Simulation und Modellierung in der Kunststofftechnik

0. *for* und *while* Schleifen

1. Hilfreiche Iterationsfunktionen

2. Funktionen selber programmieren

3. *if* , *elif* und *else* Anweisungen

4. Übungsaufgaben

for und *while* Schleifen

In [6]:
```
for i in range(2):
    for j in range(2):
        for n in range(2):
            print(i,j,n)
```

```
4 0 0
4 0 1
4 1 0
4 1 1
4 0 0
4 0 1
4 1 0
4 1 1
```

In [11]:
```
i = 0
while i <= 4:
    print(i)
    i = i + 1 # +++ Achtung! Immer angeben +++
```

```
0 True
1 True
2 True
3 True
4 True
```

In [7]:
```
%%markdown
Wenn du nicht vorher weißt, wie oft ein Codeblock wiederholt werden muss, ist es
 notwendig $while$-Schleifen einzusetzen. Meistens werden diese Schleifen einges
etzt, wenn auf etwas gewartet wird.
```

Wenn du nicht vorher weißt, wie oft ein Codeblock wiederholt werden muss, ist es notwendig $while$-Schleifen einzusetzen. Meistens werden diese Schleifen eingesetzt, wenn auf etwas gewartet wird.

Hilfreiche Iterationsfunktionen

len()

In [17]:
```
help(len)
```

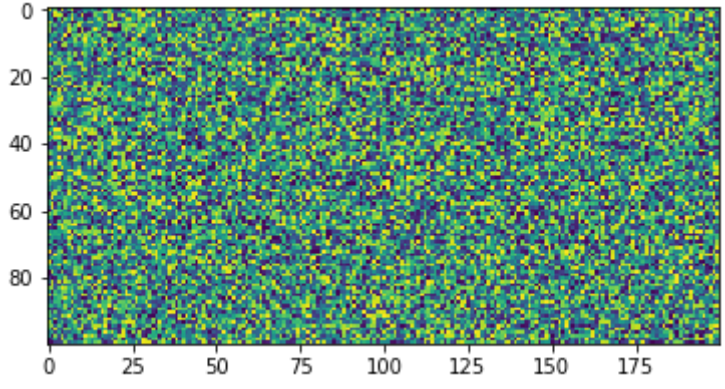
```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt

a = np.random.rand(100,200)
plt.imshow(a)
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x7f26fdbd89e8>
```



In [11]:
```
print(len(a))
print(len(a[0]))
print(len(a[:,0]))
```

```
100
200
100
```

zip()

Die *zip()*-Funktion verbindet mehrere iterierbare Objekte und erzeugt ein Tuple mit den Inhalten aus den Objekten.

In [23]:
```
help(zip)
```

```
Help on class zip in module builtins:

class zip(object)
 |  zip(iter1 [,iter2 [...]]) --> zip object
 |
 |  Return a zip object whose .__next__() method returns a tuple where
 |  the i-th element comes from the i-th iterable argument.  The .__next__()
 |  method continues until the shortest iterable in the argument sequence
 |  is exhausted and then it raises StopIteration.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signatur
e.
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
 |      Return state information for pickling.
```

In [16]:
```
### 3. $if$, $elif$ und $else$ Anweisungena = range(0,5)
a = range(8, -1, -1)
b = range(20,26)

for i in zip(a,b):
    print(i)
```

```
(8, 20)
(7, 21)
(6, 22)
(5, 23)
(4, 24)
(3, 25)
```

In [8]:
```
a = range(0,5)
b = range(20,25)

for i,j in zip(a,b):
    print('i={} und j={}'.format(i,j))
```

```
i=0 und j=20
i=1 und j=21
i=2 und j=22
i=3 und j=23
i=4 und j=24
```

enumerate()

Die *enumerate*()-Funktion zählt die Iterationsschritte.

In [21]:
```
x = ['Steffen','Arthur','Marco','Sonja','Simon','Karin','Till']
x
```

Out[21]:
```
['Steffen', 'Arthur', 'Marco', 'Sonja', 'Simon', 'Karin', 'Till']
```

In [22]:
```
for i,name in enumerate(x):
    print(i,':' + name)
```

```
0 :Steffen
1 :Arthur
2 :Marco
3 :Sonja
4 :Simon
5 :Karin
6 :Till
```

Funktionen

In [9]:
```
def name_der_funktion(a,b):
    # das ist ein Kommentar
    c = a + b
    return c
```

In [14]:
```
ergebnis = name_der_funktion(99, 66)
print(ergebnis)
```

```
165
```

In [5]:
```
def bp_some_func(x):
    r"""Brief description of the function"""
    return x**2
```

In [7]:
```
help(bp_some_func)
```

```
Help on function bp_some_func in module __main__:

bp_some_func(x)
    Brief description of the function
```

In [8]:
```
help(np.random.rand)
```

```
Help on built-in function rand:

rand(...) method of mtrand.RandomState instance
    rand(d0, d1, ..., dn)

    Random values in a given shape.

    Create an array of the given shape and populate it with
    random samples from a uniform distribution
    over ``[0, 1)``.

    Parameters
    ----------
    d0, d1, ..., dn : int, optional
        The dimensions of the returned array, should all be positive.
        If no argument is given a single Python float is returned.

    Returns
    -------
    out : ndarray, shape ``(d0, d1, ..., dn)``
        Random values.

    See Also
    --------
    random

    Notes
    -----
    This is a convenience function. If you want an interface that
    takes a shape-tuple as the first argument, refer to
    np.random.random_sample .

    Examples
    --------
    >>> np.random.rand(3,2)
```

Hinweise für die Verwendung von Funktionen und Methoden aus importierten Bibliotheken

In [ ]:

```
def bp_some_func(x, y, z=3.14, **kwargs):
    r"""Some function

    Does some stuff.

    Parameters
    ----------
    x : int
        Description of x
    y : str
        Description of y
    z : float, optional
        Description of z.  Defaults to 3.14
    **kwargs
        Arbitrary optional keyword arguments.
        w : float
            Defaults to 6.28
    # evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
    Returns
    -------
    double
        Some nonsensical number computed from some ugly formula

    """
    w = kwargs.pop("w", 6.28)
    if kwargs:
        print("Got {0} unused kwargs".format(len(kwargs)))
    return (x**2 + len(y)) * (w + z)
```

3. *if* , *elif* und *else* Anweisungen

if und *elif* werden mit Bedingungen versehen, *else* macht den Rest.

In [ ]:
```
name = input('Wie heißt du? >>>  ')

if type(name) != str:
    print('Error! Bitte ein String eingeben')
else:
    print('Hallo ' + name)
```

In [59]:
```
a = np.random.randint(0,100,1)

if a < 33:
    print('Erstes Drittel')
elif a < 66:
    print('Zweites Drittel')
else:
    print('Drittes Drittel')
print(a)
```

```
Erstes Drittel
[31]
```

Übungsaufgaben