EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Science
Department of Computer Science

Vorkurs Programming - Informatik 4 Lifescientists

Introduction to Object-Oriented Programming

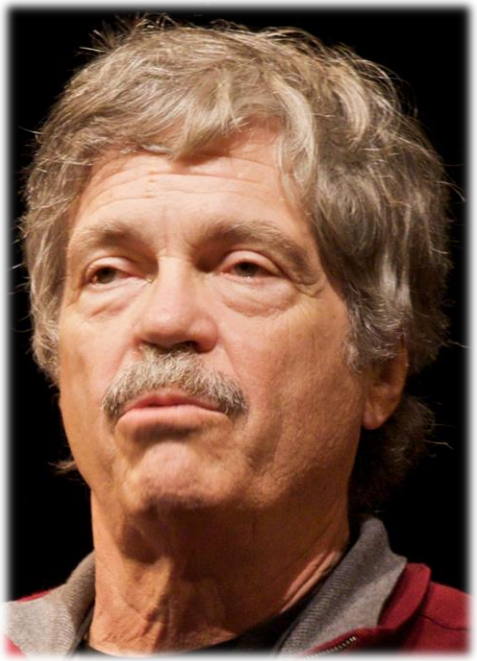
https://github.com/BioInfPrep/python_oop

26/10/2020, Samuel Wein, Oliver Alka

Concepts

What is an Object?

What is an Object?



"I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages"
-Alan Levy

- **Attributes**. Attributes define characteristics, such as the type of a cell, or its pH.
- **Methods**. Methods instruct the object to perform tasks, such as phagocytose or undergo mitosis.
- **Events**. Events fire when something happens to an object, for example, antigen presentation. (not natively present in Python)

Why do we use Objects?

- **Simplify your life.** OOP helps make it possible for you to communicate what you want the computer to do in a way that the computer can understand.
- **Define ideas consistently.** OOP creates a common way to express what you want to do so that others will understand.
- **Specify the manner used to create objects.** Each object uses specific techniques to define attributes, methods, and events.
- **Write code with less effort.** Creating an animal object means that you specify the things that make animals different from other objects only once.

Concepts of Objects

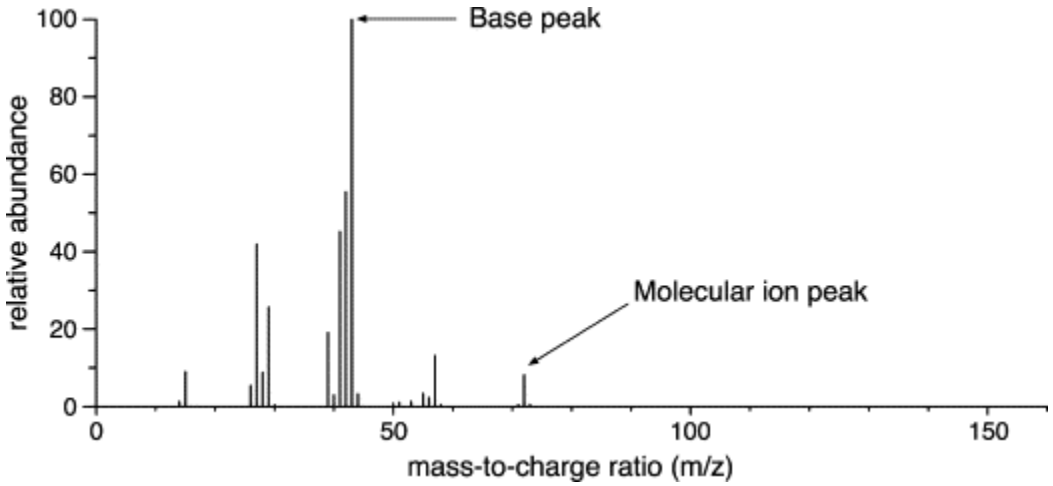
What is a

Some key concepts of OOP?

- **Classes.** A class is a template for creating objects. Eg. Cell may be a class, “that cell over there” is an object
- **Encapsulation.** Some characteristics of an object are accessible from other objects some are only accessible to the object itself
- **Inheritance.** Classes can be defined to inherit methods and attributes from another class. Eg erythrocyte inherits cell membranes, and cytoplasm
- **Polymorphism.** Functions acting on a parent class can be applied to its daughter classes

An aside about

MSPeaks



- Our preferred example class
- Represents a reading from a mass spectrometer
- Has intensity and mass-to-charge ratio (MZ)

What is a Class?

```python
# Class
class MSPeak:
    pass # null operation - placeholder


# if module is imported by another program main is not used.
# https://stackoverflow.com/questions/419163/what-does-if-name-main-do
if __name__ == "__main__":
    x = MSPeak()
    y = MSPeak()
    y_alias = y # reference to y

    print(x==y)
    print(y==y_alias)
```

What are Attributes?

```
class MSPeak:
    def __init__(self,
                 mz=None,
                 intensity=None):
        self.mz = mz
        self.intensity = intensity
```

What are Methods?

```python
# initalize the instance right after creation
class MSPeak:
    def __init__(self,
                 mz=None,
                 intensity=None):
        self.mz = mz
        self.intensity = intensity

    def show_peak(self):
        if self.mz and self.intensity:
            print("The peak can be found at mz " + str(self.mz) + " with an intensity of " + str(self.intensity))
        else:
            print("Error: Either mz or intensity or both values are missing - could not show the peak!")

if __name__ == "__main__":
    # x = MSPeak(250, 60000) # using the __init__ method "constructor"
    x = MSPeak()
    x.show_peak()
    x.mz = 250
    x.intensity = 600000
    x.show_peak()
```

What is Encapsulation?

```python
# public, protected, private attributes
class A():

    def __init__(self):
        self.__priv = "I am private" # inaccessible & invisible - can only be mutated inside the class definition
        self._prot = "I am protected" # should not be used outside of the class definition (unless in a subclass)
        self.pub = "I am public" # used freely inside/outside class definition

    def set_private_attribute(self, __priv):
        self.__priv = __priv

x = A()

print(x.pub)
x.pub = x.pub + " - changed freely!"
print(x.pub)

print(x._prot)

#x.__priv
```

What is Inheritance?

```python
# class methods vs. static methods - inheritance
class Pet: # base class
    _class_info = "pet animals"

    @classmethod
    def about(cls):
        print("This class is about " + cls._class_info + "!")

class Dog(Pet):
    _class_info = "man's best friends" # overload

class Cat(Pet):
    _class_info = "all kinds of cats" # overload

Pet.about()
Dog.about()
Cat.about()
```

What are
Static Methods?

```python
# class methods vs. static methods - inheritance

class Pet: # base class
    _class_info = "pet animals"

    @staticmethod
    def about():
        print("This class is about " + Pet._class_info + "!")


class Dog(Pet):
    _class_info = "man's best friends" # overload

    #@staticmethod
    #def about(): # overlad
    #    print("This class is about " + Dog._class_info + "!")

class Cat(Pet):
    _class_info = "all kinds of cats" # overload

Pet.about()
Dog.about()
Cat.about()

# no way to differentate what kind of class it really is!
```

What are
Class Methods?

```python
# class methods vs. static methods - inheritance
class Pet: # base class
    _class_info = "pet animals"

    @classmethod
    def about(cls):
        print("This class is about " + cls._class_info + "!")

class Dog(Pet):
    _class_info = "man's best friends" # overload

class Cat(Pet):
    _class_info = "all kinds of cats" # overload

Pet.about()
Dog.about()
Cat.about()
```

Questions?

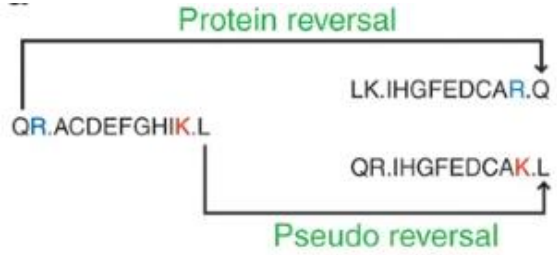
Project:

- Build a decoy database generator

What is a

Decoy database?

- False discovery rate in proteomics is determined by searching for known-bad sequences (decoys).
- Simplest way to generate decoys is to reverse input sequences.



Random

Residue	Frequency
A	0.070
C	0.023
D	0.046
E	0.070
F	0.036

Markov

Residue	Frequency
A	0.047
C	0.003
D	0.043
E	0.087
F	0.020

[STEV]+