



---

## Introduction to Object-Oriented Programming

[https://github.com/BioInfPrep/python\\_oop](https://github.com/BioInfPrep/python_oop)

13/10/2021, Mathias Witte Paz, Dominic Boeck



# 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, one of the fathers of OOP and inventor of FLEX (programming language)



- **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





# Some key concepts of OOP

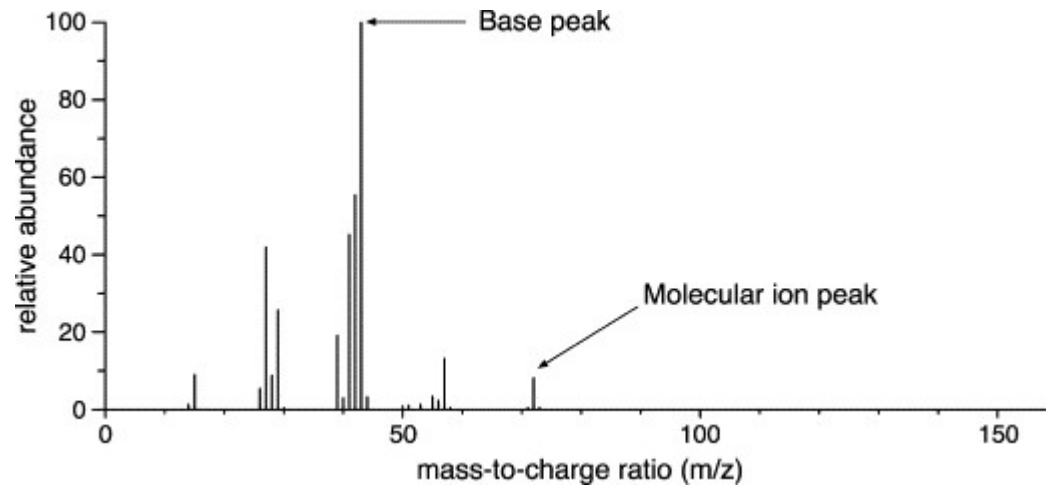


- 
- **Classes.** A class is a template for creating objects. Eg. Cell may be a class, “that cell over there” is an object
  - **Encapsulation.** Some characteristics of an object are accessible from other objects some are only accessible to the object itself
  - **Inheritance.** Classes can be defined 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

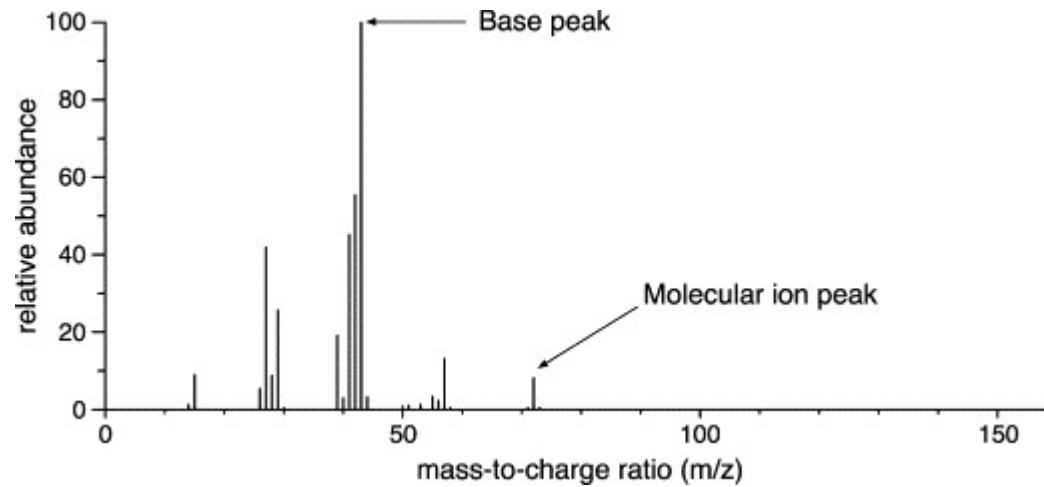


---

# An aside about MSPeaks



- Our preferred example class
- Represents a reading from a mass spectrometer



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



---

Let's start with the code!

**Folder: code\_chunks**



# What is a Class?

```
# 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?

```
# initialize 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()
```



---

## Attributes and Methods

# Hands on

- **Task 1:** Add an attribute for the retention time (rt) and one the name of each peak. Include the printing of these variables in the existing method.



# What is Encapsulation?

```
# 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
```



## Encapsulation

# Hands on

- **Task 1:** From your previous task, encapsulate the attributes to be private.
- **Task 2:** Create setter and getter functions for each attribute.



# What is Inheritance?

```
# 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?

```
# 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(): # overload
        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 differentiate what kind of class it really is!
```



# What are Class Methods?

```
# 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?



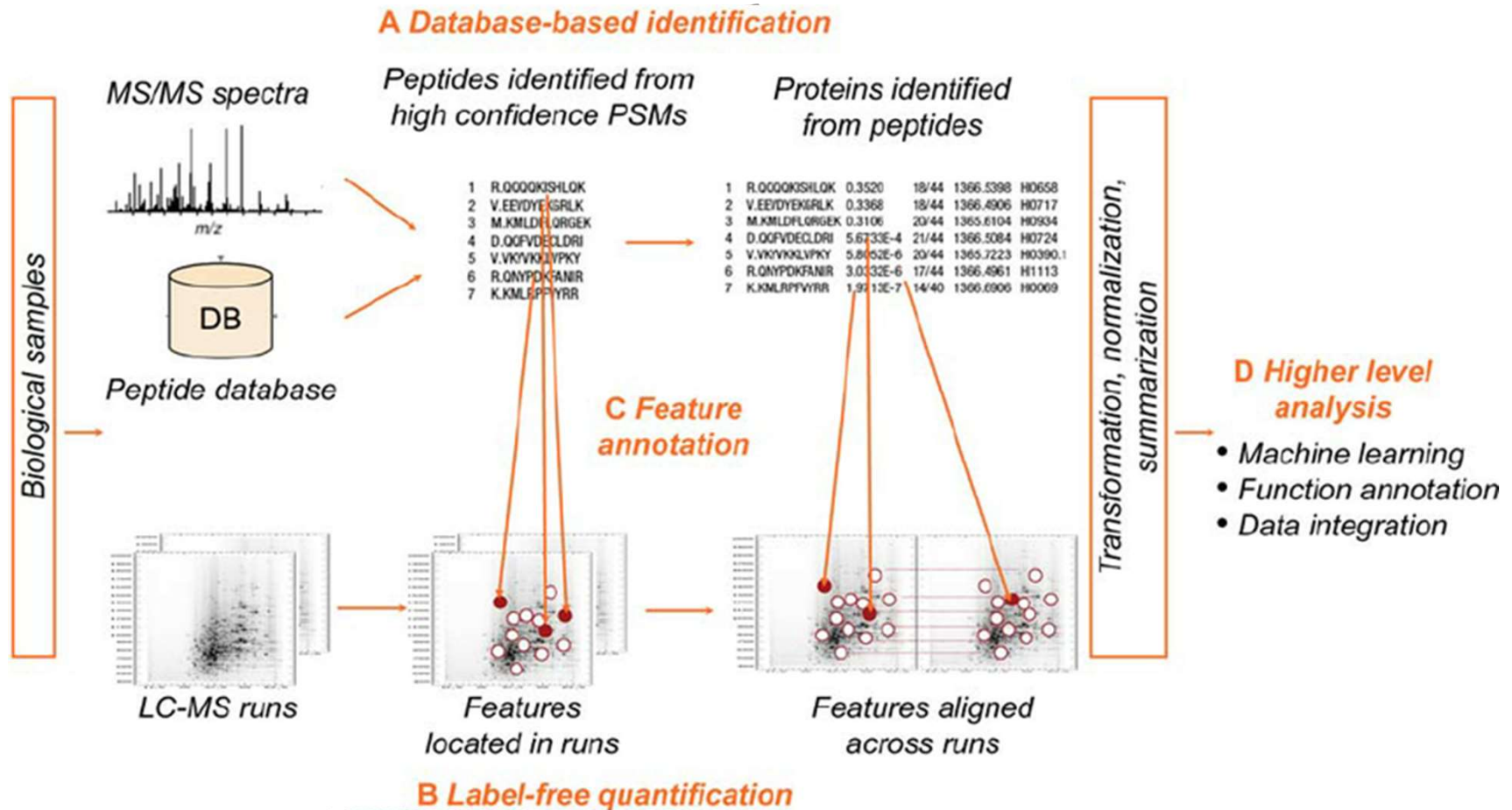


Your turn!

# Small project on proteomics



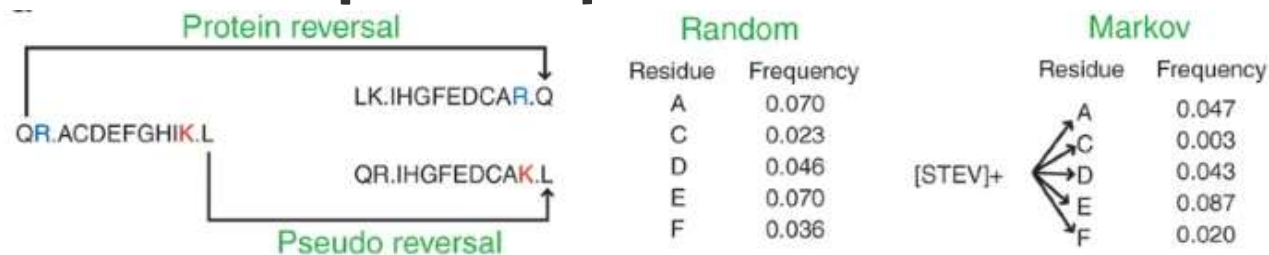
# What is proteomics?





# 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.





# Project:

- Build a decoy database generator
- Use the OOP concepts



# Class diagram

Visual Paradigm Online Free Edition

