WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

EFMD
EQUIS
ACCREDITED

Principles of Software Programming: Structured and OOP paradigms

Svitlana Vakulenko, MSc.

WS 2017

Acknowledgement: slides adopted from Dr. Jürgen Umbrich GZP SS 2014 lecture slides

This Episode

- 13:00-15:45
- Structured programming in Python
  - Branching (if-else)
  - Iteration (for, while)
- Object Oriented Programming (OOP):
  - Classes
  - Objects
  - Methods
  - Inheritance
  - UML

Programming paradigms

- **Structured** programming: all programs are seen as composed of control structures
- **Object-oriented** programming (OOP):
  - Java, C++, C#, **Python**

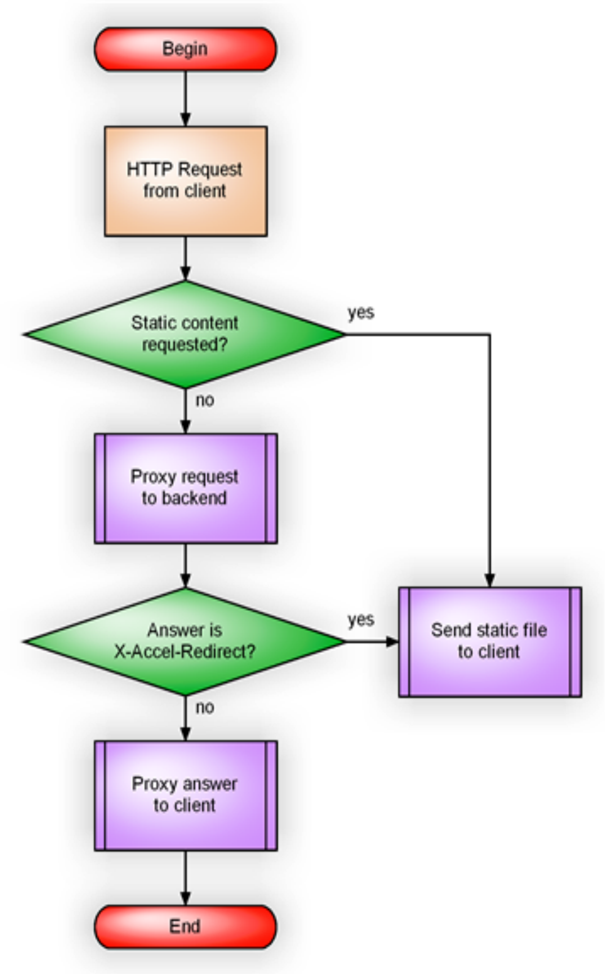
- **Functional** programming:
  - Clojure, Haskell
- **Logic** programming based on formal logic:
  - Prolog, Answer set programming (ASP), Datalog

https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms

http://docs.huihoo.com/nltk/0.9.5/en/ch05.html https://dl.acm.org/citation.cfm?id=278289

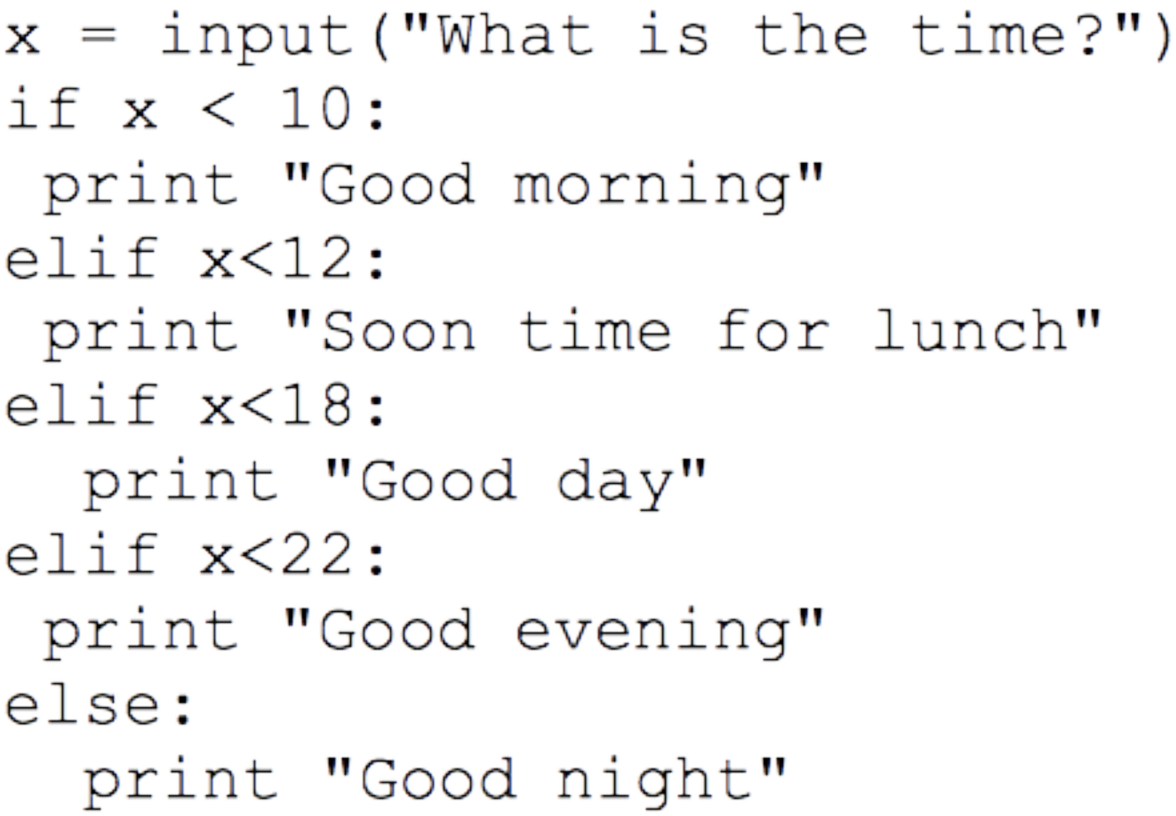
Control flow

- Algorithm - sequence of commands (computation steps)



https://python.swaroopch.com/control_flow.html

Conditions



```
x = input("What is the time?")
if x < 10:
 print "Good morning"
elif x<12:
 print "Soon time for lunch"
elif x<18:
  print "Good day"
elif x<22:
 print "Good evening"
else:
  print "Good night"
```

http://www.pythonforbeginners.com/basics/python-if-elif-else-statement/

Data Types

- **Boolean:** `True` and `False`
- Numeric Types — int, float, long
- String

http://www.diveintopython3.net/native-datatypes.html

https://docs.python.org/2/library/stdtypes.html

Operators

- **Comparison**: >, <, ==, !=, >=, <=
- Arithmetic: +, -, *, /, %, **, //
- Assignment: =, +=, -=,*=,/=, %=, **=, //=
- **Logical**: and, or, not

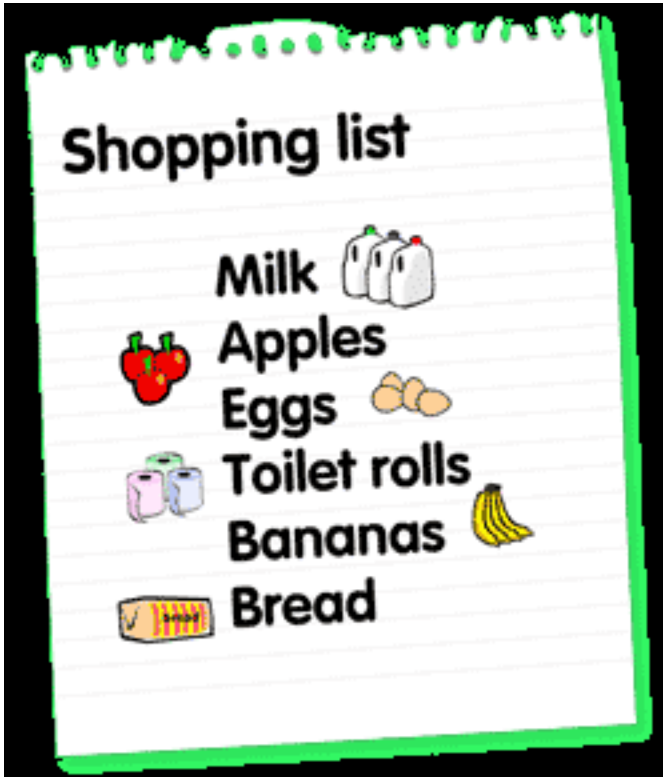
http://www.tutorialspoint.com/python/python_basic_operators.htm

Warm up: Hello, World!

Ex.1: ATM PIN

https://pixabay.com/en/atm-pin-number-field-withdraw-cash-1524869/

Data Structure: List



Shopping list

- Milk
- Apples
- Eggs
- Toilet rolls
- Bananas
- Bread

http://www.bbc.co.uk/scotland/education/wwww/buildings/standard/shopping/?item=list

Data Structure: List

```
shopping_list = ['Milk', 'Apples',
'Eggs', 'Toilet rolls', 'Bananas',
'Bread']
```

http://www.bbc.co.uk/scotland/education/wwww/buildings/standard/shopping/?item=list

List Slicing

```
shopping_list = ['Milk', 'Apples',
'Eggs', 'Toilet rolls', 'Bananas',
'Bread']

shopping_list[1]
shopping_list[-1]
shopping_list[0:-1]
```

http://www.bbc.co.uk/scotland/education/wwww/buildings/standard/shopping/?item=list
http://pythoncentral.io/how-to-slice-listsarrays-and-tuples-in-python/

List Functions

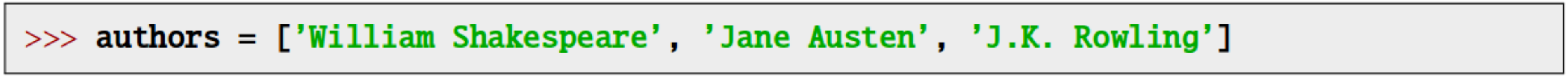
Shopping list
Milk
Apples
Eggs
Toilet rolls
Bananas
Bread

```
shopping_list = ['Milk', 'Apples',
'Eggs', 'Toilet rolls', 'Bananas',
'Bread']

len(shopping_list)

'Milk' in shopping_list
```

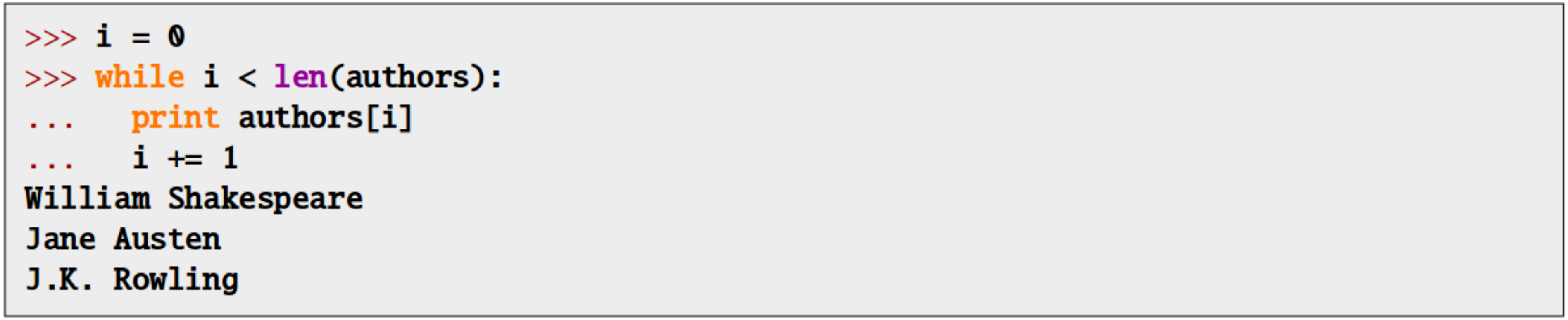
http://www.bbc.co.uk/scotland/education/wwww/buildings/standard/shopping/?item=list
https://www.programiz.com/python-programming/list

Warm up: Hello, World!

Loops



```
>>> authors = ['William Shakespeare', 'Jane Austen', 'J.K. Rowling']
```



```
>>> i = 0
>>> while i < len(authors):
...    print authors[i]
...    i += 1
William Shakespeare
Jane Austen
J.K. Rowling
```

```
for x in shopping_list:
    print (“I need ” + x)
```

http://www.icrar.org/__data/assets/pdf_file/0008/1436615/challenge09b-notes3.pdf

Create int list

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(5, 10)
[5, 6, 7, 8, 9]

>>> range(0, 10, 3)
[0, 3, 6, 9]
```

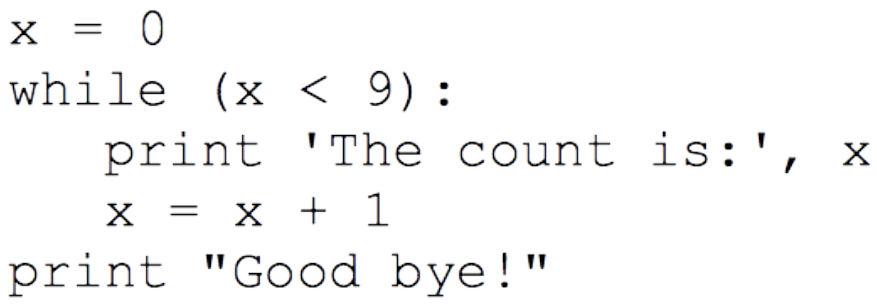
http://docs.python.org/release/1.5.1p1/tut/range.html

Counting Loop

- **for** loop

```
for x in range(9):
        print 'The count is:', x
print "Good bye!”
```

- **while** loop



```
x = 0
while (x < 9):
   print 'The count is:', x
   x = x + 1
print "Good bye!"
```

http://www.tutorialspoint.com/python/python_while_loop.htm

https://wiki.python.org/moin/WhileLoop

https://wiki.python.org/moin/ForLoop

Ex.1: ATM PIN

https://pixabay.com/en/atm-pin-number-field-withdraw-cash-1524869/

Ex.2: Hey-You

"Write a program that prints the numbers from 1 to 100. But for multiples of three print “Hey” instead of the number and for the multiples of five print “You”. For numbers which are multiples of both three and five print “HeyYou”.”

http://c2.com/cgi/wiki?FizzBuzzTest

Ex.2: Fizz-Buzz

"Write a program that prints the numbers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”."

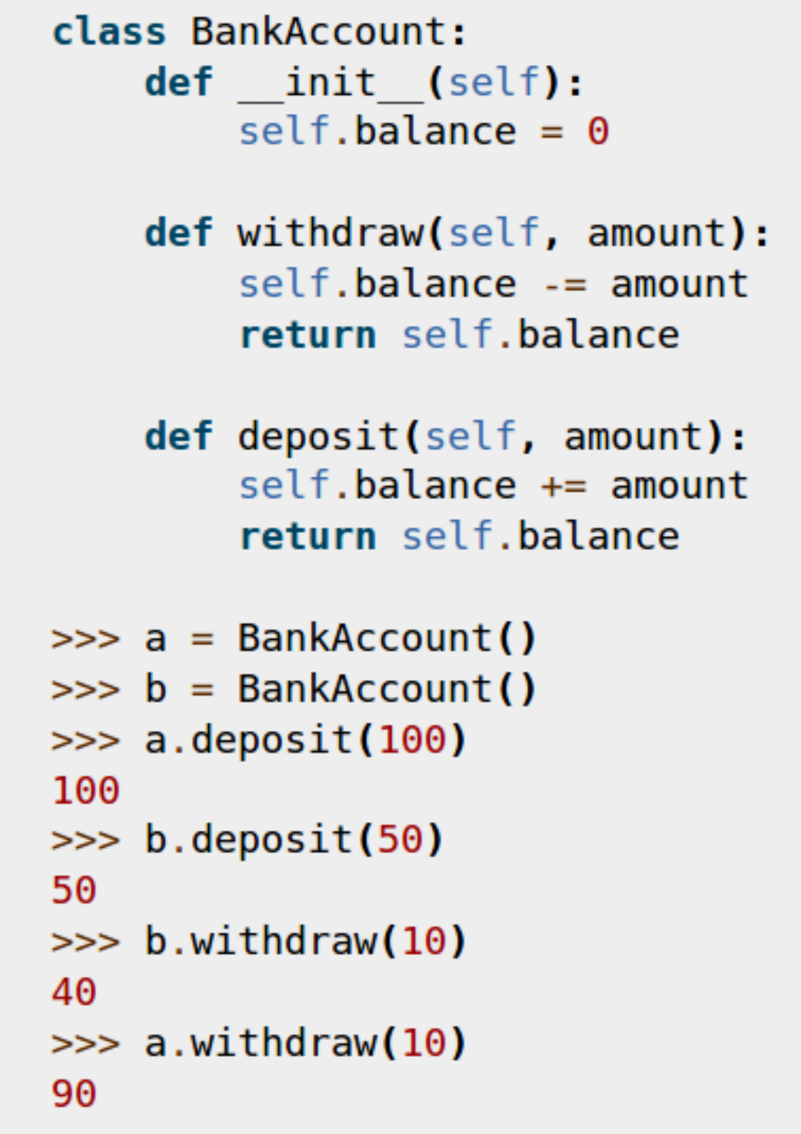
http://c2.com/cgi/wiki?FizzBuzzTest

Program for success

1. **problem**_solved = False
2. **think()** # about the problem and how to start
3. **if** do_not_know_where_to start **or** do_not_understand:
   - **say()** # tell me or ask your fellow student
4. **while not** problem_solved:
   - **try**_figure_out_solution() # do not give up!
   - **if** got_stuck:
     - **say()** # tell me or ask your fellow student
5. **say**(“I AM AWESOME!”)

Object Oriented Programming (OOP)

- Classes
  - attributes
  - methods
- Objects - instances of classes



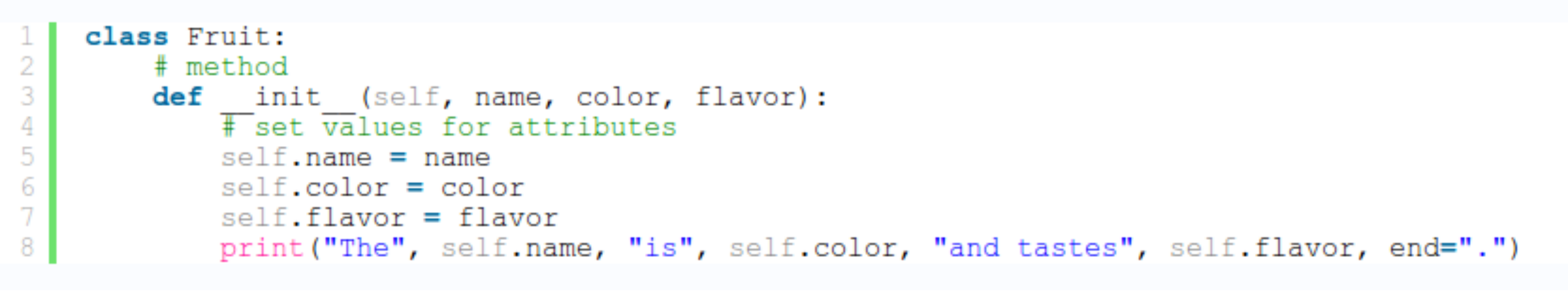
```
class BankAccount:
    def __init__(self):
        self.balance = 0

    def withdraw(self, amount):
        self.balance -= amount
        return self.balance

    def deposit(self, amount):
        self.balance += amount
        return self.balance


>>> a = BankAccount()
>>> b = BankAccount()
>>> a.deposit(100)
100
>>> b.deposit(50)
50
>>> b.withdraw(10)
40
>>> a.withdraw(10)
90
```

https://docs.python.org/3/tutorial/classes.html#classes

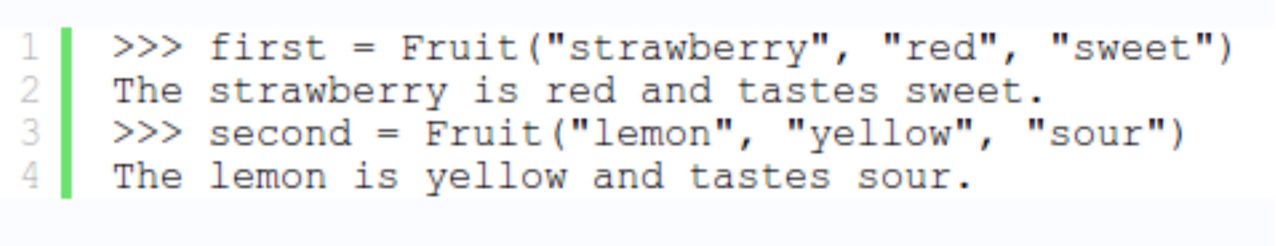
http://anandology.com/python-practice-book/object_oriented_programming.html

Constructor

- Initialisation method
- Classes act as factories for new instances of themselves
- Class is a callable object (like a function), with the call being the constructor
- Calling the class returns an instance of that class



```python
class Fruit:
    # method
    def __init__(self, name, color, flavor):
        # set values for attributes
        self.name = name
        self.color = color
        self.flavor = flavor
        print("The", self.name, "is", self.color, "and tastes", self.flavor, end=".")
```

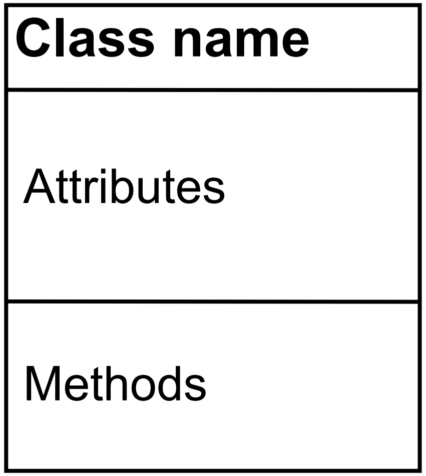


```
>>> first = Fruit("strawberry", "red", "sweet")
The strawberry is red and tastes sweet.
>>> second = Fruit("lemon", "yellow", "sour")
The lemon is yellow and tastes sour.
```

https://en.wikipedia.org/wiki/Constructor_%28object-oriented_programming%29#Python

http://learntofish.wordpress.com/2011/12/07/tutorial-object-oriented-programming-in-python-part-3/?relatedposts_exclude=1352

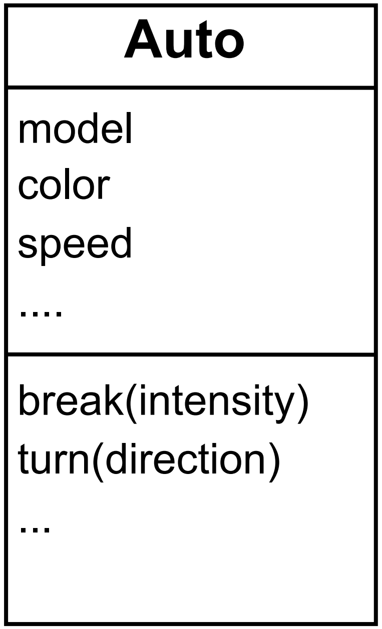
UML: Class diagram



Class name

Attributes

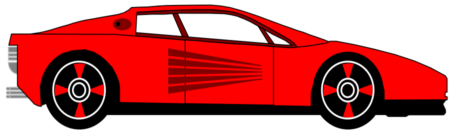
Methods

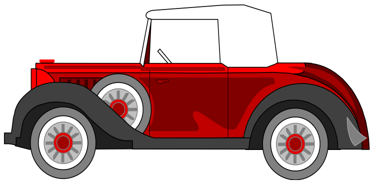


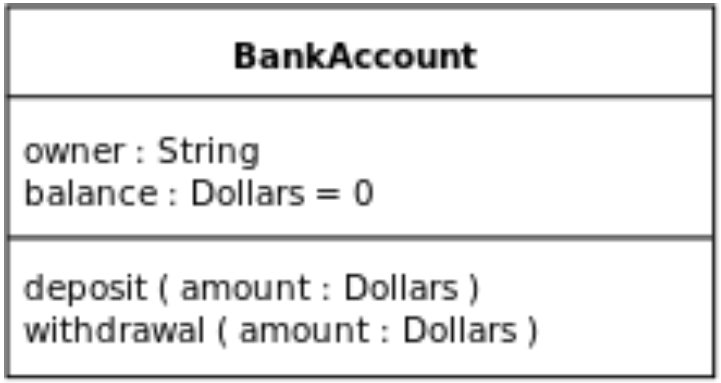
Auto

model
color
speed
....

break(intensity)
turn(direction)
...







BankAccount

owner : String
balance : Dollars = 0

deposit (amount : Dollars)
withdrawal (amount : Dollars)

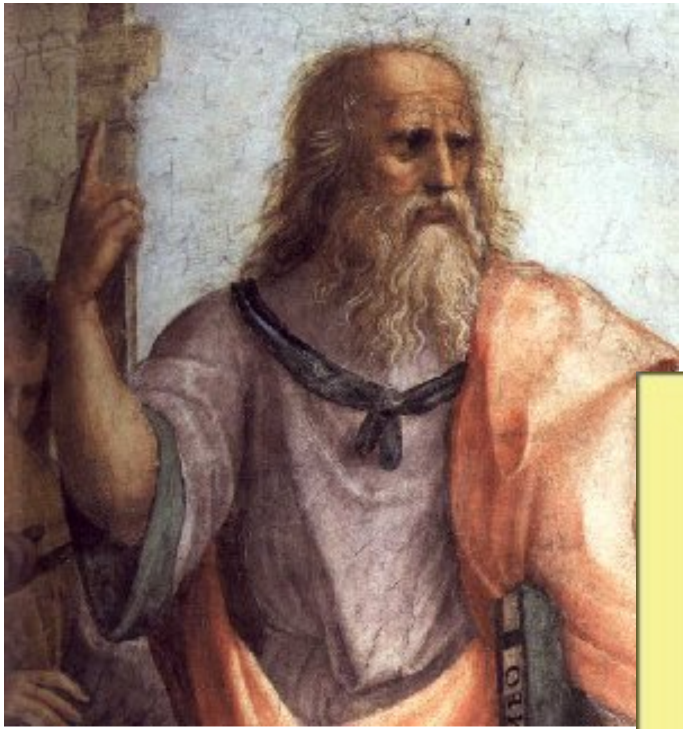
https://en.wikipedia.org/wiki/Class_diagram

OOP principles

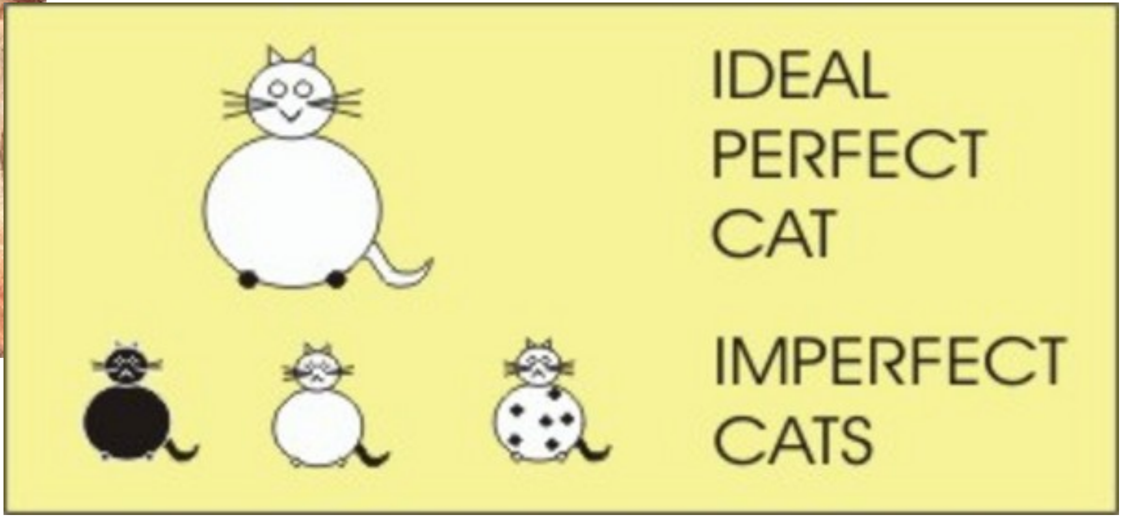
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

https://anampiu.github.io/blog/OOP-principles/

Abstraction



http://commons.wikimedia.org/wiki/Main_Page



http://www.proprofs.com/quiz-school/story.php?title=greekology

Encapsulation

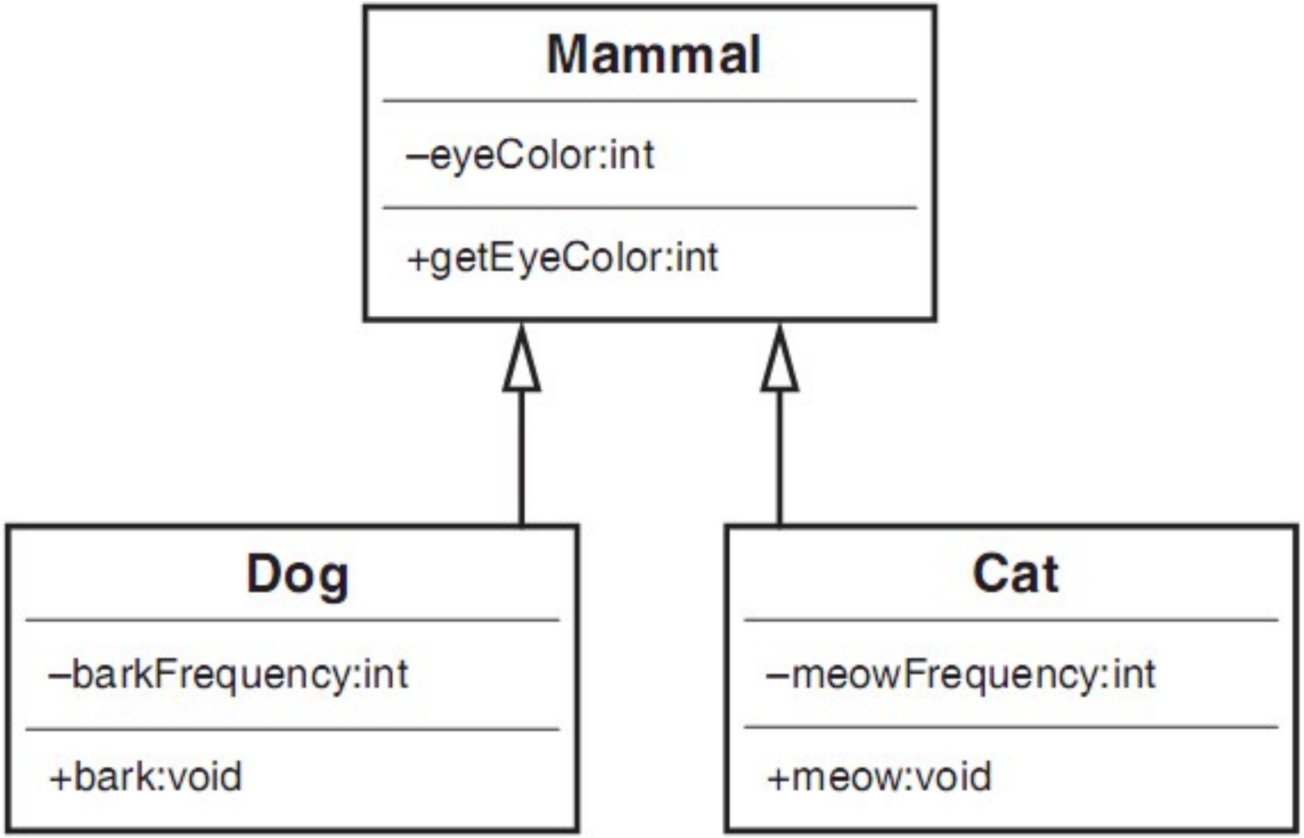
- *restrict access* to methods and variables (visibility) to prevent the data from being modified by accident
  - **public** accessible from anywhere
  - **private** can be accessed only from the same class: __ prefix

```python
class Car:

    def __init__(self):
        self.__updateSoftware()

    def drive(self):
        print 'driving'

    def __updateSoftware(self):
        print 'updating software'

redcar = Car()
redcar.drive()
#redcar.__updateSoftware()  not accesible from object.
```

https://pythonspot.com/en/encapsulation/

Inheritance

- Class hierarchy



http://guruzon.com/1/oop-concepts/inheritance-and-composition/what-is-inheritance-example-tutorial-uml-notation

Inheritance

```
class Animal:

    def __init__(self):
        print("Animal created")

    def whoAmI(self):
        print("Animal")

    def eat(self):
        print(“Eating")
```

```
class Dog(Animal):

    def __init__(self):
        super().__init__()

        print("Dog created")

    def whoAmI(self):
        print("Dog")

    def bark(self):
        print("Woof!")
```

```
d = Dog()
d.whoAmI()
d.eat()
d.bark()
```

http://zetcode.com/lang/python/oop/

Abstract class

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

https://stackoverflow.com/questions/3724110/practical-example-of-polymorphism

https://zaiste.net/posts/abstract_classes_in_python/

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

```python
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!'
```

http://zetcode.com/lang/python/oop/

https://stackoverflow.com/questions/3724110/practical-example-of-polymorphism

Ex.3: Classroom

http://www.edudemic.com/the-teachers-guide-to-polling-in-the-classroom/

Next Episode

- Monday 6 November 13:00-15:45 D2.0.031
- **Data structures** & operations:
  - list: Stack & Queue
  - string
  - set
  - tuple
  - dictionary (hash table)