

# Algorithms for Bioinformatics

## *Python* *Object-Oriented Programming*

Pedro G. Ferreira  
[dCC] @ Faculty of Sciences University of Porto

- OOP
- Classes
- Special Methods
- Inheritance
- Exercises

# Object-Oriented Programming

- OOP = objects + classes + inheritance. It provides modularity, reusability and interpretability of the code.
- Classes provide a representation of an object or entity and are the central concept in OOP.
- Classes are composed of :
  - i) data content to be handled (*attributes*).
  - ii) *methods* and functions that specify the behavior and manipulate the contents.

Camel case notation: first letter in class name in upper case. Attributes in lower case (eventually separated by underscore).

```
1. class ClassNameExample:  
2.     """ class documentation goes here """  
3.     pass # coed goes here
```

- The class is the template. The objects is the instance of the class.

- Class that represents and processes biological sequences. Two attributes: sequence and sequence biological type.

```
1. class MySeq:
2.     """Biological sequence class"""
3.     def __init__(self, seq, seq_type = "DNA"):
4.         self.seq = seq
5.         self.seq_type = seq_type
6.
7.     def print_sequence(self):
8.         print ("Sequence: " + self.seq)
9.
10.    def get_seq_biotype (self):
11.        return self.seq_type
12.
13.    def show_info_seq (self):
14.        print ("Sequence: " + self.seq + " biotype: " + self.seq_type)
15.
16.    def count_occurrences(self, seq_search):
17.        return self.seq.count(seq_search)
```

# Class

- **Self** is not a reserved keyword but a strong convention. It appears as the first argument of the methods in the class. It also refers to the created object instance in the constructor.
- The `__init__` method is the initializer of the state of the object. All classes should contain this method.
- Object instantiation:

```
1. mseq = MySeq("ATATAGATGATG")
```

- Invoking methods:

```
1. mseq.get_seq_biotype()
```

```
1. s1 = MySeq("ATAATGATAGATAGATGAT")
2. # access attribute values
3. print (s1.seq)
4. print (s1.seq_type)
5. # calling methods
6. s1.print_sequence()
7. print (s1.get_seq_biotype())
```

# Class

- In python the attributes of a class can be directly modified. It is up to the programmer to decide if there should be a method performing this action (with the necessary validation).
- The attributes starting with `__` (double underscore) should be treated as non-public, i.e. not accessed directly from outside the class.

```
1. # the type of the sequence is updated to an invalid biotype
2. # by direct alteration of the attribute
3. s1.seq_type = "time series"
4.
5. # safer alternative: class method to validate update
6. def set_seq_biotype (self, bt):
7.     biotype = bt.upper()
8.     if biotype == "DNA" or biotype == "RNA" or biotype == "PROTEIN":
9.         self.seq_type = biotype
10.    else:
11.        print "Non biological sequence type!"
12.
13. # testing the update of the attribute
14. s1.set_seq_biotype("time series")
15. s1.set_seq_biotype("dna")
```

# Special Methods

- Several methods are shared across different classes. They need to be re-implemented to provide specific behavior. These methods start and end with `__` (double underscore).

```
1. class MySeq:
2.     """--some_code_here--"""
3.     def __len__(self):
4.         return len(self.seq)
5.     def __str__(self):
6.         return self.seq_type + ":" + self.seq
7.     def __getitem__(self, n):
8.         return self.seq[n]
9.     def __getslice__(self, i, j):
10.        return self.seq[i:j]
11.
12.
13. s1 = MySeq("MKKVSJEMSSVPYW", "PROTEIN")
14. print(s1)
15. print(len(s1))
16. print(s1[4])
17. print(s1[2:5])
```

# Inheritance

- If we want to define a class a very similar behavior and information of an existing class but that represents a more specialized behavior, then class inheritance can be used.
- The new inherited class (child) will inherit the attributes and methods from the parent class. The child class can introduce new attributes and methods and can redefine existing methods. The method **super** refers to the parent class.

```
1. class MyNumSeq(MySeq):
2.     def __init__(self, num_seq, seq_type="numeric"):
3.         super().__init__(num_seq, seq_type)
4.
5.     def set_seq_biotype(self, st):
6.         seq_type = st.upper()
7.         if seq_type == "DNA" or seq_type == "RNA" or seq_type == "PROTEIN":
8.             self.seq_type = seq_type
9.         elif seq_type == "NUMERIC" or seq_type == "NUM":
10.            self.seq_type = seq_type
11.        else:
12.            print ("Non-biological or Non-numeric sequence type")
```



# Inheritance

- In an object of a child class we can call new methods or methods from the parent class.

```
1. >>> a = MyNumSeq("123456789")
2. >>> a.seq_type
3. "DNA"
4. >>> a.set_seq_biotype("numeric")
5. >>> a.seq_type
6. "NUMERIC"
7. >>> a.print_sequence()
8. Sequence: 123456789
```

- The code of a class should be organized in a module. Later, the classes from this module can be imported.

```
1. # import specific classes from a module
2. from myseq import MySeq, MyNumSeq
3. # import all classes from a module
4. import myseq
```

- Define a class to represent a rectangle. The attributes should be *height* and *length*.
  - Implement the following methods:
    - Constructor
    - Calculate area
    - Calculate perimeter
    - Calculate the length of diagonal
  - Test the class with multiple instances of different values
  - Implement a child class that extends the class rectangle to represent squares.
  
- Define a class to keep a phone book. It should have a list of name and respective phone numbers.
  - Implement the following methods:
    - Constructor (init)
    - add phone entry
    - Search by name; search by phone number
    - Print the entire book
    - Define a method to copy the phone book information.
  - Extend the previous class to emails
  - Test the class with multiple instances