EINFÜHRUNG IN PYTHON

PATRICK ABRAHAM

Zusammenfassung. Diese kurze Einführung in Python soll einen kleinen Einblick in die Welt der Algorithmen bieten und euch etwas vorbereiten auf die Programmieraufgaben welche im Studium auf euch warten.

Inhaltsverzeichnis

Teil 1. Tutorium

1. Algorithmen im Allgemeinen

Wenn man in der Wissenschaft von Algorithmen spricht, meint man generell Abläufe welche strikte Regeln befolgen. Dabei kann man damit Formeln, Ideen oder Programme beschreiben. Alles folgt einer spezifischen Reihenfolge und jeder Schritt muss brav dem anderen Folgen.

In der CoMa geht es meist darum Algorithmen zu finden, welche ein Problem bzw. eine spezifische Art von Eingabe immer lösen können. Dafür werden euch verschiedenste Modelle beigebracht und spezifische Arten an Probleme heran zu gehen.

Generell geht es darum ein wiederkehrendes Problem immer mit der selben Idee lösen zu können ohne lange über das Problem nachdenken zu müssen. Hier muss man aber generell mal darauf zu sprechen kommen, was ist eigentlich ein Problem und wie geht man an so ein Problem heran.

2. Was ist ein Problem und wie löse ich es?

Ein Problem besteht immer aus einem Anfangszustand und einem Ziel. Dieser Anfangszustand kann verschieden aussehen, sei es ein Wort, ein paar Zahlen oder vielleicht sogar einfach nur eine Form auf einem Stück Papier. Der Algorithmus soll nun diesen Anfangszustand oder auch Eingabe bzw. Input aufnehmen und diesen so manipulieren, dass das gewünschte Ziel erreicht wird.

Ein einfaches Beispiel wäre die Konstruktion eines rechten Winkels mit Hilfe von Zirkel und Lineal. Das Anfangsproblem ist hierbei ganz einfach eine Gerade G_1 durch welche man gerne an einem Punkt P eine andere Gerade G_2 in einem rechten Winkel zeichnen will. Eine Gerade ist hierbei als unbeschränkte Linie zu sehen.

Die Konstruktion ist relativ simpel:

Algorithm 1: Rechter Winkel

Input: Eine Gerade G_1 und ein Punkt P auf dieser Gerade
Output: Eine Gerade G_2 die G_1 im rechten Winkel in P schneidet.

1 Zeichne einen zweiten Punkt Q_1 auf G_1;
2 Setze die Nadel auf P und die Mine auf Q_1;
3 Zieh einen Kreis um P, es entsteht der Punkt Q_2 als Schnittpunkt mit G_1;
4 Setze die Nadel auf Q_1 und die Mine auf Q_2;
5 Ziehe einen Kreis K_1 um Q_1;
6 Setze die Nadel auf Q_2 und die Mine auf Q_1;
7 Ziehe einen Kreis K_2 umd Q_2, es entstehen die Schnittpunkte S_1 und S_2 zwischen K_1 und K_2;
8 Es entstehen zwei Schnittpunkte S_1, S_2 zwischen K_1 und K_2;
9 Zeichne die Gerade G_2 als Verbindung von S_1 und S_2;

Dabei kommt natürlich die Frage auf, wie man sich solche Algorithmen erarbeitet.

3. Wie erarbeite ich mir einen Algorithmus

Im Allgemeinen entwickelt man Algorithmen immer auf die selbe Art:

(1) Was ist mein Problem
Hierbei geht es darum zu verstehen, was man bekommt und was man am Ende erreichen will.
(2) Ein Model entwickeln
Man versucht sich klar darüber zu werden, wie man den Input modeliert, damit man mit diesem arbeiten kann. Bringt es etwas ihn zu unterteilen? Welcher Teil des Inputs ist wichtig? Was kann ich vernachlässigen?
(3) Spezifikation des Algorithmus
Die allgemeine Art oder Spezifikation des Algorithmus hilft einem dabei die richtige Formulierung schneller zu finden und somit schneller den richtigen Weg zu gehen. Am Anfang wird dies vielleicht als unwichtig erachtet was sich später, wenn ihr einige Probleme und Algorithmen schon kennt, sorgt es jedoch dafür, dass ihr eine klare Vorstellung davon habt wie ihr das Problem lösen könnt.
(4) Design des Algorithmuses
Hier kommen alle vorherigen Schritte zusammen. Was krieg ich, wie modifiziere ich es, wie übersetze ich mein Problem, etc. Ihr fangt den Algorithmus zu planen und aufzuschreiben.
(5) Überprüfen der Korrektheit des Algorithmus
Wenn man einen Algorithmus aufgeschrieben hat, dann sollte man generell überprüfen, ob auch das richtige Ergebnis herrauskommt und das unter jedem möglichen Input.
(6) Analyse des Algorithmus
Man versucht zu erkennen, welche Teile des Algorithmus am besten wann passieren sollten. Generell sagt man, dass man Sachen so selten wie möglich berechnen will und am besten nie zwei Mal das Gleiche. Auch wo es vielleicht doch Schwachstellen geben könnte kann sehr interessant werden.
(7) Implementation
Falls die voheringen Schritte geschehen sind, sollte das eigentliche Programmieren kein Problem mehr sein. Natürlich ist dies gerade zum Anfang Eins der größeren Probleme, aber genau dafür gibt es ja diesen Kurs. Hier sollt ihr lernen, wie ihr eure Gedanken in die Sprache Python übersetzen könnt. Aber zu erst werden wir lernen wie wir eine generellere Sprache der Algorithmik gesprochen wird.
(8) Testen
Testen ist essentiell, auch in der CoMa. Habt ihr wirklich keinen Denkfehler gemacht, nichts vergessen und sauber gearbeitet? Die Antwort war bei mir zumindest meist "Nein", aber dadurch das ich gut getestet habe, hatte ich meist keine Probleme bei der Abgabe.

4. Pseudocode

Pseudocode ist ein Überbegriff für die Vermischung von Umgangssprache und Algorithmik. Das Ziel ist hierbei allgemein über Algorithmen zu sprechen ohne explizite Programmiersprachen zu verwenden. Dafür ist generell alles erlaubt, was man umschreiben oder beschreiben kann. Allerdings sollte es einem Algorithmus schematisch folgen, dass bedeutet es sollte am besten von oben nach unten eine Abfolge bilden.

Hierfür gibt es einige Mittel um Wiederholungen zu vermeiden:

- **for / für** $a, b, c, \ldots$ **do**
- **while / während** Vergleich **then / dann**
- **if / wenn** Vergleich und **else / ansonsten**

Algorithm 2: Beispiel

Input: Eine Liste L an Zahlen
Output: Die Summe der Zahlen, falls diese größer als 0 ist, ansonsten 0

```
R = 0;
for zahl in L do
    R = R + zahl;
if R > 0 then
    Output R;
else
    Output 0;
```

Eure Aufgabe ist jetzt einen einfachen Algorithmus versuchen selbst zu schreiben.

Algorithm 3: Addiere die zwei größten Zahlen in einer Liste

Input: Eine Liste L von Zahlen
Output: Die Summe S als Addition der 2 größten Zahlen in L

```
/* Here be code                                    */
```

5. Pseudocode zu Python

Wiederholungen sind natürlich auch ein wichtiger Teil von Python:

```
1  for x in Iteration:
2      Mache etwas mit x
3
4  if Aussage 1:
5      Falls die Aussage 1 stimmt
6  elif Aussage 2:
7      Falls Aussage 1 nicht stimmt aber Aussage 2
8  else:
9      Falls weder Aussage 1 noch Aussage 2 stimmte
10
11 while Aussage:
12     Solange die Aussage stimmt mache ...
```

Das wichtige an Python ist die sogenannten Intendation oder Einschiebung, also die Anzahl an `Tabs` oder `Leerzeichen`. Diese muss innerhalb von einem Dokument gleichmäßig sein. Viele Programme machen diesen Einschub automatisch, daher sollte es für euch kein Problem darstellen. Meistens gibt es eine Art Leerzeichen und Tabs explixit einzublenden, um Fehler dieser Art zu finden.

Eine weitere wichtige Funktion von Python ist es natürlich Funktionen selbst zu definieren, um diese Aufzurufen. Das soll auch wieder sogenannten redundanten Code oder auch Wiederholungen im Code zu vermeiden (Siehe Algorithmus 1).

```
1  def meine_funktion(Variable1, Variable2):
2      return Variable1 + Variable2
```

Hierbei ist `return` das was `meine_funktion` zurückgibt.

6. Python Grundtypen

Jetzt folgen die Grundlagen der Python Programmiersprache. Damit Python weiß wie es mit spezifischen Eingaben umzugehen hat, gibt es sogenannte Typen. Die wichtigsten Typen sind 'Strings' oder 'Zeichenketten', 'Integers' oder 'ganze Zahlen', 'float' oder 'gebrochene Zahlen', 'Listen', 'Sets' oder 'Mengen' und 'Booleans' oder 'Zustände'. Es gibt natürlich noch viel mehr als diese 'Objekte' oder 'Typen'. Gehen wir kurz die einzelnen Typen durch:

- String
  Ein String ist wie gesagt eine Zeichenkette. `'Mein_String'`. Falls man ein anderen Typen zu einem String wandeln möchte gibt es dafür die `str()`-Funktion.

- Integer
  Ein Integer oder eine ganze Zahl ist das was der Name schon sagt, etwas ganzzahliges wie `1, 2, 3, -5, ...`. Zahlen werden generell als Integers interpretiert, außer es kommt ein `.` vor. Auch hier gibt es wieder die Funktion `int()` um eine Zahl oder ein anderen Typen zu einem Integer zu konvertieren, hierbei wird generell abgerundet.
- Floats
  Floats kommen immer dann zum Zuge, wenn Integers nicht mehr ausreichen. Auch hier gibt es die Konvertierungsfunktion `float()`.
- Listen
  Listen haben in der Eingabe generell die Form `[1, "b", 34.5]`. Wobei `[` und `]` die Grenzen und `,` die Unterteilung sind. Auf die einzelnen Teile kann man mit sogenannten Slices oder Unterteilungen zugreifen. Näheres kommt bald.
- Set
  Sets oder Mengen sehen aus wie Listen nur sind die `[]` gegen {} getauscht worden. Sets haben im Gegesatz zu Listen keine Ordnung und können nicht einfach unterteilt werden, dafür beinhalten sie aber auch jedes Element nur ein einziges Mal. Also sind {`1,1,2,2,2,2,2,3`} und {`1,2,3`} das Gleiche.
- Boolean
  Ein Boolean oder kurz Bool ist ein Zustand zwischen 'JA' und 'NEIN', 'Richtig' und 'Falsch' oder in Python `True` und `False`. Generell wenn etwas ist gilt es generell als `True`, also falls ihr fragt `if 'String'` ist die Antwort `True`. Hierbei ruft `if` die Funktion `bool()` auf das Objekt `'String'` auf und die Antwort lautet, Ja es existiert. Leere Objekte wie `''` oder auch `[]` sind `False`, hierzu zählt auch die 0.

7. Python Erweitert

Viele Typen in Python haben weitere Funktionen, sei es die append / Anhäng Funktion von Listen `meine_liste.append('füge das hinzu')` oder eine beliebige andere Funktion eines anderen Datentyps. Nicht immer kann man sich alles merken, aber es gibt Abhilfe.

Zum einen gibt es in Python selbst die sogenannte `help()` Funktion, diese ermöglicht es euch einen Einblick in die Funktion eines Objekts zu bekommen. Hier erfahrt ihr alles was dieses Objekt kann und wie ihr es verwendet. Falls ein Fehler aufkommt oder `print "test"` ohne Klammern funktioniert befindet ihr euch in einer Python2 Umgebung. Python2 wird zum Januar 2020 nicht mehr weiterentwickelt.

Manchmal will man jedoch die Möglichkeit haben durch etwas schnell durch zu scrollen, zu suchen oder sogar etwas zu kopieren. Dafür eignet sich wunderbar die offizielle Python Dokumentation `https://docs.python.org/3` hier steht alles feinsäuberlich.

Eine Funktionalität muss ich aber direkt noch ansprechen.

8. Slices

Slices oder Abschnitte sind wie der Name vielleicht schon erraten lässt die Möglichkeit nur Teile von spezifischen Objekten, genauer gesagt von iterierbaren Objekten, auszugeben. Ein Beispiel für ein iterierbares Objekt ist ein String:

```
1 x = 'Peter Lustig'
  /* Jetzt hat unser x eine Länge von 12 Buchstaben      */
2 x → 'Peter Lustig'
  /* Wenn wir jetzt nur einen Teil davon haben müssen wir
     slicen                                               */
3 x[0] → 'P'
  /* Der erste Eintrag (wir mussten 0 mal weitergehen), ist
     P                                                    */
4 x[1] → 'e'
  /* Falls wir mehr als ein Eintrag haben wollen, sieht
     dies so aus                                          */
5 x[0:3] → 'Pet'
  /* Also [Anfang:Ende]                                   */
6 x[1:4] → 'ete'
  /* Auch die Schrittgröße ist veränderbar                */
7 x[0:11:2] → 'PtrLsi'
  /* Negative Zahlen stehen immer für von hinten gezählt  */
8 x[0:-1] → 'Peter Lustig'
```

Weitere Beispiele sind Listen und Tupel. Integers sind nicht iterierbar! Falls ihr über die Zahlen eines Integers iterieren wollt, müsst ihr vorher einen String daraus machen.

9. Import und Standardbibliothek

Python kommt von Haus aus mit vielen weiteren Objekten und Funktionen, welche nicht immer erforderlich sind und welche man vorher importieren muss wenn man sie nutzen will. Ein Beispiel wäre `import math`. Jetzt findet ihr unter `math.log()` zum Beispiel die Logarithmus Funktion. Falls ihr genauer wissen wollt was `math` so beinhaltet, nutzt einfach die help Funktion `help(math)`. Weitere interessante Bibkiotheken sind `itertools`, `numpy`, `scipy` und viele weitere Module.

Falls ihr nicht immer `math.funktion()` schreiben wollt, gibt es auch die Möglichkeit `from math import *`, was soviel bedeutet wie, importiere alles aus `math`. Diese Funktionen stehen euch dann direkt zur Verfügung. Generell würde ich aber davon abraten, weil man auch gerne den Überblick über importierte Sachen verliert, bzw. nicht unbedingt weiß, was man alles importiert hat.

Teil 2. Arbeiten

10. Konsole, was ist das?

Eine Konsole/Console oder Befehlszeile ist wie der Name schon vermuten lässt die Schnittstelle des Nutzers, also euch, und dem Computer. Die meisten von euchen haben noch nie eine Konsole gesehen und für viele ist die erste Nutzung einer Konsole grausam. Sie ist klein, man kann nichts lesen und ausserdem ist die Schrift hässlich.

Es liegt meist daran, falls es keine Einstellungen zu den einzelnen Programmen bzw. Konsolen gibt, dass die Voreinstellugen verwendet werden bzw. in manchen Fällen es einfach zu Fehlern kommt.

Wir werden aber versuchen alldiese Probleme zu umgehen. Unser Unixpool hat bereits sehr solide Konsolen installiert und auch meist sind alle nötigen Schritte getan, um den Nutzer eine möglichst angenehme Nutzung zu ermöglichen. Unter euren Programmen findet sich meist eine Sparte mit dem Wort System oder auch Programmierung, dort befinden sich mehrere verschiedene Konsolen. Xterm ist dabei die meist benutzte Standardkonsole und auch wir werden auf sie zurückgreifen.

Das Erste was ihr machen müsst sobald euer Konsolenfenster erscheint ist es zu maximieren.

Eine Konsole befindet sich immer an einem spezifischen Ort, bzw. in einem spezifischen Ordner, falls nichts anderes angeben ist dies euer `home` Ordner. In diesem befinden sich Ordner wie Documents/Dokumente oder auch Downloads und Desktop. Im Studium wird es vorkommen, dass ihr diesen Aufenhaltsort verändern müsst, daher gebe ich euch jetzt die Fünf wichtigsten Konsolenbefehle an die Hand.

Algorithm 4: Nützliche Konsolen Befehle

```
cd Ordner
/* Gehe in den Ordner                                          */
mkdir Ordner
/* Erzeuge den Ordner                                          */
mv Objekt1 Objekt2
/* Bewege Obejekt1 nach Objekt2 (Umbenennen oder auch
   Objekt in Ordner verschieben)                               */
rm Objekt
/* Lösche das Objekt                                           */
rmdir Ordner
/* Lösche den Ordner                                           */

cd ..
/* Gehe einen Ordner nach 'oben', also in den Überordner
   der den derzeitigen Ordner enthält                          */
ls -a
/* Zeigt auch versteckte Objekte (mit einem . vor dem
   Namen) an                                                   */
```

11. Was ist ipython und wie nutze ich es

Ipython ist eigentlich nichts anderes als python3 oder python. Es hat aber einen sehr sehr großen Vorteil, Farben! Ipython ist an sich nichts anderes als eine Eingabemöglichkeit für euren Code. Wenn ihr in der Konsole seid schreibt ihr einfach `ipython` und dann sollte dieses in der Konsole starten. Beenden könnt ihr es mit `Strg + z` oder dem Befehl `exit()`.

12. Wo finde ich schnelle Hilfe

Hilfe findet ihr auf `https://docs.python.org/3` und mit der `help()` Funktion.

13. Umkehr Funktion

Gegeben einen String oder eine Liste, Ziel ist es das Objekt umzudrehen und auszugeben per print().

14. Listen

Als nächstes ist eure Aufgabe mehrere Funktionen zu schreiben, welche auch miteinander arbeiten sollten.

14.1. **Listen Appenden.** Die Funktion Anhaengen(Liste, Objekt) soll an die Liste das Objekt anhängen. Als Tipp Listen haben eine append Funktion. Also `[1,2,3].append(4)` hängt die 4 and die Liste an.

14.2. **Listen Mergen.** Aufgabe ist es an eine Liste alle Elemente einer anderen Liste anzufügen. Die Funktion sollte Zusammenfassen(Liste1, Liste2) heißen und eine Liste zurückgeben.

14.3. **Listen Intersect.** Als letztes soll die Funktion Schnitt(Liste1, Liste2) den Schnitt, zwischen zwei Listen als Liste zurückgeben.

15. Anfangsbuchstaben

Die Funktion soll Strings grob in Listen einteilen. Dabei soll der Anfangsbuchstabe entscheiden, in welche Liste er eingeteilt wird. Zurückgegeben soll eine Liste mit allen nicht leeren Listen werden.

16. Inverser Ackermann

Für die Aufgabe benötigt ihr das math package, welches ihr mit `import math` importieren könnt.
Gegeben eine Zahl x berechnet:

$$f(x) = \begin{cases} f\left(log_2(\lceil x \rceil)\right) + 1 & , \text{ falls } x \geq 2 \\ 0 & , \text{ ansonsten} \end{cases}$$

Findet ausserdem herraus, was die kleinste Zahl x ist mit $f(x) = 1$ bzw. $f(x) = 2, f(x) = 3$ und $f(x) = 4$.

Ohne es mit dem Computer zu berechnen, was ist die kleinste Zahl $f(x) = 5$. Falls ihr es doch versucht mit `Str + C` könnt ihr eine Schleife abbrechen.

17. Was ist ein Editor?

Zum schreiben und bearbeiten von Code der nicht direkt gelöscht wird, sobald man das Fenster schließt, braucht es einen Editor.

Da man beim Programmieren meist lange auf ein und die selbe Stelle schaut, sollte man ein Editor haben, welcher nicht unbedingt sehr anstrengend ist für die Augen. Daher empfehle ich an dieser Stelle Atom, welcher auf dem System installiert sein sollte. Diesen sollte ihr unter den Programmier-Programmen auf dem Rechner finden.

18. Wie führe ich eine .py Datei aus

Wenn ihr eine Datei abgespeichert habt, sollte dies immer mit der python typischen Endung .py sein. Diese könnt ihr dann mit der Konsole aufrufen, solange ihr im gleichen Ordner mit der Konsole seid, wie die Datei. Der Aufruf sieht dann wie folgt aus: `python test.py`. Zun interaktiven arbeiten mit einem Programm gibt es die `-i` Flag, dadurch wird das Programm geladen und ihr könnt eure Funktion frei aufrufen. Als Beispiel: `python -i test.py`.

19. Generator Function

Eure Aufgabe ist es, ein Programm `generate.py` zu schreiben, welches eine Funktion `generate(n)` beinhalten soll, welche eine Kombination aus Zahlen und Operatoren als einen String ausgeben soll. Dabei soll die Anzahl an Zahlen von der Eingabe `n` abhängen. Die Zahlen sollen zufällig sein und zwischen 0 und 9 liegen. Die Operatoren sollen zufällig + oder − sein.

Tipp: Es gibt das Modul random, welches die Funktion randint hat.

Beispiel Ausgabe:
`generate(3)` erzeugt den String `"3+2-5"`

20. Calculate Generated Stuff

Schreibt einen Interpreter der die Ausgabe von `generate` bekommt und diesen auswertet. Also `interprete("3+2-5")` soll `0` ausgeben.

21. Walking Simulator

Ihr sollt die Funktion `random_walk(n)` schreiben. Die Funktion soll ein 'Spielfeld' generieren, also Listen in Listen welche in jedem Feld ein 'o' bis auf zwei Felder, welche ein 'F' und ein 'Z' beinhalten sollen. 'F' und 'Z' sollen zufällig verteilt sein. Das Spielfeld soll $n \times n$ sein.

In jedem Schritt soll das Spielfeld in einer lesbaren Form ausgegeben werden und die Figur 'F' sich weiterbewegen.

Ziel ist es, dass die Figur nach endlich vielen Schritten im Ziel 'Z' ankommen soll, wonach die Funktion fertig ist.

TIPP: Eine zufällige Bewegung endet nach endlich vielen Schritten im Ziel, bei einem endlichen Spielfeld, mit Wahrscheinlichkeit 1.

OPTIONAL: Wände, Algorithmus für die Bewegung, Wraping, Ausgabe der Wege, etc.

22. Peano as lists in lists

Eine Peanoliste ist eine Liste die alle vorheringen Listen beinhaltet und wird mit Zahlen assoziiert. Wobei die leere Liste `[]` die Null ist.

$$0 = [], 1 = [[]], 2 = [[], [[]]], 3 = [[], [[]], [[], [[]]]]$$

Schreibt eine Funktion `peano_list(x)` welche einen Integer x als Eingabe bekommt und daraufhin die $x$te Peanoliste erzeugen soll.

Desweiteren sollt ihr Funktionen für die Addition von Peanolisten schreiben `peano_add(obj1,obj2)`, welche `obj1, obj2` addieren soll und die Addition als Peanoliste ausgeben soll.