

# Merkblatt - Datumsangaben

May 7, 2020

## 1 Datumswerte in Python

Zur Darstellung von Datumswerten gibt's in Python das `datetime`-Modul (<https://docs.python.org/3/library/datetime.html>). Damit kannst du Datumswerte repräsentieren und damit rechnen.

```
[2]: from datetime import datetime
```

Über `datetime.now()` hast du die Möglichkeit, ein Datumsobjekt zum aktuellen Datum erstellen zu lassen:

```
[3]: now = datetime.now()
print(now)
```

```
2017-09-23 17:51:05.689893
```

Alternativ kannst du auch ein spezifisches Datum angeben (hier: 20.8.2017, 20:00:00):

```
[4]: day = datetime(2017, 8, 20, 20, 0, 0)
print(day)
```

```
2017-08-20 20:00:00
```

Wenn du ein solches Datumsobjekt erstellt hast, kannst du z. B. über `.year` auf das Jahr direkt zugreifen. Du hast also direkten Zugriff auf alle einzelnen Angaben:

```
[5]: print(day.year)
print(day.month)
print(day.day)
print(day.hour)
print(day.minute)
print(day.second)
```

```
2017
```

```
8
```

```
20
```

```
20
```

```
0
```

```
0
```

Die `.timestamp()`-Methode gibt dir den entsprechenden Unix-Timestamp zu einem bestimmten Datumswert zurück. Unix-Timestamp ist einfach nur eine Zahl, die die Sekunden seit dem 01.01.1970 hochzählt.

Vorteil bei einem Unix-Timestamp ist, dass wir ihn recht kompakt speichern können; intern muss der Computer ja nur eine Zahl speichern, um einen Datumswert zu repräsentieren.

Wir können ihn hier aber verwenden, um den Zeitunterschied in Sekunden zu berechnen. :-)

```
[6]: print(day.timestamp() - now.timestamp())
```

```
-2929865.6898930073
```

### 1.0.1 date- und time- Angaben

Das `datetime`-Paket stellt uns auch weitere Klassen zur Verfügung, die wir verwenden können, um mit Datumsangaben zu arbeiten.

Beispielsweise repräsentiert `date` eine Datumsangabe, `time` eine Zeitangabe.

- `datetime`: Datumsangabe + Zeitangabe
- `date`: Nur Datumsangabe
- `time`: Nur Zeitangabe

```
[7]: from datetime import date, time
```

```
[8]: d = date(2017, 8, 20)
print(d)
```

```
2017-08-20
```

```
[9]: t = time(20, 1, 4)
print(t)
```

```
20:01:04
```

Natürlich kannst du auch Datumswerte vergleichen.

Aber Achtung! Bei `date` ist `==` beim gleichen Datum erfüllt, bei einem `datetime`-Objekt müssen natürlich sowohl die Datumsangabe als auch die Zeitangabe übereinstimmen.

Ausführlich also:

- `datetime`: Datumsangabe + Zeitangabe müssen übereinstimmen
- `date`: Datumsangabe muss übereinstimmen
- `time`: Zeitangabe muss übereinstimmen

```
[10]: print(date(2017, 8, 20) == date(2017, 8, 20))
print(datetime(2017, 8, 20, 20, 0, 0) == datetime(2017, 8, 20, 15, 0, 0))
```

```
True
False
```

### 1.0.2 datetime in date und time umwandeln

Natürlich kannst du ein `datetime`-Objekt auch in ein `date` und ein `time`-Objekt zerlegen:

```
[11]: dt = datetime(2017, 8, 20, 20, 0, 0)
      print(dt.time())
      print(dt.date())
```

```
20:00:00
2017-08-20
```

### 1.0.3 date und time in datetime umwandeln

Und natürlich geht das auch anders herum: :-)

```
[12]: print(datetime.combine(date(2017, 8, 20), time(20, 30, 0)))
```

```
2017-08-20 20:30:00
```

## 2 Datumswerte ausgeben

Natürlich ist es auch wichtig, dass wir ein Datum formatiert ausgeben möchten. Schließlich möchten wir nicht immer eine Ausgabe in der Form `2017-08-20` haben.

Dazu bietet Python verschiedene Format-Optionen an.

Dokumentation: <https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior>

```
[13]: from datetime import datetime
      now = datetime.now()
```

```
[14]: print(now)
```

```
2017-09-23 17:51:06.325082
```

```
[15]: print(now.strftime("%d.%m.%Y"))
      print(now.strftime("%Y-%m-%d"))
      print(now.strftime("%Y%m%d"))
```

```
23.09.2017
2017-09-23
20170923
```

### 2.0.1 Datumswerte einlesen

Das ganze funktioniert auch anders herum: Du kannst auch Datumswerte aus einem String extrahieren, wenn du z. B. mit den Python Funktionen später mit dem Datum weiter rechnen willst:

```
[16]: d = "18.07.2017"
```

```
[17]: print(datetime.strptime(d, "%d.%m.%Y"))
```

2017-07-18 00:00:00

### 3 Zeitdifferenzen

Mit einem `timedelta` hast du die Möglichkeit, mit Zeitdifferenzen zu rechnen. Ein `timedelta` beschreibt hier einfach nur den Zeitunterschied zwischen zwei Datumswerten.

Beispielsweise kannst du auf einen Datumswert ein `timedelta` addieren, um zum neuen Datum zu kommen:

```
[18]: from datetime import datetime, timedelta
now = datetime.now()
print(now)
print(now + timedelta(days = 20, hours = 4, minutes = 3, seconds = 1))
```

2017-09-23 17:52:13.681326

2017-10-13 21:55:14.681326

Zudem ist es so, dass, wenn du zwei Datumswerte voneinander abziehst, als Ergebnis ein `timedelta`-Objekt herauskommt:

```
[19]: day = datetime(2017, 8, 20)
td = day - now
print(td)
```

-35 days, 6:07:46.318674

```
[20]: print(datetime(2018, 1, 1) + td)
```

2017-11-27 06:07:46.318674

```
[ ]:
```