Algorithms for Bioinformatics
2018/2019

Python
Object-Oriented Programming

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

Outline

- 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).

```python
class ClassNameExample:
    """ class documentation goes here """
    pass # coed goes here
```

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

Class

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

```python
class MySeq:
    """Biological sequence class"""
    def __init__(self, seq, seq_type = "DNA"):
        self.seq = seq
        self.seq_type = seq_type

    def print_sequence(self):
        print ("Sequence: " + self.seq)

    def get_seq_biotype (self):
        return self.seq_type

    def show_info_seq (self):
        print ("Sequence: " + self.seq + " biotype: " + self.seq_type)

    def count_occurrences(self, seq_search):
        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:

```python
mseq = MySeq("ATATAGATGATG")
```

- Invoking methods:

```python
mseq.get_seq_biotype()
```

```python
s1 = MySeq("ATAATGATAGATAGATGAT")
# access attribute values
print (s1.seq)
print (s1.seq_type)
# calling methods
s1.print_sequence()
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.

```
# the type of the sequence is updated to an invalid biotype
# by direct alteration of the attribute
s1.seq_type = "time series"

# safer alternative: class method to validate update
def set_seq_biotype (self, bt):
    biotype = bt.upper()
    if biotype == "DNA" or biotype == "RNA" or biotype == "PROTEIN":
        self.seq_type = biotype
    else:
        print "Non biological sequence type!"

# testing the update of the attribute
s1.set_seq_biotype("time series")
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).

```python
class MySeq:
    """--some_code_here--"""
    def __len__(self):
        return len(self.seq)
    def __str__(self):
        return self.seq_type + ":" + self.seq
    def __getitem__(self , n):
        return self.seq[n]
    def __getslice__(self , i, j):
        return self.seq[i:j]


s1 = MySeq("MKKVSJEMSSVPYW", "PROTEIN")
print(s1)
print(len(s1))
print(s1[4])
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.

```python
class MyNumSeq(MySeq):
    def __init__(self, num_seq, seq_type="numeric"):
        super ().__init__(num_seq , seq_type)

    def set_seq_biotype (self , st):
        seq_type = st.upper()
    if seq_type == "DNA" or seq_type == "RNA" or seq_type == "PROTEIN":
        self.seq_type = seq_type
    elif seq_type == "NUMERIC" or seq_type == "NUM":
        self.seq_type = seq_type
    else:
        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.

```
>>> a = MyNumSeq("123456789")
>>> a.seq_type
"DNA"
>>> a.set_seq_biotype("numeric")
>>> a.seq_type
"NUMERIC"
>>> a.print_sequence()
Sequence: 123456789
```

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

```
# import specific classes from a module
from myseq import MySeq, MyNumSeq
# import all classes from a module
import myseq
```

Exercises

- 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
    - 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