1 The "Hangman"-Challenge

"Hangman" is a simple word-guessing game that can be implemented with python fairly easy. Most of you may know it as "Galgen-Raten" - If you have absolutely no idea what "Hangman" or "Galgen-Raten" is, you may want to check Hangman Wiki for a brief explanation.

The idea behind the "Hangman"-Challenge is to give you the opportunity to practise what you've learned so far in a playful manner. You don't need any function or methods we haven't introduced yet. If you feel otherwise - use Google or **ask**!

1.1 Exercise (60 to 90 min)

Implement the Hangman game in python. We already prepared a file that contains several words to be used in the Hangman game (Word List). We also prepared a small list of milestones you should achieve throughout the challenge, which should give you a good idea how to solve the challenge.

- (1) Take a piece of paper and write down what kind of variables you need, functions you need to design, and how the program should be structured. Be specific! Before you begin writing you should already have a good idea how the final program should look like (15 min)
- (2) Implement all necessary function and test them. A good idea for some functions would be something that prints formated output for the hangman game, somethings that selects a random word from the provided word.dat file and something that checks if the user has picked the correct character or not (30 min)
- Take your idea from (1) and your functions from (2) to create the hangman game. (15 min for the implementation and 15 min for bugfixing)

A typical implementation of the Hangman game would produce the following output each round:

```
Round 6 (4 lifes left)
----------------------

The Current Word is: T r _ _ a _ n _ _ a n
Characters already picked: a n r h t

Choose a new character or type 'Solve' to guess the word:

>>>
```

2 Python Scripting für Physiker -Handout W3

An important concept of python is the ability to store functions, classes, and variables in so-called modules. This concept is called modularisation and allows to break a program into several smaller parts that are each stored in different python scripts. The obvious benefit is here to organize a complex project by dividing it into several smaller parts, that can be maintained independently from the rest of the program.

2.1 Import Modules

Python provides several built-in modules, which are already accessible withing the interpreter, as well as several "third-party" modules, which have to be installed first (see Installation.pdf for a brief explanation). To gain access to the module scope, we first have to import of using the import keyword.

```
>>> import random           #import of the random module
>>> dir(random)             #shows all objects of the module
[ ... , 'randint', ... ,'uniform', ... ]
```

To gain access to functions and variables of a module, you use the same syntax as with methods.

```
Syntax:
Modulname.Funktion()

Beispiel:
>>> a=random.randint(0,10) #Accessing the randint function of the random module
>>> print a
4
>>> b=random.uniform(0,10) #Acessinf the uniform function of the random module
>>> print b
5.0087110427737924
```

You can also add functions and variables from a given a module to the namespace of the current script. This way you don't have to type the module name in front of the function every time you want to use it. Instead you call the function as if it is defined in the current python script.

```
>>> from math import sin, pi       #import sin and pi from module math
>>> print sin(pi)
0.0
>>> from math import sin as Sinus #Import sin and add it to the namespace as Sinus
>>> print Sinus(pi)
0.0
```

You can also add all available function and variables to the current namespace of your script.

```
>>> from math import *
>>> print cos(pi)
1.0
```

It is important to know, that all variables and functions of the given module are added to the current namespace and that they possibly overwrite existing variables and function in your current namespace.

```
>>> path = "/data/datenfile.txt"
>>> from os import *
>>> print path
<module 'posixpath' from '/usr/lib/python2.6/posixpath.pyc'>
```

2.2 Built-In Modules

As already noted, there are a lot of built-in modules already available in any standard python installation. A complete list of these modules can be found under Standard Library. Throughout the lecture we will introduce and explain some of them in detail.

2.3 "User-created" modules

To import your own functions and classes from external scripts (instead of using copy and paste!!) you can also create your own modules. In general you can use every python script as a module.

```
funktionen.py
==========
def gerade(x,a=1,b=0):
  """
  Returs the value a*x+b
  """
  return a*x+b

def parabel(x,a=1,b=0,c=0):
  """
  Returs the value a*(x+b)**2+c
  """
  return a*(x+b)**2+c
==========

>>>from funktionen import gerade,parabel
>>>print gerade(2)
2
>>>print parabel(2)
4
```

In this example the script funktionen.py lies in the same folder from which you called python. When you import a module, python automatically searches for this module in a list of paths, that includes the PYTHONPATH (which is a environment variable you can set in Windows, Linux, and Mac OS), the standard path of python extensions (this is where all your installed "third-party" modules go), and the current path (where you started the python interpreter).