User-defined data types with classes

Python's Object Paradigm

- Recall that Python data types have a type and a variable of that type has a value. Classes define a new type and a variable references an instance or object.

  Type => *Value*

  Class => *Instance*

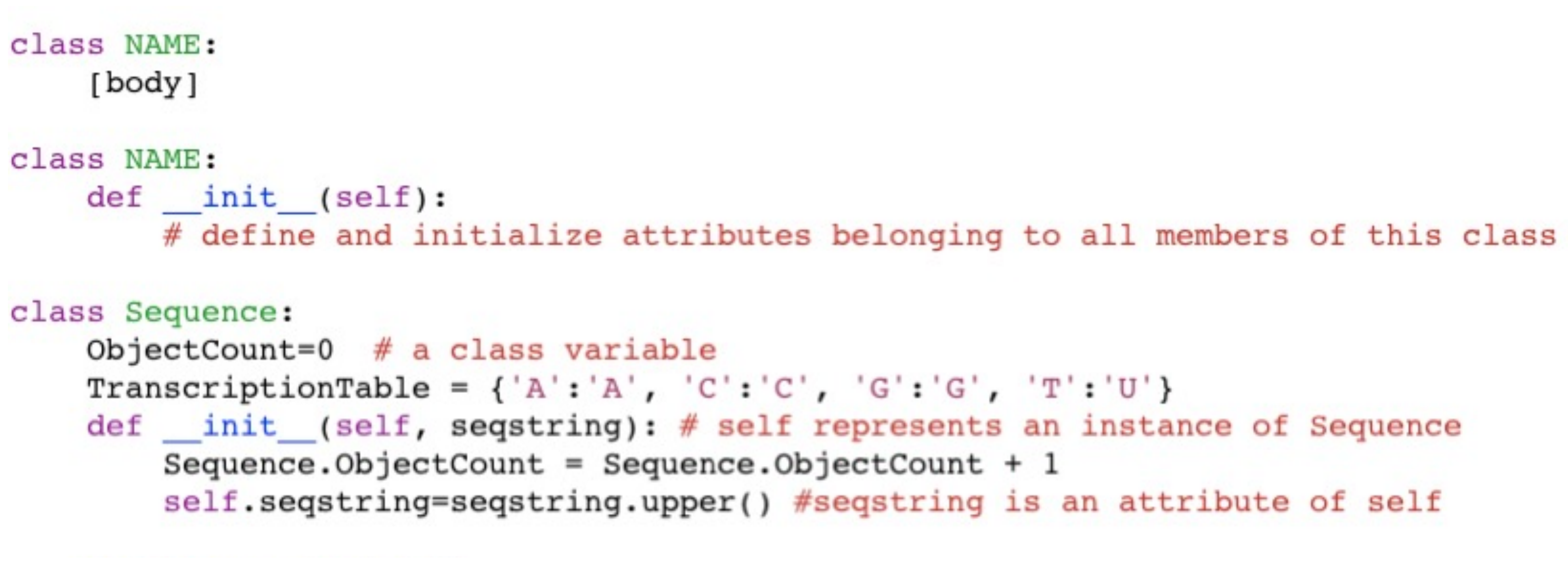
- Python's built-in ***data types*** have their own ***functions*** that can be called on it.
- A ***Class*** is a user-defined data type with its own specific attributes and methods (member functions). Any object that can be abstracted can be a class:

  e.g. person, mammal, animal; sequence, plasmid, protein

<u>Terminology:</u>

- A particular *instantiation* of a class is called an ***instance*** ("object" is essentially used as a synonym for "instance").
- Class instances have their own characteristics (attributes or properties); these are called ***data attributes (or member variables)***.
- ***Methods*** (also called ***member functions)*** are ***functions*** that belong to a specific ***data type*** or ***class*** and define how objects derived from that class "behave".

Creating Python Classes



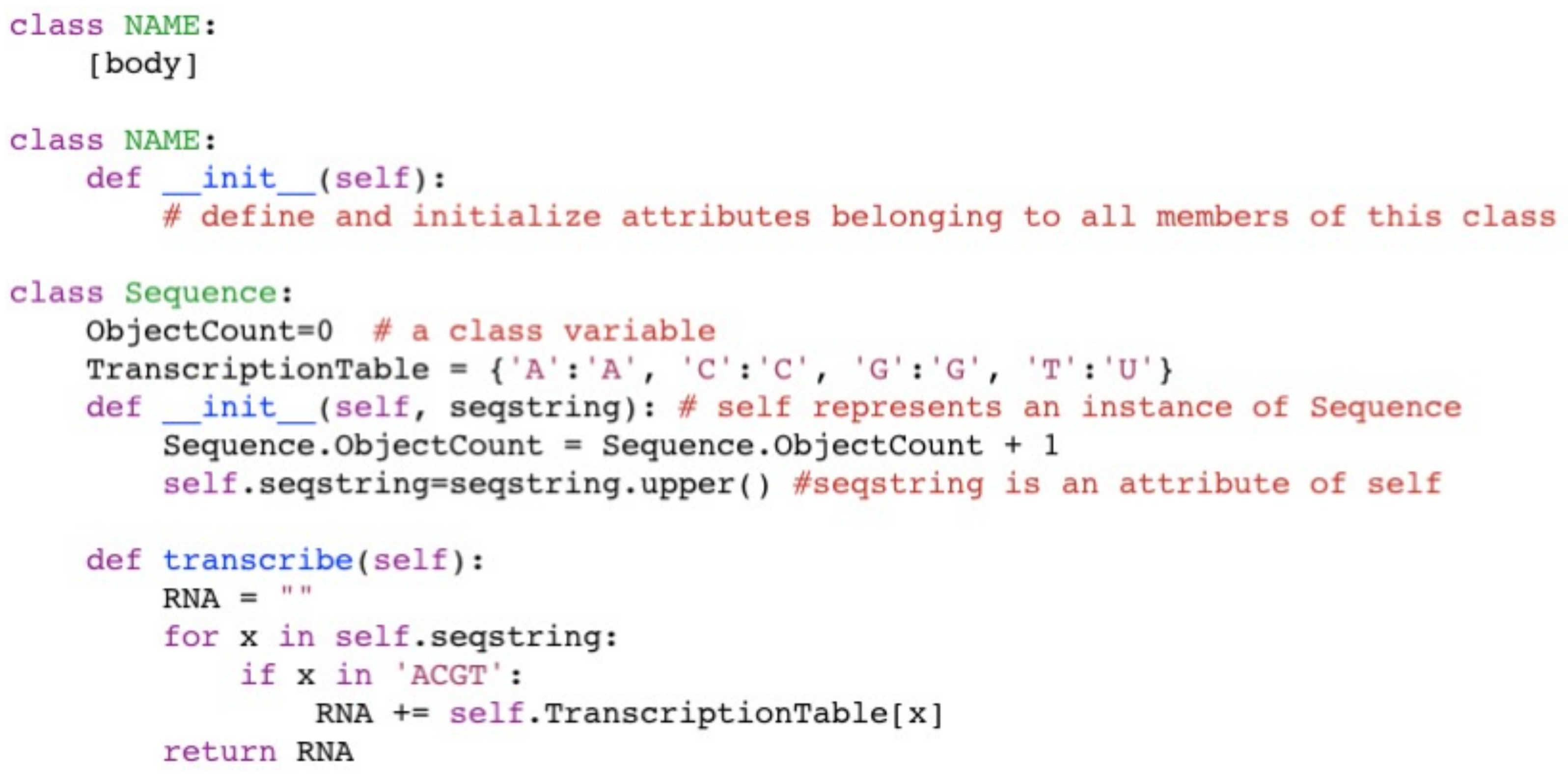
```
class NAME:
    [body]

class NAME:
    def __init__(self):
        # define and initialize attributes belonging to all members of this class

class Sequence:
    ObjectCount=0  # a class variable
    TranscriptionTable = {'A':'A', 'C':'C', 'G':'G', 'T':'U'}
    def __init__(self, seqstring): # self represents an instance of Sequence
        Sequence.ObjectCount = Sequence.ObjectCount + 1
        self.seqstring=seqstring.upper() #seqstring is an attribute of self
```

```
>>> from seq_plasmid import Sequence
>>> DNA_seq=Sequence('atgcaagtaggtcccaac')
>>> DNA_seq.seqstring
'ATGCAAGTAGGTCCCAAC'
```

Creating Python Classes



```
class NAME:
    [body]

class NAME:
    def __init__(self):
        # define and initialize attributes belonging to all members of this class

class Sequence:
    ObjectCount=0  # a class variable
    TranscriptionTable = {'A':'A', 'C':'C', 'G':'G', 'T':'U'}
    def __init__(self, seqstring): # self represents an instance of Sequence
        Sequence.ObjectCount = Sequence.ObjectCount + 1
        self.seqstring=seqstring.upper() #seqstring is an attribute of self

    def transcribe(self):
        RNA = ""
        for x in self.seqstring:
            if x in 'ACGT':
                RNA += self.TranscriptionTable[x]
        return RNA
```

```
>>> DNA_seq.transcribe()
'AUGCAAGUAGGUCCCAAC'
```

Class Methods

```
>>> from seq_plasmid import Sequence
>>> DNA_seq=Sequence('atgcaagtaggtcccaac')
>>> DNA_seq
<seq_plasmid.Sequence instance at 0x1004d2b48>
>>>
>>> DNA_seq.seqstring
'ATGCAAGTAGGTCCCAAC'
>>> DNA_seq.transcribe()
'AUGCAAGUAGGUCCCAAC'
>>>
>>> len(DNA_seq)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Sequence instance has no attribute
'__len__'
>>> len(DNA_seq.seqstring)
18
```

- If you define a method called __len__ inside your Sequence class, then you can call it on the object itself (rather than on its attribute, seqstring).
- This is called ***function or operator overloading***.

Inheritance

- ***Derived classes*** can inherit properties and methods from a parent class, but implement attributes and methods specific to their subclass.

- For example, the 'seq_plasmid.py' module defines two classes, Sequence and Plasmid:
  - Sequence contains two methods, 'transcribe', and 'restrict'.
  - Plasmid inherits these methods, but also contains the methods 'pcs' and 're_in_pcs’.
  - Functions that work with Sequence arguments will also work with Plasmids (subtype polymorphism)

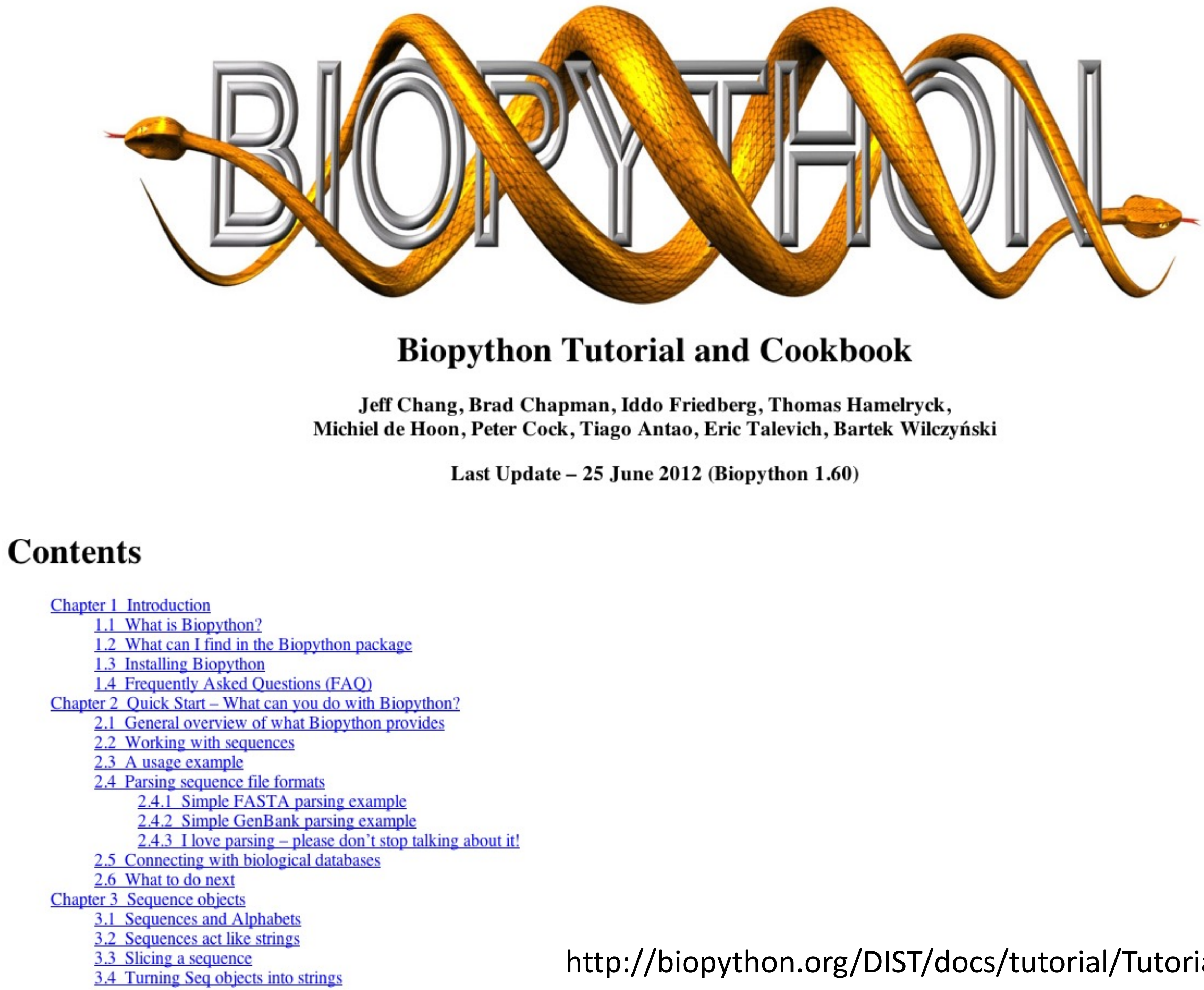
➢ The file 'seq_plasmid.py' is posted on the course website.

Inheritance

```
>>> from seq_plasmid import *
>>> test = Plasmid('acgaattcgtacagc')
>>> test.restrict('EcoRI')
True
>>> test.restrict('EcoR1')
'Unknown enzyme'
>>> test.pcs('EcoRI')
'EcoRI'
>>> test.re_in_pcs('EcoRI')
True
>>> test.re_in_pcs('EcoR1')
False
>>> test.pcs(['EcoRI','SalI'])
['EcoRI', 'SalI']
>>> test.re_in_pcs('EcoR1')
False
>>> test.re_in_pcs('EcoRI')
True
>>>
>>> test2 = Sequence('acgaattcgtacagc')
>>> test2.restrict('EcoRI')
True
>>> test2.restrict('SalI')
False
>>> test2.pcs('EcoRI')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Sequence instance has no attribute 'pcs'
```

Now introducing …

A suite of Python tools for bioinformatics and computational molecular biology.



Biopython Tutorial and Cookbook

Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck,
Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich, Bartek Wilczyński

Last Update – 25 June 2012 (Biopython 1.60)

Contents

http://biopython.org/DIST/docs/tutorial/Tutorial.html

BioPython Modules

- The BioPython package contains a large number of modules that implement specialized ***classes*** for biological sequence analysis.
- Using them typically entails creating one or more ***instances*** of a particular ***object*** type, and then calling different ***methods*** on these.

➢ Documentation for BioPython modules can be found online.

Alphabets

- A lot of BioPython modules operate on sequence data, which represent DNA, RNA, or proteins.

- Each of these sequence types has allowable symbols in the alphabet used to describe them, e.g.:

  {A,C,G,T} = unambiguous DNA

  {A,C,G,U} = unambiguous RNA

  {A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y} = standard amino acids

- The International Union of Pure and Applied Chemistry (IUPAC) has established universally accepted nomenclature conventions for nucleic acids and proteins. These are the **IUPAC alphabets**.

BioPython Seq objects

- The **Bio.Seq** module defines a **Seq object**, comprising the sequence itself and the **alphabet** that defines it:

```
In [7]: from Bio.Seq import Seq

In [8]: seq = Seq('GATTCGTATGCCATG',Bio.Alphabet.IUPAC.unambiguous_dna)

In [9]: seq1 = seq

In [10]: seq1
Out[10]: Seq('GATTCGTATGCCATG', IUPACUnambiguousDNA())

In [13]: seq1.reverse_complement()
Out[13]: Seq('CATGGCATACGAATC', IUPACUnambiguousDNA()) # returns revcom of seq1

In [14]: mrna1 = seq1.transcribe()                     # returns RNA corresponding to seq1

In [15]: mrna1
Out[15]: Seq('GAUUCGUAUGCCAUG', IUPACUnambiguousRNA())

In [16]: mrna1.back_transcribe()
Out[16]: Seq('GATTCGTATGCCATG', IUPACUnambiguousDNA()) # turns RNA back into DNA

...

In [19]: print(seq)                      # seq objects can be manipulated like string sequences

-------> print(seq)
GATTCGTATGCCATG

In [20]: seq1[2]                         # e.g. indexed sequence element
Out[20]: 'T'

In [22]: seq.lower()
Out[22]: Seq('gattcgtatgccatg', DNAAlphabet())

...

In [25]: seq.translate()                  # returns translated DNA or RNA
Out[25]: Seq('DSYAM', IUPACProtein())

In [26]: seq[1:].translate()              # use substrings (slices) to get different reading frames
Out[26]: Seq('IRMP', IUPACProtein())

In [27]: seq[2:].translate()
Out[27]: Seq('FVCH', IUPACProtein())
```

BioPython SeqRecord objects

- If we want to know more about a sequence, like its name, ID, description, etc., we need a **SeqRecord object**, which stores a **Seq** object with its associated **metadata**, as described in the **BioPython Tutorial and Cookbook**:

4.1 The SeqRecord object

The SeqRecord (Sequence Record) class is defined in the Bio.SeqRecord module. This class allows higher level features such as identifiers and features to be associated with a sequence (see Chapter 3), and is the basic data type for the Bio.SeqIO sequence input/output interface (see Chapter 5).

The SeqRecord class itself is quite simple, and offers the following information as attributes:

seq – The sequence itself, typically a Seq object.

id – The primary ID used to identify the sequence – a string. In most cases this is something like an accession number.

name – A "common" name/id for the sequence – a string. In some cases this will be the same as the accession number, but it could also be a clone name. I think of this as being analagous to the LOCUS id in a GenBank record.

description – A human readable description or expressive name for the sequence – a string.

letter annotations – Holds per-letter-annotations using a (restricted) dictionary of additional information about the letters in the sequence. The keys are the name of the information, and the information is contained in the value as a Python sequence (i.e. a list, tuple or string) with the same length as the sequence itself. This is often used for quality scores (e.g. Section 16.1.6) or secondary structure information (e.g. from Stockholm/PFAM alignment files).

annotations – A dictionary of additional information about the sequence. The keys are the name of the information, and the information is contained in the value. This allows the addition of more "unstructed" information to the sequence.

features – A list of SeqFeature objects with more structured information about the features on a sequence (e.g. position of genes on a genome, or domains on a protein sequence). The structure of sequence features is described below in Section 4.3.

dbxrefs - A list of database cross-references as strings.