Biopython
Sequence Objects

Doris Steinbauer und Klaus Leitner

21.11.2024

Einleitung und Motivation

- Ermöglicht das Bearbeiten von DNA, RNA und Protein Sequenzen
- Oft der erste Schritt in der Datenanalyse
- Potentielle Anwendungsgebiete:
  - Trimmen von Sequenzen
  - Das Identifizieren von Motifen (z.B: für Transkriptionsfaktoren Bindungsstellen)
  - Designen von Oligos (z.B.: Primer, crRNA,…)
  - Anpassung der CDS an spezifische Codon Tabelle
  - Änderungen in Sequenzen

String Eigenschaften

```python
from Bio.Seq import Seq

my_seq = Seq("GATCG")
normal_string = "GATCG"

print("Seq() Funktion " )
for letter in my_seq:
    print(letter, end =" ")
print("\nLänge: ", len(my_seq))
print("\nType von Seq Objekt: ", type(my_seq))

print("\nNormal string" )
for letter in normal_string:
    print(letter, end =" ")
print("\nLänge: ", len(normal_string))
print("\nType von String Objekt: ", type(normal_string))
```

```
Seq() Funktion
G A T C G
Länge:  5

Type von Seq Objekt:  <class 'Bio.Seq.Seq'>

Normal string
G A T C G
Länge:  5

Type von String Objekt:  <class 'str'>
```

- Seq() Objekte verhalten sich wie normale Strings (gehören aber zur Bio.Seq.Seq Klasse)
- String Eingenschaften:
  - Iterieren (Schleifen)
  - Funktionen/Methoden: `len()`, `join()`, `upper()`, `lower()`,…
  - Slicing: `my_seq[4:3]`, `my_seq[0::3]`, …
  - Verbinden von Sequenzen: `my_seq1 + my_seq2`
- Seq() Objekte können in Strings umgewandelt werden:
  - `normal_string = str(my_seq)`
- Vergleichen von Sequenzen ist der Ursprung irrelevant (ob DNA, RNA, AA oder Python String)
  - `seq1 = Seq("ACGT")`
  - `"ACGT" == seq1 -> True`

Biologische Anwendungen

- Erstellen eines Seq Objekts : `my_seq = Seq("GTACCCGAATA")`

- Calculating GC bias:
  - `from Bio.SeqUtils import gc_fraction`
  - `gc_fraction(my_seq)`
  - Ist komplizierter: `my_seq.count("G") + my_seq.count("C") / len(my_seq)`

- Reverse complement:
  - `my_seq.reverse_complement()`
  - Funktioniert nur für DNA Sequenzen

- Transcription:
  - `my_seq.transcribe()` (T->U)
  - `my_seq.back_transcribe()` (U->T)

Biologische Anwendungen

- Translation
  - Codon usage table: Darstellung mit Name oder NCBI table number
  - Zusätzliche Argumente: to_stop und cds
  - Potentielles stop codon mit "*" markiert

```
print(coding_dna.translate())
print(coding_dna.translate(to_stop=True))

print(coding_dna.translate(table="Vertebrate Mitochondrial"))
print(coding_dna.translate(table=2))

print(coding_dna.translate(table=2, to_stop=True))

# Full coding sequence -> gene object
print(gene.translate(table="Bacterial", to_stop=True))
print(gene.translate(table="Bacterial", cds=True))
```

```
MAIVMGR*KGAR*
MAIVMGR
MAIVMGRWKGAR*
MAIVMGRWKGAR*
MAIVMGRWKGAR
VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDHGWWKQHYEWRGNRWHLHGPPPPPRHHKKAPHDHHGGHGPGKHHR
MKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDHGWWKQHYEWRGNRWHLHGPPPPPRHHKKAPHDHHGGHGPGKHHR
```

Biologische Anwendungen

- Sequenzen mit partially defined sequences:
  - Beispiel: MAF (Multiple Alignment Format) file
  - Data Argument wird ein Dictionary verwendet (Nur 1 item erlaubt)
  - Es können subsequences definiert werden und output ist entweder: fully defined sequence, an undefined sequence, or a partially defined sequence.
  - Partially aligned sequences können auch durch string concatenation erstellt werden, wenn eine Sequenz partially oder fully defined ist
    - `undefined_seq = Seq(None, length=10)`
    - `seq + undefined_seq + seq`

```
seq = Seq({117512683: "TTGAAAACCTGAATGTGAGAGTCAGTCAAGGATAGT"}, length=159345973)

seq[1000:1020] # Seq(None, length=20)
seq[117512690:117512700] # Seq('CCTGAATGTG')
seq[117512670:117512690] # Seq({13: 'TTGAAAA'}, length=20)
seq[117512700:] # Seq({0: 'AGAGTCAGTCAAGGATAGT'}, length=41833273)
```

Seq Objekte - Mutability

- Seq objects (und Strings) sind immutable
- Können durch Funktion mutable gemacht werden
  - Benötigt MutableSeq class von Bio.seq module
  - Objekt mutable_seq kann in-place modifiziert werden
  - Durch Seq() Klasse kann das mutable Objekt in immutable Objekt überführt werden

```
from Bio.Seq import MutableSeq

mutable_seq = MutableSeq(my_seq)
mutable_seq[5] = "C"
mutable_seq.remove("T")
mutable_seq.reverse()

# Return to immutable
new_seq = Seq(mutable_seq)
```

Subsequences identifizieren

- Python erlaubt das Finden von subsequences in Strings
- Die Klasse Seq ermöglicht das Finden von der ersten subsequence in: Strings, bytes, bytearray, Seq, oder MutableSeq (output ist die Indexnummer)
- Methoden: `index()`, `rindex()`, `find()`, `rfind()` und `search()`
- `search()` is used to find multiple subsequences (output: Generator Objekt)

```python
from Bio.Seq import Seq, MutableSeq

seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA")

# if subsequence not found -> return ValueError
seq.index("ATGGGCCGC")
seq.index(b"ATGGGCCGC")
seq.index(bytearray(b"ATGGGCCGC"))
seq.index(Seq("ATGGGCCGC"))
seq.index(MutableSeq("ATGGGCCGC"))
seq.rindex("ATGGGCCGC")

seq.find("CC") # if subsequence not found -> return -1
seq.rfind("CC") # same as find, starts from right side
```

Strings anstatt Seq Klasse

- Mit gewissen Modul-Funktionen besteht die Möglichkeit direkt mit Strings zu arbeiten
- Personen die functional programming bevorzugen (wird nicht empfohlen)

```
from Bio.Seq import reverse_complement, transcribe, back_transcribe, translate

my_string = "GCTGTTATGGGTCGTTGGAAGGGTGGTCGTGCTGCTGGTTAG"

reverse_complement(my_string)
transcribe(my_string)
back_transcribe(my_string)
translate(my_string)
```

Zusammenfassung

- Die Seq Klasse von Bio.Seq Modul ermöglicht es biologische Sequenzen schnell und einfach zu modifizieren
  - Umwandlung von DNA <-> RNA und RNA in AA
  - unabhängig von diversen web tools

    (z.B.: reverse_complement_online_tool)

- String Eigenschaften

- Bei Seq Objekten handelt es sich um immutable Objekte. Diese können in mutable Objekte umgewandelt werden.