Principles of Software Programming

Lecture 5
Object oriented programming

Anton Yeshchenko

SS 2018

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Some slides and/or ideas were borrowed from:
MIT Introduction to Computer Science and Programming in Python
and Svitlana Vakulenko WS 2017 lecture slides

APRIL 2018

EFMD EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION OF MBAS ACCREDITED

Recap

- **Functions**
  - Return
  - Parameters
  - Method overloading
- **Control flow:**
  - if-else branches
  - loops
- **Lists:**
  - Arrays (lists)
  - create and fill Arrays
- **Types**
- **Classes**

CLASSes

- __init__(self, blabla):
  - self.bla = blabla
- Attributes and behavior
  - Always **self**
  - **Function (method) of a class**
  - **Function (method) of an instance**
  - **Function**

  - Access elements of the class

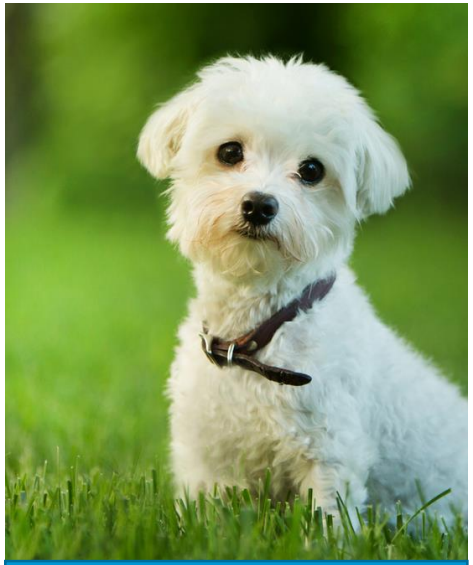
  - Dot . Operator

Huge homework!

- Questions?

Define Tom!

Our Tom can!
bark()
run()
carry_bone()
poo_in_the_park()
talk_to_a_dog(a_dog)



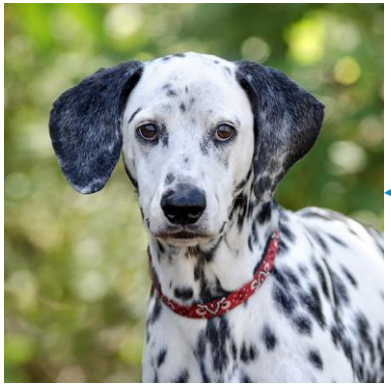
name = “Tom”

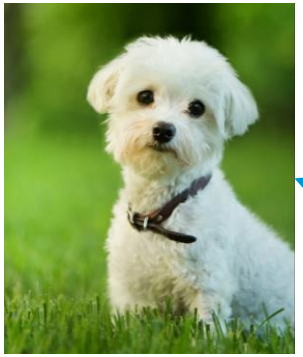
Is it enough to say that he is ‘Tom’ for him to be a dog?

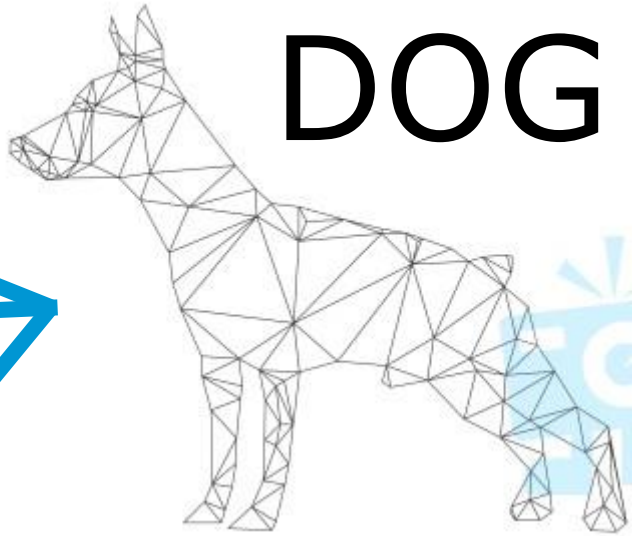
He is:
A dog
He has:
Weight=2kg
Height=20cm
Accessories = [belt, leash, gps tracker]

Classes, Objects, Instances!

- We can make a **class dog** (the blue print)
- "Tom" is an **instance** of the **class dog**







Can have weight, height, accessories,
Can bark, run, carry_done etc.

Today!

- Classes – Objects - Instances
- **OOP and principles**


OBJECT ORIENTED
PROGRAMMING
memegenerator.net

What are the objects?

- objects are **a data abstraction** that captures…

(1) an **internal representation**

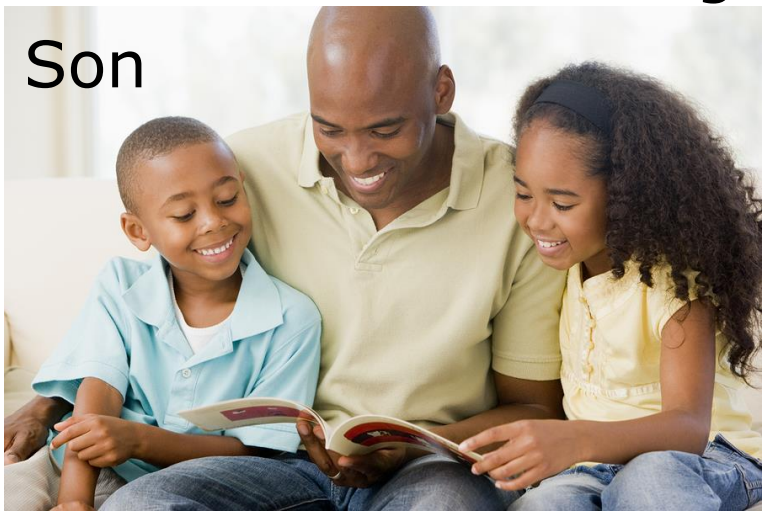
- through data attributes

(2) an **interface** for interacting with object

- through methods (aka procedures/functions)
- defines behaviors but hides implementation

Object oriented programming. WHY?????? All objects

Father Daughter

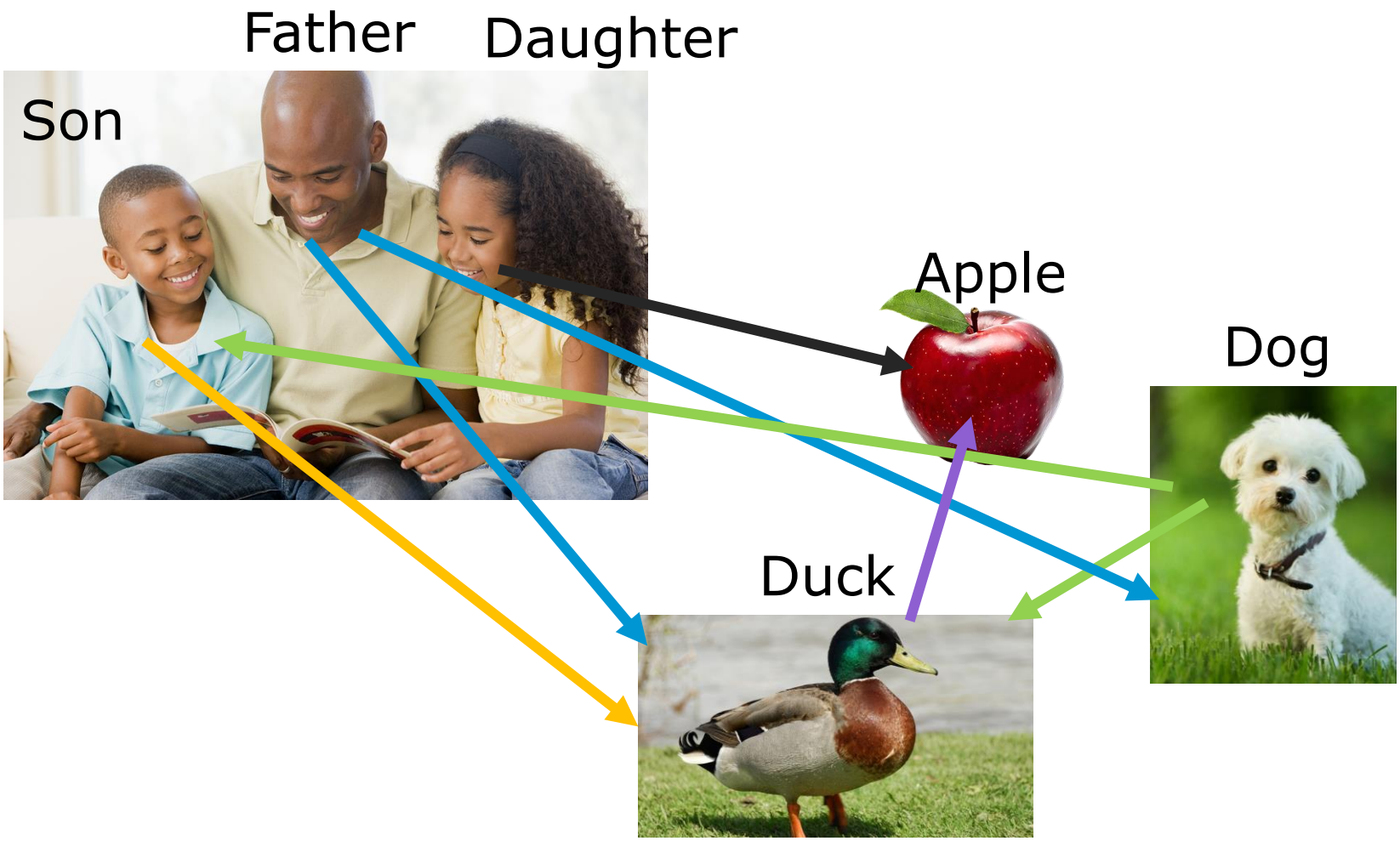


Apple

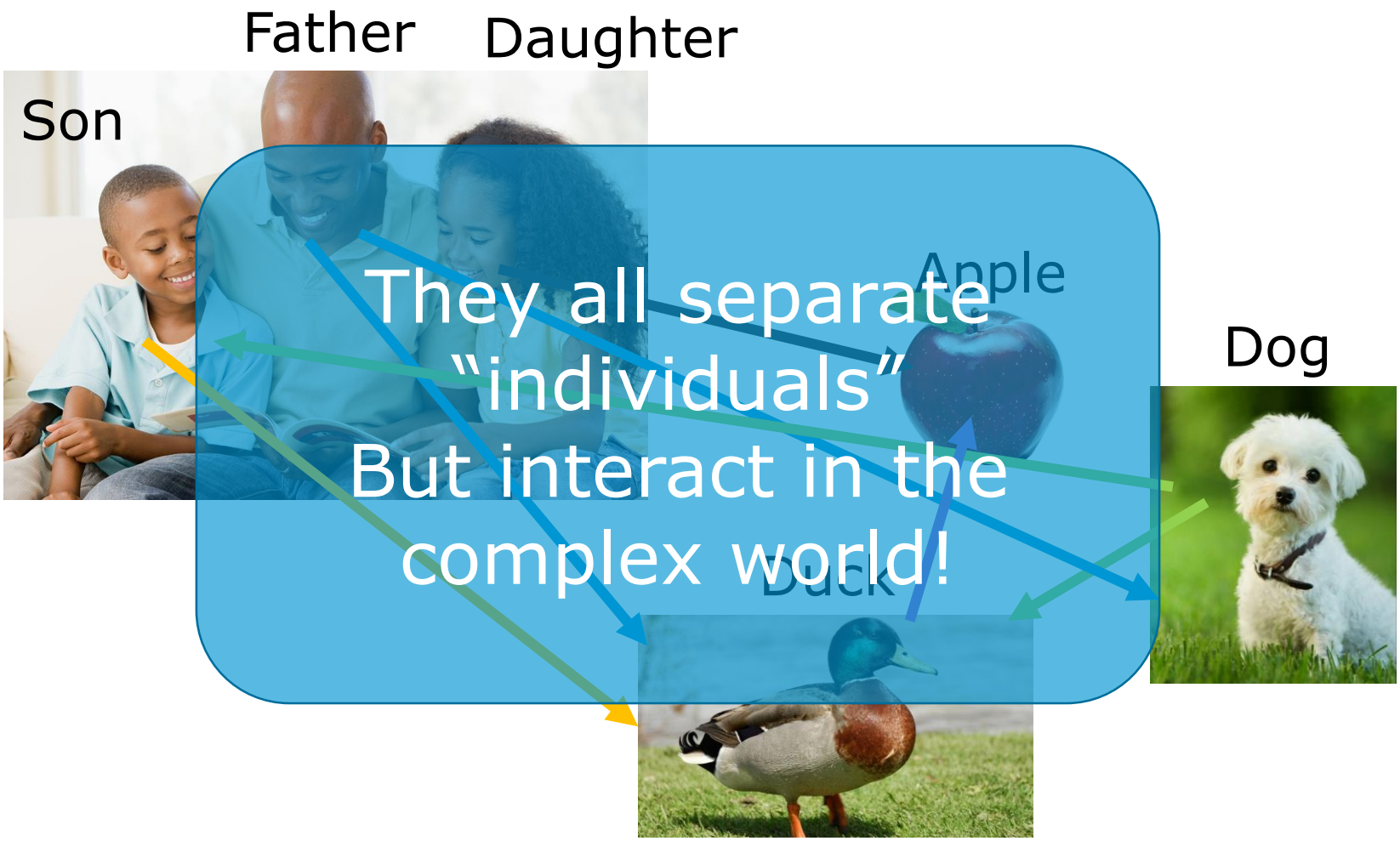
Dog

Duck

Object oriented programming. WHY?????? All objects

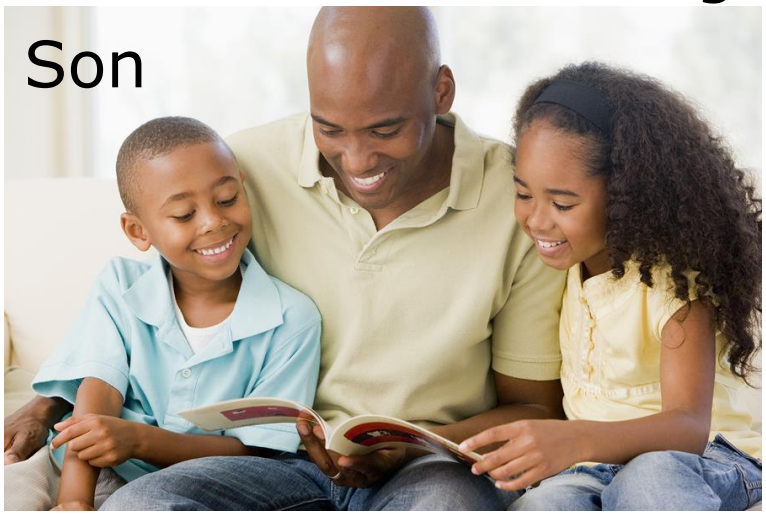


Object oriented programming. WHY?????? All objects



Object oriented programming. WHY?????? All objects

Father Daughter



Apple

Dog

divide-and-conquer (decomposition)
development
•implement and test behavior of each class separately
•increased modularity reduces complexity
Reuse!

Duck

4 principles of OOP

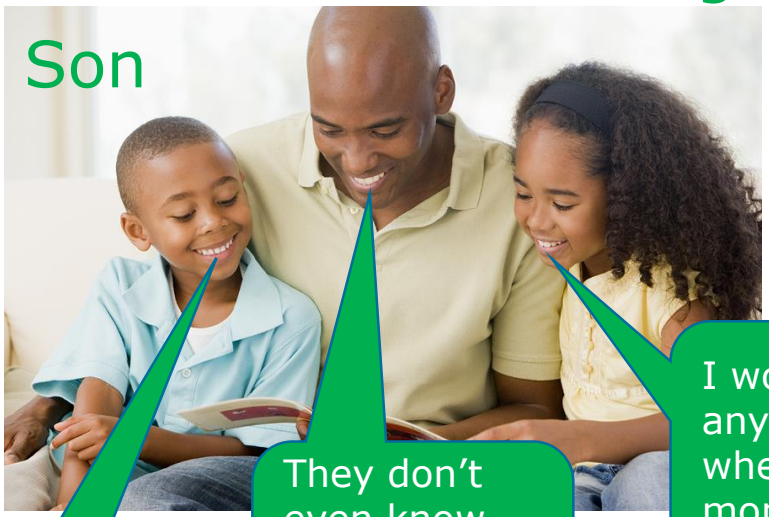
1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

Encapsulation. Everyone hides something

Father

Daughter

Son



Whatever he tells me, I know better

They don't even know that their mother left them at birth

I wont tell anyone where my money are

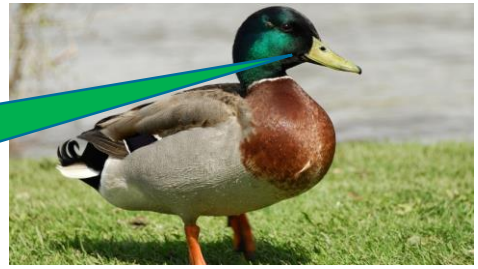
I know how many vitamins I have

Apple

I know how to please my owner, I have my own leash!

Dog

Duck



I know my hideout!

Encapsulation

- **Information hiding!** *restrict access* to methods and variables (visibility) to prevent the data from being modified by accident
  - **public** accessible from anywhere
  - **private** can be accessed only from the same class: __ prefix

```
class Car:
    def __init__(self):
        self.__updateSoftware()
        self.__a = 10
    def drive(self):
        print 'driving'
    def __updateSoftware(self):
        print 'updating software'
```

Encapsulation. Variable access

- **Hide everything from everyone. Allow access to those who really need it only.**
  - Class variables ??
  - Object variables ??
  - Block (local) variables ??
  - Global variables ??

HTTPS://STACKOVERFLOW.COM/QUESTIONS/68645/STATIC-CLASS-VARIABLES-IN-PYTHON

Encapsulation. Syntax

- **Hide everything from everyone. Allow access to those who really need it only.**
  - Class variables ??
    - Variables declared inside the class definition, but not inside a method are class or static variables:

```
>>> class MyClass:
...     i = 3
...
>>> MyClass.i
3
```


HTTPS://STACKOVERFLOW.COM/QUESTIONS/68645/STATIC-CLASS-VARIABLES-IN-PYTHON

Encapsulation. Syntax

- **Hide everything from everyone. Allow access to those who really need it only.**
  - Class variables ??
    - Variables declared inside the class definition, but not inside a method are class or static variables:

```
>>> class MyClass:
...     i = 3
...
>>> MyClass.i
3
```

```
>>> m = MyClass()
>>> m.i = 4
>>> MyClass.i, m.i
>>> (3, 4)
```


HTTPS://STACKOVERFLOW.COM/QUESTIONS/68645/STATIC-CLASS-VARIABLES-IN-PYTHON

Encapsulation. Syntax

- **Hide everything from everyone. Allow access to those who really need it only.**
  - Class variables ??
  - Object variables ??

```
class Dog:
    def __init__(self, x):
        self.x = x

tom = Dog(10)
tom.weight = 2.3

print (tom.weight, tom.x)
```

HTTPS://STACKOVERFLOW.COM/QUESTIONS/68645/STATIC-CLASS-VARIABLES-IN-PYTHON

Encapsulation. Syntax

- Class variables ??
- Object variables ??

Block (local) variables ??

- Python variables are scoped to the innermost function, class, or module in which they're assigned. Control blocks like if and while blocks don't count, so a variable assigned inside an if is still scoped to a function, class, or module.

```python
x = 10
if True:
    y = 10

print (x, y)
```

```java
public class HelloWorld{

    public static void main(String []args){

        int x = 10;
        if (true){
            int y = 10;
            System.out.println(y);
        }

        System.out.println(x);
        System.out.println(y);
    }
}
```

HTTPS://STACKOVERFLOW.COM/QUESTIONS/68645/STATIC-CLASS-VARIABLES-IN-PYTHON

Encapsulation. Syntax

- Class variables ??
- Object variables ??
- Block (local) variables ??

Global variables ??

- Defined outside of the functions
- Defined inside of the functions with **global** modifier

```
# sample.py
myGlobal = 5

def func1():
    myGlobal = 42

def func2():
    print myGlobal

func1()
func2()
```


HTTPS://STACKOVERFLOW.COM/QUESTIONS/68645/STATIC-CLASS-VARIABLES-IN-PYTHON

Encapsulation. Syntax

- Class variables ??
- Object variables ??
- Block (local) variables ??

Global variables ??

- Defined outside of the functions
- Defined inside of the functions with **global** modifier

```
# sample.py
myGlobal = 5

def func1():
    myGlobal = 42

def func2():
    print myGlobal

func1()
func2()
```

```
def func1():
    global myGlobal
    myGlobal = 42
```


HTTPS://STACKOVERFLOW.COM/QUESTIONS/68645/STATIC-CLASS-VARIABLES-IN-PYTHON

Encapsulation. Get-set-methods

In python everything is public (unless states otherwise), need to make an effort to keep secrets

```
class Car:
    def __init__(self):
        self.__updateSoftware()
        self.__a = 10
    def drive(self):
        print 'driving'
    def __updateSoftware(self):
        print 'updating software'
```

- How do you change __a if it's only accessible in the class?

Encapsulation. Get-set-methods

In python everything is public (unless states otherwise), need to make an effort to keep secrets

```
class Car:
    def __init__(self):
        self.__updateSoftware()
        self.__a = 10
    def drive(self):
        print 'driving'
    def __updateSoftware(self):
        print 'updating software'
```

- How do you change __a if it's only accessible in the class?
  - set and get methods!

Exercise 2! Encapsulation

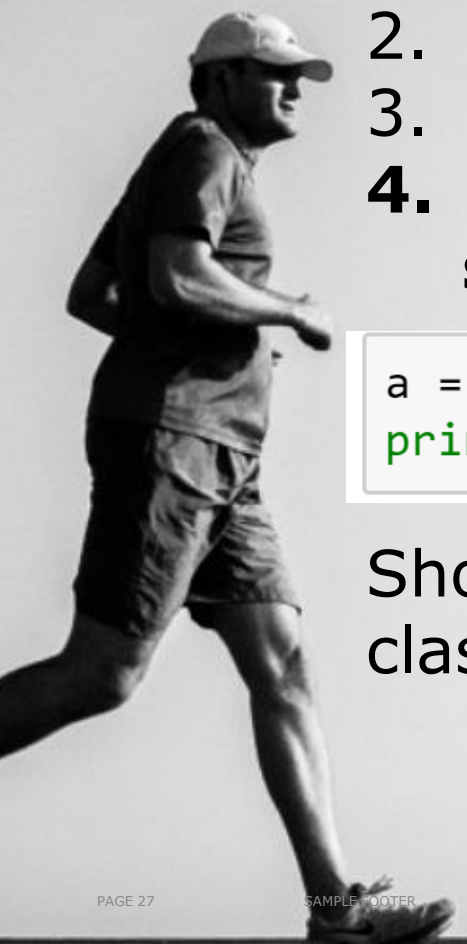
1. Define class Human
2. Make a private variable “real_secret”
3. Make a public variable “secret”

```
a = Human("I dont like apples", "I hate my classmate")
print("My secret is that " + a.secret)
```

Should print “My secret is that I don’t like apples”

https://i.ytimg.com/vi/Z91tys7dYp4/hqdefault.jpg

Exercise 2! Encapsulation



1. Define class Human
2. Make a private variable “real_secret”
3. Make a public variable “secret”
4. **Make function that prints the “real secret secret”**

```
a = Human("I dont like apples", "I hate my classmate")
print("My secret is that " + a.tell_secret()
```

Should print “My secret is that I hate my classmate”

https://i.ytimg.com/vi/Z91tys7dYp4/hqdefault.jpg

Abstraction

- Know only what you need to know!
- **Abstraction** refers to showing only the necessary details to the intended user

HTTPS://CLEANTECHNICA.COM/FILES/2017/08/TESLA-MODEL-X-SILVER.JPG

Abstraction

HTTPS://OEH-WU.AT/EVENTS/LIFE/WU-BALL-2018-AUSVERKAUFT

Abstraction

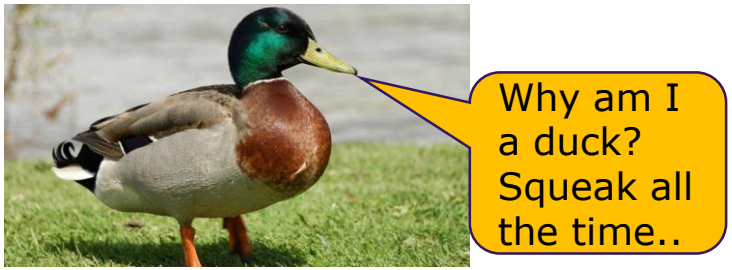
Success

CONTENT/UPLOADS/2017/11/FACEBOOK_EVENT_138012376838707.JPG

Inheritance

- **Describe behavior with class**

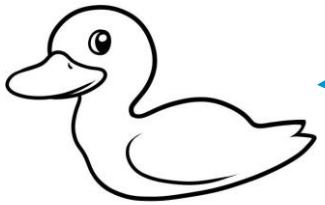
Duck

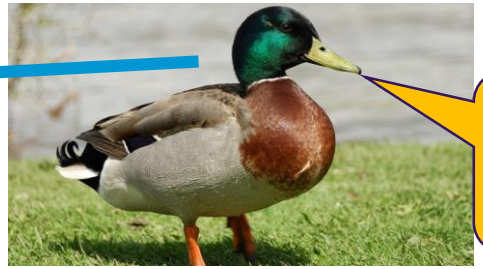


Inheritance

- **Describe behavior with class**

Duck





Why am I a duck? Squeak all the time..

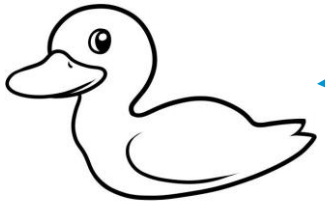
Inheritance

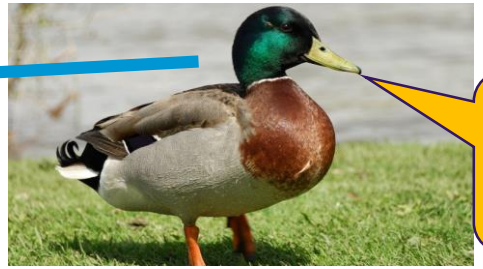
- Describe behavior with class

Great! I knew how to bark from the day I was born!

Dog

Duck





Why am I a duck? Squeak all the time..

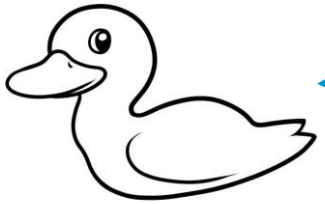
Inheritance

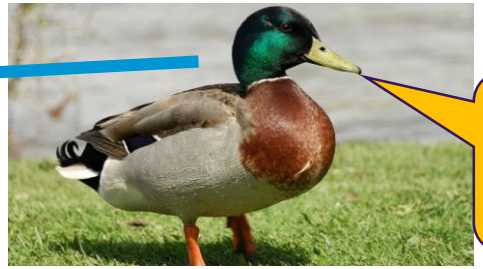
- Describe behavior with class

Great! I knew how to bark from the day I was born!

Dog

Duck

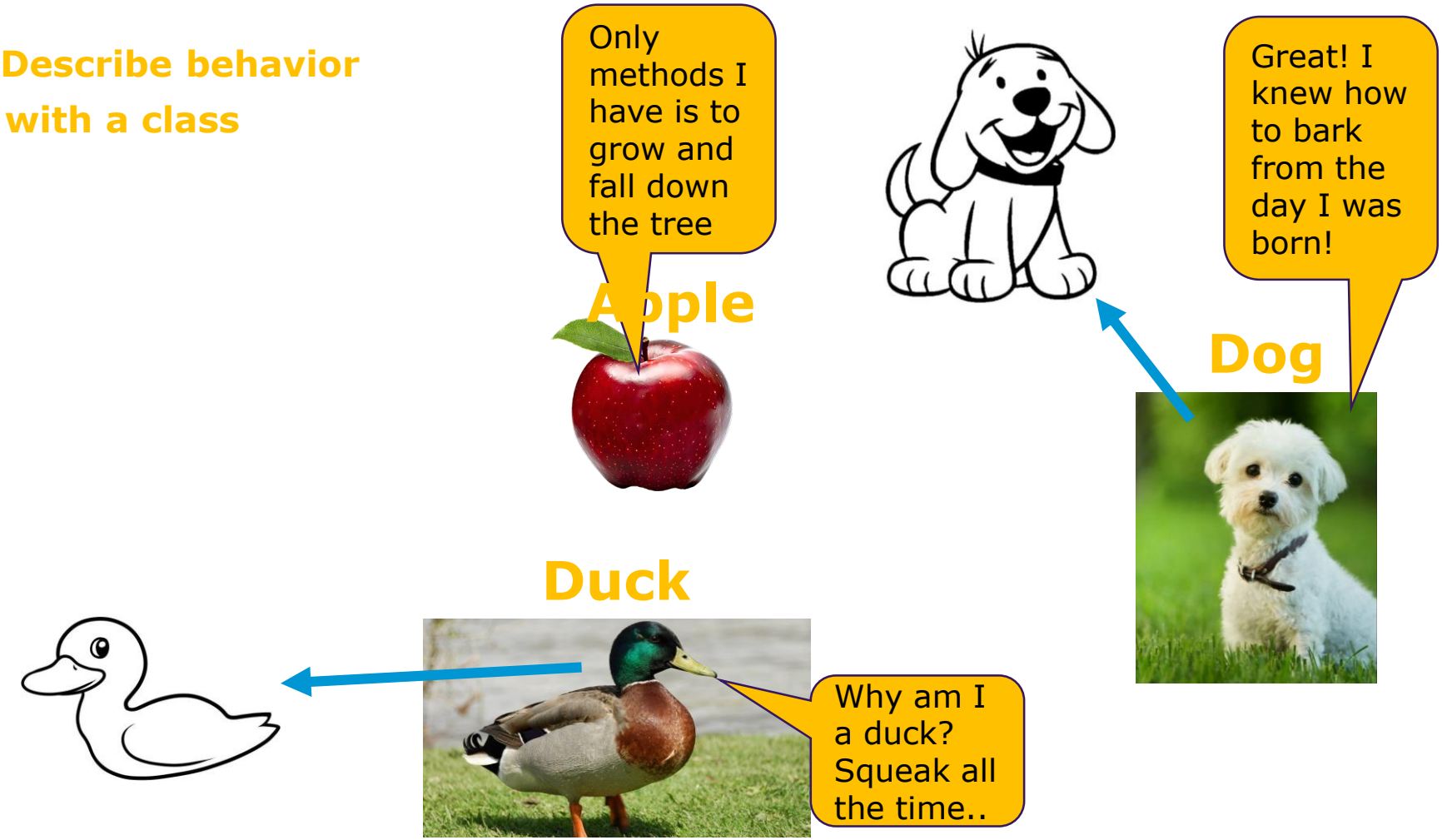




Why am I a duck? Squeak all the time..

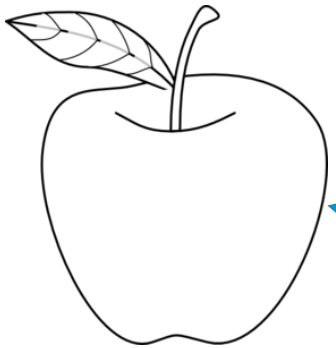
Inheritance

- **Describe behavior with a class**



Inheritance

- **Describe behavior with a class**

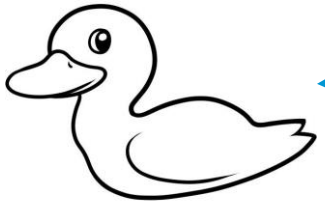


Only methods I have is to grow and fall down the tree

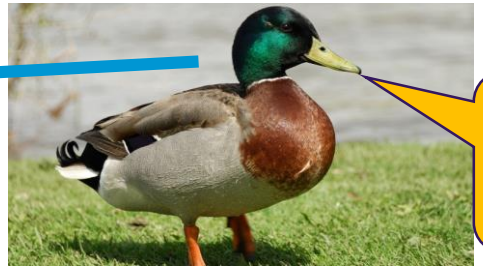
Apple

Great! I knew how to bark from the day I was born!

Dog



Duck



Why am I a duck? Squeak all the time..

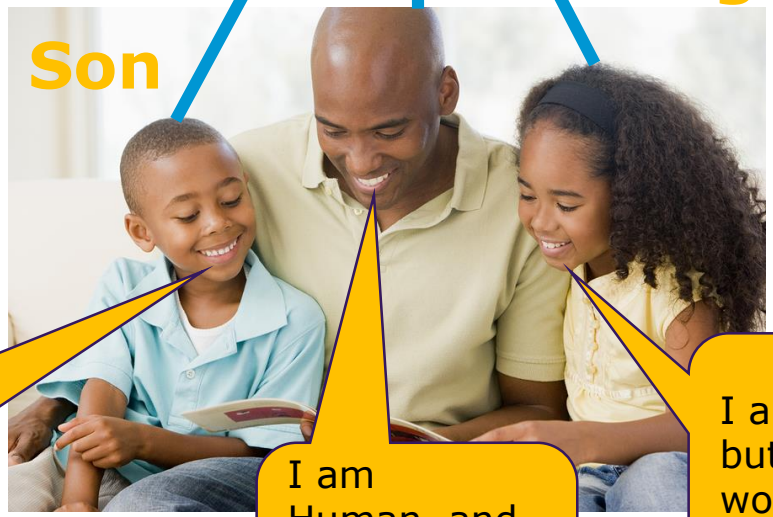
Inheritance

human

Father

Daughter

Son



I am human after all! As all of them!

I am Human, and a man. That makes me who I am

I am human, but a woman!

Inheritance

Human

Work()

Father

Daughter

Son

I am human after all! As all of them!

I am Human, and a man. That makes me who I am

I am human, but a woman!

All have method **work()** From class Human

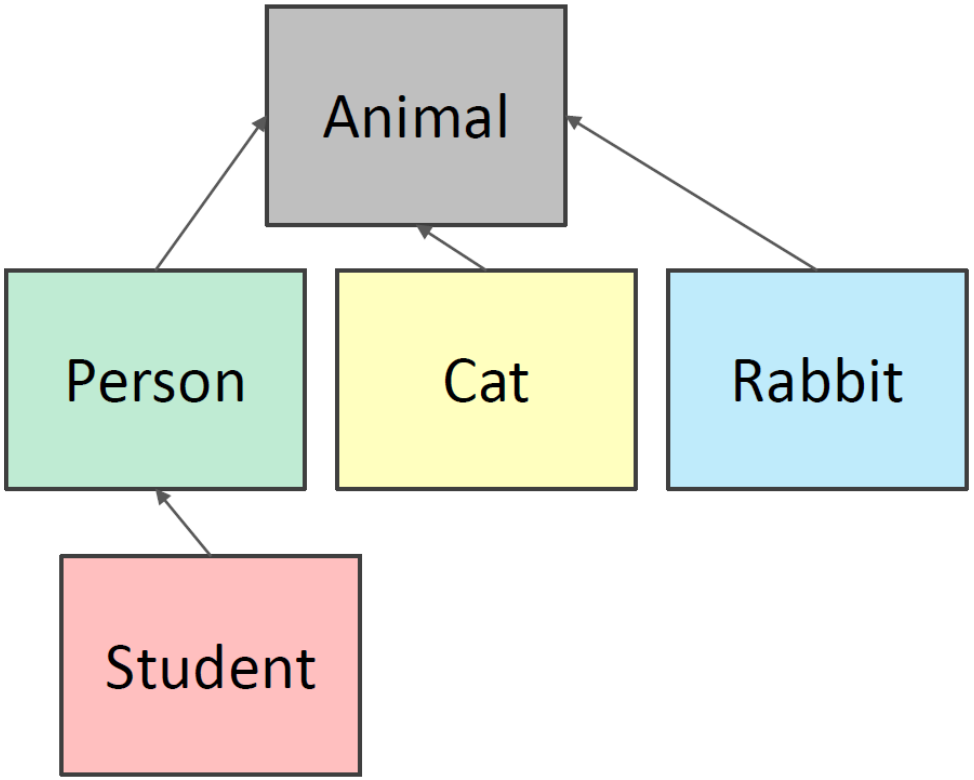
Inheritance

- Father Daugh

All have method work()
From class Human

Inheritance

- **parent class** (superclass)
- **child class** (subclass)
  - **inherits** all data and behaviors of parent class
  - **add** more **info**
  - **add** more **behavior**
  - **override** behavior



Inheritance: parent class

```
class Animal(object):
    def __init__(self, age):
        self.age = age
        self.name = None
    def get_age(self):
        return self.age
    def get_name(self):
        return self.name
    def set_age(self, newage):
        self.age = newage
    def set_name(self, newname=""):
        self.name = newname
    def __str__(self):
        return "animal:"+str(self.name)+":"+str(self.age)
```

- everything is an object
- class `object` implements basic operations in Python, like binding variables, etc

INHERITANCE: SUBCLASS

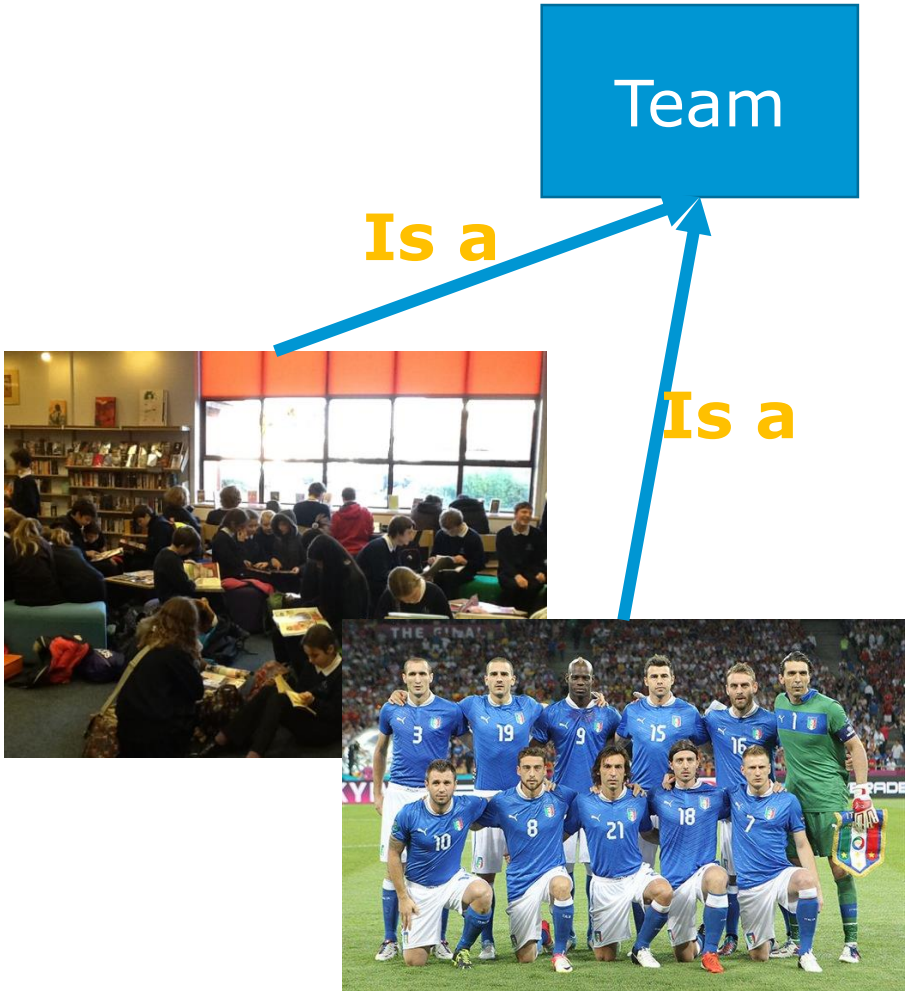
inherits all attributes of `Animal`:
`__init__()`
`age, name`
`get_age(), get_name()`
`set_age(), set_name()`
`__str__()`

```
class Cat(Animal):
    def speak(self):
        print("meow")
```

add new functionality via speak method

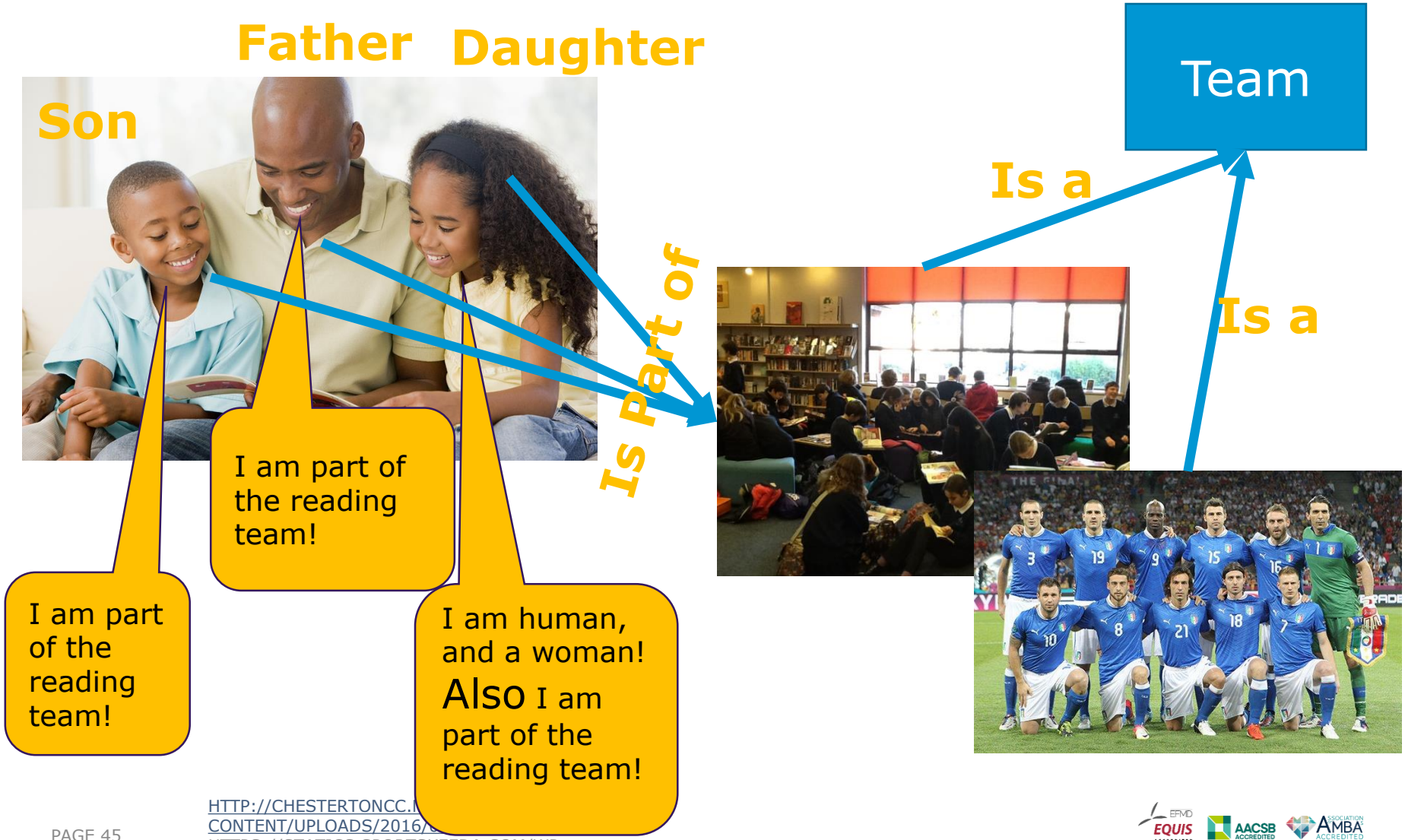
- add new functionality with `speak()`
  - instance of type `Cat` can be called with new methods
  - instance of type `Animal` throws error if called with `Cat`'s new method
- `__init__` is not missing, uses the `Animal` version

“Is a”, “is part of”



HTTP://CHESTERTONCC.NET/WP-CONTENT/UPLOADS/2016/05/WS-GENERAL-3.JPG
HTTPS://STATICS.SPORTSKEEDA.COM/WP-CONTENT/UPLOADS/2015/04/ITALY-1428968210.JPG

“Is a”, “is part of” (composition)



HTTP://CHESTERTONCC.
CONTENT/UPLOADS/2016/
HTTPS://STATICS.SPORTSKEEDA.COM/WP-CONTENT/UPLOADS/2015/04/ITALY-1428968210.JPG

“Is a”, “is part of”

- Is a
  - This is inheritance
- Is part of
  - This is an attribute of the class!

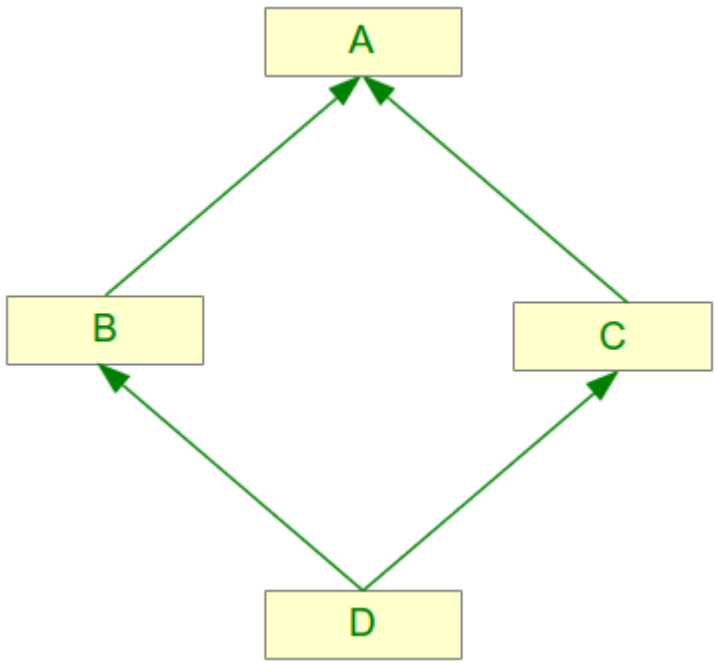
Calendar clock!

Multiple inheritance

```
class CalendarClock(Clock, Calendar):
    def __init__(self,day, month, year, hour, minute, second):
        Clock.__init__(self,hour, minute, second)
        Calendar.__init__(self,day, month, year)
    def tick(self):
        """
        advance the clock by one second
        """
        previous_hour = self._hours
        Clock.tick(self)
        if (self._hours < previous_hour):
            self.advance()
```


HTTPS://WWW.PYTHON-COURSE.EU/PYTHON3_MULTIPLE_INHERITANCE.PHP

Problems with it! Deadly diamond problem!




```python
class A:
    def m(self):
        print("m of A called")

class B(A):
    def m(self):
        print("m of B called")

class C(A):
    def m(self):
        print("m of C called")

class D(B,C):
    def m(self):
        print("m of D called")
```

Polymorphism:
Overriding and overloading. Where is what?

HUMAN

Father

Daughter

Son

write()

I have my own writing style!

earn_money()

I have family now, no more playing poker!

As every human I can

speak()

But, I can also change

speak(language, accent)

Polymorphism

- if class B inherits from class A, it doesn't have to inherit everything about class A; it can do some of the things that class A does differently
- using function/operator in different ways for different types

```
class Animal:
    def __init__(self, name):    # Constructor of the class
        self.name = name
    def talk(self):              # Abstract method, defined by convention only
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof!'
```

Exercise 2! Inheritance

1. Make a class **Human**
2. Class Human has a method **speak()** (prints something on the screen)
3. Make a class **Student** that **inherits** from Human (and overloads the **speak()** method)
4. Make a class **BachelorStudent** that also **overloads** method **speak()** with it's version

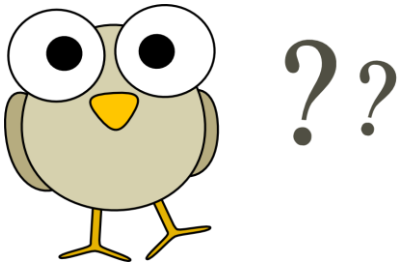
https://i.ytimg.com/vi/Z91tys7dYp4/hqdefault.jpg

Exercise 2! Inheritance

5. Make a class **Teacher** that has a method **speak()**
6. Make a class **PhD** that **inherits Teacher and** a **Student** class and **speak()** what Teacher and Student would speak!

https://i.ytimg.com/vi/Z91tys7dYp4/hqdefault.jpg

KAHOOT quiz!



HTTPS://WWW.SUEHAMPTONAUTHOR.CO.UK//WP-CONTENT/UPLOADS/2014/11/GOOGLEY-EYE-BIRDIE-HAS-QUESTIONS.PNG

Recap today! It is getting *CLASSY*!

- Class
- OOP
- What else do you remember?

Homework!

- Python jupyter notebook will be provided in email!

- **Deadline 14 April 9pm!**

HTTPS://WWW.GOOGLE.AT/IMGRES?IMGURL=HTTPS://US.123RF.COM/450WM/RACORN/RACORN1308/RACORN130802183/21246405-PORTRAIT-OF-HAPPY-PREADOLESCENT-STUDENT-SITTING-AT-DESK-IN-CLASSROOM.JPG%3FVER%3D6&IMGREFURL=HTTPS://WWW.123RF.COM/PHOTO_21171027_YOUNG-BOY-GIVING-A-THUMBS-UP-GESTURE-OF-APPROVAL-WHILE-STANDING-WITH-HIS-TEACHER-IN-THE-CLASSROOM-SE.HTML&H=300&W=450&TBNID=CE1DGGYN11UEIM&TBNH=183&TBNW=275&USG=__EZD-J8X6-KB6DS3HKYH3IPSZTEA=&DOCID=8MNLOK4KAI6UWM

See you next Tuesday!

Wrapping up **classes**!