Übungsblatt 8

Willkommen zum Praktikum zu Programmieren in Python.

Aufgabe 1. In dem Bild `ite.png` ist eine Nachricht versteckt. Welche Nachricht ist es? Schreiben Sie die Nachricht "Nein, gar nicht. Es war ganz leicht." stattdessen in das Bild und speichern Sie es ab. Verwenden Sie die Python Image Library (PIL). Importieren Sie `PIL.Image`. Mit `Image.open` öffnen Sie ein Bild, auf einem Bild holt `getpixel(pixel)` den RGB-Wert eines Pixel als 3-Tupel. Finden Sie weitere Möglichkeiten von `Image` selbst heraus. Schreiben Sie ein Skript `message.py`, das Sie von der Kommandozeile aufrufen können mit

```
$ ./message.py -w "Die Nachricht" ite.png
```

zum Schreiben der Nachricht und

```
$ ./message.py ite.png
Die Nachricht
```

zum Lesen der Nachricht. Warum verwenden wir hier `PNG` und nicht `JPG`?

Aufgabe 2. Schreiben Sie ein Programm `filter.py`, das mit zwei Dateinamen als Argumente aufgerufen wird:
`<lesebildname>`, ein Dateiname eines zu lesenden Bilds als erstes Argument
`<schreibbildname>`, ein Dateiname eines zu schreibenden Bilds als zweites Argument
Das eingelesene Bild wird nach einer "Binarisierung" wieder abgespeichert, ähnlich zu der Aufgabe mit den Graustufenbildern. Bei der Binarisierung werden alle Pixel, die über einem gewissen Helligkeits-Schwellwert liegen weiß und alle anderen schwarz. Sie können davon ausgehen, dass wir Farbbilder in RGB-Darstellung haben. Für den aktuellen Helligkeitswert h ($0 \leq h \leq 255$) nehmen wir den Mittelwert der Farbanteile (die Summe der Werte der drei Farben durch drei). Erlauben Sie noch zusätzlich einen optionalen Kommandozeilenparameter `-f <wert>` mit einer Angabe des Helligkeitsschwellwerts in ganzzahligen Prozent zwischen 0 und 100 und die Wahl einer Implementierungsvariante mit `-i <variante>`, wobei die Varianten aus `(PY|EPY|BPY|C)` sind.

- `PY` : Realisieren Sie den Filter zunächst in Python mit ähnlichen Hilfsmitteln, wie in der letzten Aufgabe und in der früheren Aufgabe mit den Graustufenbildern.
- `EPY` Pixelweiser Zugriff ist sehr langsam. Viele Bibliotheken, auch die PIL, stellen bessere Alternativen für solche Aufgaben zur Verfügung. Lesen Sie nach was die Methoden `tobytes` und `frombytes` der PIL machen. Realisisieren Sie den Filter effizienter.
- `BPY` Lesen Sie was die Funktion `load` der PIL macht. Sie erhalten ein Pixel-Access-Object, mit dem die naiven Zugriffe schneller als mit `putpixel` gehen. Realisisieren Sie den Filter effizienter.
- `C` Da diese Aufgabe eine typische Aufgabe für maschinennahes Programmieren ist schreiben Sie ein C-Modul, das die eigentlich Filterung für Sie übernimmt. Verwenden Sie `ctypes` zur Integration. Sie brauchen `tobytes` und `frombytes`.

Vergleichen Sie die Ausfühungsgeschwindigkeit (`time.process_time()` zum Messen der verbrauchten CPU-Zeit) der verschiedenen Varianten. Messen Sie nur die Zeit, die die Konvertierung braucht, nicht die Zeit zum lesen und speichern. Beachten Sie Hinweis 1. Ist die C-Variante schneller?

Aufgabe 3. Schreiben Sie reguläre Ausdrücke zum Matchen folgender Objekte.

- Postleitzahlen in Deutschland
- Datum in der Form TT.MM.JJJJ, (Tag.Monat.Jahr)
- Euro-Beträge: Optional mit durch Leerzeichen getrenntem `EUR`. Optional zwei durch Komma getrennte Nachkommastellen. Der Punkt ist erlaubt als Separator für Tausenderbeträge.
- Telefonnummern: Optional mit Länderkennung (z.B. +49) durch Leerzeichen abgesetzt und ohne 0 bei der Vorwahl. Als Trennzeichen sind Leerzeichen, / und - erlaubt. Zwischen zwei Trennzeichen sind mindestens 2 Ziffern notwendig.
- Email-Adressen: Vor dem `@` beliebige Buchstaben- und Ziffern-Kombinationen mit führendem Buchstaben. Optional getrennt durch ein Punkt oder ein `-`. Muss mit Buchstaben oder Ziffer enden. Hinter dem `@` alles in Kleinschreibung und mindestens ein Domainname plus Länderkennung.

Testen Sie Ihre regulären Ausdrücke in Python. Nutzen Sie dazu die Datei `reg.txt`. Schauen Sie sich die Struktur der Datei an und entwickeln Sie ein Skript, um Ihre Ausdrücke zu testen. Welche Zeichenketten passen sollen ist in der dritten Spalte angegeben.

Verpacken Sie die Tests in einen `unittest` unter `test_regexe.py`. Es soll fünf Tests geben. Lesen Sie die Datei an der passenden Stelle ein und bereiten Sie die Tests vor.

Aufgabe 4. Sie sollen die Straßennamen von Mannheim analysieren. Glücklicherweise gibt es alle Straßen von Mannheim unter

`mannheim.opendatasoft.com/explore/dataset/strassennamen-in-mannheim/export/`

Sie können die Daten live mit der Bibliothek `urllib.requests` abholen. Sinnvoll ist das JSON-Format. Nutzen Sie das Modul `json`. Damit uns nicht der Zugang wegen der vielen Requests gesperrt wird, speichern wir uns die Datei unter dem Namen `strassenma.json` zwischen und holen Sie uns nur, wenn `strassenma.json` noch nicht existiert.

Sie möchten folgendes wissen: Was ist der längste Straßenname in Mannheim? Was ist die durchschnittliche Länge der Straßennamen in Mannheim? Wie viele Straßennamen einer bestimmten Länge gibt es in Mannheim, aufsteigend nach Länge sortiert.

Schreiben Sie ein ausführbares Python-Skript `strassen.py`, das Ihnen diese Fragen beantwortet. Beachten Sie, dass alle Daten immer schlecht sind, das merken Sie bei den Ergebnissen. Filtern Sie schlechte Daten heraus[1].

Hinweis 1. Sie erzeugen eine „shared Library“ durch Angabe von `-fPIC -shared` als Kommandozeilenoption für `gcc`. Das Ergebnis sollte die Endung `.so` haben und kann mit Hilfe von `ctypes` mit der Funktion `cdll.LoadLibrary` geladen werden. Beachten Sie, dass Sie den aktuellen Pfad zum Laden noch mit zum Dateinamen dazu nehmen müssen. Als Kommandozeilenoption zum `gcc` nehmen wir üblicherweise noch
`gcc -O2 -Wall -C11 -pedantic ....`

moodle Python_Barth

[1] Ein Bonuspunkt, wer es schafft, dass die Stadt die Daten fixed ;-).