Hinweise zur Installation einer Python-Umgebung

Grundsätzlich ist es zu sagen, dass Python auch in verschiedenen Entwicklungsumgebungen IDE's[1] integriert ist bzw. integriert werden kann, wie es z. B. bei Eclipse der Fall ist[2]. Python besitzt von Haus aus eine eigene Mini-Entwicklungsumgebung namens IDLE. Dies wird gleich mit dem Python-Interpreter mitinstalliert und ist unter Windows über das Startmenü und in Linux (Unix) über das Shell-Kommando *idle* aufrufbar (eine korrekte Installation vorausgesetzt). Insbesondere kann diese so auf den Linux-Rechnern in den Fischer-Räumen aufgerufen werden. (Zu IDLE noch ein kleiner Hinweis: IDLE startet gewöhnlich als Konsole in den interaktiven Modus, was nett zum Experimentieren aber nicht wirklich hilfreich zum Programmieren ist. Es lässt sich aber über das *File* Menü mit *New Window* ein neues Fenster öffnen in den man einen Python Programm eingeben und auch gleich ausführen kann. Falls man IDLE lieber gleich in diesen Modus starten möchte, so kann dies über das Verändern der Einstellungen erreicht werden. Eigentlich ist das alles, was man über IDLE wissen muss.

Für alle die, die lieber doch noch eine kleine Einführung in IDLE (One Day of IDLE Toying) lesen möchten.; unter
http://wiki.python.org/moin/BeginnersGuide/NonProgrammers?highlight=%28BeginnersGuide%2F%29
finden Sie diese.

Wir wollen hier nicht weiter auf IDE's eingehen. Erwähnt sei trotzdem noch die IDE *ActivePython* mit integriertem Python-Interpreter, welche für Windows, Linux, Mac OS und weitere Betriebssysteme verfügbar ist. Sie kann unter folgender Adresse heruntergeladen werden:
http://www.activestate.com/Products/Download/Download.plex?id=ActivePython.

ActivePython ist, wie ich meine, etwas bequemer als IDLE. Sie ist recht einfach aufgebaut und macht das Arbeiten mit Python sehr angenehm, auch wenn sie bei weitem nicht so mächtig – was nebenbei kein Nachteil sein muss – wie Eclipse und vergleichbare IDE's ist.

Im Weiteren gehen wir davon aus, dass Sie Windows verwenden; schlicht und einfach deshalb, da es wohl das meist verbreitete Betriebssystem ist (ob zu recht sei dahingestellt). Nichts desto trotz werden die meisten hier aufgeführten Hinweise auch für Linux-, Mac- und andere Benutzer nützlich sein.

Einrichten einer Python-Umgebung

Haben Sie sich eine IDE besorgt oder wollen eine benutzen, so können Sie den Abschnitt über den Python-Interpreter und den Texteditor überspringen. Ansonsten benötigen Sie, um auf Ihrem Computer in Python programmieren zu können, im Grunde nur einen Python-Interpreter und einen Texteditor.

Python-Interpreter

Den Python-Interpreter bekommen Sie, wie nicht anders zu erwarten, unter http://www.python.org/. Dort findet man z. B. unter **Documentation links** den Link Beginner's Guide to Python - **start here if you're new to programming** zu Beginner's Guide, wo man wiederum unter **BeginnersGuide**/Download die Python Interpreter bekommt.
Bitte nehmen Sie immer die letzte, als stabil eingestufte Version! Für Windows war es am 04.10.2005 Python 2.4 for Windows**.** Linux- und Mac-Benutzer finden hier auch den geeigneten Interpreter. (Die geeignete Datei für das jeweilige Betriebsystem zu finden dürfte kein Problem darstellen.)

---

[1] IDE = Integrated Development Environment bzw. Integrated Design Environment

[2] Für mehr Infos zu Python für Eclipse findet man z.B. unter http://www.eclipse.org/, wenn man nach Python sucht.

Texteditor

Als Texteditor ist unter Windows z. B. der Freeware Editor PsPad (http://www.pspad.com/de/) zu empfehlen, welches das Python-Syntaxhighlighting unterstützt und auch sonst sehr leistungsstark ist. (Ein Nachteil von PsPad gegenüber einer Python-IDE ist, dass es nicht das automatische Einrücken unterstützt, welches für Python von Bedeutung ist.) Da PsPad aber auch bestens für andere Aufgaben geeignet ist, ist es alle mal einen Blick wert.

Ein paar Einstellungen

Ist der Python-Interpreter und ein Texteditor installiert, müssen nur noch ein paar Kleinigkeiten gemacht werden.

- Der Pfad zu dem Python-Interpreter sollte in die **Path-Umgebungsvariable** aufgenommen werden. (Wenn Sie nicht wissen wie, hilft bestimmt eine kurze Internet Recherche unter Angabe des Betriebssystems und des Schlagwort *Umgebungsvariable*.)

- Erstellen Sie dann ein **Verzeichnis für Ihre eigenen Python Programme**.

- Sie werden, im Rahmen der Übung zu der Vorlesung, von uns auch fertige Programme bekommen, die Sie ändern bzw. nutzen sollen. Die Programme, die Sie als Bibliothek (Library) nutzen sollen, werden in einem Unterordner *PRG1_lib* abgelegt werden. Diesen Ordner samt Unterordner können Sie in das gleiche Verzeichnis, von dem aus Sie Ihre Programme starten, ablegen. Da Sie aber wahrscheinlich mehrere Projekt-Ordner für ihre Programme anlegen wollen (was übersichtlichkeitshalber auch ratsam wäre), sollten Sie einen (Haupt-) Ordner für ihre Projekte anlegen (z. B. python>Projekte>PRG1) und dort Ihre Projektordner, sowie Bibliothek (-ordner) ablegen. Eine Verzeichnisstruktur könnte z. B. so aussehen:
  python>Projekte>PRG1>uebung1>
  python>Projekte>PRG1>uebung2>
  :
  python>Projekte>lib>*PRG1_lib*>
  In diesem Fall müssen Sie den Python-Interpreter mitteilen, wo er die Bibliotheken findet (in diesem Beispiel wäre das der Pfad „python/Projekte/lib/“. Dies tun Sie, indem Sie den betreffenden Pfad in die **Umgebungsvariable *PYTHONPATH*** ablegen.

Ein Paar Hinweise zum Schluss

Zuletzt ein paar Hinweise zu Python-Programmen. (Als Programmieranfänger werden Sie möglicherweise nicht alle hier aufgeführten Hinweise verstehen. In den meisten Fällen sollte Ihnen aber das Internet bzw. Ihr Tutor helfen können.)

Als Beispiel soll das folgende einzeilige Programm *Test1.py* dienen, mit folgenden Inhalt:

```
|print "Hallo Welt!"
```

Dabei bezeichnet „|“ den linken Rand der Datei.

Die Hinweise:

1) Die Python Programme sollten in einer Textdatei mit der Endung „py“ stehen, wie es auch bei unserem Beispielprogramm der Fall ist.

2) Ausgeführt wird ein Python-Programm, mit dem Kommando „python *programmname.py*“[3]. Also wird unser Beispielprogramm wie folgt ausgeführt:
```
>python Test1.py
```

3) Achten Sie darauf, dass Ihr Programm in der ersten Spalte anfängt. Ein Leerzeichen (hier dargestellt als „_“) vor dem *print*-Befehl führt schon zu einem Syntax Error, mit einer leider wenig hilfreichen Fehlermeldung (siehe unten).

```
|print "Hallo Welt!"
```

Ausgabe:
```
Hallo Welt!
```

```
|_print "Hallo Welt!"
```

Ausgabe:
```
 :
 print "Hallo Welt!"
 ^
 SyntaxError: invalid syntax
```

4) Beachten Sie, dass Python sehr lässig mit Variablentypisierung umgeht. Dies kann sehr verwirrend sein. So sind z.B. Kommandozeilen-Argumente und auch Variablen, die von stdin (standard input stream) gelesen werden Strings (Zeichenketten). So wird z.B. das folgende Programm:

```
data = 5
if data > 10:
  print "Der Wert von data ist groesser 10."
else:
  print "Der Wert von data ist kleiner gleich 10."
```

korrekt funktionieren, also ausgeben:

```
 Der Wert von data ist kleiner gleich 10.
```

Wogegen

```
import sys
print "Geben Sie bitte eine Zahl ein: ",
data = sys.stdin.readline()
 if data > 10:
  print "Der Wert von data ist groesser 10."
else:
  print "Der Wert von data ist kleiner gleich 10."
```

Python veranlasst einen String mit einem Integer zu vergleichen (*data hat nämlich im Grunde den Wert: **5\n,** welchen man z.B. mit dem Befehl* `print repr(data)` *ausgeben kann*). Was für die Eingabe von 5 zur folgenden Ausgabe führt:

```
Geben Sie bitte eine Zahl ein: 5
Der Wert von data ist groesser 10.
```

[3] Manchmal ist es nützlich ein Programm im interaktiven Modus auszuführen (z.B. um dieses zu testen). Im interaktiven Modus wird das Programm nach der Abarbeitung nicht wieder verlassen. So kann man z.B. ein Python-Programm, welches „Vorarbeiten“, wie Importieren von Modulen, Definitionen von Variablen/Funktionen usw. erledigt, im interaktiven Modus ausführen und an der Stelle wo das „Vorprogramm“ die Vorarbeit erledigt hat, einsteigen und weitermachen. Man führt ein Python-Programm im interaktiven Modus aus, in den man die Option **–i** dem Programmnamen voranstellt. Z.B. führt `python –i Test1.py` unser Beispielprogramm im interaktiven Modus aus.

5) Geben Sie Kontrollmeldungen (z.B. bei Fehlersuche) nicht mit dem Befehl `print` aus, sondern mit `sys.stderr.write (<Meldung als String>)`. Der Befehl `print` schreibt die Meldung intern auf den *stdout* (standard output stream), wogegen `sys.stderr.write` die Meldung auf *stderr* (standard error stream) schreibt. Auch wenn beide Streams standardmäßig auf den Bildschirm umgelenkt (ausgegeben) werden, wodurch der Eindruck entsteht, dass es egal ist wie man die Meldungen ausgibt, so gibt es mindestens zwei gute Gründe es richtig zu machen. Der erste wird uns zwar in der Vorlesung kaum begegnen, nichts desto trotz ist es gut zu wissen, dass man Streams (z.B. in eine Datei) umleiten kann und so z.B. die regulären Ausgaben von den Fehler- und Kontrollmeldungen „trennen“ kann.

   Der zweite ist, dass die Reihenfolge der Ausgabe der Streams untereinander, im Gegenteil zu der Reihenfolge in einem Stream selbst, nicht unbedingt chronologisch[4] ist. Das heißt z.B., dass die Stelle in der ein Fehler tatsächlich auftritt, nicht unbedingt aus der Stelle der letzten Meldung, die mit `print` ausgegeben wurde, ableitbar ist. (Es kann vorkommen, dass ein Programm mal nach, mal vor einer regulären Ausgabe mit einem Fehler abbricht und so der Eindruck entsteht, dass der Fehler in dem Programm zu verschiedenen Zeitpunkten passiert. So etwas kann bei der Fehlersuche verwirren).

[4] zeitlich geordnet