Bibliotheksmodul `random`

Dieses Modul enthält verschiedene Zufallszahl-Generatoren, die man zur Simulation von zufälligen Prozessen benötigt, z.B.

```
randint(a, b)        # int-Stichprobe einer Gleichverteilung in [a, b]
gauss(mu, sigma)     # float-Stichprobe einer Normalverteilung
random()             # float-Stichprobe einer Gleichverteilung in [0, 1)
uniform(a, b)        # float-Stichprobe einer Gleichverteilung in [a, b]
seed()               # initialisiert den Generator mit der Uhrzeit
seed(x)              # initialisiert den Generator mit x
choice(x)            # wählt ein Element aus  x  aus (gleiche Wahrscheinlichkeit)
```

```
randint(1, 6)                          # Wurf eines fairen Würfels
choice([1, 2, 3, 4, 5, 6, 6, 6])      # dieser Würfel wirft gern 6
```

Die folgende Funktion liefert eine Stichprobe aus `{0, 1}` mit `P(1) = p`

```
def bernoulli(p):
     if random() <= p:
          return 1
     else:
          return 0
```

oder kürzer:

```
def bernoulli(p):
     return int(random() <= p)     # True -> 1, False -> 0
```

Die **Bernoulli-Verteilung** braucht man für Sieg-Niederlage Simulationen, wenn man mit der Wahrscheinlichkeit `p` gewinnt (Resultat ist `1`).

Eine Stichprobe der **Binomialverteilung(n, p).** Diese bekommt man als Summe von n Bernoulli-Verteilungen:

```
def binomial(n, p):
     return sum(bernoulli(p) for i in range(n))
```

Mit der Binomialverteilung modelliert man Ereignisse wie die Anzahl der Einzel-Gewinne unter `n` Versuchen.