Editorial

Liebe Leserinnen und Leser,

Sie halten kein Lehrbuch in Händen. Stattdessen präsentiert dieses Sonderheft Projekte, die jeweils für sich ein Problem vorstellen und es mit Python lösen. Trotzdem können Sie Ihre Python-Skills mit diesem Heft vom blutigen Anfänger bis zur Fachgröße steigern. Unsere Sammlung enthält nämlich vom Computerspiel „Knäueljagd“ (S. 28) für programmierende Kinder über schnelle Projekte wie ein Bilder-Sortier-Skript (S. 60) bis zum Details erfindenden tiefen Convolutional Network in der KI (S. 84) alle Schwierigkeitsgrade.

Statt einem Semester mit Vorlesungen haben Sie eine Zusammenstellung mit Beispielen ohne didaktisches Konzept. Mit denen können Sie einfach loslegen – ohne Studienordnung, ohne ein halbes Jahr Vorlauf, ohne Mathe-Vorkurs. Passen Sie unsere Projekte dabei gern an Ihre eigenen Bedürfnisse an: So lernen Sie nämlich am meisten. Das beste dabei ist aber, dass Sie an Ihren und unseren Projekten nicht nur lernen. Jedes Projekt löst auch ein konkretes Problem.

Ein bisschen haben wir aber dennoch für eine gute Lernkurve gesorgt: Nach dem Einstieg am Anfang des Heftes finden Sie bei den Alltagshelfern ab Seite 40 die richtigen Herausforderungen. Für ambitionierte Hobbyisten bieten unsere nerdigen Hardcore-Projekte ab Seite 154 gutes Futter und unsere Reihe zum automatischen Testen ab Seite 120 ist genau richtig für Berufsprogrammierer, denen Code-Qualität und Performance am Herzen liegen. Und mit unseren KI-Projekten ab Seite 62 werden Sie zum Data-Scientist.

Stürzen Sie sich also einfach in die Python-Projekte, die Ihnen am meisten Spaß machen! Lernen werden Sie dabei ganz automatisch.

Pina Merkert

Inhalt

PYTHON LERNEN

Mit einer klaren und übersichtlichen Syntax eignet sich Python hervorragend als Einstiegssprache. Wir erklären die ersten Schritte mit dem sicheren und alltagstauglichen Passwort-Manager c't SESAM.

6 Elegante Entschleunigung
8 Python lernen mit c't Sesam
20 c't SESAM objektorientiert erweitern
28 Spiele mit Python und Pygame, Teil 1
34 Spiele mit Python und Pygame, Teil 2

PYTHON ALS ALLTAGSHELFER

Bereits mit wenigen Zeilen Code programmieren Sie nützliche Helfer. Mit diesen Projekten lernen Sie Frameworks kennen, die komplizierte Probleme ganz einfach lösen.

40 Zwei Faktoren mit OAuth2
42 Auf das Google-Fit-API zugreifen
48 Texterkennung mit Tesseract
54 Python statt Bash
56 PDFs aus Code
60 Urlaubsbilder sortieren mit EXIF-Daten

KI MIT NEURONALEN NETZEN

Ein Blick hinter den Hype ums maschinelle Lernen: Mit unseren KI-Projekten klappt der Einstieg in die Entwicklung künstlicher Intelligenzen. Die Projekte zeigen an konkret nutzbaren Beispielen, wie man KIs erfolgreich trainiert.

62 KI für Einsteiger
70 Sacred verwaltet KI-Experimente
78 8-Bit-KI mit TensorFlow-Lite
84 Bilder skalieren mit TensorFlow 1
90 Mit LSTMs Texte verschlagworten
98 Den Stil eines Malers auf Fotos anwenden
104 TensorFlow erkennt schlechte Bilder
112 Listen in der PyQt5-GUI zur KI

AUTOMATISCHES TESTEN

Automatische Tests finden Fehler im Code und sparen Zeit. Der zusätzliche Programmieraufwand lohnt sich schon nach wenigen Releases.

120 Automatisches Testen mit unittest
126 Django testen
130 Web-GUIs testen mit Selenium
136 Mutationstests mit MutMut
140 Hypothesis erfindet Textbeispiele

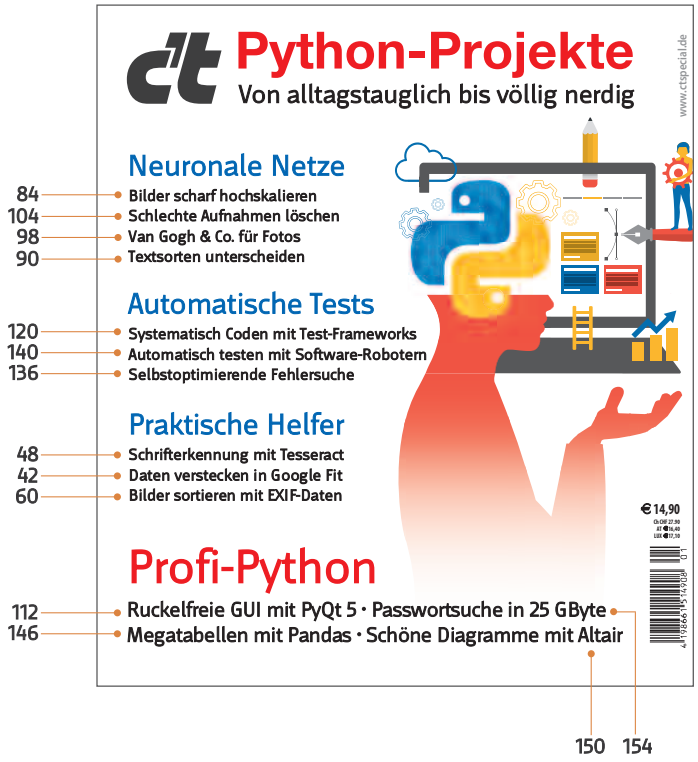
PYTHON FÜR PROFIS

Python vereinfacht viele aufwendige und komplexe Berechnungen. Mit den richtigen Tools gelingt alles von Wissenschaft bis Web-Anwendung.

146 Binäre Suche in 25 GB Passwörtern
150 Genom-Datamining mit Pandas
154 Diagramme mit Altair
162 COVID-19-Rechenmodelle
166 Beliebige Bäume in flachem SQL
172 Pandoc in Flask
176 API-Calls in Threads in PyQt5

ZUM HEFT

3 Editorial
178 Impressum



Programmieren lernen mit c't SESAM

Programmierkenntnisse eröffnen am Computer neue Möglichkeiten. Damit der Rechner folgt, muss man aber erst mal dessen Sprache lernen. Python macht den Einstieg leicht: Wenige Zeilen Code reichen, um einen sicheren und alltagstauglichen Passwort-Manager auf die Beine zu stellen.

Von Pina Merkert

Für den Bau einer Kathedrale bedarf es Hunderter hoch spezialisierter Baumeister und viel Zeit. Bei komplexer Software ist es ähnlich: Niemand programmiert mal eben so eine Photoshop-Alternative oder einen Browser. Aber man muss weder Baumeister noch Handwerker sein, um in der eigenen Wohnung einen Dübel zu setzen.

Ganz ähnlich verhält es sich beim Programmieren. Gut 20 Zeilen Python reichen zum Schreiben eines Passwort-Managers, der mehr als ein Spielzeug ist. Für den schnellen Einstieg ins Programmieren erklären wir anhand dieses Beispiels die wichtigsten Konzepte.

Ein Programm entwerfen

Ein Programm ist eine Anleitung, die der Computer von oben bis unten abarbeitet. Bevor Sie die erste Zeile Code schreiben, sollten Sie eine klare Vorstellung haben, was der Computer machen soll. Programmieren fordert Sie heraus, große Probleme so klein aufzuteilen, dass der Rechner die Teilprobleme lösen kann. Mit zunehmender Erfahrung erarbeiten Sie immer mehr Lösungen für solche Teilprobleme.

Jedes gelöste Teilproblem kann im nächsten Programm wieder eingesetzt werden, um ein komplexeres Problem zu lösen. Wir empfehlen Ihnen, mit einfachen Beispielen anzufangen und sich erst nach und nach an komplexere Probleme zu wagen. Wenn Sie mit Python gelernt haben, große Probleme in lösbare Teilprobleme aufzubrechen, können Sie diese Erfahrung bei allen anderen Programmiersprachen und auch abseits vom Computer nutzen.

Für die Planung eines Programms empfehlen sich je nach Programm unterschiedliche Vorarbeiten. Für grafische Programme skizzieren Sie am besten zuerst die gewünschte Oberfläche. Für Datenbankanwendungen sollten Sie sich vorab überlegen, welche Daten das Programm speichert und unter welchen Bedingungen es darauf zugreift. Ein einfacher Passwort-Manager auf der Konsole muss nur die Eingabe und Verarbeitung der Daten in der richtigen Reihenfolge gewährleisten.

Passwort-Meister

Sichere Passwörter sind ein Problem: Sie sollen länger als 10 Zeichen sein, dürfen in keinem Wörter-

Passwort-Konzepte

Normal: Der Anwender muss sich viele Passwörter merken und schickt sie dann übers Netz an die Dienste, um sich auszuweisen.

Master-Passwort: Eine kleine App erzeugt für jeden Dienst ein eigenes, unknackbares Passwort. Der Anwender muss sich nur noch sein Master-Passwort merken.

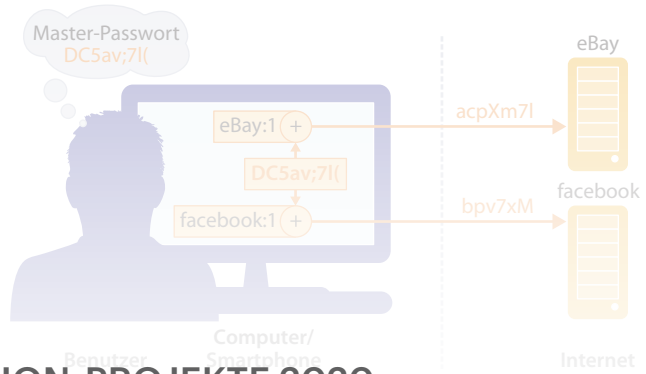


Bild: Albert Hulm

Auf das Google-Fit-API zugreifen

Google Fit ist nicht nur eine App, die automatisch sportliche Aktivitäten erkennt. Unter gleichem Namen bietet Google auch ein API zum Zugriff auf den dahinterliegenden Cloud-Service an. Mit dem API lassen sich zahlreiche Daten auslesen und schreiben, die die Google-App nicht anzeigt. Wir erklären, wie Sie mit Python an alle Daten kommen.

Von Pina Merkert

Google Fit speichert Workouts, Schritte, Gewicht, Herzfrequenz und vieles mehr im „Google Fitness Store“. Auf den können auch andere Apps, Webanwendungen und Programme über das Google-Fit-API zugreifen. Für Android hat Google den API-Zugriff über die Play-Services implementiert. Für alle anderen Plattformen bietet der Konzern ein REST-API an. Das API nutzen bereits viele Anwendungen und speichern dort auch Daten, die Googles Fitness-App gar nicht anzeigen kann, beispielsweise Ernährungsdaten. Wir wollten natürlich Zugriff auf alles und auch selbst Daten in Googles Fitness-Cloud ablegen.

Dafür muss man sich zunächst per OAuth2 bei Google anmelden (siehe S. 40). Google erlaubt das nur Apps, die Google kennt, sodass man die eigene Anwendung zuvor in der Cloud-Developer-Console anlegen muss (siehe Kasten). Bei der Anmeldung legt man auch fest, auf welche Daten die eigene Anwendung zugreift, und Google lässt diesen Datenzugriff auch vom Nutzer abnicken, bevor das API Daten liefert.

Um die Daten anzuzeigen, haben wir ein grafisches Programm mit PyQt5 geschrieben. Für die Anmeldung bei Google zeigt die eine URL an, die man in einem aktuellen Browser wie Firefox oder Chrome öffnet und dort die Anmeldung durchspielt. Am Ende des Prozesses steht dort ein Code, den man wieder in die Anwendung kopiert. Das funktioniert auch in der Konsole, weshalb die hier

gezeigten API-Zugriffe auch ohne Qt und GUI funktionieren.

Datenquellen

Eine Grundidee des Google-Fit-API besteht darin, dass alle Daten eine Quelle und einen Typ haben. Stammen Datenpunkte beispielsweise von einem Bluetooth-Sensor, der mit einer App auf dem Smartphone verbunden ist, legt die App eine Datenquelle für den Sensor in Verbindung mit dieser App an. Auf diese Weise registrieren Apps meist einige verschiedene Datenquellen bei Google Fit. Für jeden unterschiedlichen Sensor und Typ gibt es eine eigene Quelle. Auch aggregierte Daten wie die Summe der gelaufenen Schritte über einen Tag bekommen eine eigene Datenquelle.

Ohne besondere Freischaltung durch Google können Apps die Daten anderer Apps nicht löschen oder verändern, die meisten Daten aber lesen. Will eine Anwendung wie unser Python-Programm Daten editieren, muss es daher eigene Datenquellen registrieren und die veränderten Daten aus diesen Quellen höher priorisieren als die aus anderen Apps.

Außerdem gibt es unterschiedliche Typen von Daten. Google liefert einige Standardtypen wie Aktivitäten, Herzfrequenzen, Geschwindigkeiten, verbrauchte Kalorien, Ernährungsdaten, Größe, Gewicht oder Blutdruck. Zusätzlich können Anwendungen in ihren Datenquellen eigene Typen definieren. Diese sind aber zunächst privat, sodass nur diese Anwendung die Daten lesen kann. Sollen die Daten auch anderen Apps zur Verfügung stehen, können sich Entwickler bei Google melden und dort beantragen, dass ihre Datenquellen „shared“ werden und damit von allen Apps gelesen werden dürfen.

Will eine Anwendung wie unsere nicht nur die selbst gespeicherten Daten auslesen, muss sie zunächst die Liste aller Datenquellen abrufen. Das geht mit einem GET-Request an `https://www.googleapis.com/fitness/v1/users/me/dataSources`. Die URL für dieses API setzt sich aus der Adresse des API (`https://www.googleapis.com/fitness/`), einer Versionsnummer (`v1/`), der Angabe des Nutzers (momentan unterstützt Google ausschließlich `users/me/`) und dem Endpunkt (`dataSources`) zusammen. Die Anfragen an das REST-API nutzen alle dieses Schema und hängen lediglich noch weitere Informationen an die URL an.

Das Google-Fit-API antwortet grundsätzlich mit Daten im JSON-Format. Für die Nutzung mit `requests_oauth` ist das praktisch, da die Bibliothek mit der `.json()`-Methode gleich den Inhalt parsen und als Python-`dict` oder -`list` zurückgeben kann. Das Abfragen aller Datenquellen geht daher in zwei Zeilen:

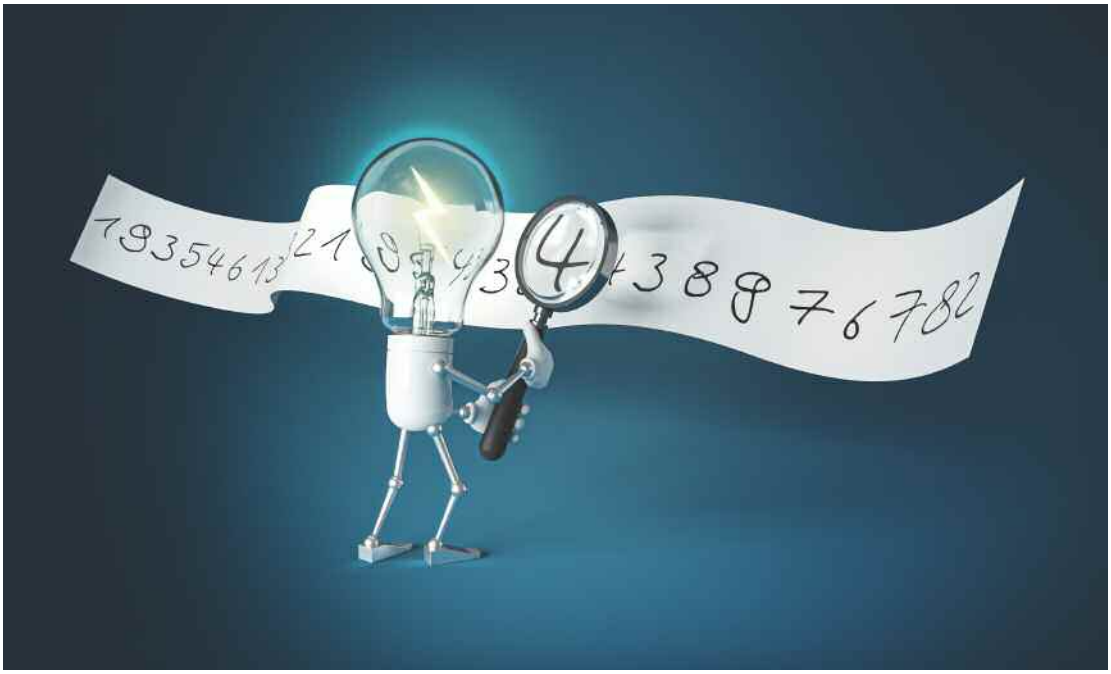
```
response = google_fit.get(
   "https://www.googleapis.com" +
   "/fitness/v1/users/me/dataSources")
data_sources = response.json()[
                        'dataSource']
```

In der Antwort steht bei jeder Datenquelle eine `"dataStreamId"`, die Sie verwirrenderweise beim Abfragen der Datensätze als `"dataSourceId"` angeben müssen.

Workouts laden

Mit der Liste der Datenquellen kann man nun alle Daten eines bestimmten Typs laden, beispielsweise `"com.google.activity.segment"`. Fitnessaktivitäten haben diesen Typ. Wie alle Daten in Google Fit ha-

Ausschnitt aus der Antwort des API, angezeigt in Firefox: In ‚points‘ stehen die Workouts mit Zeiten und der Nummer der Aktivität in ‚value‘.



KI für Einsteiger

Neuronale Netze klingen nach Gehirn und komplexer Biologie. Maschinelles Lernen leiht dort aber nur primitive Ideen. Eigentlich erstellt der Rechner nämlich nur selbstständig Statistiken – und die wenigen dafür nötigen Zeilen Python kann jeder auf dem heimischen Rechner leicht selbst programmieren. Um die Mathematik dahinter kümmert sich das Framework Keras automatisch.

Von Pina Merkert

Computer sind dumme Maschinen. Sie arbeiten zwar Millionen von Befehlen pro Sekunde ab, es muss aber stets ein Programmierer exakt festlegen, was die Maschine wann tun soll. Das ist das Prinzip klassischer Programme von Menschenhand.

Einen ganz anderen Ansatz verfolgt künstliche Intelligenz in Form des maschinellen Lernens (ML): Der Rechner bleibt zwar gleich dumm, aber Framework-Programmierer haben lernfähige Algorithmen vorgegeben, die sich selbstständig an eine Herausforderung anpassen können. Diese Anpassung erfolgt in einer Trainingsphase, in der die Algorithmen Tausende verschiedener Beispiele zu sehen bekommen und interne Parameter dabei so verändern, dass sie möglichst gut den Beispielen entsprechen. Ein Lernalgorithmus startet also zumeist in einem Zustand, in dem er kein Problem zufriedenstellend löst. Durch das Training spezialisiert er sich auf die Aufgabe, die der Datensatz implizit vor-

gibt. Erst nach dem Training meistert der Algorithmus die gestellte Herausforderung, was Data Scientists als „Inferencing“ bezeichnen.

Es gibt bereits seit vielen Jahrzehnten solche lernfähigen Algorithmen. Die „klassischen Verfahren“ bauen auf einfachen Ideen auf. K-Nearest-Neighbour sucht beispielsweise eine vorgegebene Zahl (k) an Beispielen aus den Trainingsdaten, die am ehesten zu einer neuen Eingabe passt, und mittelt für seine Vorhersage die Ausgaben aus diesen Beispielen. Entscheidungsbäume lernen, anhand einer Kaskade von Wenn-Dann-Entscheidungen die passenden Ausgaben zu liefern. In der ersten Kinect hat diese einfache Idee gereicht, damit Microsoft damit Körperteile erkannte. Bayesian-Networks, Hidden-Markov-Models und Support-Vector-Machines erstellen alle automatisierte Statistiken und liefern teils erstaunlich gute Ergebnisse. Die Qualität der Vorhersagen hängt nämlich nur bedingt vom verwendeten Algorithmus ab. Auch klassische effiziente ML-Algorithmen liefern mit guten Trainingsdaten akkurate Resultate.

Gute Daten sind teuer

Trainingsdaten zu erzeugen war aber lange Zeit ein zu großer Kostenfaktor. Oft war es billiger, wenn ein Programmierer das Problem durchschaute und dann den für die Lösung notwendigen Code per Hand zusammenschrieb. Das änderte sich jedoch mit dem „Deep Learning“. Das „tiefe“ Lernen nutzt neuronale Netze, einen Lernalgorithmus, den es bereits seit den 1950er-Jahren gibt. Dessen simulierte Neuronen lassen sich leicht stapeln, was für tiefe Strukturen sorgt. Die Tiefe nutzt das Deep Learning dazu, aus wenig vorstrukturierten Datensätzen automatisch Features zu extrahieren. Mit denen kann das Netzwerk erlernen, die richtigen Ausgaben zu berechnen, obwohl kein Programmierer die Daten aufwendig per Hand vorverarbeitet hat. Man braucht für diese Art von Lernalgorithmus zwar noch mehr Trainingsbeispiele als für die klassischen Verfahren, spart aber Programmierarbeit.

Maschinelles Lernen ist damit etwas für Programmierer, die es nicht so genau nehmen wollen: Statt eines exakten Algorithmus wählt man nur ein grobes Modell für die Berechnung. Dazu sammelt man den Datensatz mit Beispielen und lässt den Rechner selbst herausfinden, wie die Details aussehen müssen, damit das Modell zu den Beispielen passt.

Gerade beim Deep Learning bestehen die Modelle aber aus unüberschaubar vielen Neuronen mit Millionen von Parametern, die sich während des Trainings alle gleichzeitig ändern. Mancher fühlt sich daher von der Komplexität solcher Systeme erschlagen. Die zugrunde liegenden Ideen sind aber ausgesprochen einfach und die Netzwerke wenden lediglich die immer gleiche Idee wieder und wieder an. Mit dem Code aus diesem Artikel trainieren Sie selbst auf betagten Notebooks im Handumdrehen ein neuronales Netz. Keine Angst vor komplexen Netzen: Der beschriebene Code fängt ganz klein an.

Ein Neuron

Maschinelles Lernen beginnt stets mit einem Datensatz (siehe Tabelle). Im ersten Beispiel geht es darum, aus dem Status von Pinas smartem Fahrradschloss und dem Stromverbrauch gemessen an ihrem Lichtschalter vorherzusagen, ob Pina gerade schläft oder nicht. Eine einfache Funktion, um das zu berechnen, wäre:

```
y_ = max(w1 * x1 + w2 * x2 + b, 0)
```

Bei `x1` handelt es sich um den Status des Schlosses, bei `x2` um den gemessenen Stromverbrauch in Watt. `w1`, `w2` und `b` sind Parameter, die der Rechner aus den Daten lernen soll. Die `max()`-Funktion mit `0` ist eine Aktivierungsfunktion, die sogenannte „rectified linear unit“ (ReLU). Die ReLU sorgt dafür, dass das Ergebnis `y_` nie negativ wird. Diese einfache Berechnung als Funktion zu definieren reicht als Modell schon aus.

Beim Training geht es nun darum, die drei Parameter automatisch so zu wählen, dass es für die Daten aus der Tabelle keinen Unterschied zwischen der errechneten Vorhersage für den Schlafstatus `y_` und dem gemessenen Status `y` gibt. Da der Rechner dabei nicht sofort richtig liegt, definiert man die Loss-Funktion, ihr Wert muss klein

Datensatz: Pina schläft?

Zeit	x1: Schloss	x2: Strom	y: Pina schläft
15:00	0	0	0
17:00	0	0	0
19:00	1	0	0
21:00	1	20	0
23:00	1	20	0
01:00	1	0	1
03:00	1	0	1
05:00	1	0	1

Bild: Albert Hulm

Automatisches Testen mit unittest

Warum sollte man Software per Hand testen, wenn Unit-Tests die Arbeit automatisch erledigen? Der Zusatzaufwand zum Programmieren der Tests lohnt sich schon nach wenigen Releases und die Tests helfen, Fehler zu vermeiden und sauber zu strukturieren.

Von Pina Merkert

Programmieren wäre einfach, wenn der Code nicht funktionieren müsste. Stattdessen versuchen Programmierer, so gut es geht, ihren Code zu verstehen, damit der in allen Fällen ohne Fehler genau das tut, was die Auftraggeber wünschen. Die Prüfung nach der Auslieferung frisst eine Menge Zeit. Meist wollen die Prüfer auch noch Zeit sparen und übersehen dann Fehler.

Die Infamie manueller Tests offenbaren die weiteren Releases: Falls der Test zeigt, dass der Programmierer etwas ändern muss, wiederholt sich der gesamte Zyklus. Am Ende der zweiten Iteration testen die Prüfer dann dieselben Features noch mal, was mit zunehmender Zahl an Iterationen zu einer ziemlich stupiden Arbeit verkommt. Und immer, wenn ein Mensch am Rechner stupide Arbeiten verrichtet, keimt die Idee, ob man diese denn nicht automatisieren könnte.

Genau dazu dienen automatische Tests. Das sind kleine Programme, die anderen Programmen

auf die Finger schauen und prüfen, ob deren Ausgaben den Erwartungen entsprechen. Wir haben diverse Tests für unseren Flask-Microservice (siehe Seite 172) geschrieben und nutzen den dort vorgestellten Markdown-Konverter als Beispiel, wie automatische Tests einer Python-Anwendung aussehen. Den Code finden Sie im Git-Repository über ct.de/wvau.

Randfällig

Jeder automatische Test prüft nur einen Aspekt einer Funktion. Dadurch muss man beim Programmieren der Tests nicht lange nachdenken: Zu je einem Beispiel gehört eine bestimmte erwartete Ausgabe. Um eine Funktion jedoch vollständig zu testen, sind oft Dutzende von Beispielen nötig. Schließlich sollen die Tests jeder Verzweigung folgen und jeden Randfall abdecken.

Dass die Tests dabei jeweils ähnlich aussehen und Code duplizieren, muss den Tester nicht kümmern. Jeder Test soll für sich einfach zu schreiben und leicht zu verstehen sein. Performance und Ästhetik spielen eine untergeordnete Rolle.

Viel wichtiger ist dagegen, dass die Tests alle Fehler finden, die sich im Code verstecken könnten. Für eine Funktion, die eine Liste mit Zahlen überprüft, ob sie größer als 5 sind, bedeutet das beispielsweise folgende Tests:

- Die Funktion wirft einen Fehler, wenn man keine Liste übergibt.
- Die Funktion wirft einen Fehler, wenn die Liste nicht nur Zahlen enthält.
- Die Funktion gibt `False` zurück, wenn man eine Liste mit Werten übergibt, bei der einer der Werte exakt 5 ist.
- Die Funktion gibt `True` zurück, wenn man eine Liste mit genau einer Zahl übergibt, die minimal größer als 5 ist, beispielsweise 5,001.

Je nach Programmiersprache ist es auch sinnvoll, negative Zahlen und verschieden lange Datentypen zu prüfen. Im Zweifel sollte man lieber einen Test mehr als zu wenig schreiben.

Häppchenweise

Da jemand die Tests programmieren muss, fällt beim testgetriebenen Entwickeln zwangsläufig mehr Arbeit für den Programmierer an. Damit die Tests schnell durchlaufen, sucht man deswegen, die Tests so klein und einfach wie möglich zu schreiben. Testet ein automatischer Test nur eine einzelne Funktion (im Tester-Jargon eine „Unit“), handelt es sich um einen Unit-Test.

Damit die Tests einzelne Funktionen aufrufen können, liegen sie im gleichen Repository wie der produktive Code, meist in einem Unterordner `tests/`. Jeder Test prüft nur ein einzelnes Feature mit einem bestimmten Satz an Eingaben. Um dafür möglichst wenig Code zu brauchen, bringt Python das Modul `unittest` mit. Für andere Programmiersprachen gibt es stets mindestens ein Test-Framework (oft sogar mehrere), das nach der gleichen Idee funktioniert wie das hier vorgestellte `unittest` für Python.

Ein Unit-Test sieht beispielsweise so aus:

```
import unittest
class TestPandocStringConverter(
                    unittest.TestCase):
  def test_single_string(self):
    span_list = convert_list([
        {'t': 'Str', 'c': 'Foo.'}
    ], [])
    self.assertEqual([{
        "type": "span-regular",
        "text": "Foo."
    }], span_list)
```

Nach dem Import von `unittest` definiert der Test eine Klasse, die von `unittest.TestCase` erbt. Alle in solch einer Klasse definierten Tests führt das Framework aus, wenn der Code in der gleichen Datei `unittest.main()` aufruft:

```
if __name__ == '__main__':
  unittest.main()
```

In der Klasse kann man nach Herzenslust Methoden definieren, die im `TestCase`-Objekt Eigenschaften setzen. Automatisch ausgeführt werden solche Funktionen aber nicht. Erst wenn der Name einer Methode mit `test` beginnt, führt `unittest` sie automatisch als eines der Test-Programme aus.

Der `TestCase` bringt zusätzlich eine ganze Reihe von Prüf-Funktionen mit, die mit `assert` anfangen und ganz ähnlich wie Pythons Schlüsselwort `assert` funktionieren. Das Verb „to assert“ bedeutet bei automatischen Tests „zusichern“. Eine Assertion sichert zu, dass an dieser Stelle im Programm die angegebene Bedingung erfüllt ist. Bleibt eine Zusicherung unerfüllt, wirft der Python-Interpreter einen Fehler, [illegible] sprechen benennt.

Im Beispiel prüft `self.assertEqual()`, ob in der Variable `span_list` (zweiter Parameter) die im ersten

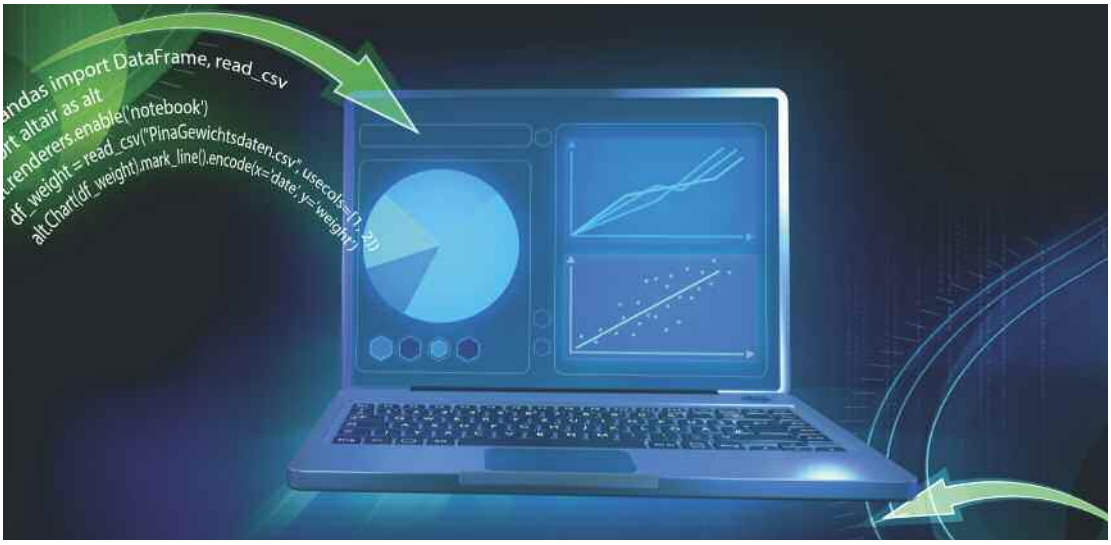


Diagramme mit Altair

Mit etwas Übung schreibt man Code schneller, als man klickt. Mit den Frameworks Pandas und Altair bereitet man in einer Handvoll Python-Zeilen Datensätze auf und erzeugt schicke Diagramme. Altair basiert auf dem JavaScript-Framework Vega, sodass sich die Plots mühelos im Web einbinden lassen – auch als interaktive Grafiken.

Von Pina Merkert

Ein Klick auf den Download-Link befördert eine 25 Megabyte große CSV-Datei ins Download-Verzeichnis. Ein Doppelklick darauf startet LibreOffice und danach heißt es warten. Der Import dauert nämlich auf einem nicht ganz taufrischen i5 über 30 Sekunden. Als die Riesen-Tabelle endlich erscheint, ungeduldig die Formel für den Durchschnitt in die freie Spalte ganz rechts tippen und anschließend eine Minute lang scrollen, um die Formel auf alle Zeilen zu übertragen. Das fühlt sich alles ziemlich zäh an. Führt man sich vor Augen, dass Office die Daten alle in sein Interface rendern muss, fällt auf: Das Programm ist eigentlich nicht langsam. Es ist nur nicht für so große Tabellen gemacht.

Damit das alles flotter geht, muss das grafische Interface weichen. Stattdessen kommt Python zum Einsatz, genauer die Bibliothek Pandas. Pandas nutzt unter der Haube Numpy und damit schnellen C-Code, um Arrays effizient zu speichern. Statt in `numpy.ndarray` (ohne Spaltennamen) landet alles in Objekten vom Typ `pandas.DataFrame`. So ein `DataFrame` ist gewissermaßen ein Tabellenblatt für Programmierer, das die gleichen Funktionen wie Excel bereitstellt. Für Numpy-Veteranen bekannt, für

Python eher ungewöhnlich: Die Daten in `DataFrames` sind hart typisiert. Wenn Sie ein DataFrame anlegen, müssen Sie daher gleich festlegen, ob dort Gleitkommazahlen, Ganzzahlen, Strings oder Zeitangaben in den Spalten stehen. Außerdem sind Spalten generell benannt, was in großen Tabellen für Übersicht sorgt.

Das Framework Altair zeichnet aus einem DataFrame mit einer Handvoll Zeilen ein hübsches Diagramm. Es bietet dafür eine kürzere und logischere Syntax als andere Python-Plotting-Bibliotheken wie Matplotlib. Altair erfindet das Rad nicht neu, sondern setzt intern auf der JavaScript-Plotting-Bibliothek Vega (oder Vega-Lite) auf, weshalb das Framework ohne Mehraufwand neben PNGs und SVGs auch Webseiten exportiert. Altairs integrierte Web-Affinität nutzt man am bequemsten, indem man den gesamten Python-Code gleich in einem Jupyter-Notebook schreibt, dort erscheinen die Diagramme direkt in der Weboberfläche des Notebook.

Damit die Datenanalyse mit Python Spaß macht, brauchen Sie also Pandas, Altair, Vega und Jupyter. Wie Sie diese Pakete am leichtesten beschaffen, hängt vom Betriebssystem ab.

Installation: Anaconda

Unter Windows haben wir gute Erfahrungen mit der Python-Distribution Anaconda gemacht, deren Installer Sie unter `www.anaconda.com/distribution/` herunterladen. Anaconda arbeitet mit „Umgebungen“, die die Abhängigkeiten verschiedener Python-Projekte voneinander abschotten. Lassen Sie den Installer ruhig eine Default-Umgebung einrichten. Die trägt alle nötigen Pfade in Umgebungsvariablen ein. Anaconda aktiviert die Umgebung auch gleich automatisch in allen neu gestarteten Konsolenfenstern. Wenn Sie das stört, schalten Sie es einfach nach der Installation mit folgendem Befehl ab:

```
conda config --set auto_activate_base false
```

Wenn Sie nun eine getrennte Umgebung namens „datavis“ für Ihr Projekt einrichten wollen, geht das mit folgendem Befehl:

```
conda create -n datavis python=3.8
```

Nutzen Sie auf jeden Fall Python 3 und nicht die [illegible] conda nicht findet, geben Sie den ganzen Pfad an, beispielsweise:

```
~/anaconda3/bin/conda create ↵
                ↳-n datavis python=3.8
```

Unter Windows erzeugen und aktivieren Sie die Umgebungen über ein Menü. Linux-Nutzer von Anaconda aktivieren die Umgebung mit folgendem Befehl:

```
conda activate datavis
```

Die Bibliotheken installieren dann folgende Befehle auf der Konsole (unter Windows starten Sie dafür innerhalb der Umgebung eine Anaconda-Konsole):

```
conda install numpy pandas vega altair jupyter
```

Installation: Pip

Linuxer sparen ein paar Befehle mit der Python-Umgebung des Systems. Virtualenv erzeugt eine virtuelle Umgebung:

```
mkdir datavis && cd datavis
python3 -m venv env
source env/bin/activate
```

Pip installiert die Bibliotheken danach mit folgenden Befehlen:

```
pip install wheel numpy pandas vega altair jupyter
```

Wie gebe ich Geld aus?

Mit der perfekt vorbereiteten Arbeitsumgebung kann die Analyse losgehen. Starten Sie dafür den lokalen Webserver des Jupyter-Notebooks:

```
jupyter notebook
```

Der Befehl öffnet auch gleich ein Browser-Fenster mit der URL `127.0.0.1:8888/tree`, das eine Übersicht anzeigt. Mit dem Menü unter „New“ (oben rechts) starten Sie ein neues Notebook für Python3.

In dessen erste Zelle importieren Sie Pandas, Numpy und Altair:

```
import pandas as pd
import numpy as np
import altair as alt
```

Den Code einer Zelle im Jupyter-Notebook führen Sie mit Shift+Enter aus.

Nun gilt es, einen Datensatz zu besorgen. Für ei[illegible]e ich eine CSV-Datei aus meinem Online-Banking exportiert und jeder meiner Ausgaben über ein Jahr eine Kategorie zuge-