Ćwiczenia: Podstawy programowania w języku Python

dr inż. Piotr Błaszczyk

Szczecin, 2007

1 Cel

Ćwiczenie ma na celu poznanie podstaw programowania w języku Python, powszechnie używanego w bioinformatyce.

2 O języku Python

Python stanowi użyteczne narzędzie do programowania na potrzeby bioinformatyki, gdyż został zaprojektowany jako możliwie prosty i łatwy w użyciu język wysokiego poziomu, jednocześnie posiadający szerokie możliwości. Pomimo swojej prostoty i przejrzystości, Python umożliwia tworzenie rozbudowanych i potężnych aplikacji do poważnych zastosowań.

Python jest językiem o dynamicznym systemie typów, co oznacza, że nie potrzeba deklarować zmiennych przed ich użyciem, a ta sama zmienna może odnosić się do różnych typów danych, takich jak tekst czy dane numeryczne. Język ten wspiera również programowanie obiektowe, co znajduje zastosowanie w bardziej złożonych problemach.

Python może być obsługiwany interaktywnie poprzez swoją konsolę (interpreter), co ułatwia szybką naukę programowania w tym języku, gdyż wykonanie wpisanej komendy z miejsca pokazuje wyniki naszych działań. Możemy również pisać programy w postaci skryptów, czyli plików tekstowych zawierających kod Python i wykonywanych następnie z użyciem interpretera Python.

3 Zastosowanie w biologii molekularnej

Jak wiadomo, DNA stanowi strukturę złożoną z *sekwencji* cząsteczek zwanych nukleotydami, które reprezentowane są za pomocą inicjałów nazw ich zasad: A(denina), C(ytozyna), G(uanina) oraz T(ymina). DNA składa się właściwie z dwóch takich sekwencji nukleotydów owiniętych wokół siebie w postaci słynnej podwójnej helisy.

Sekwencja pojedynczej nici DNA, np. ATGCCTTCGG, jest zatem reprezentowana przez *sekwencję* liter oznaczających poszczególne zasady. Z uwagi na charakter oddziaływań między cząsteczkami, nukleotydy łączą się w pary: adenina zawsze z tyminą, a cytozyna zawsze z guaniną. Na podstawie danej sekwencji możemy zatem jednoznacznie określić, jak wygląda sekwencja do niej komplementarna.

To ćwiczenie unaoczni nam, jak w języku Python radzić sobie z sekwencjami tekstowymi. Wiedza ta pozwoli nam na operowanie fragmentami DNA, które można przedstawić również jako sekwencje tekstowe.

4 Podstawy interaktywnej pracy z interpreterem Python

Wywołaj aplikację Terminal (menu Aplikacje / Akcesoria). Środowisko Python jest zainstalowane w systemie, więc wystarczy wpisać polecenie "python" w terminalu i nacisnąć klawisz Enter, aby przejść do konsoli (powłoki) interpretera Python (ang. *Python shell*). W terminalu zobaczymy zapis podobny do poniższego:

```
% python
Python 2.4.3 (#1, Jun 13 2006, 11:46:08)
[GCC 4.1.1 20060525 (Red Hat 4.1.1-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Jeśli skończymy pracę, klawiszami Ctrl-D zamykamy konsolę Python i powracamy do powłoki systemowej (bash). Pozostając jednak nadal w konsoli Python — linia poleceń to trzy znaki "większy":

```
>>>
```

Praca w konsoli Python polega na wpisaniu polecenia i wciśnięciu klawisza Enter aby otrzymać wynik. Np. wpisując działanie arytmetyczne odkryjemy, że Python może służyć jako kalkulator:

```
>>> 8*7
56
```

Jak wspomniano powyżej, zmienne w Pythonie nie wymagają deklaracji. Wystarczy, że wpiszemy nazwę zmiennej i przypiszemy jej wartość znakiem równości:

```
>>> a = 876
>>> b = 14.7
>>> a/b
59.591836734693878
```

Powyżej stworzyłem dwie zmienne, *a* i *b*, przypisując jednocześnie liczbę całkowitą pierwszej zmiennej, a liczbę rzeczywistą drugiej zmiennej. W trzeciej linii wpisano dzielenie, a w czwartej otrzymano jego wynik. Z kolei wpisując coś w stylu:

```
>>> imie = 'Katarzyna'
```

deklarujemy zmienną "imie" i przypisujemy jej wartość w postaci ciągu znaków 'Katarzyna' (ciągi znaków przedstawiamy w pojedynczym ' lub podwójnym " cudzysłowie). Aby wywołać zawartość zmiennej wpisujemy jej nazwę i wciskamy Enter:

```
>>> imie
'Katarzyna'
```

lub posługujemy się komendą „print":

```
>>> print imie
Katarzyna
```

Podobnie jak w arkuszu kalkulacyjnym, możemy manipulować ciągami znaków, np. łączyć je ze sobą:

```
>>> imie + ' Kowalska'
'Katarzyna Kowalska'
```

5 Operatory

Powyżej podano, że operatorem „+" możemy łączyć ze sobą ciągi znaków. Znak „+" jest oczywiście również operatorem arytmetycznym oznaczającym dodawanie, jeżeli argumenty będą miały typ numeryczny a nie tekstowy. Inne użyteczne dla nas operatory matematyczne to: - (odejmowanie), * (mnożenie), / (dzielenie), ** (potęgowanie). Operatorem przypisania wartości do zmiennej jest znak równości (=).

6 Podstawowe typy danych

6.1 Ciągi znaków (łańcuchy) - ang.*string*

Python posiada szereg wbudowanych metod do obsługi tekstu[1]. I tak, przy ich pomocy funkcji „len()" policzyć znaki w tekście:

```
>>> len(imie)
9
```

Możemy też dowolnie zmieniać litery na wielkie (metoda „upper()" od ang. *uppercase*, proszę zwrócić uwagę na inną składnię):

```
>>> imie.upper()
'KATARZYNA'
```

lub małe (*lowercase*):

```
>>> imie.lower()
'katarzyna'
```

Możemy też łatwo zliczać znaki (lub ciąg znaków) występujące w danym tekście:

```
>>> imie.count('a')
3
>>> imie.count('K')
1
>>> imie.count('rzy')
1
```

Ta ostatnia właściwość języka Python staje się bardzo przydatna w biologii, np. do wyszukiwania powtarzających się sekwencji nukleotydów w niciach DNA:

```
>>> dna = 'TGGAAGATTATATCTAATATCCTCTCTATGGTGGGGTTTAGTAGGGTTGTCATTAAGAAT'
>>> dna.count('T')
23
>>> dna.count('GGTT')
2
```

Z powyższego wynika bowiem, że tymina w danej sekwencji występuje 23 razy, zaś dwukrotnie powtarza się sekwencja „GGTT". Inna ciekawa metoda to „replace":

```
>>> dna.replace('T','U')
'UGGAAGAUUAUAUCUAAUAUCCUCUCUAUGGUGGGGUUUAGUAGGGUUGUCAUUAAGAAU'
```

Wynik działania możemy przypisać innej zmiennej:

```
>>> rna = dna.replace('T','U')
>>> dna
'TGGAAGATTATATCTAATATCCTCTCTATGGTGGGGTTTAGTAGGGTTGTCATTAAGAAT'
>>> rna
'UGGAAGAUUAUAUCUAAUAUCCUCUCUAUGGUGGGGUUUAGUAGGGUUGUCAUUAAGAAU'
```

[1] Pełna lista metod obiektów tekstowych: http://docs.python.org/lib/string-methods.html (ang.)

Ponadto, każdy ciąg znaków w Pythonie stanowi tablicę jednowymiarową, co oznacza, że możemy dowolnie przywoływać poszczególne znaki przy pomocy *indeksu*, czyli numeru znaku w ciągu (rozpoczynając od 0). Na przykład w sekwencji 'TGGAAGATTATATCTAATATCCTCTCTATGGTGGGGTTTAGTAGGGTTGTCATTAAGAAT' litera 'T' na początku ma indeks 0, więc możemy wg tego indeksu ją wywołać następująco:

```
>>> dna[0]
'T'
>>> dna[3]
'A'
```

6.1.1 Zadania

Za pomocą interpretera Python wykonaj na sekwencji DNA TGGAAGATTATATCTAATATCCTCTCTATGGTGGGGTTTAGTAGGGTTGTCATTAAGAAT następujące operacje:

1. Zmiennej o nazwie „dna" przypisz wartość w postaci powyższej sekwencji.
2. Oblicz długość sekwencji przy użyciu odpowiedniej funkcji.
3. Oblicz, ile razy w sekwencji występuje każda z zasad.
4. Oblicz, ile razy występuje sekwencja „GG".
5. Oblicz, ile razy występuje sekwencja „TAT".
6. Oblicz, ile razy występuje sekwencja „ATA".
7. Przekształć sekwencję DNA w mRNA, zamieniając tyminę na uracyl, a wynik tego działania przypisz nowej zmiennej „mrna".
8. Zbadaj, ile kodonów fenyloalaniny (UUU lub UUC) znajduje się w otrzymanej sekwencji mRNA
9. Zbadaj, ile kodonów leucyny (UUA, UUG, CUU, CUC, CUA lub CUG) znajduje się w sekwencji mRNA

6.2 Listy

Listy w języku Python to uporządkowane zbiory dowolnych obiektów, włączając w to inne listy. Elementy list mogą być dowolnie wstawiane, usuwane i podmieniane na inne. Listy tworzone są jako serie obiektów oddzielone przecinkami i zawarte w prostokątnych nawiasach, np. możemy zdefiniować listę zasad i ją wywołać podobnie jak dla łańcuchów:

```
>>> zasady = ['A', 'C', 'G', 'T']
>>> zasady
['A', 'C', 'G', 'T']
```

Metoda „append" pozwala dodać element do listy (na jej koniec):

```
>>> zasady
['A', 'C', 'G', 'T']
>>> zasady.append('U')
>>> zasady
['A', 'C', 'G', 'T', 'U']
```

Metoda „remove" pozwala usunąć element z listy:

```
>>> zasady.remove('U')
>>> zasady
['A', 'C', 'G', 'T']
```

Metoda „insert" pozwala wstawić element w danej pozycji listy, przesywając istniejące elementy w prawo:

```
>>> zasady.insert(2,'U')
>>> zasady
['A', 'C', 'U', 'G', 'T']
```

Podobnie jak dla ciągów znaków, możemy liczyć elementy listy i odwoływać się do elementów poprzez ich indeksy:

```
>>> len(zasady)
5
>>> zasady.count('U')
1
>>> zasady[0]
'A'
>>> zasady[4]
'U'
```

Z ciągów znaków możemy łatwo tworzyć listy funkcją „list":

```
>>> dna
'TGGAAGATTATATCTAATATCCTCTCTATGGTGGGGTTTAGTAGGGTTGTCATTAAGAAT'
>>> list(dna)
['T', 'G', 'G', 'A', 'A', 'G', 'A', 'T', 'T', 'A', 'T', 'A', 'T', 'C', 'T', 'A', 'A',
'T', 'A', 'T', 'C', 'C', 'T', 'C', 'T', 'C', 'T', 'A', 'T', 'G', 'G', 'T', 'G', 'G',
'G', 'G', 'T', 'T', 'T', 'A', 'G', 'T', 'A', 'G', 'G', 'G', 'T', 'T', 'G', 'T', 'C',
'A', 'T', 'T', 'A', 'A', 'G', 'A', 'A', 'T']
>>>
```

Listy możemy odwracać (*reverse*), sortować, usuwać z nich wybrane elementy oraz zamieniać je na inne:

```
>>> zasady
['A', 'C', 'G', 'T', 'U']
>>> zasady.reverse()
>>> zasady
['U', 'T', 'G', 'C', 'A']
>>> zasady.sort()
>>> zasady
['A', 'C', 'G', 'T', 'U']
>>> zasady[3] = 'U'
>>> zasady
['A', 'C', 'G', 'U']
```

6.2.1 Zadania

1. Utwórz listę czterech zasad występujących w DNA (stosując ich pełne nazwy, tj. „adenina", „cytozyna" itd) w zmiennej o nazwie „zasady".
2. W liście „zasady" dokonaj zamiany elementu „tymina" na „uracyl".
3. Odwróć kolejność zasad.
4. Nowej zmiennej „dnarev" przypisz listę utworzoną z sekwencji DNA (zmienna „dna" z poprzedniego zestawu zadań), a następnie odwróć listę.

7 Literatura uzupełniająca

Więcej informacji na temat języka Python:

- http://pl.wikipedia.org/wiki/Python — opis języka Python w Wikipedii
- http://www.python.org/ — strona oficjalna projektu (ang.)
- http://wiki.python.org/moin/BeginnersGuide (ang.)
- http://docs.python.org/tut/tut.html — Python Tutorial (ang.)
- http://www.pasteur.fr/formation/infobio/python/ — Introduction to Programming using Python (ang.)
- http://docs.python.org/ref/ref.html — Python Reference Manual (ang.)

Więcej informacji na temat programowania w biologii molekularnej:

- http://www.onlamp.com/pub/a/python/2002/10/17/biopython.html — wstęp do języka Python dla bioinformatyków (ang.)
- http://www.pasteur.fr/recherche/unites/sis/formation/python/ — Python course in Bioinformatics (ang.)