Oken Gymnasium Offenburg
Physik-Kurs
Leitung: Herr Mechling
Kurstufe 2

Pisole

Ein modulares Kivy Konsolen Widget

Version 0 - 20.August 2016

Jonas Teufel
Wilhelm-Störk-Straße 6
77652 Bohlsbach
Email: jonseb1998@gmail.com

Inhaltsverzeichnis

1 Zielsetzung

Basierend auf den Erfahrungen mit dem *JTShell Projekt*,[1] welches eine ähnliche Konsolen Anwendung bereitstellt, in ihrer Natur jedoch der einer Linux Shell ähnelt, d.h. dass die Anwendung relativ statisch an das Betriebssystem gebunden ist und daher nur sehr schwer mit anderen Projekten bzw. Anwendungen verbunden werden kann, stattdessen ist die JTShell nur vorgesehen, um Anwendungen und einzelne utility Funktionalitäten zu starten und bestenfalls zu überwachen.
Die Konsole als solches Konzept sollte jedoch als mögliches Benutzerinterface für jedes mögliche weitere Projekt genutzt, oder zumindest als Erweiterung dessen eingesetzt werden können.
Das Pisole Projekt soll sich deshalb mit der Entwicklung eines Python Packages beschäftigen, welches es ermöglicht, eine Konsolen-Schnittstelle weitgehend unabhängig vom eigentlich Subjekt der nutzenden Projekte bereitzustellen. Hierbei sind folgende Ziele von größter Bedeutung:

Übersichtlichkeit Da das Package mehr zu Entwicklungs- und weniger zu Anwendungszwecken eingesetzte werden soll, muss der interne Aufbau, also der Code übersichtlich und simpel gestaltet werden, so dass dieser einfach verstanden und angepasst werden kann.

Entwicklerfreundlichkeit Hiermit sei vor allem gemeint, dass das Package einfach in ein Projekt zu integrieren sein soll. Die Nutzung sollte Beispielsweise keine allzu aufwändigen Installationsaufwand voraussetzen, keine Abhängigkeiten oder Restriktionen aufweisen. Bestenfalls wird das Package in den Projekt Ordner kopiert, von einem Modul importiert und funktioniert sofort.

Modularität Ähnlich dem vorgehenden Ziel soll es demnach möglich sein, die Funktionen des Pisole Projektes in einer vielfältigen Auswahl an Anwendungen nutzen zu können. Das Package soll also einen Standardisierten UI-Software Baustein bereitstellen.

[1] Jonas Teufel. „JTShell-Eine erweiterbare Python Kommandozeilen Umgebung“. In: (2016).

2 Projektstruktur

2.1 Die Konsolen Klasse

Die letztendlich für die Nutzung intendierten Objekte werden durch die SimplePisoleConsole Klasse erstellt, welche sich im *pisole.py* Modul des Packages befinden. Jene Klasse ist eine Subklasse der Python 'threading.Thread' Klasse.
Die Klasse selbst ist ein Thread, da die Konsole ihre Befehle und Funktionen unabhängig vom Programmfluss der eigentlichen grafischen kivy Benutzerschnittstelle ausführen können soll. Wäre es nun zum Beispiel nicht so, würde ein eingegebener Befehl, welchem eine recht umfangreiche Funktion zu Grunde liegt innerhalb des zentralen kivy loops ausgeführt werden und alle restlichen UI Elemente für einige Sekunden blockieren. Deshalb läuft parallel zum eigentlichen UI die ganze Zeit der Pisole-Thread, welcher sich ausschließlich um die Befehlausführung kümmert.

2.1.1 Befehl Ausführung

Innerhalb der Hauptschleife des Pisole Threads werden die vom Nutzer eingegebenen Befehle ausgeführt. Die Schleife wirkt dabei zunächst als eine Art Überwacher, welche die meiste Zeit wartet bis ein neuer String im Buffer des Widgets zur Ausführung bereit steht. Dieser String wird dann aus jenem Buffer entfernt und dem Widget wird über die Methode "new_command()" signalisiert, dass ein neuer Befehl ausgeführt wird und ein dementsprechend neues Ausgabe Label angelegt werden soll.
Bevor der String jedoch verwendet werden kann muss er durch die "translate" Funktion des *translate.py* Moduls so übersetzt werden, dass die Funktionen wie intendiert vom Python Interpreter ausgeführt werden.

```
translated_input = translate.translate(input_string, "self")
try:
    compiled_input = compile(translated_input, "<string>",
                             "exec")
    exec(compiled_input)
except Exception as exception:
    self.print_error(exception)
```

Innerhalb eines try-except Blocks wird dann der übersetzte String mit der Python built-in

Funktion 'compile'[2] compiliert und dann dynamisch über die built-in 'exec'[3] ausgeführt. Sollte eine Exception während dieses Prozesses auftauchen so wird diese, unabhängig von der Art des Fehlers, aufgefangen und dem Nutzer über die Ausgabeanzeige des Widgets mithilfe der 'print_error' Methode angezeigt.

2.1.2 Output Methoden

Da die den Befehlen zu Grunde liegenden Funktionen nicht nur ausgeführt werden sollen, sondern letztendlich auch da sind, um dem Nutzer Rückmeldung über deren Status und Ausgang zu erstatten, benötigt die Konsolen Klasse natürlich auch Methoden, über welche Strings zum ausgeben an das Widget weiter geleitet werden.
Hierfür wird die 'print_info' Methode für Zwischenmeldungen zum Programmstatus, die 'print_error' Methode für auftretende Exceptions und die 'print_result' Methode für Meldungen zum Ausgang des Ausführung genutzt.
Innerhalb dieser Methoden werden die übergebenen Strings wiederum als Parameter zur Instanzierung der 'Message' Objekte genutzt. Dies ist wichtig, denn diese Message Objekte besitzen die Methode 'get_kivy', wodurch die simplen String-Nachrichten, um Elemente der kivy Markup Language erweitert werden, wodurch sie ihr jeweiliges spezifisches formatiertes Aussehen auf dem Ausgabe Label des UI erhalten. Bsp.

```
def print_info(self, string):
    info_message = message.InfoMessage(string)
    self.console_widget.println(info_message.get_kivy())
```

[2]Python Software Foundation. *Built-in Functions*. 2016. URL: https://docs.python.org/3/library/functions.html (besucht am 21. 08. 2016).
[3]Ebd.

2.2 Der Übersetzungsvorgang

Die für eine Ausführung nötige Übersetzung geschieht mittels der im *translate.py* befindlichen Funktion 'translate'. Jene Funktion basiert auf mehreren Algorithmen, welche im Endeffekt auf oberster Ebene in folgende Funktionen vereinfacht werden können:
Aus dem vom Nutzer eingegebenen String wird eine Liste aller auftretenden Befehle erstellt. Aus dieser Liste werden dann zunächst die Python-nativen built-in Funktionen herausgefiltert. Jedem Befehl wird über eine Reihe von String-Operationen ein Parameter, dessen String durch einen Parameter der 'translate' Funktion wählbar ist, hinzugefügt, wichtig ist hierbei, dass dieser die erste Position einnimmt, so dass alle im String auftretenden Befehle an der selben Position einen zusätzlichen Parameter erhalten.
Im Fall des Pisole Projektes ist heißt dieser Parameter 'self', denn die Funktionen innerhalb des Strings werden letztendlich innerhalb der Hauptschleife eines SimplePisoleConsolen Objektes ausgeführt, in wessen Namensraum 'self' für die Übergabe einer Referenz auf das Objekt selbst verantwortlich ist. Die Befehle erhalten also eine Referenz auf das Konsolen Objekt, welches sie ausführt. Dies ist in sofern essentiell, dass die Funktionen somit die verschiedenen 'print' Befehle der Konsole ausführen und damit mit dem UI selbst interagieren können.

2.3 Das Widget

Das Widget 'SimpleConsoleWidget' des *consolewidget.py* Moduls basiert grundlegend auf einem kivy GridLayout, um die verschiedenen Teile zu verbinden. Optisch ist die Konsole natürlich an den gängigen Konsolen, wie der Windows Cmd und der Linux Shell inspiriert, weicht jedoch auch relativ stark von diesen ab, da der Nutzer die Eingabe nicht direkt in der Konsole selbst, also im gleichen Feld, wie die Eingaben tätigen kann, sondern diese statisch in einem kleinen separaten Feld am unteren Ende des Widgets getätigt werden.

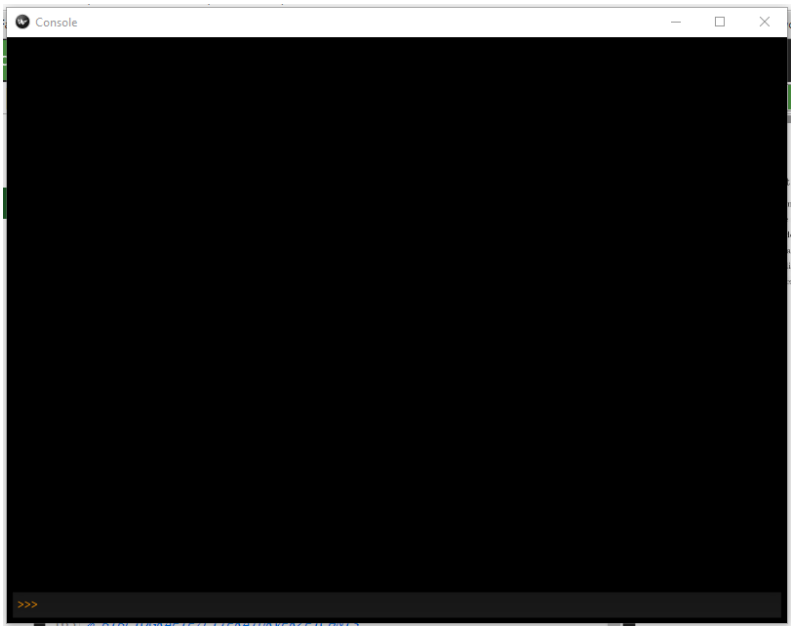

Abbildung 1: Konsolen Widget

Diese Eigenschaft wurde gewählt, da die Pisole Konsole keine eigene Scriptsprache implementiert, sondern den Syntax der Programmiersprache Python direkt mit eigenen Befehlen übernimmt. Da Python's Syntax auf Einrückungen basiert, soll eine im Fenster verankerte statische Codeeingabe die Nutzung von Mehrzeiligem Code simpler und übersichtlicher gestalten.

2.3.1 Optische Parameter

Die Erstellung eines Konsolen Widgets bietet neben den möglichen Anpassungen eines kivy GridLayouts drei spezifische Parameter zur Anpassung der Optik der Konsole.

Eingabe Prompt Der Eingabe Prompt, d.h. jener String, hinter welchem der Nutzer die Eingabe tätigen kann, wird durch den Parameter 'prompt' bestimmt. Es ist jedoch

anzumerken, dass die Änderung lediglich die erste Zeile des Eingabe Fensters betrifft, werden weitere Zeilen für eine mehrzeilige Eingabe hinzugefügt, werden diese jeweils mit dem prompt '¿¿' beginnen.
Im Normalfall hat 'prompt' den Wert '¿¿¿'

Der Hintergrund Der Hintergrund des Konsolen Widgets kann momentan zwar nicht in der Farbe modifiziert werden, d.h. die Konsole wird immer schwarz sein, jedoch ist es möglich die Graustufe dieses Schwarztones mit Hilfe des 'background' Parameters anzupassen.
Dieser Parameter ist ein Float Wert und bewegt sich zwischen 1 und 0.1.1, wobei 1 komplett schwarz und 0.1 komplett weiß darstellt. Welcher Wert auch gewählt wird, es bleibt der Farbunterschied zwischen dem eigentlichen Konsolenfenster und der Eingabeleiste erhalten, diese wird immer leicht heller sein als der Hintergrund, damit die Abgrenzung der beiden Bestandteile deutlich wird.
Im Normalfall hat 'background' den Wert 1, d.h. das Fenster ist schwarz.

Syntax Highlighting Da die Konsole mehr auf einer Programmiersprache basiert und die Code Eingabe optisch vom Rest getrennt ist, bietet sich die Möglichkeit der Syntax Hervorhebung.
Dies wird durch das Modul *pygments*[4] ermöglicht, genutzt wird von dem Widget der Eingabeleiste der Lexer für den Python 3 Syntax und das Konsolen Widget bietet 3 verschiedene Hervorhebungs-Styles, zwischen welchen gewählt werden kann.
Hierzu kann der 'style' Parameter auf die Strings 'orange', 'green' oder 'monokai' eingestellt werden.
Im Normalfall hat 'style' den Wert 'orange', die Syntax Hervorhebung basiert also auf verschiedenen Orange-Tönen.

2.3.2 Die Eingabeleiste

Die Eingabeleiste des Konsolen Widgets basiert auf dem kivy-nativen Widget 'CodeInput', welches speziell für die Eingabe von Code entwickelt wurde und beispielsweise die Nutzung eines pygments-Lexers zur Syntax Hervorhebung erst ermöglicht.
Nachdem eine Eingabe in die Liste getätigt wurde kann diese durch die Betätigung der Enter Taste bestätigt werden, wodurch sie aus der Leiste verschwindet und weiter verar-

[4]Georg Brandl. *Pygments*. 2014. URL: `http://pygments.org/` (besucht am 21. 08. 2016).

beitet wird, was bedeutet, dass diese durch ein Event und die Callback Funktion zunächst in einen Buffer innerhalb des eigentlichen Konsolen Objektes abgelegt wird.
Die Eingabeleiste kann durch Betätigung der Tabulator Taste um jeweils eine weitere Zeile für mehrzeilige Python Code Segmente erweitert werden.

Abbildung 2: Mehrzeilige Befehlseingabe

Die maximale Anzahl der Zeilen beschränkt sich hierbei auf 12.
Zusätzlich bietet die Eingabeleiste die Möglichkeit schon getätigte Befehle wieder in die Leiste zurück zu holen, ohne diese ausschreiben zu müssen. Innerhalb des Widgets wird eine Liste aller bereits ausgeführten Befehle ohne Doppelung gespeichert, um durch diese Liste chronologisch durch zu scrollen werden die Pfeiltasten nach oben und unten benutzt.

2.3.3 Das Ausgabe Fenster

Das Ausgabe Fenster des Konsolen Widgets basiert auf dem kivy-nativen Widget 'ScrollView', damit die Ausgabe history auch über den Fensterrand hinaus noch erhalten und nachlesbar bleibt.
Die Ausgaben sind so organisiert, dass für jeden neuen auszuführenden Befehl nicht nur einen neue Zeile, sondern ein ganz neues Label zur ScrollView hinzugefügt wird, welches alle Ausgaben des jeweiligen Befehls enthält. Das Widget wird neben der ScrollView auch noch einem separaten Listen Objekt in chronologischer Reihenfolge hinzugefügt, damit das SimpleConsoleOutput Objekt selbst noch Zugriff auf eine Referenz des Labels hat, 'print' Befehle jeglicher Art werden auf dem Level dieses Objektes nämlich in strings heruntergebrochen und einfach zur text Variable des zuletzt zugefügten Labels angehängt.

2.3.4 Die Ausgabe Funktionalität

Eine wichtige Eigenschaft des Konsolen Widgets insgesamt ist, dass der der vom Nutzer in die Eingabeleiste eingetragene String vom Objekt zur Verarbeitung durch andere Prozesse bereit gestellt wird und gleichermaßen einfach von externen Prozessen Strings ausgegeben werden können. Dies bedeutet, dass nicht einfach ein Event und eine callback Funktion

genutzt werden können, um Strings von einem zum anderen Widget zu transferieren. Stattdessen wird ein Benutzer Input, sollte er durch eine Enter Taste bestätigt werden durch ein callback zunächst in einen Buffer, genauer eine Listen Variable, innerhalb des Konsolen Objektes namens 'entered_strings_list' gespeichert, bis diese abgerufen werden. Analog funktioniert es auch mit den auszugebenden Strings, wird ein 'print' Befehl auf der Eben des Konsolen Widgets aufgerufen, so wird der String zunächst in einen Buffer namens 'print_buffer' gelegt. Nach einem Intervall einer Länge mehrerer Millisekunden prüft ein kivy Clock Event dann dauerhaft, ob ein String in diesem Buffer enthalten ist und gibt diesen an das Output Widget weiter, sollte dies der Fall sein.

3 Entwicklungseinbindung

3.1 Im Dateiverzeichnis

Ein großer Vorteil des Projektes ist dessen Modularität, es hat nahezu keine Abhängigkeiten von externen Bibliotheken, außer natürlich *kivy* selbst, es ist schließlich auch für die Nutzung in kivy Projekten intendiert.
Der *pisole* Ordner muss lediglich in den Projektordner kopiert werden. Dabei ist es wichtig, dass dieser sich damit im gleichen Ordner befindet, wie all die Module, welche dessen Funktionalitäten Nutzen, d.h. das main Modul des Projektes, welches letztendlich das letzte Glied der Python Import-Kette ist muss sich im gleichen Verzeichnis befinden wie der pisole Ordner.
Dieser etwas einschränkende Umstand ist dem Konzept des Python Import Systems geschuldet und kann kaum umgangen werden.
Alternativ kann pisole auch genutzt werden, indem der Ordner in einen Dateipfad eines Pythonpath gesetzt wird.

3.2 Im Code

Im Code, vorausgesetzt die Limitationen der Dateiverzeichnis Einbindung wurden eingehalten, muss lediglich das *pisole.py* Modul des pisole Ordners genutzt bzw. importiert werden. Dies geschieht bestenfalls wie folgt:

```
import pisole.pisole as pisole
# oder
from pisole.pisole import SimplePisoleConsole
```

Die Konsole kann dann einfach genutzt werden, indem das Widget zu einem beliebigen Layout hinzugefügt und der Thread gestartet wird:

```
console = pisole.SimplePisoleConsole()
layout.add_widget(console.get_widget())
console.start()
```

3.3 Neue Befehle hinzufügen

Neue Befehle können der Konsole hinzugefügt werden, indem diese einfach als neue Funktionen innerhalb des *commands.py* Moduls eingefügt werden, welches sich im Dateisystem auf gleicher Ebene wie das main-Modul und der pisole Ordner befinden sollte.

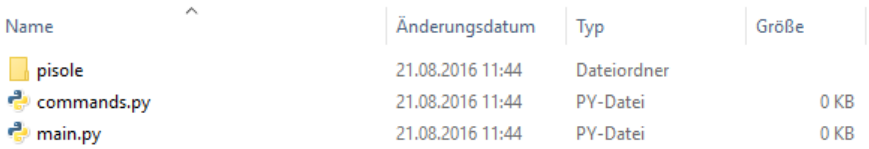

Abbildung 3: Dateistruktur

Dies ist weniger eine Beschränkung, sondern viel mehr ein Vorteil, da Funktionen, also Befehle, innerhalb dieses Moduls auf alle möglichen anderen Strukturen des Projektes zugreifen können, wodurch sinnvollere Befehle erstellt werden können.
Die eigentlichen Funktionen sind in ihrer Form kaum beschränkt, sie können alle möglichen Arten von Parametern haben, jedoch muss der erste Parameter immer *'console'* sein. Dies bedeutet nicht, dass der Parameter immer genau diesen Namen haben muss, sondern dass er immer diese Funktion übernimmt. Der erste Parameter ist für jede Funktion reserviert, um eine Referenz auf das Konsolen Objekt selbst übergeben zu bekommen. Mit den Methoden dieses Objektes werden letztendlich Ausgaben an das eigentliche Widget weiter geleitet. Die Methoden werden genutzt, wie folgt:

```
def func(console, parameter):
    # Fur Statusinformationen zur Ausfuhrung
    console.print_info("Info")
    # Fur Fehler
    console.print_error(Exception)
    # Fur Ruckgabewerte / Abschlussinformationen
    console.print_result("Fertig")
    return True
```

Literatur

Brandl, Georg. *Pygments*. 2014. URL: `http://pygments.org/` (besucht am 21. 08. 2016).

Foundation, Python Software. *Built-in Functions*. 2016. URL: `https://docs.python.org/3/library/functions.html` (besucht am 21. 08. 2016).

Teufel, Jonas. „JTShell-Eine erweiterbare Python Kommandozeilen Umgebung“. In: (2016).

Abbildungsverzeichnis

Tabellenverzeichnis