Inheritance and Polymorphism

Inheritance and Polymorphism

The power of OOP

Review of OOP

Person
`firstName: str`
`lastName: str`
`birthYear: int`
`birthMonth: int`
`birthDay: int`
`__init__(name="", DOB="")`
`__str__(): returns str`
`setName(name)`
`setDOB(DOB)`
`getName(): returns str`
`getDOB(): returns str`
`canVote(): returns bool`

Review of OOP

Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool

https://yuml.me/

Review of OOP (continued)

Review of OOP (continued)

```
class Person():
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
        self.birthMonth = 0
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
        self.birthMonth = 0
        self.birthDay = 0
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
        self.birthMonth = 0
        self.birthDay = 0
        self.birthYear = 0
```

Review of OOP (continued)

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
        self.birthMonth = 0
        self.birthDay = 0
        self.birthYear = 0
        return
```

Review of OOP (continued)

Review of OOP (continued)

```
class Person():
```

Review of OOP (continued)

```
class Person():
    ⋮
```

Review of OOP (continued)

```
class Person():
     ⋮
    def __str__(self):
```

Review of OOP (continued)

```
class Person():
          ⋮
    def __str__(self):
        """ Docstring for __str__ here """
```

Review of OOP (continued)

```
class Person():
    ⋮
    def __str__(self):
        """ Docstring for __str__ here """
        return f"{self.firstName} ..."
```

Review of OOP (continued)

Review of OOP (continued)

```
class Person():
```

Review of OOP (continued)

```
class Person():
    ⋮
```

Review of OOP (continued)

```
class Person():
    ⋮
    def setName(self, name):
```

Review of OOP (continued)

```
class Person():
    ⋮
    def setName(self, name):
        """ Docstring for setName here """
```

Review of OOP (continued)

```
class Person():
     ⋮
    def setName(self, name):
        """ Docstring for setName here """
        self.firstName = name.split()[0]
```

Review of OOP (continued)

```
class Person():
         ⋮
    def setName(self, name):
        """ Docstring for setName here """
        self.firstName = name.split()[0]
        self.lastName = name.split()[1]
```

Review of OOP (continued)

```
class Person():
          ⋮
    def setName(self, name):
        """ Docstring for setName here """
        self.firstName = name.split()[0]
        self.lastName = name.split()[1]
        return
```

Review of OOP (continued)

Review of OOP (continued)

```
class Person():
```

Review of OOP (continued)

```
class Person():
    ⋮
```

Review of OOP (continued)

```
class Person():
     ⋮
    def getName(self):
```

Review of OOP (continued)

```
class Person():
     ⋮
    def getName(self):
        """ Docstring for getName here """
```

Review of OOP (continued)

```
class Person():
    ⋮
    def getName(self):
        """ Docstring for getName here """
        return f"{self.firstName} {self.lastName}"
```

Review of OOP (continued)

Review of OOP (continued)

The `Person` class is saved in the `person.py` file.

Review of OOP (continued)

The `Person` class is saved in the `person.py` file.

So to use:

Review of OOP (continued)

The `Person` class is saved in the `person.py` file.

So to use:

```
import person
```

Review of OOP (continued)

The `Person` class is saved in the `person.py` file.

So to use:

```
import person
tyler = person.Person()
```

Re-using a class

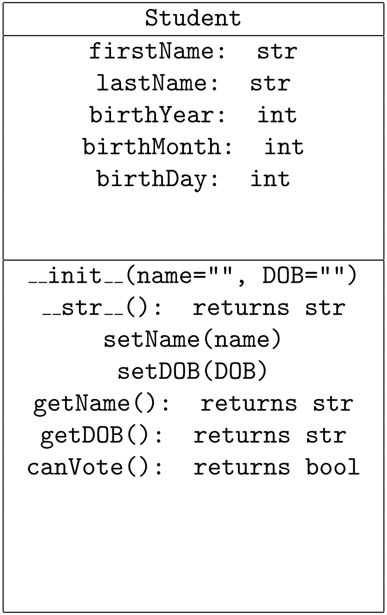
Re-using a class

What happens if we want to create a special kind of Person?

Re-using a class

What happens if we want to create a special kind of Person?

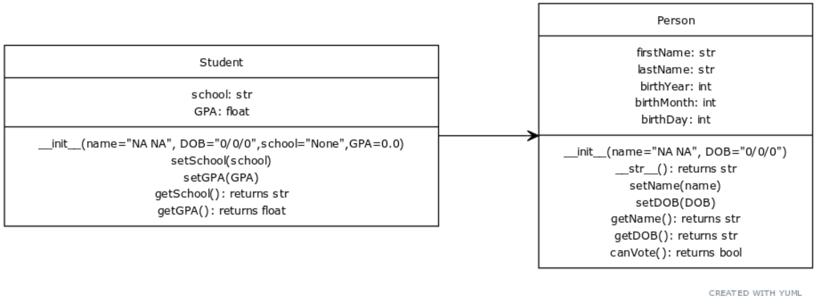
Ex: Want a Student class for describing college students?



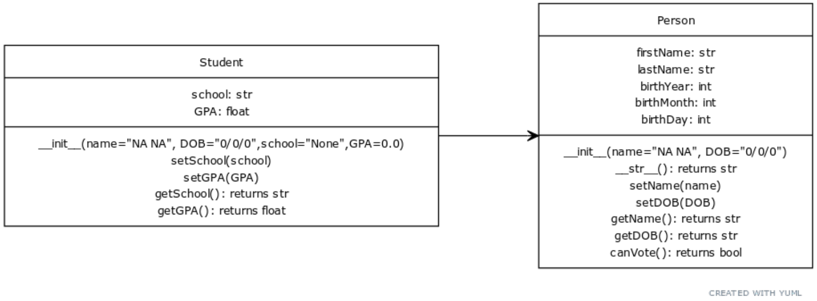
```
Student
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool
```

Student
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
school: str
GPA: float
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool
setSchool(school)
setGPA(GPA)
getSchool(): returns str
getGPA(): returns float

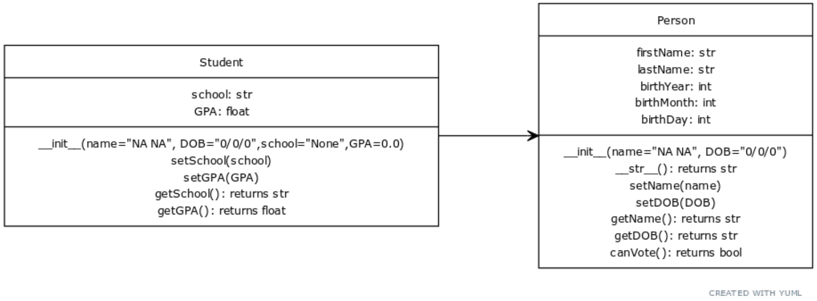
Inheritance



Inheritance

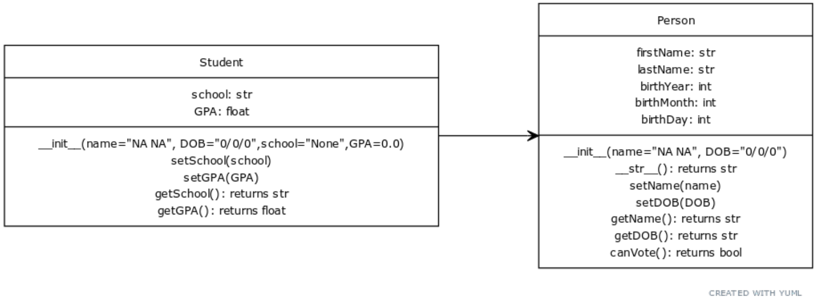


Inheritance



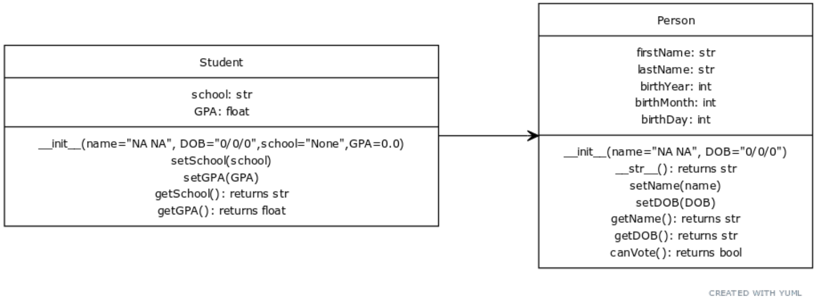
Child class

Inheritance



Child class inherits from

Inheritance



Child class inherits from Parent class

Extending classes in Python

Extending classes in Python

```
import person
```

Extending classes in Python

```
import person

class Student(
```

Extending classes in Python

```
import person

class Student( person.Person
```

Extending classes in Python

```
import person

class Student( person.Person ):
```

Extending classes in Python

```
import person

class Student( person.Person ):
    """ Docstring for Student here """
```

Extending classes in Python

```
import person

class Student( person.Person ):
    """ Docstring for Student here """
    def __init__(self, name="", DOB="",
                 school="", GPA=0.0):
        """ New constructor here """
```

Extending classes in Python

```
import person

class Student( person.Person ):
    """ Docstring for Student here """
    def __init__(self, name="", DOB="",
                       school="", GPA=0.0):
        """ New constructor here """
        super().__init__(name=name, DOB=DOB)
        # Do extra stuff for Student class here
        ...
        return
```

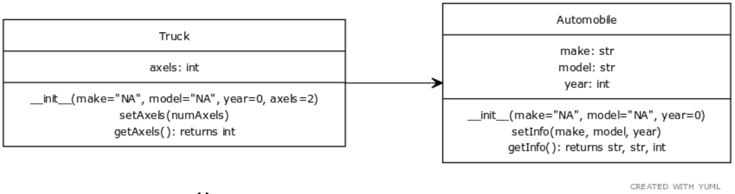
Extending classes in Python

```
import person

class Student( person.Person ):
    """ Docstring for Student here """
    def __init__(self, name="", DOB="",
                 school="", GPA=0.0):
        """ New constructor here """
        super().__init__(name=name, DOB=DOB)
        # Do extra stuff for Student class here
        ...
        return

    def setSchool(self, school):
        ...
```

Concept Check!

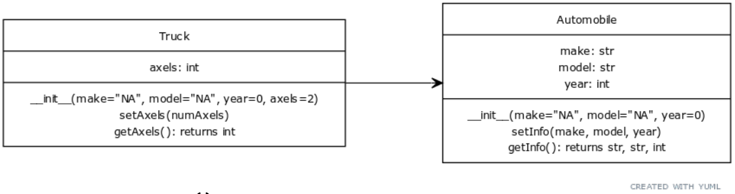

Truck
axels: int
__init__(make="NA", model="NA", year=0, axels=2)
setAxels(numAxels)
getAxels(): returns int
Automobile
make: str
model: str
year: int
__init__(make="NA", model="NA", year=0)
setInfo(make, model, year)
getInfo(): returns str, str, int
CREATED WITH YUML

```
myTruck = Truck()
myCar = Automobile()
```

Based on the above, which of the following are legal?

1. `myTruck.setInfo("Chevy", "Silverado", 2010)`
2. `myCar.setInfo("Toyota", "Corolla", 2015)`
3. `myTruck.setAxels(2)`
4. `myCar.setAxels(2)`

Concept Check!



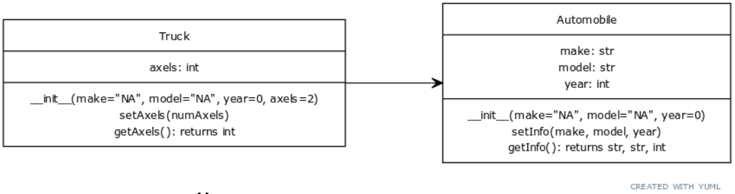
Truck
axels: int
__init__(make="NA", model="NA", year=0, axels=2)
setAxels(numAxels)
getAxels(): returns int
Automobile
make: str
model: str
year: int
__init__(make="NA", model="NA", year=0)
setInfo(make, model, year)
getInfo(): returns str, str, int
CREATED WITH YUML

```
myTruck = Truck()
myCar = Automobile()
```

Based on the above, which of the following are legal?

1. `myTruck.setInfo("Chevy", "Silverado", 2010)` ✓
2. `myCar.setInfo("Toyota", "Corolla", 2015)`
3. `myTruck.setAxels(2)`
4. `myCar.setAxels(2)`

Concept Check!



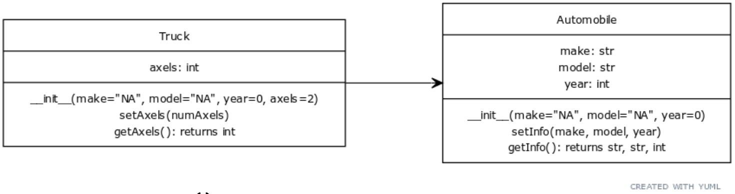
Truck
axels: int
__init__(make="NA", model="NA", year=0, axels=2)
setAxels(numAxels)
getAxels(): returns int
Automobile
make: str
model: str
year: int
__init__(make="NA", model="NA", year=0)
setInfo(make, model, year)
getInfo(): returns str, str, int
CREATED WITH YUML

```
myTruck = Truck()
myCar = Automobile()
```

Based on the above, which of the following are legal?

1. `myTruck.setInfo("Chevy", "Silverado", 2010)` ✓
2. `myCar.setInfo("Toyota", "Corolla", 2015)` ✓
3. `myTruck.setAxels(2)`
4. `myCar.setAxels(2)`

Concept Check!



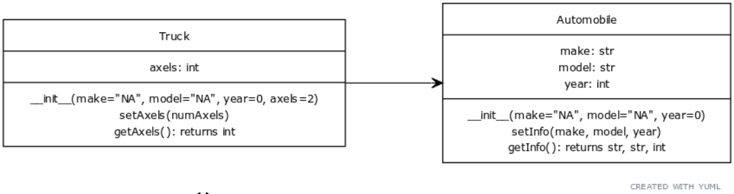
Truck
axels: int
__init__(make="NA", model="NA", year=0, axels=2)
setAxels(numAxels)
getAxels(): returns int
Automobile
make: str
model: str
year: int
__init__(make="NA", model="NA", year=0)
setInfo(make, model, year)
getInfo(): returns str, str, int
CREATED WITH YUML

```
myTruck = Truck()
myCar = Automobile()
```

Based on the above, which of the following are legal?

1. `myTruck.setInfo("Chevy", "Silverado", 2010)` ✓
2. `myCar.setInfo("Toyota", "Corolla", 2015)` ✓
3. `myTruck.setAxels(2)` ✓
4. `myCar.setAxels(2)`

Concept Check!


Truck
axels: int
__init__(make="NA", model="NA", year=0, axels=2)
setAxels(numAxels)
getAxels(): returns int
Automobile
make: str
model: str
year: int
__init__(make="NA", model="NA", year=0)
setInfo(make, model, year)
getInfo(): returns str, str, int
CREATED WITH YUML

```
myTruck = Truck()
myCar = Automobile()
```

Based on the above, which of the following are legal?

1. `myTruck.setInfo("Chevy", "Silverado", 2010)` ✓
2. `myCar.setInfo("Toyota", "Corolla", 2015)` ✓
3. `myTruck.setAxels(2)` ✓
4. `myCar.setAxels(2)` ✗

Polymorphism

Polymorphism

Perform different tasks with the same command, depending on context:

Polymorphism

Perform different tasks with the same command, depending on context:

```
def read_info_from_user(person):
```

Polymorphism

Perform different tasks with the same command, depending on context:

```
def read_info_from_user(person):
    """ Fill a person's information from keyboard
```

Polymorphism

Perform different tasks with the same command, depending on context:

```
def read_info_from_user(person):
    """ Fill a person's information from keyboard

    Args:
        person (Person or child class):  blank person
    """
```

Polymorphism

Perform different tasks with the same command, depending on context:

```
def read_info_from_user(person):
    """ Fill a person's information from keyboard

    Args:
        person (Person or child class):  blank person
    """
    person.readInfo()
    return
```

Polymorphism (continued)

Polymorphism (continued)

Person class and Student class have different readInfo() methods:

Polymorphism (continued)

Person class and Student class have different readInfo() methods:

```
class Person():

    ...

    def readInfo(self):
        """ Read name and DOB from user """
        ...
```

Polymorphism (continued)

Person class and Student class have different readInfo() methods:

```
class Person():

    ...

    def readInfo(self):
        """ Read name and DOB from user """
        ...

class Student(Person):

    ...

    def readInfo(self):
        """ Read name, DOB, school, and GPA """
        ...
```