Course	
Term	
Week	
Date	
Chapter. Topic	11. Inheritance

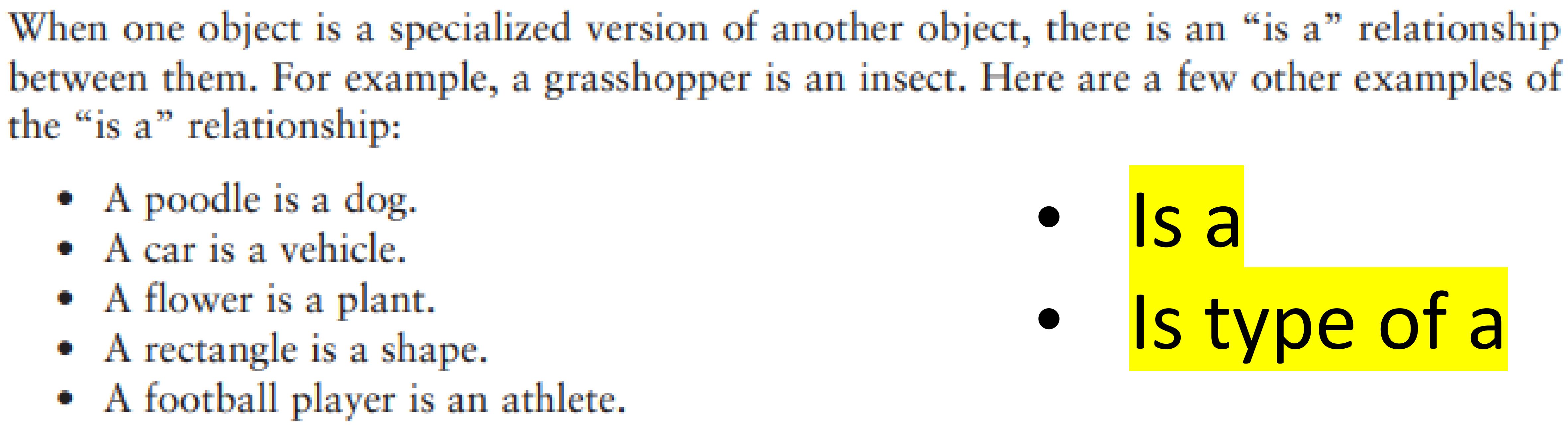
Inheritance

Siva R Jasthi

Computer Science and Cybersecurity

Metropolitan State University

Inheritance and the “Is a” Relationship



When one object is a specialized version of another object, there is an “is a” relationship between them. For example, a grasshopper is an insect. Here are a few other examples of the “is a” relationship:

- A poodle is a dog.
- A car is a vehicle.
- A flower is a plant.
- A rectangle is a shape.
- A football player is an athlete.

- Is a
- Is type of a

Inheritance and the “**Is a**” Relationship

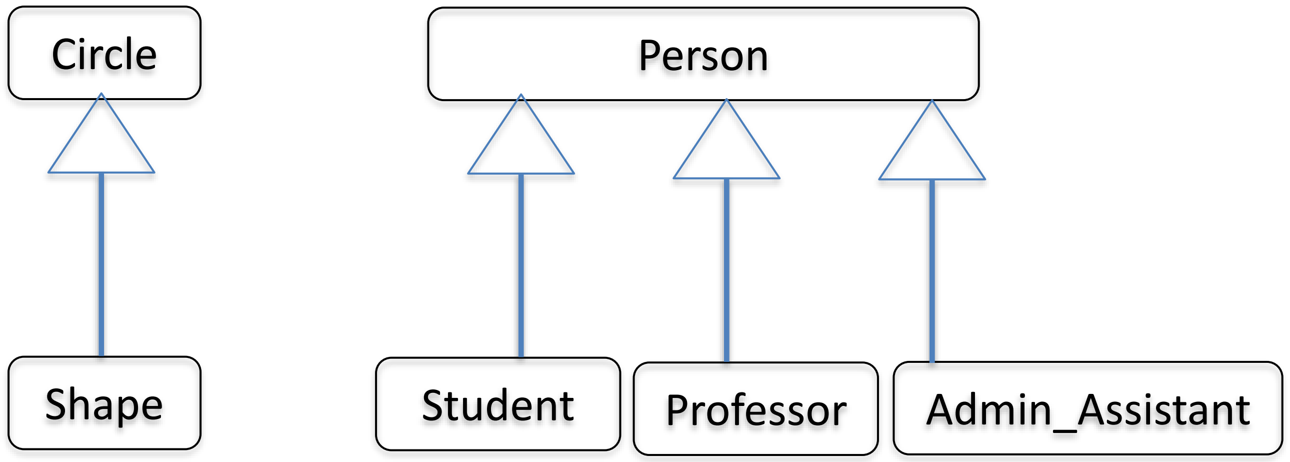
-

Circle is a Shape

Student is a Person
Professor is a Person
Admin_Assistant is a Person

ipod is a MusicPlayer
SamsungMusicPlayer is a MusicPlayer

Computer is an Elephant (doesn’t make sense)



Inheritance

-

Inheritance allows us to define a class that inherits all the methods and properties from another class.

The parent class is the class being inherited from, also called the **base** class.

A child class is a class that inherits from another class, also called a **derived** class.

Parent class = super class, base class, ancestor class
Child class = sub class, derived class, descendant class

Revisiting objects and classes (link)

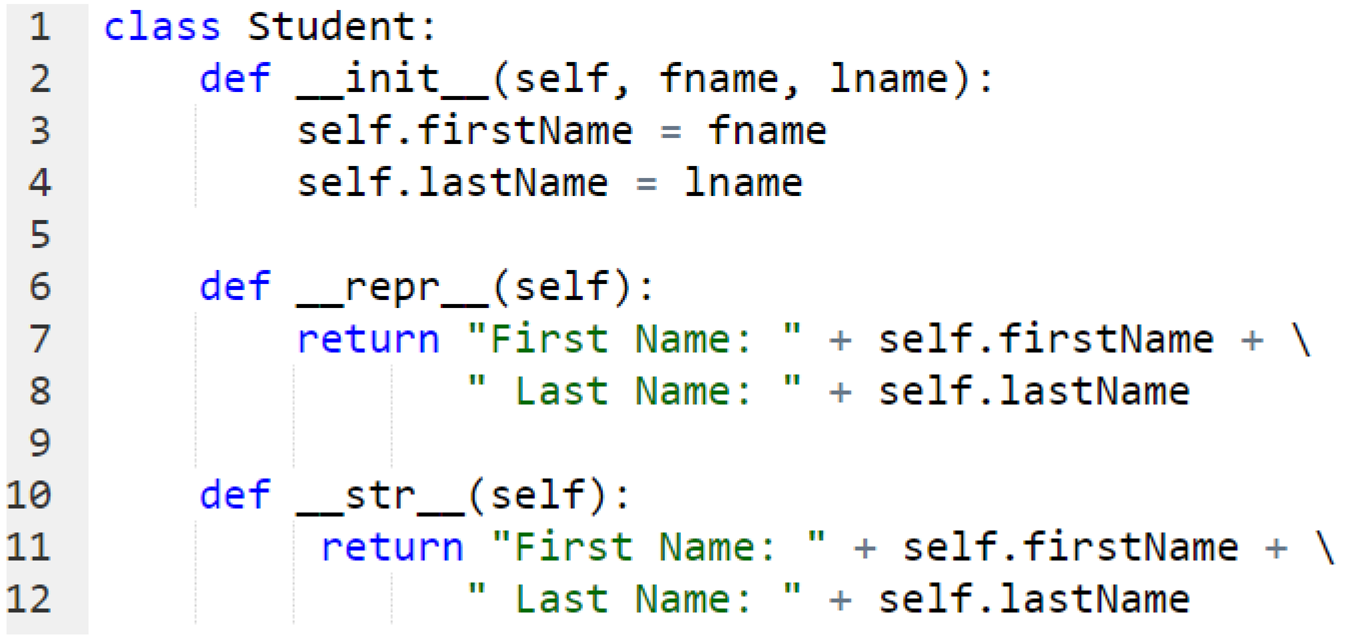
-

You define one class.

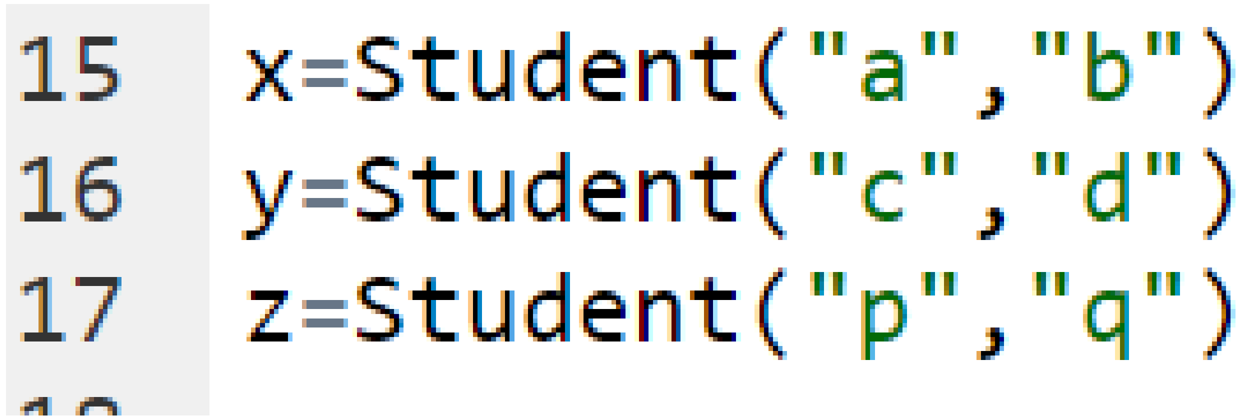
And you create many instances of the same class.

How many classes are defined here?

How many instances are defined here?



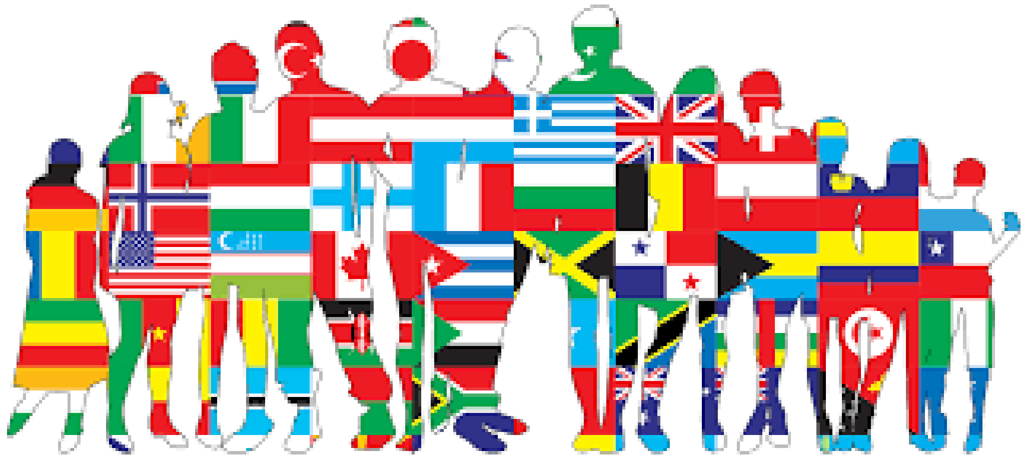
```
class Student:
    def __init__(self, fname, lname):
        self.firstName = fname
        self.lastName = lname

    def __repr__(self):
        return "First Name: " + self.firstName + \
               " Last Name: " + self.lastName

    def __str__(self):
         return "First Name: " + self.firstName + \
               " Last Name: " + self.lastName
```



```
x=Student("a","b")
y=Student("c","d")
z=Student("p","q")
```

Student and International Student

Suppose we want to define another class called "InternationalStudent".



International Student **is also a** Student.

However, International Student has two additional attributes.

- country
- language

What are our options for defining "InternationalStudent" class?

Without Inheritance

Class Student:

    id, name, email

Class InternationalStudent:

    id, name, email, country, language

InternationalStudent
is a (is a type of a)
Student

Without inheritance, you will be duplicating all the variables and all the methods in different classes.

This leads to ‘**code maintenance nightmare**’. If you need to fix any issue, you need to fix in all the classes.

And another biggest drawback is .. There is no relationship between Student and InternationalStudent classes.

With Inheritance

-

```
Class Student:
     id, name, email

Class InternationalStudent(Student):
     country, language
```

With inheritance, you inherit all the variables and behavior of the parent.
It is not needed to duplicate the code in the child class.

If you need to fix any issue, you need to fix in either parent class or child class.

And we are also establishing parent-child relationship (hierarchy)

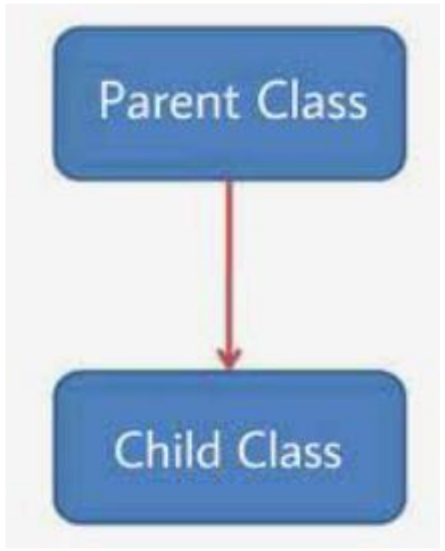
Inheritance Syntax

-

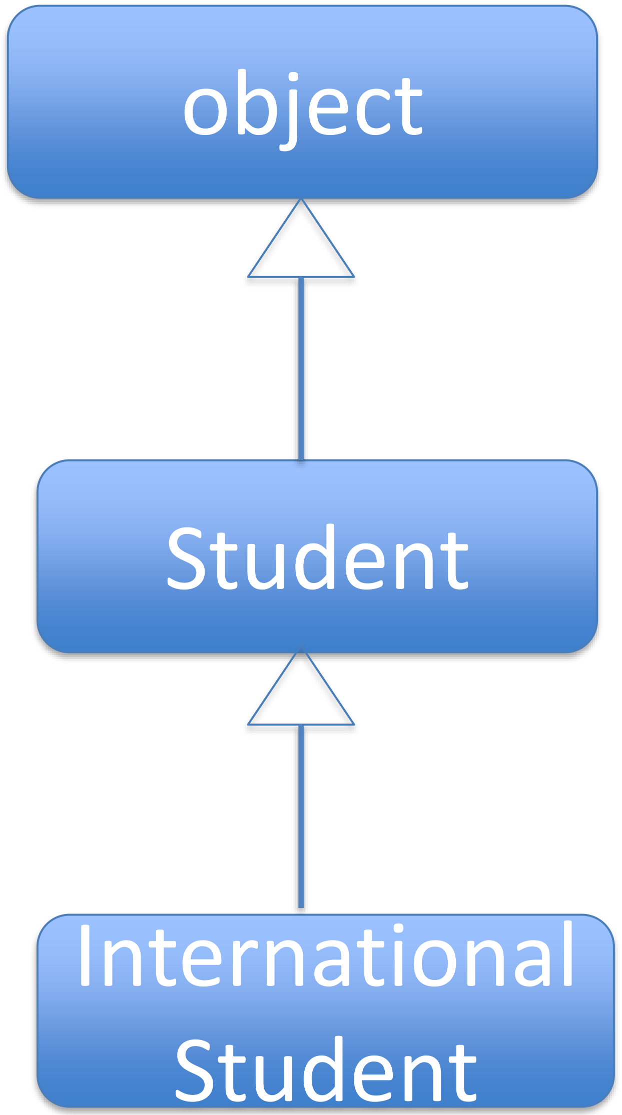
class ChildClassName(ParentClassName)

Here are some examples

- class InternationalStudent(Student)
- class CryptoQuote(Quote)
- class ScrambledQuote(Quote)



https://www.w3schools.com/python/python_inheritance.asp



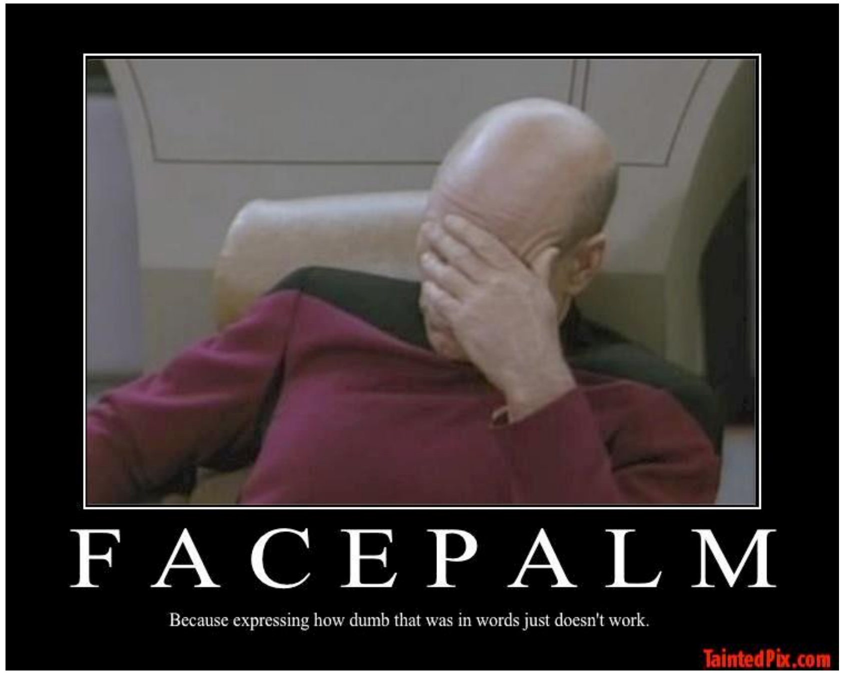
Root of all classes is called “object”

By default, all classes in python inherit from a root class called “object”.

And we create objects (instances) of these classes.

It can be confusing. Keep the following in context.

- class: a class you are defining
- object: the root class of python
- instance: specific instances you are creating


FACEPALM
Because expressing how dumb that was in words just doesn't work.
TaintedPix.com

Really..? Does my class inherit from “object”?

Oh yes!

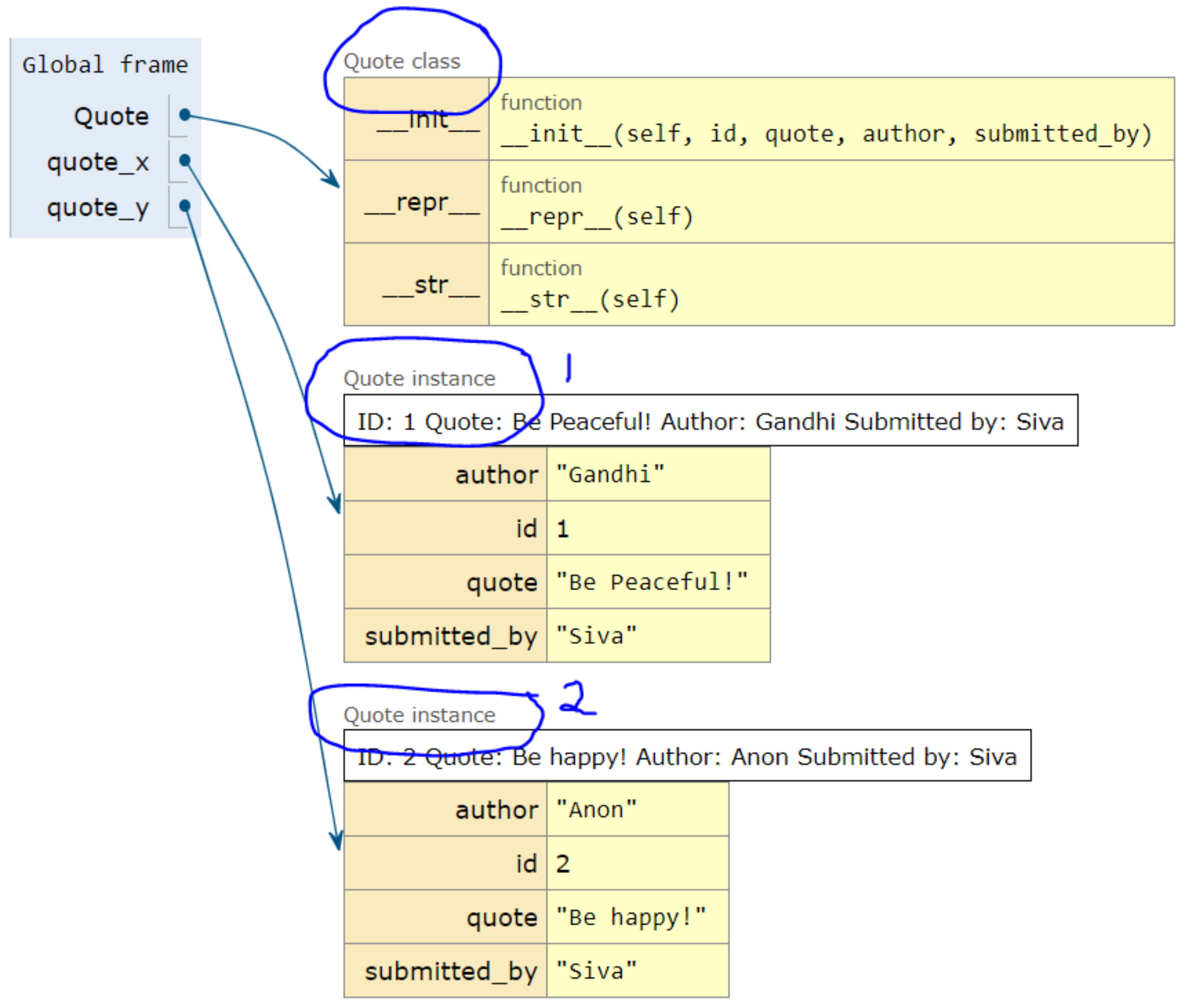
class Quote:

is equivalent to

class Quote(object):

Let us try this out on pythontutor ([link]())

Key Point: Since all python classes inherit from “object” by default, we do not need to be explicit about it.



Global frame

Quote	●
quote_x	●
quote_y	●

Quote class

__init__	function `__init__(self, id, quote, author, submitted_by)`
__repr__	function `__repr__(self)`
__str__	function `__str__(self)`

Quote instance

ID: 1 Quote: Be Peaceful! Author: Gandhi Submitted by: Siva

author	"Gandhi"
id	1
quote	"Be Peaceful!"
submitted_by	"Siva"

Quote instance

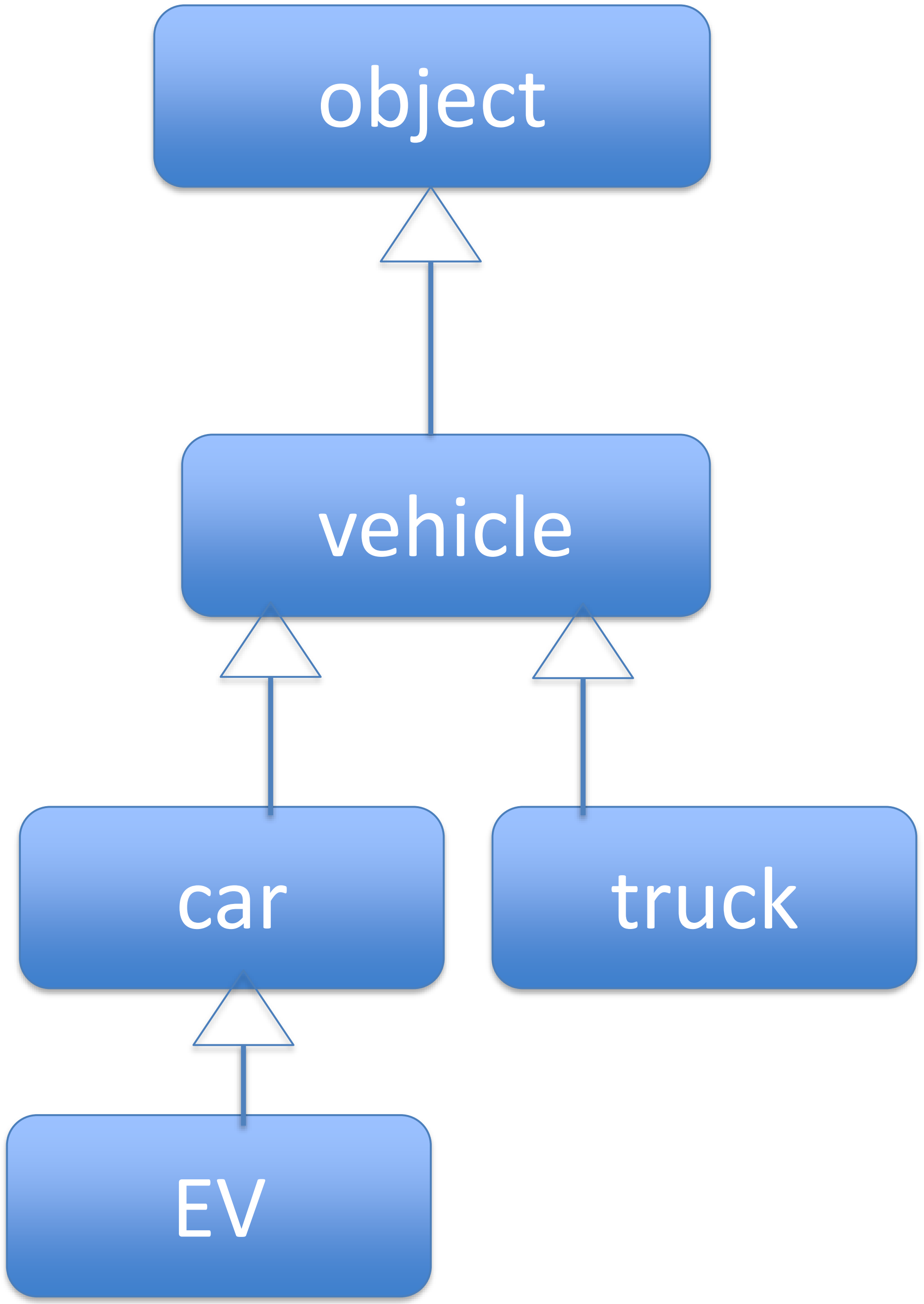
ID: 2 Quote: Be happy! Author: Anon Submitted by: Siva

author	"Anon"
id	2
quote	"Be happy!"
submitted_by	"Siva"

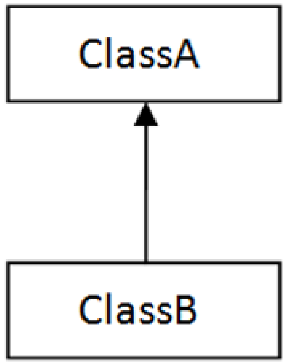
Inheritance Syntax

-

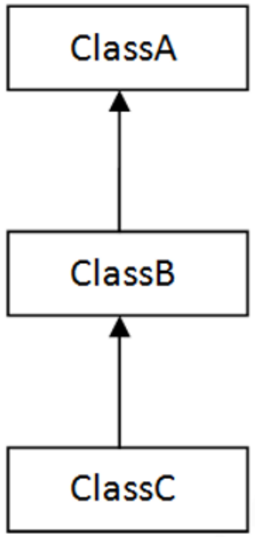
```
Vehicle
car(vehicle)
truck (vehicle)
EV (car)
```



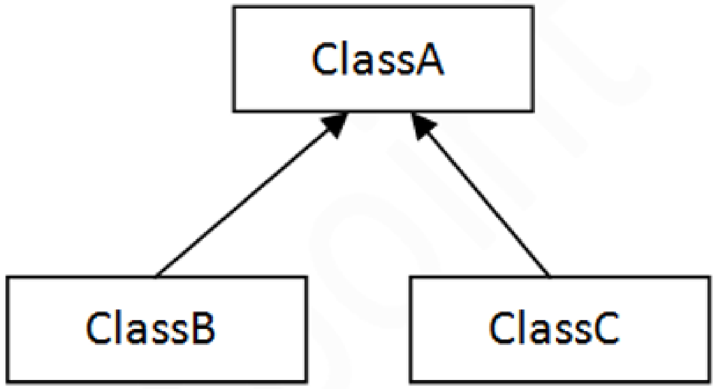
Hierarchy:



Single-Level Inheritance
(Parent-Child)

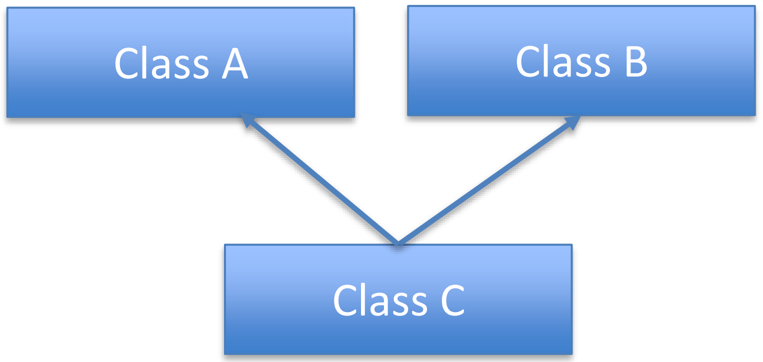


Multi-Level Inheritance
(Grand Parent - Parent- Child)



Multiple sub-classes

Many classes can inherit from the same parent.



- Children, Parents, Grand Parents, Great Grand Parents and so on…
- And at the top of the hierarchy is our “object”

Hierarchy:

These two methods are useful in querying the hierarchy of the classes.

- isinstance(x, Y) tells you whether x is an instance of Y. Here, x is an instance and Y is a class.

- issubclass(X,Y) tell you whether x is a subclass of y. Both x and y must be classes

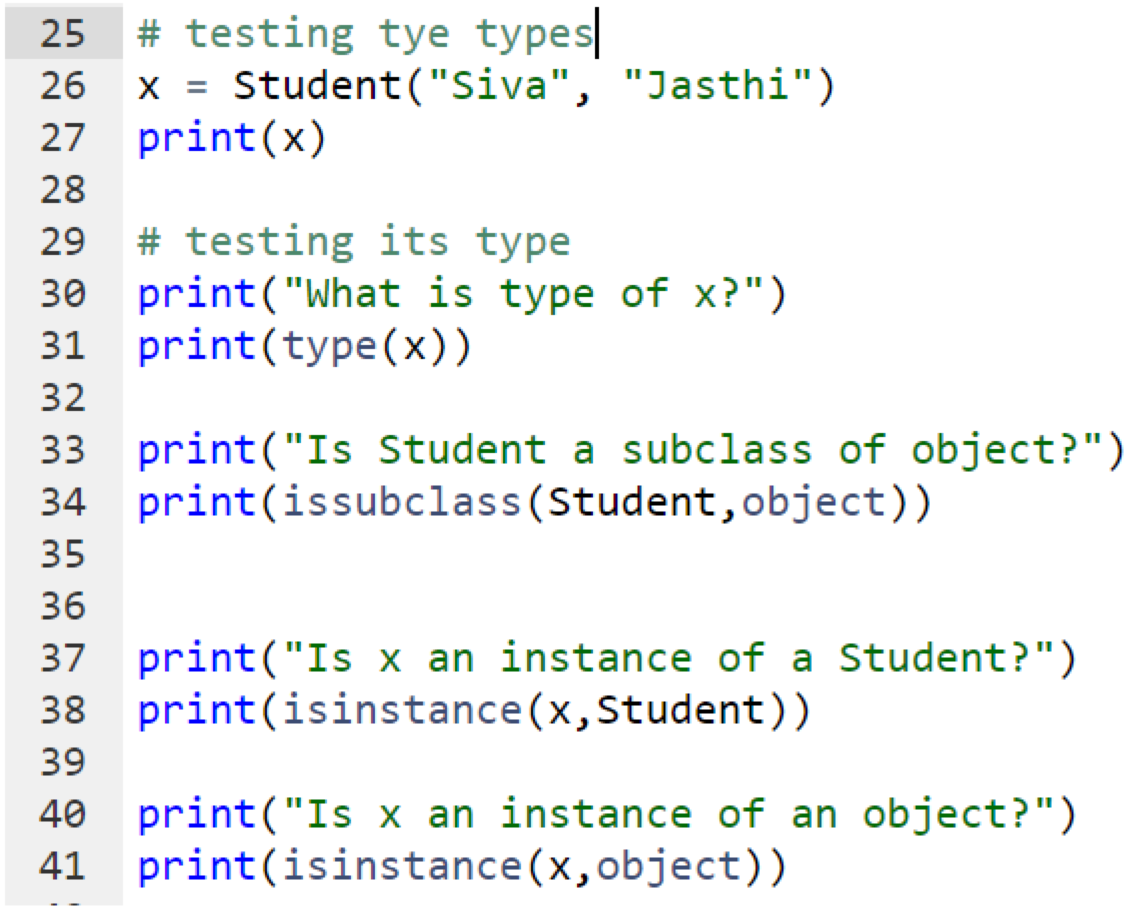
Let us revisit our example (link)

What is its type?

Is class x subclass of class y?

Is x instance of class Y?

Does x belong to this family?



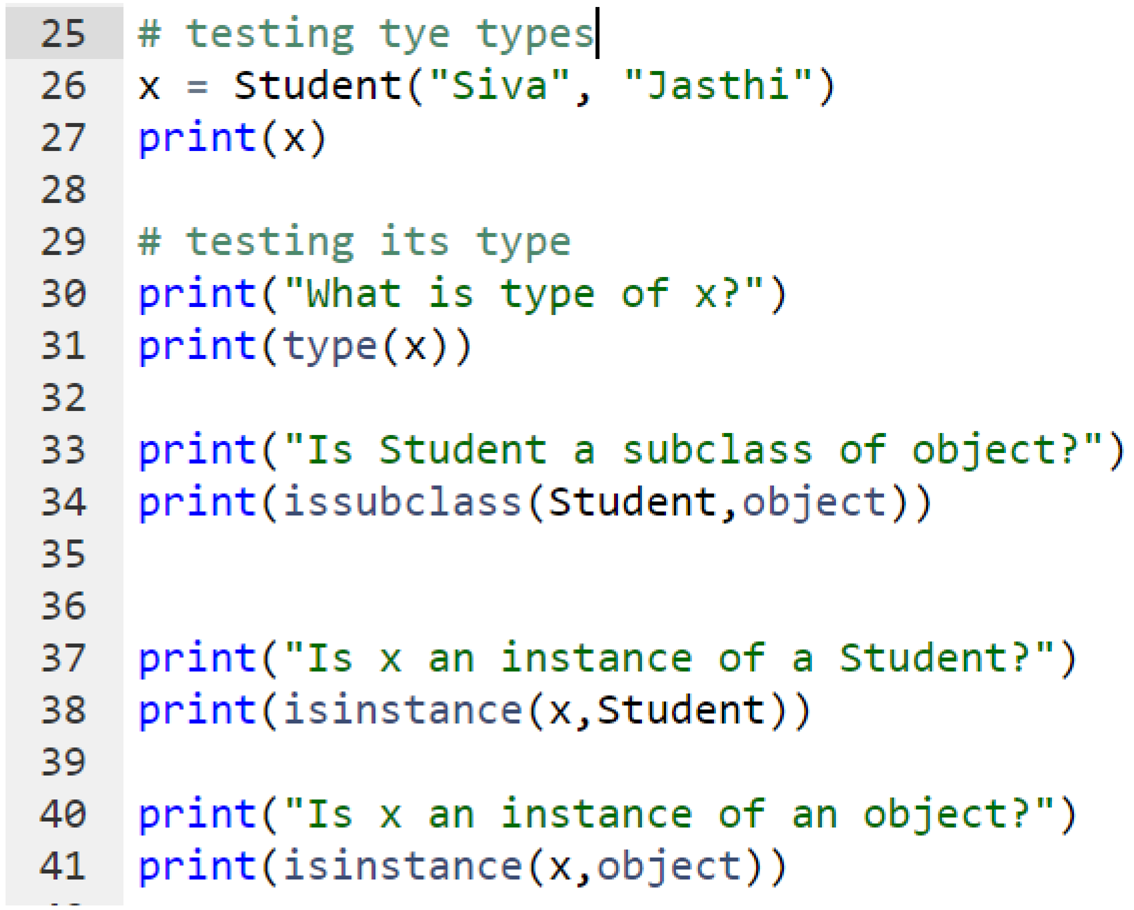
```
# testing tye types
x = Student("Siva", "Jasthi")
print(x)

# testing its type
print("What is type of x?")
print(type(x))

print("Is Student a subclass of object?")
print(issubclass(Student,object))


print("Is x an instance of a Student?")
print(isinstance(x,Student))

print("Is x an instance of an object?")
print(isinstance(x,object))
```

Useful methods for testing the “type” (link)

For knowing the type of an instance
type(instance_name)

For knowing whether x is a subclass of y
issubclass(SUBCLASS, PARENTCLASS)

For knowing whether an instance belongs to a class
isinstance(instance_name, PARENTCLASS)



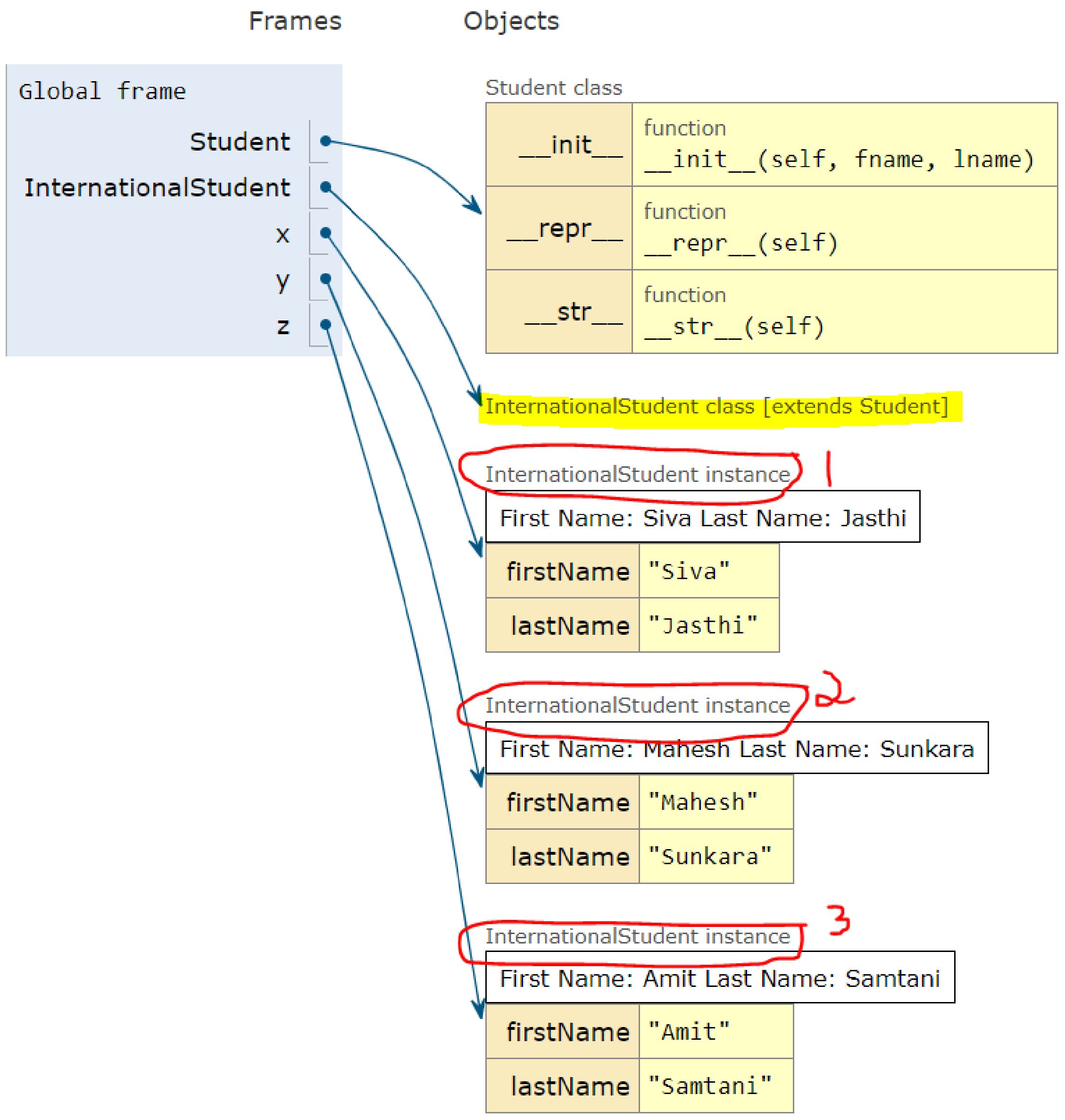
```
# testing tye types
x = Student("Siva", "Jasthi")
print(x)

# testing its type
print("What is type of x?")
print(type(x))

print("Is Student a subclass of object?")
print(issubclass(Student,object))


print("Is x an instance of a Student?")
print(isinstance(x,Student))

print("Is x an instance of an object?")
print(isinstance(x,object))
```

Object > Student > InternationalStudent (link)

Does my subclass really inherits the methods of the parent?

Let us try it out.

```python
class Student:
    def __init__(self, fname, lname):
        self.firstName = fname
        self.lastName = lname

    def __repr__(self):
        return "First Name: " + self.firstName + \
              " Last Name: " + self.lastName

    def __str__(self):
         return "First Name: " + self.firstName + \
              " Last Name: " + self.lastName


class InternationalStudent(Student):
    pass

# creating instances
x=InternationalStudent("Siva","Jasthi")
y=InternationalStudent("Mahesh","Sunkara")
z=InternationalStudent("Amit","Samtani")
```



Frames
Objects
Global frame
Student
InternationalStudent
x
y
z
Student class
__init__
function __init__(self, fname, lname)
__repr__
function __repr__(self)
__str__
function __str__(self)
InternationalStudent class [extends Student]
InternationalStudent instance
1
First Name: Siva Last Name: Jasthi
firstName "Siva"
lastName "Jasthi"
InternationalStudent instance
2
First Name: Mahesh Last Name: Sunkara
firstName "Mahesh"
lastName "Sunkara"
InternationalStudent instance
3
First Name: Amit Last Name: Samtani
firstName "Amit"
lastName "Samtani"

Object > Student > InternationalStudent (link)

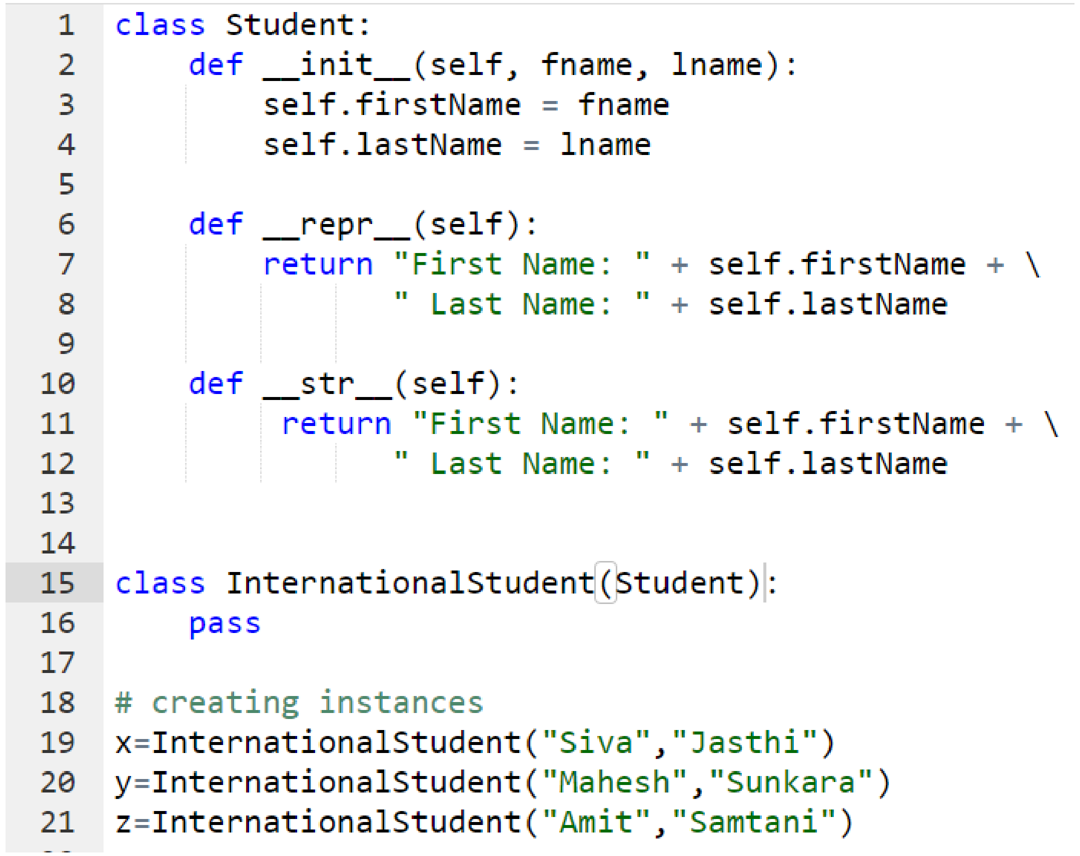
InternationalStudent extends from Student.

InternationalStudent inherits from Student.

But the real power comes when child classes:

[Case 1] Add additional attributes
[Case 2] Overwrite parent’s behavior
[Case 3] Add additional behavior (methods)

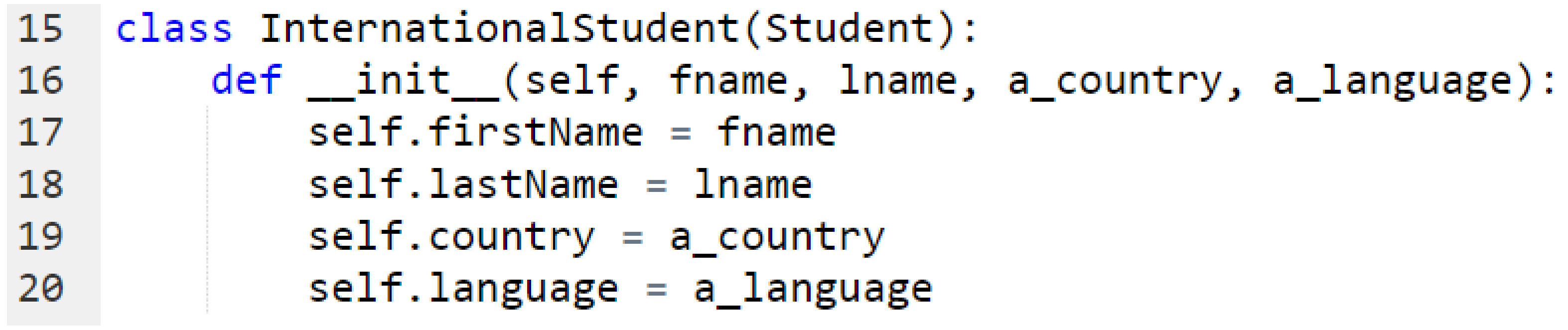
We will see these three examples.



```
class Student:
    def __init__(self, fname, lname):
        self.firstName = fname
        self.lastName = lname

    def __repr__(self):
        return "First Name: " + self.firstName + \
                " Last Name: " + self.lastName

    def __str__(self):
         return "First Name: " + self.firstName + \
                " Last Name: " + self.lastName


class InternationalStudent(Student):
    pass

# creating instances
x=InternationalStudent("Siva","Jasthi")
y=InternationalStudent("Mahesh","Sunkara")
z=InternationalStudent("Amit","Samtani")
```

InternationalStudent (Case 1) Link

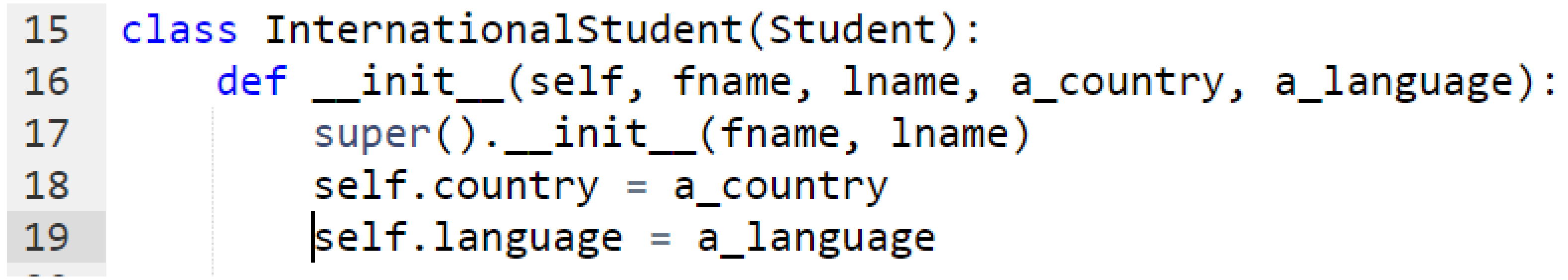
[Case 1] Add additional attributes (being explicit)
Assign each value to the attribute explicitly



```
class InternationalStudent(Student):
    def __init__(self, fname, lname, a_country, a_language):
        self.firstName = fname
        self.lastName = lname
        self.country = a_country
        self.language = a_language
```

Case 1.
Adding Additional Attributes

[Case 1] Add additional attributes (deferring to the parent)
Use “super()” to defer the assignment to the parent class.



```
class InternationalStudent(Student):
    def __init__(self, fname, lname, a_country, a_language):
        super().__init__(fname, lname)
        self.country = a_country
        self.language = a_language
```

InternationalStudent (Case 2) (see link below)

[Case 2] Overwriting the parent’s methods

In the previous code, we have a major problem.

Printing the InternationalStudent is NOT printing the country and language. What do we need to do here?

Overwrite the __str__ and __repr__ methods of the parents.

(In fact, we also overwrote __init__ method to support additional attributes)

Case 2. Overwriting Parent’s method

http://pythontutor.com/visualize.html#code=class%20Student%3A%0A%20%20%20%20def%20__init__%28self,%20fname,%20lname%29%3A%0A%20%20%20%20%20%20%20%20self.firstName%20%3D%20fname%0A%20%20%20%20%20%20%20%20self.lastName%20%3D%20lname
%0A%0A%20%20%20%20def%20__repr__%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20%22First%20Name%3A%20%22%20%2B%20self.firstName%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Last%20Name%3A%20%2
2%20%2B%20self.lastName%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20__str__%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20%22First%20Name%3A%20%22%20%2B%20self.firstName%20%2B%20%5C%0A%20
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Last%20Name%3A%20%22%20%2B%20self.lastName%0A%20%0A%20%0Aclass%20InternationalStudent%28Student%29%3A%0A%20%20%20%20def%20__init__%28self,%20fname,%20lname,%20a_country,%20a_lan
guage%29%3A%0A%20%20%20%20%20%20%20%20super%28%29.__init__%28fname,%20lname%29%0A%20%20%20%20%20%20%20%20self.country%20%3D%20a_country%0A%20%20%20%20%20%20%20%20self.language%20%3D%20a_language%0A%20%20%0A%20%20%20%20
def%20__repr__%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20%22First%20Name%3A%20%22%20%2B%20self.firstName%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Last%20Name%3A%20%22%20%2B%20self.lastNam
e%20%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Country%3A%20%22%20%2B%20self.country%20%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Language%3A%20%22%20%2B%20self.lang
uage%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20__str__%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20%22First%20Name%3A%20%22%20%2B%20self.firstName%20%2B%20%5C%0A%20%20%20%20%20%20%
20%20%20%20%20%20%20%20%20%20%22%20Last%20Name%3A%20%22%20%2B%20self.lastName%20%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Country%3A%20%22%20%2B%20self.country%20%20%2B%20%5C%0A%20%20%20%20%
20%20%20%20%20%20%20%20%20%20%20%22%20Language%3A%20%22%20%2B%20self.language%0A%20%20%20%20%0A%20%20%20%20%0A%23%20creating%20instances%0Ax%3DInternationalStudent%28%22Siva%22,%22Jasthi%22,%20%22India%22,%20%22Telugu%22%29%
0Ay%3DInternationalStudent%28%22Mahesh%22,%22Sunkara%22,%20%22India%22,%22Telugu%22%29%0Az%3DInternationalStudent%28%22Amit%22,%22Samtani%22,%20%22India%22,%20%22Marathi%22%29%0A%20%0A%23priting%20the%20instances%0Aprint%28x%29%0Apri
nt%28y%29%0Aprint%28z%29%0A&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false

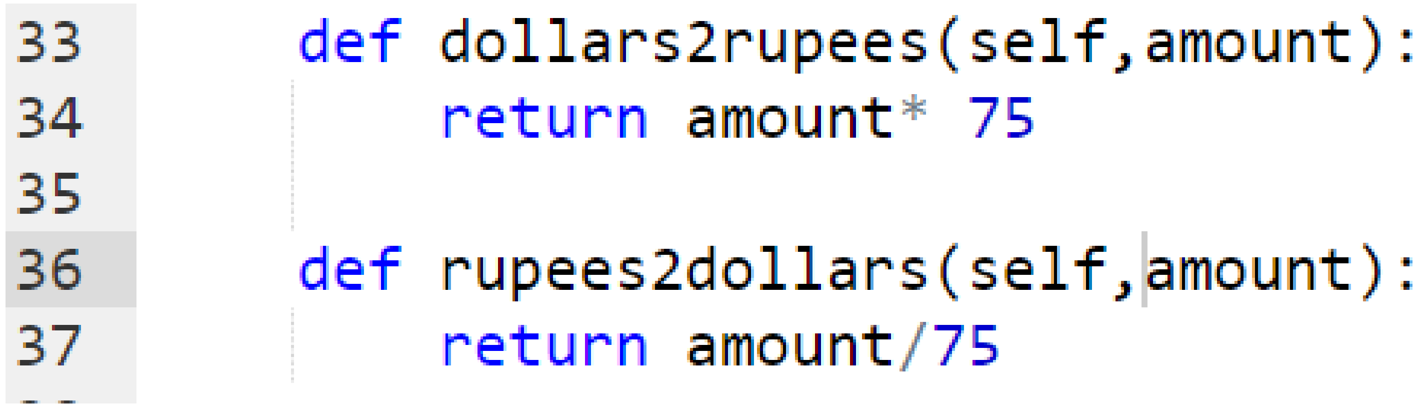
InternationalStudent (Case 3) (see link below)

[Case 3] Providing additional behavior / methods

Child classes can have their own methods.

Let us assume InternationalStudent can convert rupees to dollars and dollars to rupees.

Case 3.
Providing additional methods



```
    def dollars2rupees(self,amount):
        return amount* 75

    def rupees2dollars(self,amount):
        return amount/75
```

```
class Student:    def __init__(self, fname, lname):       self.firstName = fname       self.lastName = lname    def __repr__(self):       return "First Name: " + self.firstName + \            " Last Name: " + self.lastName              def __str__(self):       return "First Name: " + self.firstName + \
" Last Name: " + self.lastName  class InternationalStudent(Student):    def __init__(self, fname, lname, a_country, a_language):       super().__init__(fname, lname)       self.country = a_country       self.language = a_language     def __repr__(self):       return "First Name: " +
self.firstName + \            " Last Name: " + self.lastName  + \            " Country: " + self.country  + \            " Language: " + self.language              def __str__(self):       return "First Name: " + self.firstName + \            " Last Name: " + self.lastName  + \            " Country: " +
self.country  + \            " Language: " + self.language              def dollars2rupees(self,amout):       return amount * 75          def rupees2dollar(self,amout):       return amount/75          # creating instancesx=InternationalStudent("Siva","Jasthi", "India",
"Telugu")y=InternationalStudent("Mahesh","Sunkara", "India","Telugu")z=InternationalStudent("Amit","Samtani", "India", "Marathi") #priting the instancesprint(x)print(y)print(z)print(x.dollars2rupees(100))print(x.rupees2dollar(7500))
```

Who is powerful? Child or Parent

Case 1.
Adding Additional Attributes

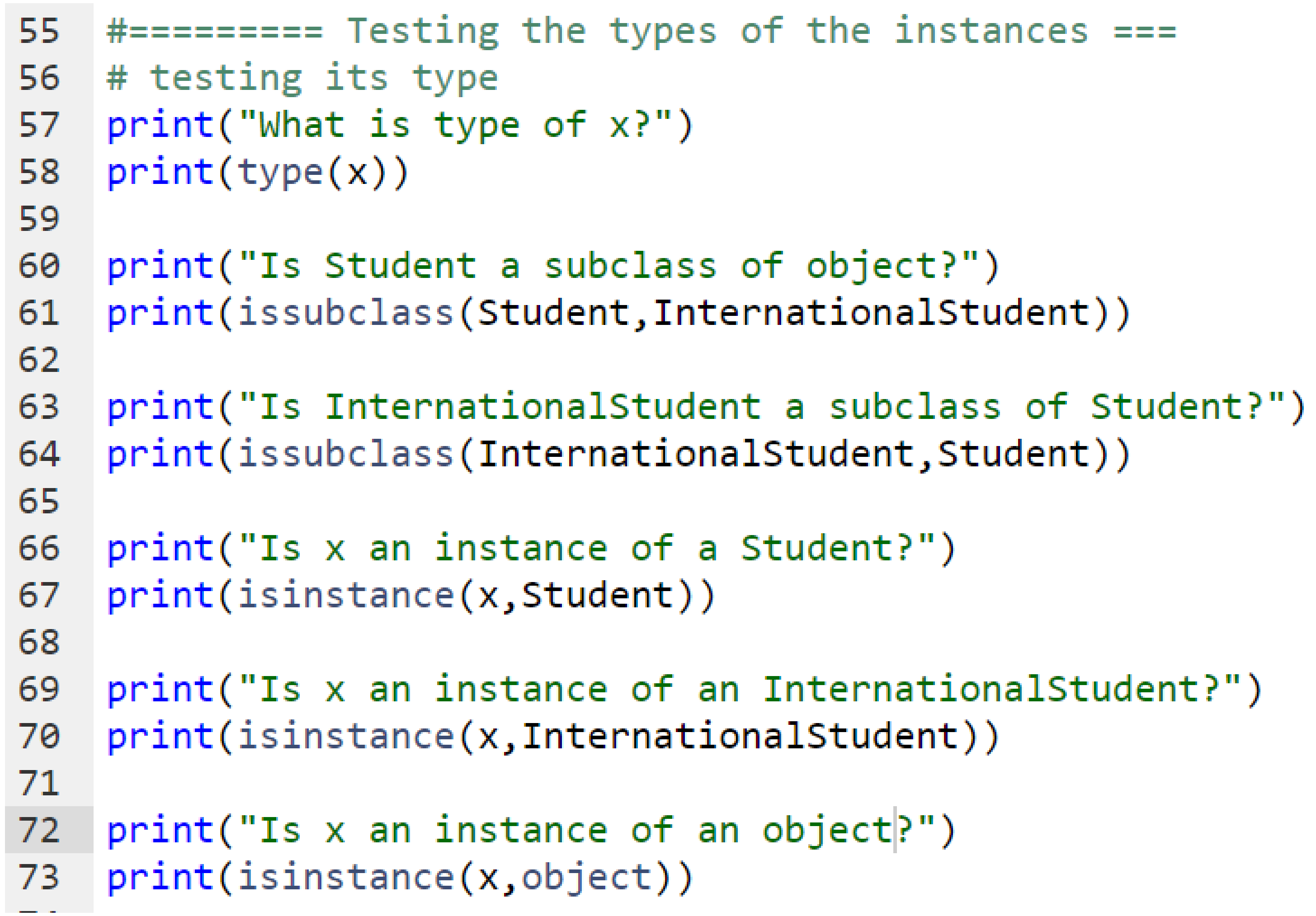
Case 2.
Overwriting Parent’s method

Case 3.
Providing additional methods

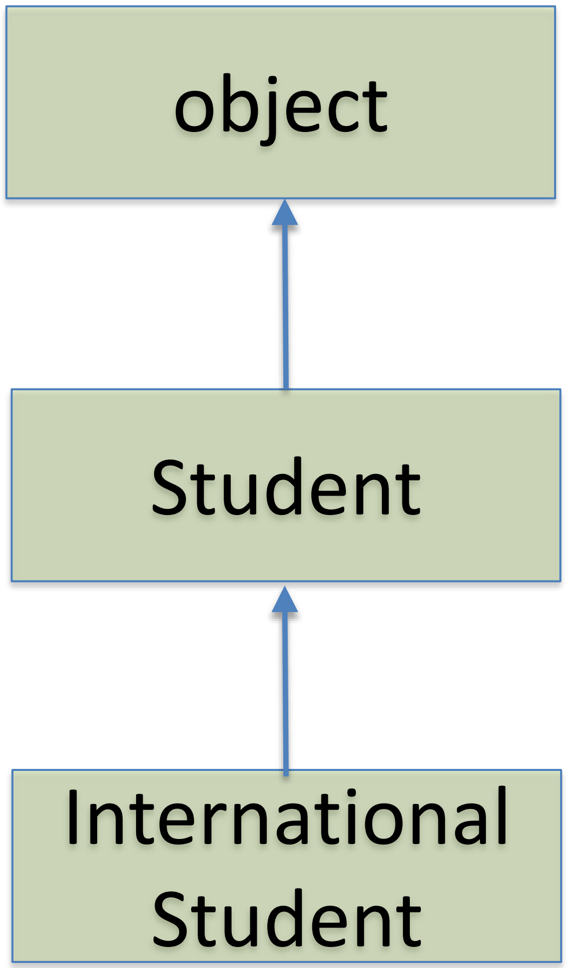
In fact, Children are more powerful than the parents.

- They can decide to ignore the parents and do their own thing
- They can have additional methods (behavior)
- They can have additional attributes

Let us revisit “type” testing (see link below)



```python
#========== Testing the types of the instances ===
# testing its type
print("What is type of x?")
print(type(x))

print("Is Student a subclass of object?")
print(issubclass(Student,InternationalStudent))

print("Is InternationalStudent a subclass of Student?")
print(issubclass(InternationalStudent,Student))

print("Is x an instance of a Student?")
print(isinstance(x,Student))

print("Is x an instance of an InternationalStudent?")
print(isinstance(x,InternationalStudent))

print("Is x an instance of an object?")
print(isinstance(x,object))
```



http://pythontutor.com/visualize.html#code=class%20Student%3A%0A%20%20%20%20def%20__init__%28self,%20fname,%20lname%29%3A%0A%20%20%20%20%20%20%20%20self.firstName%20%3D%20fname%0A%20%20%20%20%20%20%20%20self.lastName%20%3D%20lname%0A%0A%20%20%20%20def%20__repr__%28self%29%3A%0A%20%20%20%20%20%20%20%20
return%20%22First%20Name%3A%20%22%20%2B%20self.firstName%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Last%20Name%3A%20%22%20%2B%20self.lastName%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20__str__%28self%29%3A%0A%20%20%20%20%20%20%20%20retur
n%20%22First%20Name%3A%20%22%20%2B%20self.firstName%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Last%20Name%3A%20%22%20%2B%20self.lastName%0A%20%0A%20%0Aclass%20InternationalStudent%28Student%29%3A%0A%20%20%20%20def%20__init__%28self,%20fname,%20lname,%20a_country,%20a_language%
29%3A%0A%20%20%20%20%20%20%20%20super%28%29.__init__%28fname,%20lname%29%0A%20%20%20%20%20%20%20%20self.country%20%3D%20a_country%0A%20%20%20%20%20%20%20%20self.language%20%3D%20a_language%0A%20%20%0A%20%20%20%20def%20__repr__%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20%22First%20Name%3A
%20%22%20%2B%20self.firstName%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Last%20Name%3A%20%22%20%2B%20self.lastName%20%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Country%3A%20%22%20%2B%20self.country%20%20%2B%20%5C%0A%20%20%20%20%20%20%2
0%20%20%20%20%20%20%20%20%20%22%20Language%3A%20%22%20%2B%20self.language%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20__str__%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20%22First%20Name%3A%20%22%20%2B%20self.firstName%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20
%20%20%20%20%22%20Last%20Name%3A%20%22%20%2B%20self.lastName%20%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Country%3A%20%22%20%2B%20self.country%20%20%2B%20%5C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%20Language%3A%20%22%20%2B%20self.language%0A%20%2
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20dollars2rupees%28self,amount%29%3A%0A%20%20%20%20%20%20%20%20return%20amount*%2075%0A%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20rupees2dollars%28self,amount%29%3A%0A%20%20%20%20%20%20%20%20return%20amount/75%0A%20%20%20%20%0A
%20%20%20%20%0A%23%20creating%20instances%0Ax%3DInternationalStudent%28%22Siva%22,%22Jasthi%22,%20%22India%22,%20%22Telugu%22%29%0Ay%3DInternationalStudent%28%22Mahesh%22,%22Sunkara%22,%20%22India%22,%22Telugu%22%29%0Az%3DInternationalStudent%28%22Amit%22,%22Samtani%22,%20%22India%22,%20%22Marathi%22%29%0A%20
%0A%23priting%20the%20instances%0Aprint%28x%29%0Aprint%28y%29%0Aprint%28z%29%0A%0Aprint%28x.dollars2rupees%28100%29%29%0Aprint%28x.rupees2dollars%287500%29%29%0A%0A%0A%23%3D%3D%3D%3D%3D%3D%3D%3D%3D%3D%20Testing%20the%20types%20of%20the%20instances%20%3D%3D%3D%0A%23%20testing%20its%20type%0Aprint%28%22
What%20is%20type%20of%20x%3F%22%29%0Aprint%28type%28x%29%29%0A%0Aprint%28%22Is%20Student%20a%20subclass%20of%20object%3F%22%29%0Aprint%28issubclass%28Student,InternationalStudent%29%29%0A%0Aprint%28%22Is%20InternationalStudent%20a%20subclass%20of%20Student%3F%22%29%0Aprint%28issubclass%28InternationalStudent,Student%
29%29%0A%0Aprint%28%22Is%20x%20an%20instance%20of%20a%20Student%3F%22%29%0Aprint%28isinstance%28x,Student%29%29%0A%0Aprint%28%22Is%20x%20an%20instance%20of%20an%20InternationalStudent%3F%22%29%0Aprint%28isinstance%28x,InternationalStudent%29%29%0A%0Aprint%28%22Is%20x%20an%20instance%20of%20an%20object%3F%22%29%0
Aprint%28isinstance%28x,object%29%29%0A%0A&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false

UML diagram showing inheritance

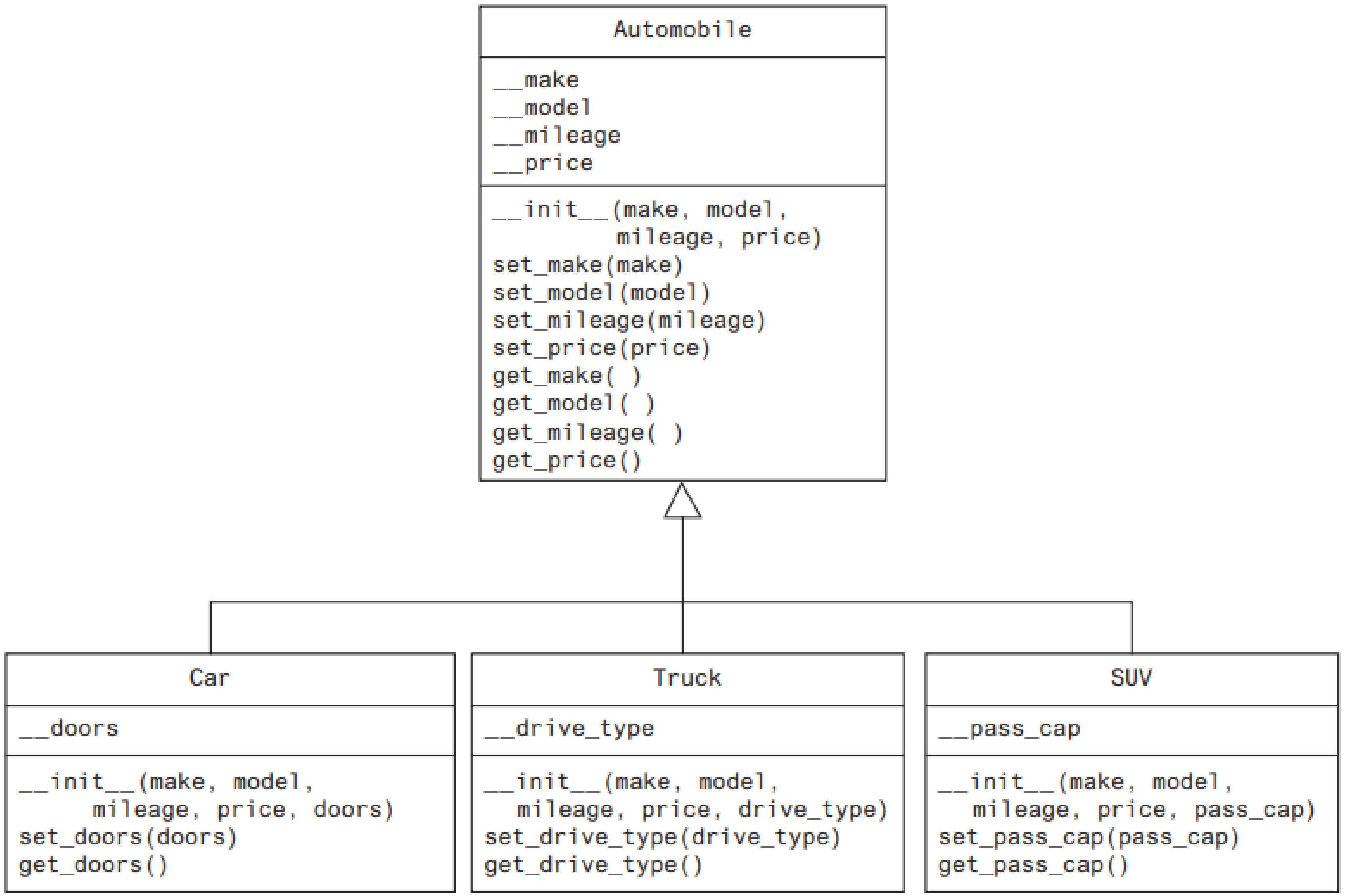
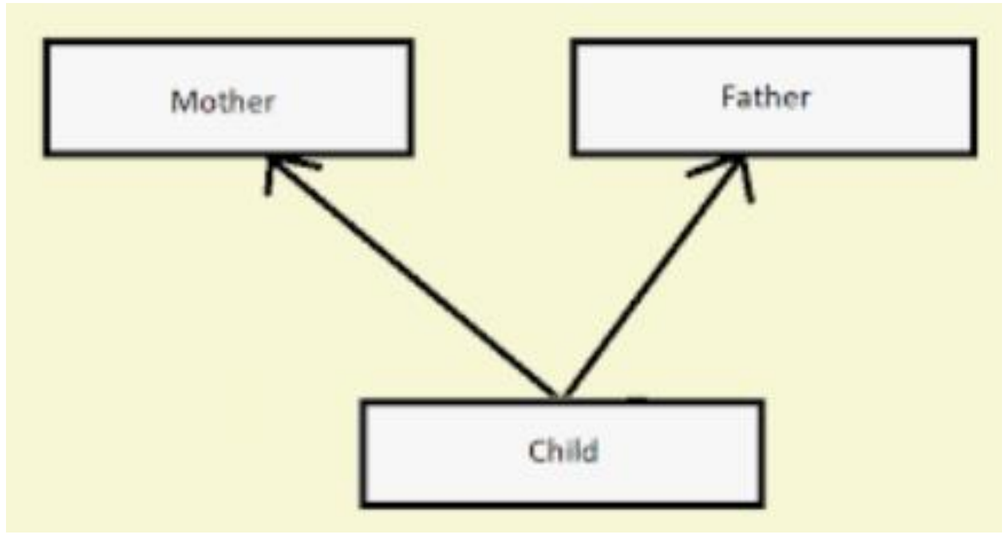


Figure 11-2 UML diagram showing inheritance

Multiple Inheritance

-

Good article on Multiple inheritance
https://data-flair.training/blogs/python-multiple-inheritance/



Python supports Multiple Inheritance as well.

If “Mother” class implements a method in one way, and “Father” class implements the same method in a different way, which method will be inherited by “Child”?

If “Mother” has a variable with one value (x=20) and “Father” has the same variable with a different value (x=30) which value is inherited by “Child”?

Due to such complications, JAVA does not support multiple inheritance. But python supports it through a mechanism called “Method Resolution Order” (MRO)

MRO: Left to right; depth-first

Inheritance Summary

Child has everything a Parent or Grand Parent has.

So, if you write a program for a Parent, it will work fine for Child also.

So, wherever a Parent is expected, you can substitute a Child.

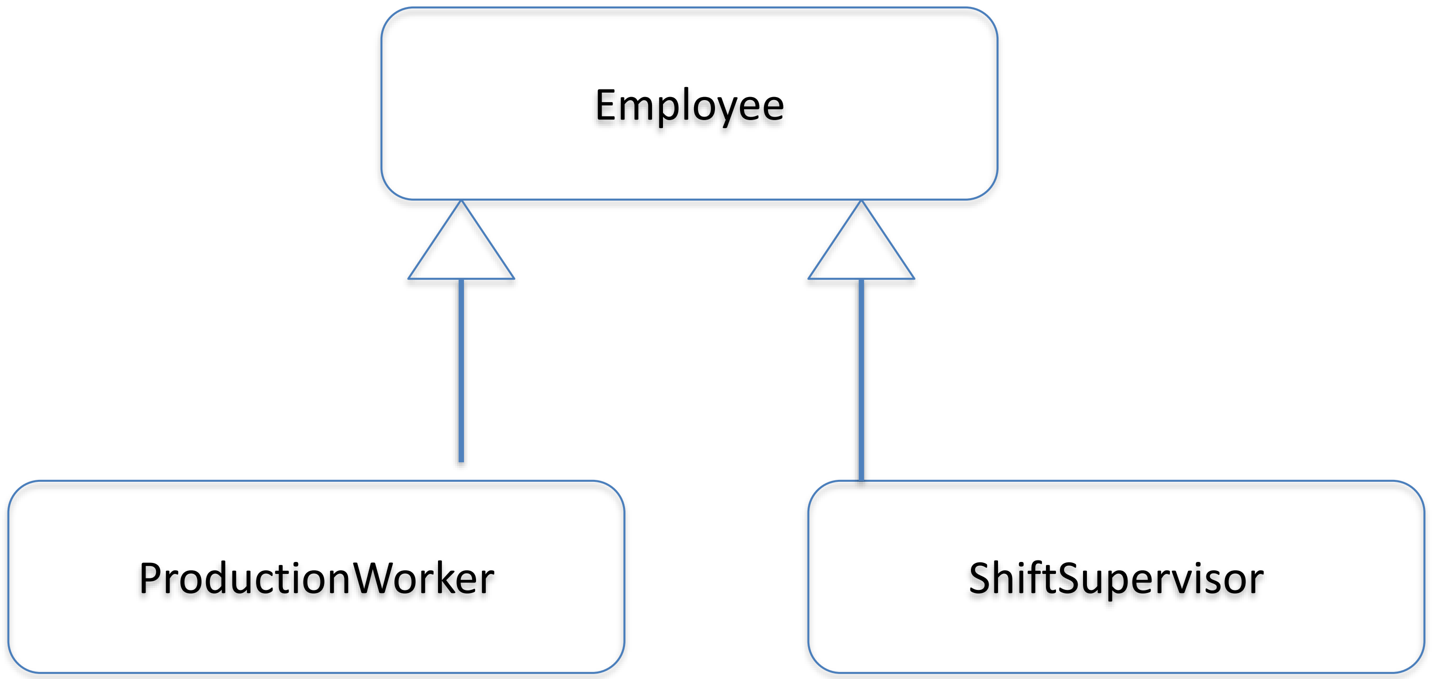
This is called “Principle of Substitutability”.

This is also called “Polymorphism”.

Programming Assignment 1 and 2 from Chapter 11

Employee (emp_name, emp_no)
ProductionWorker (shift_no, hourly_pay_rate)
ShiftSupervisor (annual_salary, annual_bonus)

1. Each class should encapsulate its data
2. Each class should have __repr__ and__str__ methods
3. Each class should have getters
4. Each class should have setters
5. Each class should try out creating the instances and printing those instances


Employee
ProductionWorker
ShiftSupervisor