1 Python Scripting für Physiker - Handout W2

2 Built-in functions

Python has a comparably huge standard library, to there are several useful built-in functions in Python. A short selection will be presented here:

- print a
  Prints variable a to the screen. print is implemented as both a keyword and a function in Python 2.x.

```
>>> print "Hello World"
Hellow World
>>> a,b,c="Hello","World",1.
>>> print c,a,b
1.0 Hello World
```

- int(arg),float(arg),str(arg) ...
  Converts to integer, float, str etc. if possible, and returns the converted value.

```
>>> print int(7.9)
7
```

  Note: int(arg) shaves off decimal places, so it basically rounds down.

- raw_input(prompt)
  Opens an input prompt on the screen and waits for input, terminated by enter.

```
>>> a=raw_input("Enter String:")
Enter String:
...Hello World
>>> print a
Hello World
```

- min(args), max(args), sum(args)
  Returns the minimum, maximum or sum value of a the given arguments or sequence.

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

- len(obj)
  Returns the length, or number of elements, of a sequence. For strings the number of characters is returned.

```
>>> a="Hello"
>>> print len(a)
5
>>> a=[8.,"Spam",[1,2,3],9.]
>>> print len(a)
4
```

- range(start, stop [, step])
  Returns a list of integers. [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]
Note: The stop value is \textit{not} part of the returned sequence.
```

3 Function definition

Of course Python allows the programmer to define his or her own functions. The keyword **def** (short for define) is used for that.

```
Syntax:

def function_name(args):    # defines function 'function_name'
   statement1
   statement2
   ...
   return expression        # expression is returned
```

A simple example of the definition and calling of a function:

```
>>> def multiply(a,b):       # Defines the function 'multiply' with arguments a,b
>>>     result=a*b            # Variable 'result' stores the value of the multiplication of a and b
>>>     return result         # Returns the value of 'result'
>>> c=multiply(2,4)
>>> print c
8
```

Note: The return type depends on the arguments:

```
>>> d=multiply(2,"vier")
>>> print d
viervier
```

4 Built-in Type: Dictionary

A useful Python data type is the **Dictionary**. It's basic functionality is that of a hash in other languages, it stores pairs of keys and associated values. This allows the programmer to easily map values to "human-readable" keys.

```
Syntax:
dict = {'key':value,'key2':value2} # Initializes a dictionary
dict['key3']=value3                # Adds a key value pair to the dictionary

>>> Max = {'Name' : 'Mustermann', 'Alter': 25}
>>> Max["Beruf"],Max["Tel"]="Taxifahrer",123456
>>> print Max["Alter"]             # returns the value associated with key "Alter"
25
>>> "Beruf" in Max                 # Checks whether the key "Beruf" exists in Max
True
>>> for key in Max:                # Iterates over all keys in Max
>>>       print key,": ",Max[key]
Beruf : Taxifahrer
Tel : 123456
Name : Mustermann
Alter : 25
```

The dictionary elements are the key-value pairs. It has to be noted that unlike lists or tuples the order of the dictionary elements does not depend on the order in which the elements were added to the dictionary, it is in fact arbitrary.

5 String formatting

Oftentimes it is useful to format strings in a certain fashion, for example when exporting measurement data. String formatting borrows its syntax heavily from C, so the basic string formatting control characters are the same.

- Line feeds and tabulators
  Line feeds are created by \n and tabulators are created by \t:

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

- Placeholders
  It can be useful to use a certain output string format multiple times, but keep some parts of the string variable. It is possible to define such strings with the use of placeholders. A placeholder for a certain data type is defined via %type. This syntax borrows heavily from C, too.

```
Syntax :
%s : string; %i : iteger; %f : float; ...
%4.3f : the float number will have a minimum length of 4,
and will be rounded to the third decimal digit

string_name="%s%f"%(var1,var2)      # defines a string where the placeholders are
                                    # replaced by str(var1) and float(var2), respectively
>>> Day=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.
```

- Placeholders and mapping keys
  To aid in keeping the complete picture of a formatting string, Python offers the possibility to use Dictionaries to fill placeholder variables:

```
>>> 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"}
```

  All placeholders with the key 'food' get replaced by the associated dictionary entry.

6 File Input/Output

Working with datasets necessitates the possibility to read and write files. Python offers the very simple function **open** to open file objects for reading and writing from and to text files:

```
>>> file1=open("file.txt","r")   # opens "file.txt" for reading
>>> file1=open("file.txt","w")  # opens "file.txt" for writing from the beginning
                                 # Warning!: existing "file.txt" is overwritten
>>> file1=open("file.txt","a")  # opens "file.txt" for writing, but written data
                                 # is appended to the end of the file
                                 # if the file does not exists, this is the same as
                                 # opening a file for writing ('w')
```

The created object 'file1' can now be used to read from the file or write to it. A single line can be rea,d as a string, by the readline method of the file object.

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

1 2 3
>>> file1=open("file.txt","r")
>>> file1.readline()        # returns the first line of "file.txt"
'x y z{\backslash}n'
>>> file1.readline()        # returns the second line of "file.txt"
'\n'
>>> file1.readline()        # returns the third line of "file.txt"
'1 2 3'
```

Note: When there are no further lines in the file object, 'readline()' returns an empty string ''

The method 'split' on strings can split a string in multiple substrings:

```
>>>'1 2 3'.split()
['1','2','3']
>>>int('1 2 3'.split()[0])
1
```

To write to the file 'file1' use the write method:

```
>>> file1=open("file2.txt","w")    # creates 'file2.txt' and opens for writing
>>> file1.write("x y z")           # writes 'x y z' to the end of 'file2.txt'
>>> file1.write("1 2 3")           # writes '1 2 3' to the end of 'file2.txt'
```

Note: Line feeds have to be added explicitly, the contents of 'file2.txt' would now be: x y z1 2 3

```
>>> file1=open("file2.txt","w")    # creates 'file2.txt' and opens for writing
>>> file1.write("x y z\n")           # writes 'x y z' to the end of 'file2.txt'
>>> file1.write("1 2 3")           # writes '1 2 3' to the end of 'file2.txt'
```

'file2.txt' now would have the content: x y z 1 2 3

It is important to note that 'write' does not immediately write the lines to the output file. To ensure that the lines are written to the file, close the file object using its 'close' method. The file will be written to the file system, and the associated file object is destroyed.

```
>>> file1=open("file2.txt","w")
>>> file1.write("x y z\n")
>>> file1.write("1 2 3")
>>> file1.close()
>>> less datei2.txt
```

'file2.txt' now has the content: x y z 1 2 3

7 Exercises

1. Define a function which calculates a number of K points of a polynomial $P(x) = x^0 + x^1 + ... + x^{N-1} + x^N$ in the range $x_s < x < x_e$. N is to be determined by the user when the function is called. Example:

```
>>> x,y = poly(0,10,5)  # xs=0; xe=10; K=5
Choose polynomial order
... 2
>>> print x,y
[0,2,4,6,8] , [1,7,21,43,73]
```

2. Define a funciton which saves the result of (1) to a file. Example:

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

3. Define a function which reads in the file created in (2) Example:

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

4. (Advanced) Create an object which has a method *next*() which returns the next prime number in the range $N_s < N < N_e$. The method may *not* calculate all prime numbers up to N with each call. Example:

```
>>> prime_nums=primes(10,100)
>>> print prime_nums.next()
11
>>> print prime_nums.next()
13
>>> print prime_nums.next()
17
```