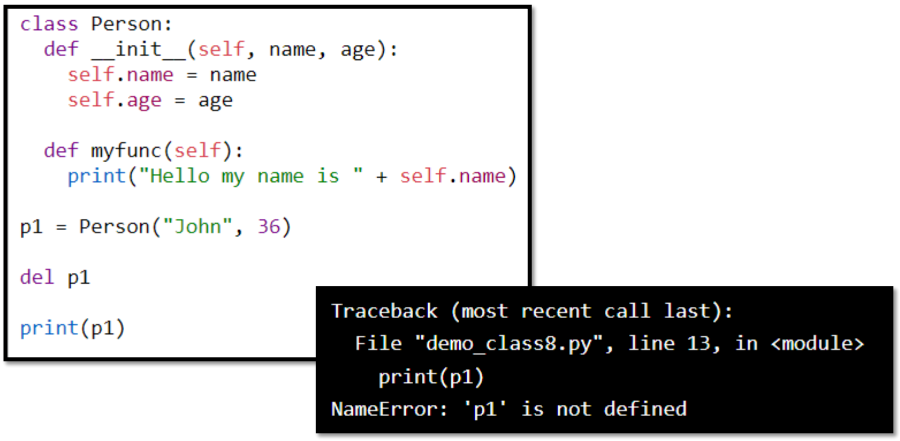


```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1.age

print(p1.age)
```

```
Traceback (most recent call last):
  File "./prog.py", line 13, in <module>
AttributeError: 'Person' object has no attribute 'age'
```

Class

- Delete Objects by using the del keyword:

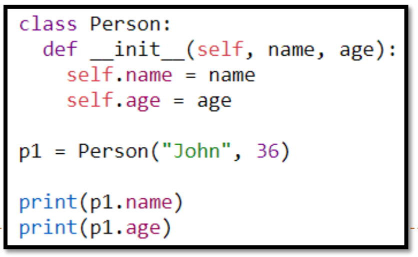


```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1

print(p1)
```

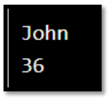
```
Traceback (most recent call last):
  File "demo_class8.py", line 13, in <module>
    print(p1)
NameError: 'p1' is not defined
```

The __init__() Function

- built-in _ _init_ _() function.
- All classes have a function called _ _init_ _(), which is always executed when the class is being initiated.
- Use the _ _init_ _() function to assign values to object properties, or other operations that are necessary to do when the object is being created:
- The _ _init_ _() function is called automatically every time the class is being used to create a new object.



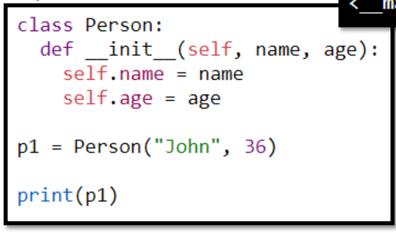
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```



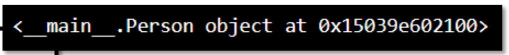
```
John
36
```

The __str__() Function

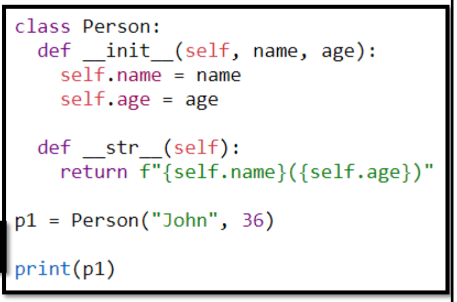
- The _ _str_ _() function controls what should be returned when the class object is represented as a string.
- If the _ _str_ _() function is not set, the string representation of the object is returned:



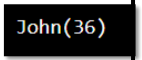
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1)
```



```
<__main__.Person object at 0x15039e602100>
```



```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```



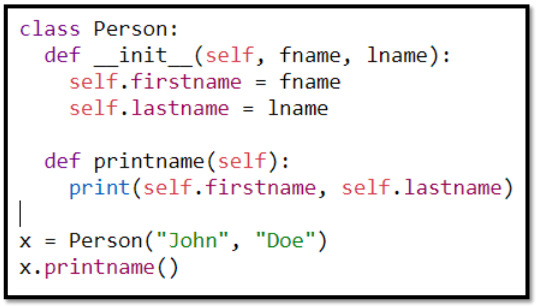
```
John(36)
```

Python Inheritance

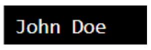
- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.

Create a Parent Class

- Any class can be a parent class, so the syntax is the same as creating any other class:



```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

x = Person("John", "Doe")
x.printname()
```



```
John Doe
```

Create a Child Class

- To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Add the __init__() Function

- When you add the _ _init_ _() function, the child class will no longer inherit the parent's _ _init_ _() function.
- The child's _ _init_ _() function **overrides** the inheritance of the parent's _ _init_ _() function.
- To keep the inheritance of the parent's _ _init_ _() function, add a call to the parent's _ _init_ _() function:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Use the super() Function

- Python also has a super() function that will make the child class inherit all the methods and properties from its parent
- By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

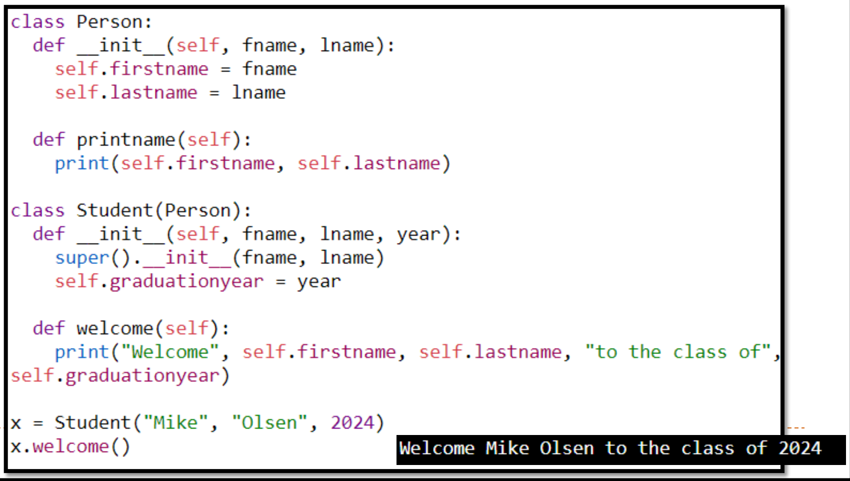
Add Properties

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
print(x.graduationyear)
```

```
2019
```

Add Methods

- If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.

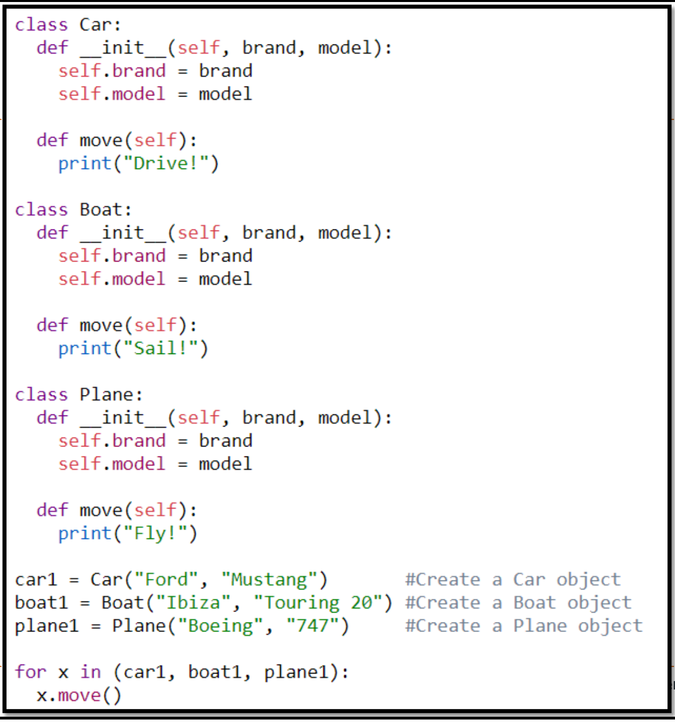


```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)

x = Student("Mike", "Olsen", 2024)
x.welcome()
```

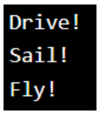
```
Welcome Mike Olsen to the class of 2024
```

Python Polymorphism

- The word "polymorphism" means "many forms“
- in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.
- An example of a Python function that can be used on different objects is the len() function, which returns:
  - the number of characters for strings
  - he number of items in the tuple for tuples
  - the number of key/value pairs in the dictionary for dictionaries
- Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.



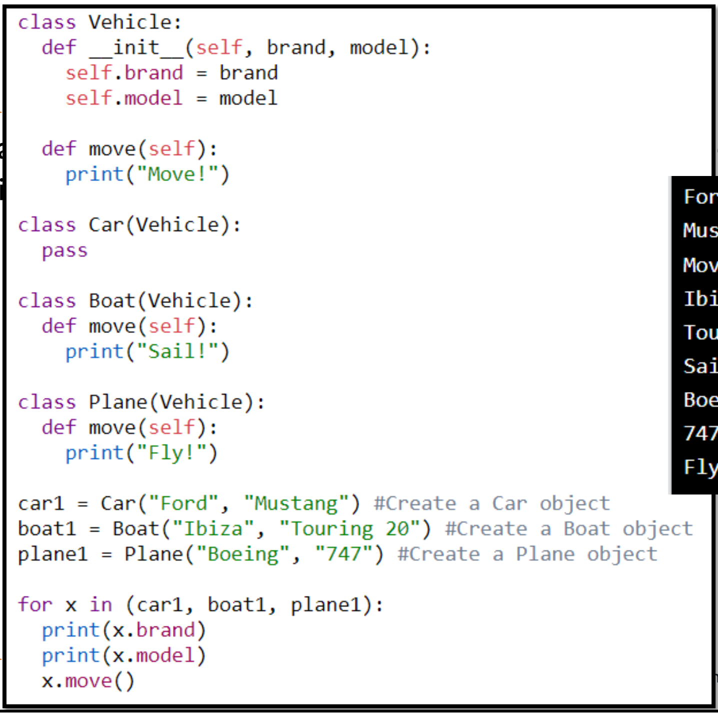
```python
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")

class Plane:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang")       #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747")     #Create a Plane object

for x in (car1, boat1, plane1):
  x.move()
```



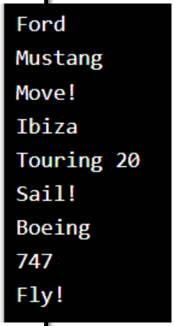
```
Drive!
Sail!
Fly!
```

Ca es

wi



```python
class Vehicle:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Move!")

class Car(Vehicle):
  pass

class Boat(Vehicle):
  def move(self):
    print("Sail!")

class Plane(Vehicle):
  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang") #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747") #Create a Plane object

for x in (car1, boat1, plane1):
  print(x.brand)
  print(x.model)
  x.move()
```



```
Ford
Mustang
Move!
Ibiza
Touring 20
Sail!
Boeing
747
Fly!
```

The END