Bioinformatical problem solving with Python

Wednesdays 17:30-19:00, M801

alexander.nater@uni-konstanz.de

Covered topics so far

- Objects and data types (string, int, float, list, dict, etc.)
- Control structures: loops and conditions
- Custom functions and lambdas
- Modules and file handling
- Regular expressions
- Object-oriented Python

Not covered topics so far

- Error handling (exceptions)
- Numerical and scientific libraries (SciPy, NumPy, pandas, etc.)
- Plotting libraries (Matplotlib)
- Software design

```
class Sequence:
    """Stores a DNA sequence as string."""
    def __init__(self, my_name, my_dna):
        self.name = my_name
        self.dna = my_dna

    def get_gc_content(self):
        g_count = self.dna.count('G')
        c_count = self.dna.count('C')
        return (g_count + c_count) / len(self.dna)

my_sequence = Sequence("Human gene 123",
"ATGCGGTCT")
```

Exercise

Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.

Custom classes

```
class DNARecord():
    def __init__(self, sequence, gene_name, species_name):
        self.sequence = sequence
        self.gene_name = gene_name
        self.species_name = species_name

    def complement(self):
        complement_table = str.maketrans('ACTG', 'TGAC’)
        dna = self.sequence.upper()
        return dna.translate(complement_table)

    def get_fasta(self):
        safe_species_name = self.species_name.replace(' ', '_’)
        header = ">{}_{}".format(self.gene_name, safe_species_name)
        return "{}\n{}\n".format(header, self.sequence)

d1 = DNARecord('ATATATTATTATATTATA', 'COX1', 'Homo sapiens’)
print(d1.get_fasta())
print(d1.complement())
```

Usage example

Writing a list of DNARecord objects in fasta format to a file:

```
with open("high_at.fasta", "w") as output:
    for d in my_dna_records:
    if len(d.sequence) > 100:
        output.write(d.get_fasta())
```

Usage example

```
def translate_dna(dna_record):
    gencode = {
        'ATA':'I', 'ATC':'I', 'ATT':'I', 'ATG':'M', 'ACA':'T', 'ACC':'T',
        'ACG':'T', 'ACT':'T', 'AAC':'N', 'AAT':'N', 'AAA':'K', 'AAG':'K',
        'AGC':'S', 'AGT':'S', 'AGA':'R', 'AGG':'R', 'CTA':'L', 'CTC':'L',
        'CTG':'L', 'CTT':'L', 'CCA':'P', 'CCC':'P', 'CCG':'P', 'CCT':'P',
        'CAC':'H', 'CAT':'H', 'CAA':'Q', 'CAG':'Q', 'CGA':'R', 'CGC':'R',
        'CGG':'R', 'CGT':'R', 'GTA':'V', 'GTC':'V', 'GTG':'V', 'GTT':'V',
        'GCA':'A', 'GCC':'A', 'GCG':'A', 'GCT':'A', 'GAC':'D', 'GAT':'D',
        'GAA':'E', 'GAG':'E', 'GGA':'G', 'GGC':'G', 'GGG':'G', 'GGT':'G',
         'TCA':'S', 'TCC':'S', 'TCG':'S', 'TCT':'S', 'TTC':'F', 'TTT':'F', 'TTA':'L',
        'TTG':'L', 'TAC':'Y', 'TAT':'Y', 'TAA':'_', 'TAG':'_', 'TGC':'C', 'TGT':'C',
        'TGA':'_', 'TGG':'W'}
    last_codon_start = len(dna_record.sequence) – 2
    protein = "«
    for start in range(0, last_codon_start, 3):
        codon = dna_record.sequence[start:start+3]
        aa = gencode.get(codon.upper(), 'X')
        protein = protein + aa
    return protein
```

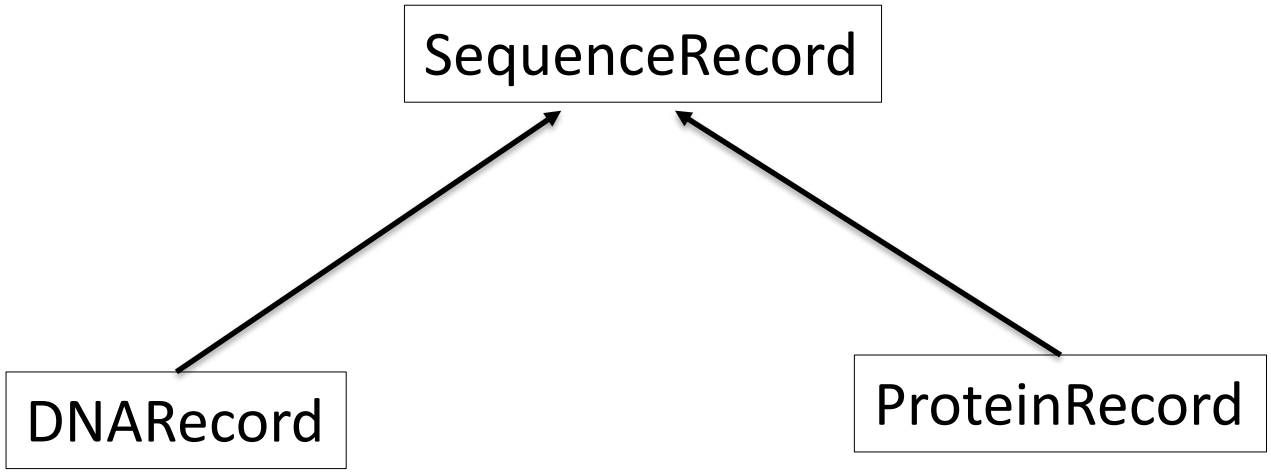
Inheritance

```
class ProteinRecord():
    def __init__(self, sequence, gene_name, species_name):
        self.sequence = sequence
        self.gene_name = gene_name
        self.species_name = species_name

    def get_hydrophobic(self):
        aa_list = "AILMFWYV“
        hp_count = sum(1 for r in self.sequence.upper() if r in aa_list)
        return hp_count / len(self.sequence) * 100

    def get_fasta(self):
        safe_species_name = self.species_name.replace(' ', '_’)
        header = ">{}_{}".format(self.gene_name, safe_species_name)
        return "{}\n{}\n".format(header, self.sequence)

d1 = ProteinRecord('MSRSLLLRFLLFLLLLPPLP', 'COX1', 'Homo sapiens’)
print(d1.get_fasta())
print(str(d1.get_hydrophobic()))
```

Inheritance

```
class SequenceRecord():
    def __init__(self, sequence, gene_name, species_name):
        self.sequence = sequence
        self.gene_name = gene_name
        self.species_name = species_name

    def get_fasta(self):
        safe_species_name = self.species_name.replace(' ', '_’)
        header = ">{}_{}".format(self.gene_name, safe_species_name)
        return "{}\n{}\n".format(header, self.sequence)
```

Inheritance



Inheritance

```
class DNARecord(SequenceRecord):
    def complement(self):
        complement_table = str.maketrans('ACTG', 'TGAC’)
        dna = self.sequence.upper()
        return dna.translate(complement_table)

class ProteinRecord(SequenceRecord):
    def get_hydrophobic(self):
        aa_list = "AILMFWYV«
        hp_count = sum(1 for r in self.sequence.upper() if r in aa_list)
        return hp_count / len(self.sequence) * 100
```

Calling base class constructor

```
class DNARecord(SequenceRecord):
    def __init__(self, sequence, gene_name, species_name, genetic_code):
        super().__init__(sequence, gene_name, species_name)
        self.genetic_code = genetic_code

    def complement(self):
        complement_table = str.maketrans('ACTG', 'TGAC’)
        dna = self.sequence.upper()
        return dna.translate(complement_table)
```