Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatikdienste
SWS / SIS

Benno Luthiger / Uwe Schmitt

Zürich, 2016

EuroPython 2016, Konferenz-Bericht

Die seit 2002 jährlich stattfindende EuroPython ist mit inszwischen mehr als 1300 Besuchern die weltweit zweitgrößte Konferenz zur Programmiersprache Python. Im Gegensatz zu den spezialisierten Konferenzen wie DjangoCon (für Webprogrammierer) oder der EuroSciPy (für wissenschaftliches Rechnen) spricht sie ein breit gefächertes Publikum an.

Wie immer von Freiwilligen organisiert fand sie 2016 zum zweiten Mal in Bilbao statt. Die Hauptkonferenz dauerte fünf Tage und bot in mehreren parallelen Tracks mehr als 100 Vorträge sowie Tutorien an. Es ging ein Tag mit Schulungen für Anfänger voraus, und am Ende folgten zwei Tage „Sprints“ diverser Opensource Projekte.

Häufig vertretene Themen der Europython 2016

Die EuroPython zeigte unter anderem auf in welchen Gebieten die Programmiersprache Python sowie damit entwickelte Software und Werkzeuge zum Einsatz kommen. Hier eine Liste der Schwerpunkte:

- Die direkte Messung von Gravitationswellen durch den Forschungsverbund LIGO-VIRGO war eine große wissenschaftliche Sensation im Februar 2016 und bestätigt erneut Einsteins allgemeine Relativitätstheorie sowie bisher offene Theorien der Astrophysik. Die Keynote von James Rollins sowie der Vortrag von Elena Cuoco beschäftigten sich mit diesem Thema. Hier kommt Python sowohl in der Steuerung der Anlagen, sowie bei der Messwerterfassung und Auswertung weitgehend zum Einsatz.

- Python in der Auswertung und Visualisierung wissenschaftlicher Daten (“data science”), inbesondere im Gebiet des maschinellen Lernens. In den letzten Jahren wurde auf dem Gebiet der tiefen neuronalen Netze (“deep learning”) erstaunliche Fortschritte erziehlt. Der Sieg 4:1 Sieg von Googles AlphaGo gegen einen der führenden Go Spieler im März 2016 zählt dazu, ebenso wie große Verbesserungen beim automatischen Bildverstehen (“Computer Vision”) autonomer Fahrzeuge.

- Python 3.5 führt die neuen Schlüsselworte “async” und “await” für kooperatives Multitasking ein. Es gab sowohl einführende als auch forgeschrittene Vorträge zu diesem Thema. Gerade beim Implementieren effizienter Netzwerkserver ist dieses Konzept von Bedeutung.

- Microservices: Containersysteme wie Docker werden genutzt um sogenannte Microservices zu implementieren. Hierbei wird gemäß der Unix-Philosophie ein System in kleine und einfach gehaltene Komponenten zerlegt. Das Verpacken einer solchen Komponente in einen Container erlaubt es (ähnlich wie bei der Programmierung) die nach außen sichtbare Schnittstelle (z.B. in Form von Webservices) von der aktuellen Umsetzung innerhalb des Containers zu entkoppeln. Vorteile sind Skalierbarkeit und Robustheit, aber auch die Implementierung innerhalb eines Teams gestaltet sich so einfacher.

Auswahl interessanter Vorträge

(Keynote) Jameson Rollins: LIGO: The Dawn of Gravitational Wave Astronomy

Die Wissenschaftler am LIGO-Projekt haben in den USA zwei Detektoren für das Messen von Gravitationswellen entwickelt. Die Detektoren sind seit 2015 im Betrieb und die Wissenschaftler sind fähig, Signale in der Grössenordnung von 10^{-20} m zu messen. Zum Vergleich: ein Atom hat eine Grösse von 10^{-10} m, ein Proton ist 10^{-17} m gross.

Mit diesen Detektoren waren die Wissenschaftler in der Lage, erste Ereignisse zu messen. Im ersten Fall konnten sie am 14. September 2015 die Kollision zweier riesigen schwarzer Löcher vor 1,4 Milliarden Jahren beobachten. Vor der Kollision betrugen die Massen der schwarzen Löcher 36.2 bzw. 29.1 Sonnenmassen. Das resultierende schwarze Loch war allerdings nur 62.3 Sonnenmassen schwer. Das bedeutet, dass beim Zusammenprall die Energie von 3 Sonnenmassen in Form von Gravitationswellen abgestrahlt wurde.

Bei einem zweiten Ereignis am 26. Dezember 2015 konnte die Kollision eines grossen mit einem kleinen schwarzen Loch nachgewiesen werden. Bemerkenswert an diesem Ereignis war, dass das kleinere schwarze Loch, welches immerhin etwa die Masse der Sonne hatte, beim Umkreisen des grösseren schwarzen Lochs immer mehr beschleunigt wurde. Kurz vor der Kollision war das kleine Loch auf eine Geschwindigkeit von mehr als die halbe Lichtgeschwindigkeit beschleunigt.

Die beobachteten Signale liegen erstaunlicherweise im hörbaren Frequenzbereich und wurden während des Vortrages auch vorgespielt. http://www.space.com/31916-what-gravitational-waves-sound-like-video.html stellt diese auch zur Verfügung.

Elena Cuoco : Python in gravitational waves research communities

Elena Cuoco vom Virgo-Projekt gab einen Überblick über die Herausforderungen beim Bau und Betrieb des mehrere Kilometer großen Interferometers in Pisa sowie deren Lösungen. Python kommt hier zu großem Teil bei der Steuerung der einzelnen Komponenten sowie in der Auswertung der Messignale zum Einsatz. Die Präsentation ist unter https://ep2016.europython.eu/media/conference/slides/pyhton-in-gravitational-waves-research-communities.pdf verfügbar. Die vollständigen Messdaten sowie Jupyter-Notebooks zu deren Auswertung sind unter https://losc.ligo.org/about/ frei verfügbar.

Liana Bakradze: Learn Python The Fun Way

Es gibt diverse Projekte, welche das Ziel haben, dass Kinder und Jugendliche auf spielerische Weise Programmieren lernen können. Die Rednerin stellte drei Projekte vor, welche dieses Ziel anstreben und dabei recht erfolgreich sind:

1. CodeCombat: https://codecombat.com/
2. CodinGame: https://www.codingame.com/
3. CheckiO: https://checkio.org/

Bei allen dreien werden die Schüler dazu angeleitet online einfache Videospiele in Form von Skripten zu steuern um vorgegebene Aufgaben zu lösen. Leider sind alle drei Plattformen bisher in Englisch gehalten.

(Keynote) Nicholas Tollervey: MicroPython on the BBC micro:bit

Auf Initiative der BBC wurden im Vereinigten Königreich eine Million Mikrokontroller mit Namen micro:bit an 11 bis 12 jährige verschenkt. Der Microcontroller kann mit Python programmiert werden und ist darauf ausgelegt Schüler zum Experimentieren anzuregen. Features wie der Neigungssensor, das LED-Display und die Möglichkeit einen Lautsprecher anzuschließen, ein einfaches Radio-Protokoll zur Kommunikation mit anderen micro:bit sowie Pythonmodule wie der Sprachsynthesizer sollen zum spielerischen Programmieren einladen. Die Vofführungen während des Vortrags sorgten für so manchen Lacher, das zugehörige Video kann unter https://www.youtube.com/watch?v=JVaF6uZuSIU angeschaut werden.

Honza Král: Designing a Pythonic Interface

Welche Prinzipien machen ein gutes (Python) Interface aus?

Simplifying access, hiding complexity: Ein API soll den Zugriff auf gewisse Funktionalität vereinfachen und Komplexität verbergen. Das API ist ein Dienst für den Code, welcher die eigentliche Arbeit ausführt. Das API erfüllt einen Vertrag. Dieser Vertrag kann explizit sein, d.h. in Spezifikationen dokumentiert, oder implizit, wenn er gewissen Erwartungen des Kontexts (z.B. der Kultur) entspricht. Das API ist unbestimmter als der Code der Applikation.

Hide mechanics, not meaning: Das Interface muss Komplexität verbergen, darf aber nicht Bedeutung verstecken. Beispielsweise soll der Benützer eines APIs für Zugriffe über http sich nicht mit Sockets befassen müssen, aber Methoden für POST, GET, PUT und DELETE zur Verfügung haben. Die Methoden des APIs sollen so gestaltet sein, dass etwas möglich wird und nicht zum Ausdruck bringen, wie das gewünschte Resultat erreicht wird. D.h. das *what* ist Teil des APIs, das *how* soll verborgen werden.

Be consistent (if it makes sense): Das API soll gewohnte Muster wieder verwenden. Im Falle von Zweideutigkeiten soll der Benützer nicht raten müssen. Das API soll konsistent sein und keine Spezialfälle aufweisen, welche die Regeln brechen. Das gilt beispielsweise für die Benennung.

Be friendly: Das API soll freundlich gestaltet sein. D.h. es soll die Methoden und Funktionalität aufweisen, die es dem Benützer erleichtern, mit dem API zu arbeiten. Im Python-Kontext, wo viel interaktiv gearbeitet wird, sollen beispielsweise die Methoden *__dir__*, *__repr__* und *__doc__* implementiert sein.

Iterative build: Das API soll iterative Erzeugung (*iterative build*) ermöglichen. Dies wird erleichtert, wenn das Interface in Form einer sprechenden Schnittstelle (*fluent interface* bzw. *method chaining*) implementiert ist.

Fail by default: Wenn eine Funktion fehlschlägt, soll die Methode standardmässig versagen und das entsprechend zurückmelden.

Be flexible, things change: Das API soll flexibel sein und an Änderungen angepasst werden können. Beispielsweise soll für der Zugriff auf die zugrundeliegende Bibliothek für fortgeschrittene Benützer möglich sein.

Vincent Warmerdam: *The Joy of Simulation: for Fun and Profit*

Der Referent zeigt, wie mit Hilfe von Sampling-Methoden bestimmte Probleme ebenso effektiv gelöst werden können wie mit Wahrscheinlichkeitsrechnung. Sampling meint in diesem Fall die Entnahme einer Stichprobe mit dem Ziel, repräsentative Fakten über die Zusammensetzung einer Menge zu erhalten.

Mit Sampling kann beispielsweise der Erfolg im Monopoly-Spiel optimiert werden oder, wenn zusätzlich generative Methoden angewendet werden, können Pokemon- oder Ikea-Möbel-Namen erzeugt werden.

Slides: http://koaning.io/theme/notebooks/simulation_2016.pdf

Dougal Matthews: Effective Code Review

Warum sind Code-Reviews nützlich? Code-Reviews sind eine effiziente Möglichkeit, Fehler im Code zu entdecken. Gemäss dem Referenten beträgt die Rate, Fehler mit Unit-Tests zu entdecken, 25%, für funktionale Test 35% und für Integrationstests 45%. Im Vergleich dazu können mit Code-Reviews zu 55-60% Fehler entdeckt werden.

Was erwarten die beteiligten Entwickler von Code-Reviews? Fehler finden, Wissenstransfer, besseres Teamgefühl, Erkennen von alternativen Lösungsansätzen. Beim Code-Review geht es ebenso um Diskussion über Code und über Zusammenarbeit.

Grundlage und Voraussetzung für Code-Reviews sind Richtlinien im Projekt: Schreiben von Tests, schreiben von Dokumentation, testen von relevanten Plattformen, Einhalten von Styleguides.

Der Code-Review soll klein und in sich geschlossen sein. Je grösser die Patches sind, desto grösser ist die Wahrscheinlichkeit, dass der Reviewer einige Fehler übersieht. 10 – 500 LOCs sind akzeptabel.

Jedermann soll Code-Reviews machen, sowohl Neulinge (Juniors) wie auch erfahrene Entwickler (Seniors). Reviews sind Gelegenheit zum Lernen, Prüfen und Lehren. Es sollen nicht nur problematische Stellen bezeichnet, sondern auch überraschende Lösungen anerkannt werden. Der Review soll in höflichem Ton erfolgen und eine positive Wortwahl verwenden.

Slides: https://speakerdeck.com/d0ugal/effective-code-review

Adam Dangoor: Another pair of eyes: Reviewing code well

Adam Dangoor stellte die Interaktionen im Reviewprozess in den Mittelpunkt seiner Präsentation.

Wer soll einen Review durchführen? Eine Microsoft-Studie weist darauf hin, dass die wertvollsten Hinweise von Personen kommen, die früher schon Code des Projekts begutachtet haben. Auf der anderen Seite gilt, dass der Wissenstransfer am wirkungsvollsten ist, wenn der Code von einer Person begutachtet wird, welche den Code noch nicht kennt. Insofern kann es eine effektive und effiziente Einführung in ein Projekt sein, wenn ein Entwickler, der neu zu einem Projekt gestossen ist, gleich eine Anzahl Review-Aufgaben zugewiesen bekommt.

Beim Review ist der Kontext zu Berücksichtigen. Wird ein Performance-Problem an einer Code-Stelle beklagt, wo Performance kein Thema ist, so kann das zu Irritationen führen. Das befolgen von Styleguides sollte durch entsprechend eingerichtete Werkzeuge sichergestellt werden, so dass es nicht Sache des Entwickler und Gegenstand einer Reviews ist. In einem Review-Kommentar sollten nicht neue Themen aufgeworfen werden. Für solche sollten eigene Issues erzeugt werden.

Slides: https://ep2016.europython.eu/media/conference/slides/another-pair-of-eyes-reviewing-code-well.pdf

Marc-Andre Lemburg: So you think your Python startup is worth $10 million...

Der Referent berichtete von einem Auftrag, den er von einer grösseren Softwarefirma erhielt. In diesem Auftrag ging es darum, den Wert eines Startups, den der Auftraggeber zu kaufen beabsichtigte, einzuschätzen.

Eine Softwarefirma hat einen geschäftlichen Wert, bestehend aus Marktanteil (Kunden, Grösse im Markt), Kosteneffizienz, Innovationsstärke, Risiken (betreffend der Geschäftstätigkeit) etc. Zusätzlich hat

sie einen IT-Wert, welcher aus der Qualität der Entwickler, der Qualität der entwickelten Software-Produkte, der Code-Qualität und Risiken (betreffend den technischen Fähigkeiten) besteht. Beim Auftrag ging es darum, den IT-Wert des Übernahmekandidaten einzuschätzen. Zu diesem Zweck wurde der IT-Ansatz eingeschätzt sowie das Team und das System. Für diese Einschätzung wurden verschiedene Modelle (z.B. COCOMO) zur Hilfe gezogen. Am Schluss wurden zusätzliche Faktoren, insbesondere die IT-Risiken einbezogen und dem Ganzen gegenübergestellt, mit welchen Kosten die Software selber entwickelt werden könnte.

Die berücksichtigten Faktoren waren die Qualität der Entwickler, der Architektur, des Datenmodells, der eingesetzten Algorithmen sowie die Erweiterbarkeit. Zur Ermittlung der weichen Faktoren wurde dem Entwicklerteam diverse Fragen gestellt. Die Code-Qualität konnte mit entsprechenden Werkzeugen gemessen werden: Lesbarkeit, Wartbarkeit, Komplexität, Testabdeckung.

Sebastian Neubauer: Infrastructure as Code: "pip install" your environment

Mit CRUD (Create, Read, Update, Delete) wird üblicherweise ein REST-Interface beschrieben (POST, GET, PUT, DELETE). Wenn wir ein unveränderliches System wollen, müssen wir die Update-Möglichkeit (PUT) eliminieren. Unveränderlichkeit ist ein bestimmtes Konzept, wie der Status eines Systems verändert werden kann, nämlich dadurch, dass das System gelöscht und neu aufgebaut wird.

Infrastruktur ist alles, was toten Code lebendig macht. „Infrastruktur als Code“ bedeutet, dass die Infrastruktur auf automatisierte Art zugegriffen werden kann, d.h. über eine Schnittstelle, welche von Maschinen konsumiert werden kann.

Unveränderliche Infrastruktur verfügt über ein REST-API ohne Update-Funktion (PUT). Unveränderlichkeit ist wichtig, damit die Prozesse zum Erstellen der Infrastruktur automatisiert werden können. Automatisierung ist wichtig, weil damit menschliche Fehler reduziert werden können, das System reproduzierbar und damit billiger und schneller wird. Wenn Infrastruktur Code ist, dann kann sie unter eine Versionenverwaltung gestellt werden.

Beispiel für ein eine Infrastruktur als Code mit Docker (um die Instanz der Infrastruktur zu erzeugen, mit docker-py steht eine bequeme Bibliothek zur Verfügung), RESTful API (für die Schnittstelle) und Flask (als Implementierung der Schnittstelle).

Slides: https://ep2016.europython.eu/media/conference/slides/infrastructure-as-code-pip-install-your-environment.pdf

Stephan Jaensch: Building Service interfaces with OpenAPI / Swagger

Mit Swagger kann der Kontrakt eines Services spezifiziert werden. Mit Swagger kann dieser Kontrakt (d.h. das Interface) zusätzlich so dokumentiert werden, dass Kontrakt/Interface sowohl für Maschinen wie auch für Menschen lesbar sind.

Python hat Binding zu swagger. Für swagger 1.0 gab es swaggerpy, für swagger 2.0 ist bravado das Binding (entstanden aus swaggerpy). In der Präsentation werden die Probleme besprochen, welche sich durch eine Migration von swagger 1.x zu swagger 2.0 ergeben können.

Slides: https://ep2016.europython.eu/media/conference/slides/building-service-interfaces-using-OpenAPI.pdf

Christie Wilson, Michael Tom-Wing: System Testing with pytest and docker-py

Was ist ein Systemtest? Test, welcher die Software im ganzen System testet, z.B. den Code in der zusammen mit dem Service, welcher mit der DB interagiert.

Systemtests sind einerseits wertvoll, weil viele Fehler gefunden werden können. Andererseits ist es aufwändig, sie zu erzeugen und zu unterhalten. Auch dauert es lange, bis sie abgearbeitet sind.

Bewährte Vorgehensweise:

- Bei jedem einzelnen Test von einem frischen Zustand ausgehen (Docker hilft dabei). Auf diese Weise können Abhängigkeiten von einzelnen Tests vermieden werden.
- Tests sollen sowohl lokal wie auch auf Build-Servern laufen können.
- Der Test soll schnell fehlschlagen und eine informative Meldung im Fehlerfall anzeigen.

Docker-Py ist eine Python-Bibliothek für den Einsatz von Docker. Docker-Py bietet ein REST-Interface an. Vorgehen: Docker Client-Instanz erzeugen, mit dem Client ein Docker-Image herunterladen, damit ein Container erzeugen, den Container starten und am Schluss wieder entfernen.

Slides: https://ep2016.europython.eu/media/conference/slides/system-testing-with-pytest-and-docker-py.pdf

Mariano Anaya: Clean code in Python

Was ist sauberer Code (clean code)? Bei sauberem Code gibt jede Methode das Resultat zurück, das man von ihr erwartet. Schöner Code gibt einem das Gefühl, dass die Sprache für die Behandlung des Problems gemacht wurde. Sauberer Code erhöht die Code-Qualität und damit die Software-Qualität. Sie fördert die Lesbarkeit des Codes und macht agile Entwicklung möglich. Der Gegensatz von sauberem Code sind beispielsweise Code-Duplikate und Code, bei welchem die Absicht nicht erkennbar ist. Voraussetzung für sauberen Code ist DRY (don‘t repeat yourself).

Python bietet viele Hilfsmittel für sauberen Code:

- Mit *Dekoratoren* (Annotationen) können Code-Duplikationen verhindert und kann Programmlogik separiert werden.

- Mit den *magischen Methoden* (z.B. __contains__()) und Context-Managern kann pythonischer Code erzeugt werden. Details der Implementierung und die interne Komplexität können versteckt werden.

- *Properties* können helfen, die Lesbarkeit des Codes zu verbessern.

Slides: https://ep2016.europython.eu/media/conference/slides/clean-code-in-python.pdf

Jose Manuel Ortega: Ethical hacking with Python tools

Mit Python sind diverse Bibliotheken entwickelt worden, welche im Sicherheitsbereich relevant sind. Der Referent stellt einige der verfügbaren Bibliotheken vor.

Penetration-Testing:

- Sparta (http://sparta.secforce.com/)

- theHarvester (https://github.com/laramies/theHarvester)

- w3af (Web Application Attack and Audit Framework, http://w3af.org/)

Werkzeuge:

- Scapy: Aufnahme und Analyse von Netzwerk-Paketen

- FiMap: RFI/LFI-Gefährdung entdecken

- XSScrapy: XSS-Gefährdung entdecken

Werkzeuge im Python-API:

- Socket

- Requests

3rd-Party Klassenbibliotheken:

- BeautifulSoup

- Shodan

- BuiltWith

Analyse von Metadaten:

- Metadaten von pdf-Dokumenten: PyPDF2

- Metadaten von Bild-Dateien: PIL

Port Scanning:

- Python nmap
- pywebfuzz: Schwachstellen von Webauftritten analysieren

Weitere Werkzeuge:

- Metasploit: python-msfrpc
- Nexpose

Der Referent hat auf GitHub verschiedene Skripte erstellt, welche den Einsatz der Werkzeuge zeigen:

https://github.com/jmortega/python-pentesting

https://github.com/jmortega/europython_ethical_hacking

Slides: http://www.slideshare.net/jmoc25/ethical-hacking-with-python-tools

Mike Bright: Jupyter for everything else

Jupyter ist die Weiterentwicklung (bzw. Ablösung) von IPython notebooks. IPython notebooks wurden als Werkzeug entwickelt, um den Erkundungsprozess beim Programmieren zu unterstützen. Während IPython stark auf Python ausgerichtet war, ist Jupyter offen für viele Sprachen. Aktuell stehen für Kernels (*execution environments*) für mehr als 50 Sprachen zur Verfügung..

Slides: https://ep2016.europython.eu/media/conference/slides/jupyter-for-everything-else.pdf

Schlussbemerkungen / Anregungen:

- Die *DevOps*-Perspektive sollte für die ID weiter geprüft werden. Zu klären ist beispielsweise, ob mit Python und den Python-APIs (z.B. docker-py) Gluecode einfacher implementiert werden kann als mit anderen Technologien.
- Code-Reviews sollten als Möglichkeit, Erfahrungen und Wissen zu vermitteln, intensiver eingesetzt werden.

Luthiger Benno (ID SWS), Uwe Schmitt (ID SIS)