Übung 7 - Python

Timur Olzhabaev

01.06.2018

Einführung

Diese Übung ist eine Einführung in die Programmiersprache **Python** (`https://www.python.org/`). Hierbei handelt es sich um eine einfach zu lernende, *höhere* Programmiersprache (d.h. von den Details des Computers stark abstrahiert), die wegen ihrer Einfachheit und der großen Anzahl verfügbarer Programmbibliotheken im Bereich der Wissenschaften sehr beliebt ist.

Das Ziel ist es den allgemeinenen Programmablauf und grundlegende Elemente der Syntax zu verstehen, um damit anschließend einfache Probleme im Kontext biologischer Sequenzen zu lösen. Weiterhin sollen einige Grundlagen für kommende Übungen gelegt werden, in denen komplexere Sequenzverarbeitung behandelt wird.

Aufgabe 1 Erstellen Sie ein Verzeichnis für die aktuelle Übung mit geeignetem Namen an geeigneter Stelle in ihrem Homeverzeichnis und wechseln Sie hinein, z.B. mit:

```
cd ase
mkdir exercise_7_python
cd exercise_7_python
```

Programmausführung und -ablauf

Python lässt sich als *interaktive* Session, in welcher eingegebene Anweisungen direkt ausgeführt werden, mit dem direkten Aufruf des Python-Interpreters `python` in der Kommandozeile starten (und mit dem Aufruf der Funktion

`exit()` oder `CTRL-D` beenden). Darüber hinaus lassen sich mehrere Anweisungen (und entsprechend ganze Programme) in einem Python-*Skript* (einfache Textdatei mit dem Dateisuffix `.py`) für einfache wiederholte Ausführung zusammenfassen.

Aufgabe 2 Erstellen Sie eine Python-Skriptdatei namens `hello_world.py` im Verzeichnis der aktuellen Übung und öffnen Sie diese mit einem Texteditor (z.B. **Kate**). Schreiben Sie die folgenden zwei Zeilen, bestehend aus einem Kommentar und dem Aufruf einer Ausgabefunktion, in die Datei und speichern Sie diese.

```
# Print a simple string to the console.
print("hello, world.")
```

Führen Sie nun das Skript aus, indem Sie es als Argument dem Python-Interpreter übergeben:

```
python hello_world.py
```

In der Ausgabe auf der Konsole sollten Sie nun den der `print()` Funktion übergebenen Text sehen. Mit '#' anfangende Zeilen sind Kommentare zu Dokumentationszwecken und werden ignoriert. Alle anderen Zeilen werden als Anweisungen interpretiert und, sofern der *Kontrollfluss* es nicht anders vorgibt, nach einander von oben nach unten einmal ausgeführt.

Syntax Grundlagen

Variablen

In Python bestehen Variablen aus einem *Bezeichner* (beliebige zusammenhängende folge von Buchstaben, Zahlen und '_' — darf kein Kennwort der Programmiersprache sein und nicht mit einer Zahl beginnen) und können eine vielzahl unterschiedlicher *Typen* von *Werten* mit dem Zuweisungsoperator '=' zugewiesen werden. Nützlich sind dabei *literale* Konstanten. Dies sind Ausdrücke, die direkt einen Wert repräsentieren und einer Variable zugewiesen werden können. Häufig verwendete Typen sind hierbei *Ganzzahlen* (*integer*), *Gleitkommazahlen* (*floating point*), *Zeichenketten* (*string*) und *Wahrheitswerte* (*boolean*):

```
some_integer_variable = 12
some_floating_point_variable = 12.345
```

```
some_string_variable = "This is free text."
some_boolean_variable = True
another_boolean_variable = False
```

Die Werte der Variablen können durch eine neue Zuweisung jederzeit überschrieben werden.

Arithmetik & Relationen

Wie in jeder Programmiersprache erlaubt Python grundlegende Arithmetik auf Variablen und Literalen mit den allgemein bekannten Operatoren:

```
a = 2
b = 4

should_be_six = a + b
also_six = 3 + 3
should_be_minus_two = a - b
should_be_eight = a * b
should_be_two = b / a
```

Die Vergleichsoperatoren '==' (*gleich*), '!=' (*ungleich*), '<=' (*kleiner-gleich*), '>=' (*größer-gleich*), '<' (*kleiner*) und '>' (*größer*) erzeugen jeweils einen Wahrheitswert als ergebnis:

```
a = 0
b = 1
c = 1

should_be_true = a == a
should_be_false = a != a
should_be_true = a < b
should_be_false = b < c
should_be_true = b <= c
```

Die Operatoren lassen sich nicht nur auf Zahlen, sondern auch auf andere einfache Datentypen anwenden, wie z.B. Zeichen und Zeichenketten:

```
should_be_true = "a" == "a"
should_be_false = "aa" == "ab"
```

Aufrufen von Funktionen

Python verfügt über eine Anzahl nützlicher Funktionen. Eine Funkton kann eine Anzahl von Parametern über die Parameterklammern nehmen (Reihenfolge der Parameter wird beachtet) und einen Wert zurückgeben. Die bereits bekannte `print()` Funktion nimmt eine Zeichenkette als Parameter, erzeugt jedoch keinen sinnvollen Rückgabewert. Eine Funkton mit Rückgabewert ist z.B. `len()`, welche als Parameter eine Zeichenkette (oder andere Datentypen, welche mehrere Daten zusammenfassen) akzeptiert und die Anzahl der Elemente (d.h. Länge) zurück gibt:

```
some_string = "abcde"
should_be_five = len(some_string)
```

Indexierung

Datentypen, die wie Zeichenketten aus einer Reihe von Elementen bestehen, erlauben es auf einzelne Elemente zuzugreifen. Dabei wird der *Index* des gewünschten Elements der Variable über die *Indexklammern* `[]` übergeben:

```
some_string = "abcde"
the_letter_c = some_string[2]
```

Dabei ist zu beachten, dass in Python die Indizes bei `0` beginnen, d.h. `[0]` gibt das erste Element, `[1]` das zweite, usw.

Bedingte Anweisungen und Verzweigungen

Häufig ist es notwendig einige Anweisungen nur unter bestimmten Bedingungen auszuführen. Dies geschieht mit dem Kennwort `if` gefolgt von einem Ausdruck, welcher einen Wahrheitswert ergibt und einem '`:`'. Anweisungen in den folgenden Zeilen, **welche um 4 Leerzeichen eingerückt sind**, werden ausgeführt, wenn der Wahrheitswert `True` ergab. Ist es notwendig im gegenteiligen Fall (d.h. wenn der Asudruck `False` ergab) ebenfalls Anweisungen auszufpühren, so kann dies nach dem `if`-Block mit dem Kennwort `else:` erzeugt werden:

```
a = 10
b = 20
```

```
if a < b:
    print("This is printed when a is smaller than b.")
else:
    print("This is printed when a is larger or equal than b.")

print("This is always printed.")
```

Bedingte Anweisungen können beliebig geschachtelt werden, wobei die Einrückung eines inneren Blocks entsprechend 4 Leerzeichen weiter sein muss, als die des äußeren.

Schleifen

Ist es notwendig Anweisungen einige Male zu wiederholen, so stehen Schleifenausdrücke zur Verfügung, wie z.B. die `while` Schleife, welche Anweisungen solange wiederholt, wie eine bestimmte Bedingung zutrifft. Wie bei dem `if` Ausdruck (und in Python generell, wann immer ein bestimmter Kontext von der Umgebung unterschieden werden soll), werden die zu wiederholenden Anweisungen um 4 Leerzeichen eingerückt. Der folgende Code nutzt die `while` Schleife um alle Buchstaben einer Zeichenkette einzeln in der Konsole auszugeben:

```
some_string = "This is free text."
i = 0
length = len(some_string)

while i < length:
    print(some_string[i])
    i = i + 1
```

Dabei wird zunächst eine Indexvariable `i` erzeugt und auf `0` gesetzt und die Länge der Zeichenkette gespeichert. Die Schleife läuft dann so lange, wie die Indexvariable kleiner als die Länge ist und in jeder Iteration wird der Buchstabe auf Position `i` ausgegeben und anschließend `i` um `1` erhöht.

Die Bedingung der folgenden Schleife wird immer zutreffen und die Schleife läuft endlos:

```
i = 0
while True:
    i = i + 1
```

Ein Skript mit dieser Schleife wird nicht von selbst enden, da das Ende der Datei (und somit der Anweisungen) nie erreicht wird. Es lässt sich nur manuell mit dem Interrupt-Befehl `CTRL-C` beenden.

Sequenzidentität

Aufgabe 3 Erstellen Sie ein neues Pythonskript, welches die Sequenzidentität zweier gleich langer Sequenzen berechnet und in `%` ausgibt. Dabei sollen die Matches an allen Positionen gezählt und normalisiert werden. Gehen Sie dabei folgendermaßen vor:

1. Erzeugen Sie zwei Sequenzvariablen und schreiben Sie ihre Testsequenzen direkt in das Skript.
2. Lassen Sie die längen der Sequenzen jeweils einer Variable zuweisen.
3. Prüfen Sie, ob die Sequenzen die gleiche Länge haben. Ist das nicht der Fall, soll das Programm nur ausgeben, dass die Sequenzen nicht gleich lang sind.
4. Sind die Sequenzen gleich lang, zählen Sie die Anzahl der Matches an allen Positionen mit Hilfe einer Schleife und eines Vergleichs der Zeichen an der aktuellen Position. Sind diese gleich, sollte eine Zählervariable um 1 erhöht werden.
5. Berechnen Sie aus der Anzahl der Matches und der Länge der Sequenzen die prozentuale Sequenzidentität.
6. Geben Sie diese in die Konsole aus. Nutzen Sie die `str()` Funktion, um eine Zahl in eine Zeichenkette umzuwandeln.