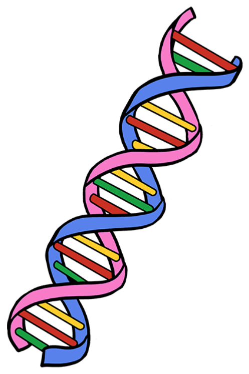
CS2204 Homework: DNA Analysis

Objectives



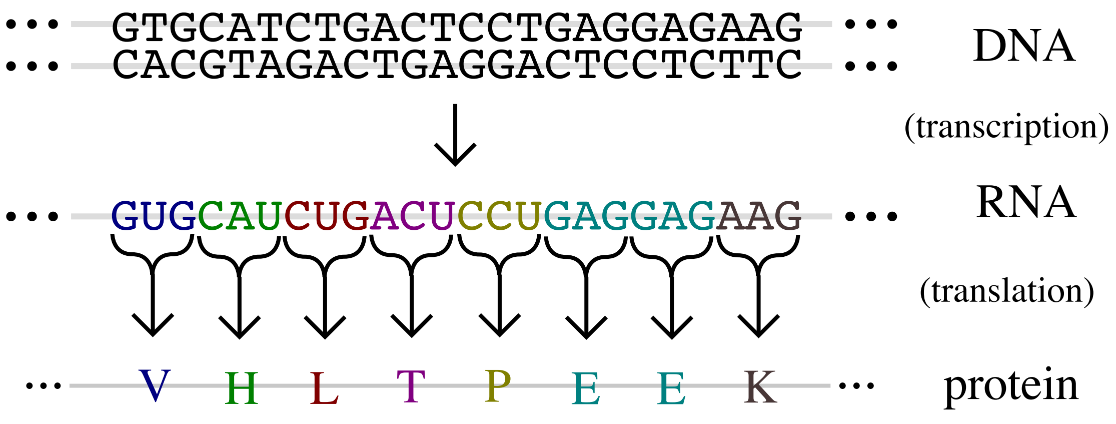
- Learn to create new types/classes
- Learn to use *aggregation*
- Understand magic (*dunder*) methods for operator overloading
- Learn to generate *exceptions* and/or use *assertions*
- Practice writing your own tests

Background

In this Homework assignment we are developing two new classes to store and analyze DNA squence information. *Note*: the approach and tasks are obviously a simplified version of *real* sequence analysis.

Genes are commonly represented in computer software as a sequence of the characters A, C, G, and T. Each letter represents a **nucleotide**, and the sequence of three (3) nucleotides is called a **codon**, which encodes a specific amino acid. We will call an arbitrary sequence of codons as **gene**, coding for a protein molecule. Note, that in the actual DNA molecule the nucleotide bases are always in pairs (as shown below), but because the pairing is deterministic (G-C, T-A), it is sufficient to store one side of these pairs in software.

As part of the protein synthesis process, the DNA molecule is transcribed to an RNA molecule. In a very simplified view the 'T' bases are replaced by 'U' bases in the original sequence.



Source: *Wikipedia: Genetic Code*

Your tasks will be to develop Python classes to represent the *Codon* and *Gene* concepts and implement some key methods to work with these objects. Note, that there is no *inheritance* relationship between these two concepts. Instances of the Gene class will **contain** a sequence of instances of the Codon class, which is called *aggregation*.

Tasks

You need to finish the `dna.py` source file. Note: you can write your own testing code at the end of this source file. A sample DNA sequence is also provided (`dna_sample.txt`) with some example code how to work with it. Feel free to change / add / delete any testing code. The outputs of this testing code are not going to be graded. The validator program will use you classes, directly.

1. Create a proper constructor (`__init__`) method for the `Codon` class. The parameter for this method is supposed to be a string with exactly three characters from `"A"`, `"C"`, `"T"`, `"G"`. Both upper case and lower case letters should be accepted. If the method is called with an invalid parameter, it should generate an exception (see Hints). (10 pts)
2. Implement the string conversion method (`__str__`) for the `Codon` class. The resulting string should use the sqare brackets and should always capitalize the base letters - even if the object was created with lower or mixed-case letters. (10 pts) E.g. `[ACT]`
3. Implement the equality operator (`__eq__`) for the `Codon` class (10 pts). E.g. the following test should return `True`:

   ```
   Codon("act") == Codon("ACT")
   ```

4. Create a transcribe method for the `Codon` class. This should return an upper case string, similarly to the string conversion class. However, the string should be enclosed in angle (`<` and `>`) brackets and the `"T'` characters are replaced by `"U'` (10 pts). E.g. `<ACU>`
5. Implement the constructor for `Gene`. The input parameter is an arbitrary long string (it can be empty) containing the nucleotide base characters (case insensitive). You should build and store a list of Codon objects from this sequence. If the the length of the sequence is not an integer multiple of 3, dicard the last characters. (10 pts)
6. Implement the string conversion method (`__str__`) for the `Gene` class. See the docstring for details. (10 pts)
7. Implement the transcribe method for the `Gene` class. See the docstring for details. (10 pts)
8. Implement the containment test (`__contains__`) for the Gene class to be able to check if a given codon is part of the gene sequence. (15 pts). E.g. the following test should return `True`:

   ```
   Codon("ACT") in Gene("GGCACT")
   ```

9. Implement a method for calculating the GC content in the gene, which can be used for gene classification and/or estimating some stability properties of the DNA molecule. The method

should return the ratio of the number of `G + C` bases relative to all bases in the gene (15 pts) as a `float`. As a reference test: the provided DNA sample has the GC conent of ~0.49915

Hints

Exceptions: your code should generate exceptions in a few cases when it receives invalid inputs. The actual type of exception does not matter. You can solve these requirements with (a combination) of the following approaches:

- rely on the built-in exception logic in Python functions and expressions (e.g. `len(None)` generates an exception)
- use the `assert` statement (preferrred approach). E.g.

```
var = "hello"
assert type(var) is str
```

- use the `raise` statement. E.g.

```
var = "hello"
if type(var) is not str:
  raise ValueError("var is not a string")
```

Grading

You can use the attached `validator.py` program to check your work. It will also estimate your final score for the homework. The program is in the same folder as your homework assignment. Open the `validator.py` script and run it in the Spyder environment to track your progress.

Penalties

Points will be deducted if you fail to set `__author__` variable (-10 pts) and for **each PEP 8 style errors** (-1 pt for each) in your program.

Submission

Please, upload the final version of the following file(s) (**and only those file(s)**) to Brightspace:

- `dna.py`