1 Python Scripting für Physiker - Handout W2

2 Eingebaute Funktionen

Es gibt in Python eine Reihe von nützlichen vorimplementierten Funktionen. Es folgt eine kleine Auswahl häufig verwendeter Funktionen

- print a
  Gibt a auf dem Bildschirm aus. (Eigentlich "Keyword")

```
>>> print "Hallo Welt"
Hallo Welt
>>> a,b,c="Hallo","Welt",1.
>>> print c,a,b
1.0 Hallo Welt
```

- int(arg),float(arg),str(arg) ...
  Konvertiert, falls möglich, arg in integer,float,string... und gibt den Konvertierten Wert zurück

```
>>> print int(7.9)
7                    # Beachte : int(arg) rundet nicht.
```

- raw_input(prompt)
  Gibt promt auf dem Bildschirm aus und wartet auf eine Eingabe

```
>>>a=raw_input("Enter String:")
Enter String:
...Hallo Welt
>>>print a
Hallo Welt
```

- min(args)/max(args)/sum(args)
  Gibt den minimalen/maximalen Wert bzw. die Summe der übergebenen Argumente zurück.

```
>>>print min(5,3)
3
>>>a=[3,1,5,2,4]
>>>print min(a)
1
Beachte:
>>>a=[[3,2],[8,1],[2,2]]
>>>print min(a)
[2,2]
```

- len(obj)
  Gibt die Länge von obj zurück. Mit Länge ist die Anzahl der Einträge gemeint.

```
>>> a="Hallo"
>>> print len(a)
5                    # Der String a enthaelt 5 Zeichen
>>> a=[8.,"Spam",[1,2,3],9.]
```

```
>>> print len(a)
4                    # Die Liste a enthaelt 4 Eintraege
Beachte:
len funktioniert nur mit Sequenzen (wie Listen, Dictionaries oder Strings) ...
```

- range(start, stop [, step])
  Gibt eine Liste mit integern zurück [start, start + step, start + 2 * step, ..., start + n * step < stop]

```
>>>print range(4,10,1)
[4,5,6,7,8,9]
>>>print range(4,10,3)
[4,7]
Beachte:
Der stop Wert wird nicht erreicht.
```

3 Funktionen definieren

Zusätzlich zu den oben beschriebenen eingebauten Funktionen gibt es in Python die Möglichkeit sich eigene Funktionen zu definieren.

```
Syntax:

def function_name(args):    # Definiert Funktion 'function_name'
   statement1
   statement2
   ...
   return expression        # expression wird zurueck gegeben
```

Es folgt ein einfaches Beispiel für eine Definition und den Aufruf einer Funktion

```
>>>def multiply(a,b):       # Definiert die Funktion 'multiply' mit den Argumenten a,b
>>>   result=a*b            # Das Produkt von a und b wird berechnet
>>>   return result         # Das Ergebnis wird zurueck gegeben
>>>c=multiply(2,4)
>>>print c
8
Beachte:
Der Typ der Rueckgabe von multiply haengt von den Typen der Argumente a,b ab.
>>>d=multiply(2,"vier")
>>>print d
viervier
```

4 Built-in Type: Dictionary

Ein weiterer nützlicher Daten Typ in Python sind Dictionaries. Ihr Hauptmerkmal ist, dass sie nicht wie Listen und Strings mit Zahlen indiziert werden, sondern mit sogenannten 'keys' oder auch Schlüsselwörtern. Das erlaubt uns Informationen in einem 'lesbaren' Format zu speichern.

```
Syntax:
dict = {'key':value,'key2':value2} # Initialisierung eines Dictionaries
dict['key3']=value3                # Hinzuf gen eines Dictionary Eintrages

>>>Max = {'Name' : 'Mustermann', 'Alter': 25}
>>>Max["Beruf"],Max["Tel"]="Taxifahrer",123456
>>>print Max["Alter"]              # Auslesen eines Dictionary Eintrages
25
>>>"Beruf" in Max                 # Ueberpr fung ob key "Beruf" im Dictionary
True                               # vorhanden ist
>>>for key in Max:                # Schleife ueber alle Schluesselwoerter
>>>     print key,": ",Max[key]    # des Dictionary
Beruf : Taxifahrer
Tel : 123456
Name : Mustermann
Alter : 25
```

Ein Dictionary ist eine **unsortierte** Sequenz von '(Schlüsselwort, Wert)-Paaren'. Die Reihenfolge der aufgerufen Schlüsselwörtern innerhalb der Schleife im obigen Beispiel ist zufällig und nicht abhängig von der Reihenfolge der Initialisierung.

5 Stringformatierung

Oftmals ist es nützlich mit Strings zu arbeiten die in einer bestimmten Formatierung vorliegen, z.B. beim Schreiben von Dateien. In Python ist es besonders einfach formatierte Strings zu erzeugen.

- Zeilenumbrüche und Tabulatoren
  Zeilenumbrüche werden durch n und Tabulatoren durch t realisiert

```
>>> a="a\tb\nc\td"
>>> print a
a       b
c       d
```

- Platzhalter
  Oft ist es nützlich ein gewisses Format immer wieder zu verwenden, obwohl sich der Inhalt immer wieder ändert. Dies ist möglich durch die Verwendung von Platzhaltern. Ein Platzhalter wird geschrieben durch %(Typ).

```
Syntax :
%s : string; %i : iteger; %f : float; ...
%4.3f : float wird auf die 3. Dezimale gerundet und mit einer minimalen
        Laenge von 4 eingefuegt.
string_name="%s%f"%(var1,var2)      # ein string wird erzeugt. Dabei werden die
                                    # beiden Platzhalter durch die Werte von
                                    # str(var1) und float(var2) ersetzt
>>> Tag=10.
>>> Monat="Oktober"
>>> a="Heute ist der %f. %s."%(Tag,Monat)
>>> print a
Heute ist der 10.0000000. Oktober.
>>> a="Heute ist der %i %s."%(Tag,Monat)
>>> print a
Heute ist der 10 Oktober.
```

- Platzhalter with Mapping Keys
  Um bei String Formatierungen nicht die Übersicht zu verlieren bietet Python noch eine weitere Möglichkeit Variablen den zugehörigen Platzhaltern zuzuweisen. Dies geschieht mittels eines Dictionaries.

```
>>> a="""Well, there's egg and bacon; egg sausage and
         bacon; egg and %(food)s; egg bacon and %(food)s;
         egg bacon sausage and %(food)s; %(food)s bacon
         sausage and %(food)s;"""%{"food":"SPAM"}
Es werden alle Platzhalter mit dem key 'food' ersetzt durch
den zugehoerigen Dictionary Eintrag.
```

6 Datei Input/Output

Beim Arbeiten mit dem Computer ist es unerlässlich auf Dateien zu zugreifen um z.B. Messdaten einlesen oder Ergebnisse speichern zu können. Hierzu gibt es die eingebaute Funktion 'open' die ein Objekt erzeugt über dass die gewählte Datei bearbeitet werden kann.

```
Oeffnen einer Datei mit Lesezugriff
>>> file1=open("datei.txt","r")
Erzeugen einer Datei mit Schreibzugriff
>>> file1=open("datei.txt","w")  # Vorsicht ! eine ggf. bestehende Datei "datei.txt"
                                 # wird geloescht
Oeffnen/erzeugen einer Datei mit Schreibzugriff
>>> file1=open("datei.txt","a")  # Falls Datei "datei.txt" existiert wird sie geoeffnet
                                 # mit der Moeglichkeit Daten ans Ende der Datei
                                 # anzuhaengen. Ansonsten wird eine neue Datei erzeugt
```

Das oben erzeugte Objekt 'file1' kann jetzt dazu verwendet werden um aus der Datei zu lesen oder in sie zu schreiben. Das auslesen erfolgt über die Methode 'readline' (mehr zu Objekten und Methoden folgt später).

```
datei.txt ascii-datei
x y z

1 2 3
>>> file1=open("datei.txt","r")
>>> file1.readline()       # gibt die 1. Zeile aus datei.txt als String zurueck
'x y z{\backslash}n'
>>> file1.readline()       # gibt die 2. Zeile aus datei.txt als String zurueck
'\n'
>>> file1.readline()       # gibt die 3. Zeile aus datei.txt als String zurueck
'1 2 3'
Beachte : Nach dem Ende der Datei gibt 'readline' einen leeren String zurueck
>>> file1.readline()       # gibt einen leeren String zurueck
''
Anmerkung : die Stringmethode 'split' kann dafuer verwendet werden eingelesene Zeilen
weiter zu verarbeiten. Hierbei sind auch die Typenkonvertierungen int(arg),float(arg)
hilfreich.
>>>'1 2 3'.split()
['1','2','3']
>>>int('1 2 3'.split()[0])
1
```

Um in 'file1' zu schreiben gibt es die Methode 'write'.

```
>>> file1=open("datei2.txt","w")  # erzeugt leere Datei 'datei2.txt' mit Schreibzugriff
>>> file1.write("x y z")          # schreibt 'x y z' ans Ende von 'datei2.txt'
>>> file1.write("1 2 3")          # schreibt '1 2 3' ans Ende von 'datei2.txt'
Beachte : Zeilenumbrueche muessen explizit mitgeschrieben werden
datei2.txt haette folgenden Inhalt
x y z1 2 3

>>> file1=open("datei2.txt","w")  # Die bestehende Datei 'datei2.txt'
>>> file1.write("x y z\n")        # wird dabei geloescht
>>> file1.write("1 2 3")
datei2.txt haette nun folgenden Inhalt
x y z
1 2 3
```

Wichtig ist zu beachten dass Änderungen in der nicht sofort beim Aufruf von 'write' geschrieben werden. Um sicher zu stellen dass alles wie gewuenscht in die Datei geschrieben wird ist es wichtig die Datei am Ende mit der Methode 'close' zu schliessen. Gleichzeitig wird auch das zugehörige obj 'file1' zerstöert.

```
Im obige Beispiel
>>> file1=open("datei2.txt","w")
>>> file1.write("x y z\n")
>>> file1.write("1 2 3")
Beachte : der folgende Aufruf wuerde noch eine leere Datei 'datei2.txt' anzeigen
>>> less datei2.txt
Erst bei Aufruf
>>> file1.close()
hat datei2.txt den erzeugten Inhalt
>>> less datei2.txt
x y z
1 2 3
```

7 Übungen

1. Schreiben sie eine Funktion, die eine Anzahl von K Punkten des Polynoms $P(x) = x^0 + x^1 + ... + x^{N-1} + x^N$ im Wertebereich $x_s < x < x_e$ berechnet. N soll bei bei Aufruf der Funktion vom Nutzer bestimmt werden.

   ```
   Beispiel:
   >>> x,y = poly(0,10,5)  # xs=0; xe=10; K=5
   Waehle Grad des Polynoms:
   ... 2
   >>> print x,y
   [0,2,4,6,8] , [1,7,21,43,73]
   ```

2. Schreiben sie eine Funktion, die das Resultat aus (1) in einer Datei speichert.

   ```
   Beispiel:
   >>> save(x,y)
   ```

3. Schreiben sie eine Funktion, welche die Datei aus (2) einliest.

   ```
   Beispiel:
   >>> x,y=load()
   ```

```
>>> print x,y
[0,2,4,6,8] , [1,7,21,43,73]
```

4. (Advanced) Erstellen sie ein Objekt, welche bei jedem Aufruf der Methode $next()$ die nächst höhere Primzahl im Wertebereich $N_s < N < N_e$ zurückgibt. Die Methode darf hierfür **nicht** bei jedem Aufruf alle Primzahlen bis N berechnen!

```
Beispiel:
>>> primzahlen=primes(10,100)
>>> print primzahlen.next()
11
>>> print primzahlen.next()
13
>>> print primzahlen.next()
17
```