PythonTutorial

November 22, 2017

1 Einführung in Python

Felix Riese | 22.11.2017

1.1 Allgemeines

- Interpreter, kein Compiler. Langsamer als C++.
- Anspruch: gut lesbar, knapper Programmierstil.
- Blöcke nicht durch Klammern sondern durch Einrückung dargestellt
- GroSSe Community, zahlreiche Pakete verfügbar

1.2 The Zen of Python

by Tim Peters > Beautiful is better than ugly.

> Explicit is better than implicit.
>
> Simple is better than complex.
>
> Complex is better than complicated.
>
> Flat is better than nested.
>
> [...]

Ausgabe über `import this` in der Python-Konsole.

1.3 Installation von Python und Jupyter Notebooks

1. Installation von **Anaconda**
2. Start von Jupyter Notebooks: `jupyter notebook`

1.4 Ausgabe in Python

Die Ausgabe funktioniert in Python 3 mit der Funktion `print()`:

```
In [1]: print("Mit 'print' kann man Informationen auf der Konsole ausgeben.")
```

```
Mit 'print' kann man Informationen auf der Konsole ausgeben.
```

Hier müssen die Dateitypen nicht beachtet werden:

```
In [2]: print("Ein String", 3.141, True, complex(4, 2))

Ein String 3.141 True (4+2j)
```

1.5 Variablen

In Python wird der Variablentyp nicht bei der Definition der Variablen mit angegeben (*dynamic typing*).

```
In [4]: var_string = "blablabla"
        var_number = 42             # a_number is an integer
        var_number = 3.141          # a_number now is a float
        var_bool = True
```

Man kann Variablen casten in andere Dateitypen und mit `type()` den Typ der Variablen bekommen.

```
In [5]: print(var_number)
        print(type(var_number))
        print(int(var_number))
        print(var_bool)

3.141
<class 'float'>
3
True


In [6]: print(float(var_string))


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-6-782b578021c2> in <module>()
    ----> 1 print(float(var_string))


        ValueError: could not convert string to float: 'blablabla'
```

Variablennamen können frei gewählt werden aus Buchstaben, Zahlen und Unterstrichen. Sie dürfen aber nicht mit einer Zahl beginnen. Die folgenden Wörter dürfen nicht als Variablennamen verwendet werden:

and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield

1.6 Ausgabe von Variablen

Alte Syntax:

```
In [7]: print("Alte Ausgabe von %s, z.B. Floats %.2f." % ("Variablen", 3.141))
```

```
Alte Ausgabe von Variablen, z.B. Floats 3.14.
```

Neue Syntax:

```
In [8]: print("Neue Ausgabe von {0}, z.B. Floats {1:.2f}.".format("Variablen", 3.141))
```

```
Neue Ausgabe von Variablen, z.B. Floats 3.14.
```

Schön formattierte Ausgabe:

```
In [9]: print("{0:<20} {1:>8} {2:^8}".format("Name", "Gröe", "Alter"))

        print("-"*37)

        print("{name:<20} {groesse:>8.2f} {alter:^8.0f}".format(name="Max", groesse=1.79, alte
        print("{name:<20} {groesse:>8.2f} {alter:^8.0f}".format(name="Anne", groesse=1.68, alt
        print("{name:<20} {groesse:>8.2f} {alter:^8.0f}".format(name="Rita", groesse=1.735646,
```

```
Name                    Gröe  Alter
-------------------------------------
Max                     1.79    26
Anne                    1.68   103
Rita                    1.74    9
```

1.7 Kommentare

```
In [10]: # Kommentar in einer Zeile

         """
         Mehrzeilige
         Kommentare
         """

         print("Kein Kommentar")
```

```
Kein Kommentar
```

1.8 Listen

Es gibt fünf Array-ähnliche Typen in Python:
List, Tuple, Dictionary, Set, Frozenset
Listen werden über eckige Klammern `[]` oder über `list()` definiert:

```
In [11]: example_list = ["test", "cooltest", "again", "..."]
         empty_list = list()
         emptylist2 = []
```

Die Elemente einer Liste erhält man auch über eckige Klammern:

```
In [12]: example_list = ["test", "cooltest", "again", "..."]
         print(example_list[0])    # Element 0
         print(example_list[1:3])  # Element 1 und 2
         print(example_list[-1])   # Letztes Element
```

```
test
['cooltest', 'again']
...
```

Die Länge von Listen wird über `len()` ausgegeben:

```
In [13]: print(len(example_list))
```

```
4
```

Die wichtigsten **Listen-Operationen** sind im Folgenden zusammengefasst:

```
In [15]: my_list = ["test", "test", 3.141, False]
         print(my_list)

         my_list.append(3)
         print(my_list)

         my_list.insert(1, ["inserted", "list on", "2nd position"])
         print(my_list)

         my_list.remove("test")
         print(my_list)

         del my_list[-1]
         print(my_list)

         my_list[0] = "eins"
         print(my_list)
```

```
['test', 'test', 3.141, False]
['test', 'test', 3.141, False, 3]
['test', ['inserted', 'list on', '2nd position'], 'test', 3.141, False, 3]
[['inserted', 'list on', '2nd position'], 'test', 3.141, False, 3]
[['inserted', 'list on', '2nd position'], 'test', 3.141, False]
['eins', 'test', 3.141, False]
```

1.9 Tuple

In Python sind Tupel ähnlich zu Listen mit dem Unterschied, dass sie nicht veränderbar sind:

```
In [16]: my_tuple = ("one", "two three", "four")
         print(my_tuple)
         print(my_tuple[1])

('one', 'two three', 'four')
two three
```

1.10 Dictionaries

In Dictionaries haben die Elemente einen Namen statt einem Index. Man definiert sie über geschweifte Klammern { }:

```
In [17]: english_german = {"hello": "hallo",
                           "day": "Tag",
                           "car": "Auto"}

         print(english_german["car"])

Auto
```

Man kann sich sowohl die *keys* als auch die *values* einzeln als Liste ausgeben lassen:

```
In [18]: print(english_german.keys())
         print(english_german.values())

dict_keys(['hello', 'day', 'car'])
dict_values(['hallo', 'Tag', 'Auto'])
```

1.11 Sets

Mengen (*sets*) sind ungeordnet und jedes Element kommt nur einmal darin vor:

```
In [19]: my_set = set([1, 1, 4, 3, 3, 6, 1])
         print(my_set)
```

```
{1, 3, 4, 6}
```

Frozensets sind nicht veränderbare Mengen in Python:

```
In [20]: my_fset = frozenset([1,2,1])
         print(my_fset)
```

```
frozenset({1, 2})
```

1.12 Funktionen

Funktionen sind in Python wie folgt definiert:

```
In [21]: def my_function(par1, par2):
             result = par1 - par2
             return result

         print(my_function(4, 9))
```

```
-5
```

Parameter von Funktionen können auch *default*-Werte haben:

```
In [22]: def another_function(par1, par2=0):
             return par1 + par2

         print(another_function(5))
```

```
5
```

1.13 If/else-Abfragen

In Python gibt es selbstverständlich if/else-Abfragen, jedoch keine *switch-case*-Ausdrücke. Dafür kann *elif* verwendet werden:

```
In [23]: var = 10

         if type(var) != int:
             print("Keine ganze Zahl.")
         elif var < 5:
             print("Kleiner als 5.")
         elif var < 10:
             print("Kleiner als 10.")
         else:
             print("Mindestens 10.")
```

```
Mindestens 10.
```

1.14 Schleifen

In Python gibt es *while* und *for* Schleifen:

```
In [24]: var = 5
         while var > 1:
             print(var)
             var -= 1
```

```
5
4
3
2
```

Man kann die Schleifen mit *break* beenden und mit *continue* den aktuellen Durchlauf überspringen.

for-Schleifen sehen in Python wie folgt aus:

```
In [26]: for i in range(5):
             print(i)

         for j, x in enumerate(["A", "B", "Z"]):
             print(j, " ",x)
```

```
0
1
2
3
4
0   A
1   B
2   Z
```

for-Schleifen, die Listen erstellen sollen, können auch zusammengefasst werden. Das wird *list comprehension* genannt und sieht so aus:

```
In [27]: square_numbers = [i**2 for i in range(10)]
         print(square_numbers)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

1.15 Klassen

```
In [28]: class Vektor:
             """Hier wird die Klasse Vektor beschrieben."""

             def __init__(self, x=0, y=0):
```

```
            """Dies ist der Konstruktor."""
            self.x = float(x)
            self.y = float(y)

        def __str__(self):
            return "({0}, {1})".format(self.x, self.y)

        def __add__(self, other):
            return Vektor(self.x + other.x, self.y + other.y)

        def norm_squared(self):
            return self.x**2 + self.y**2

    vector1 = Vektor(2, -1)
    vector2 = Vektor(3, 1)

    print(vector1, "---", vector2.norm_squared(), "---", vector1 + vector2)
```

```
(2.0, -1.0) --- 10.0 --- (5.0, 0.0)
```

1.16 Wichtige Bibliotheken

Im Folgenden werden die wichtigsten Python-Bibliotheken vorgestellt: * Numpy * Matplotlib * Pandas * Scikit-Learn

1.16.1 Numpy

Numpy ist das wichtigste Paket für den Umgang mit groSSen Arrays (z.B. Matrizen). Da Numpy gröSStenteils in C geschrieben ist, ist es sehr schnell.

```
In [29]: import numpy as np

         # Mathematische Funktionen und Konstanten, z.B. Sinus und Pi
         print("sin(pi/2) = ", np.sin(np.pi/2))

         # Komplexe Zahlen
         print("e^(i*pi) = ", np.exp(1.0j*np.pi))
```

```
sin(pi/2) =  1.0
e^(i*pi) =  (-1+1.22464679915e-16j)
```

Numpy Arrays sind vergleichbar mit Listen, jedoch viel performanter mit groSSen Datenmenge.

```
In [30]: # List vs. Array
         some_list = [1, 2, 3]
         some_array = np.array([1, 2, 3])
```

```
         print(some_list)
         print(some_array)
```

```
[1, 2, 3]
[1 2 3]
```

```
In [31]: # Mehrdimensional
         print(np.array([[1,2], [3,4]]))
```

```
[[1 2]
 [3 4]]
```

```
In [34]: # Man kann "Dummy-Arrays" erzeugen:
         print(np.zeros((5, 5)))
```

```
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

Zufallszahlen können über `numpy.random` generiert werden:

```
In [36]: # Seed
         np.random.seed(1)

         # Fünf standardnormalverteilte Zufallszahlen
         np.random.normal(0, 1, 5)
```

```
Out[36]: array([ 1.62434536, -0.61175641, -0.52817175, -1.07296862,  0.86540763])
```

```
In [37]: # Drei gleichverteilte Zufallszahlen zwischen -1 und 1
         np.random.uniform(-1, 1, 3)
```

```
Out[37]: array([-0.16161097,  0.370439  , -0.5910955 ])
```

In diesem Beispiel erzeugen wir 16 ganzzahlige Zufallszahlen, die wir in eine Matrix *reshapen* und dann *slicen*:

```
In [38]: import numpy as np

         ndim_array = np.random.randint(0, 100, 16).reshape((4,4))
         print(ndim_array, "\n")

         # Erste Zeile
         print("Erste Zeile: ", ndim_array[0])
```

```
        # Dritte Spalte
        print("Dritte Spalte: ", ndim_array[:, 2])

        # Diagonale
        print("Diagonale: ", np.diag(ndim_array))
```

```
[[28 29 14 50]
 [68 87 87 94]
 [96 86 13  9]
 [ 7 63 61 22]]

Erste Zeile:  [28 29 14 50]
Dritte Spalte:  [14 87 13 61]
Diagonale:  [28 87 13 22]
```

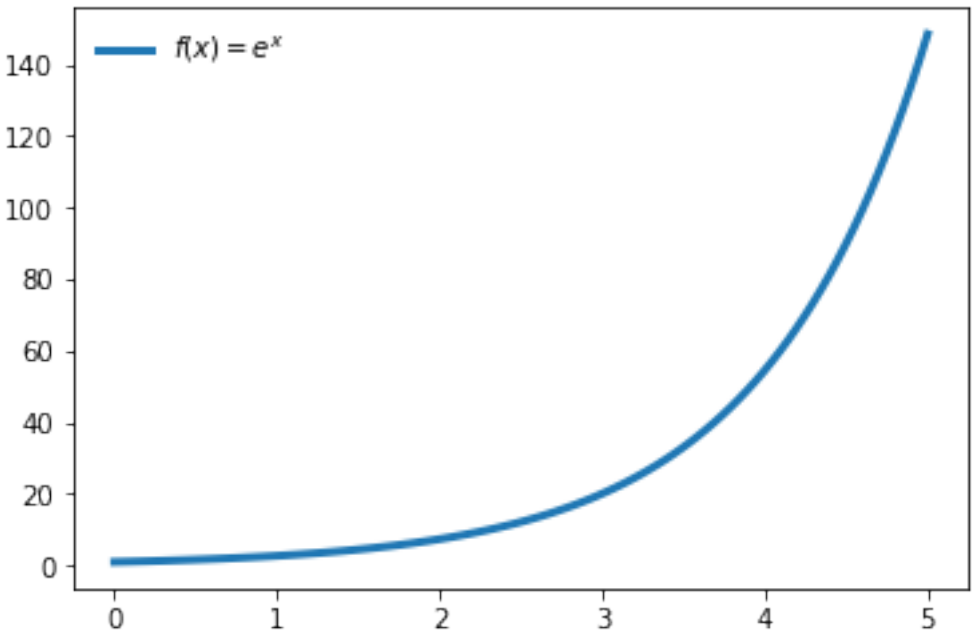
1.16.2 Matplotlib

Matplotlib ist das wichtigste Plot-Paket in Python.

```
In [39]: import matplotlib.pyplot as plt

        x = np.linspace(0, 5, 100)
        y = np.exp(x)

        plt.plot(x, y, linewidth=3, label="$f(x)=e^{x}$")
        plt.legend(loc="best", frameon=False)
        plt.show()
```



Scatter-Plots:

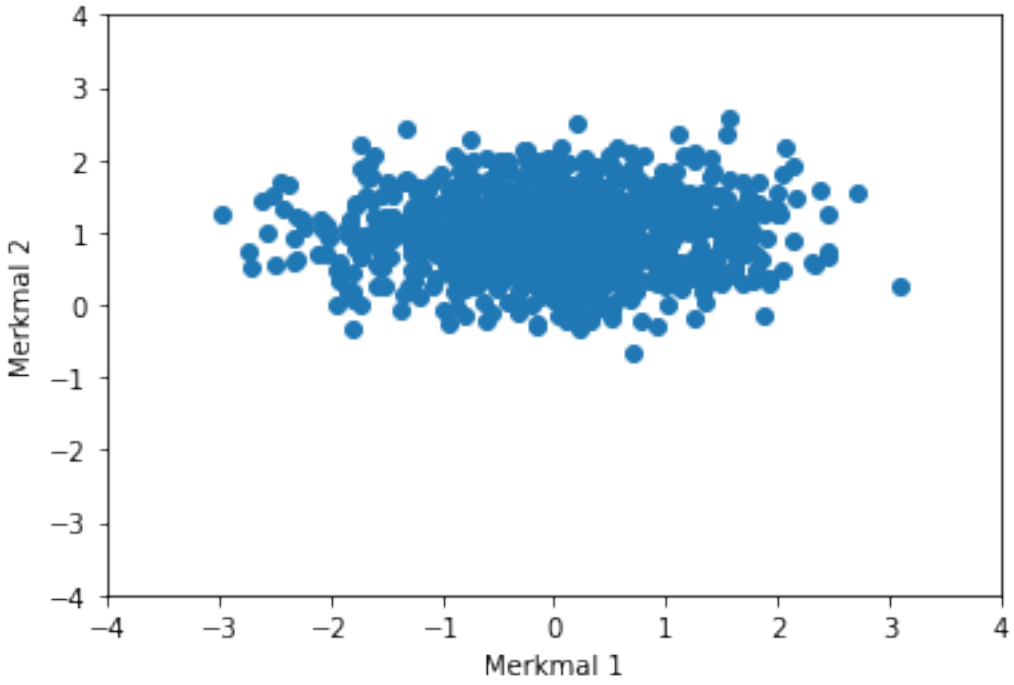
```
In [40]: merkmal_1 = np.random.normal(0, 1, 1000)
         merkmal_2 = np.random.normal(1, 0.5, 1000)

         # Scatterplot
         plt.scatter(merkmal_1, merkmal_2)

         # Einstellung der Achsen
         plt.xlim(-4, 4)
         plt.ylim(-4, 4)

         # Beschriftung der Achsen
         plt.xlabel("Merkmal 1")
         plt.ylabel("Merkmal 2")

         plt.show()
```



Histogramme:

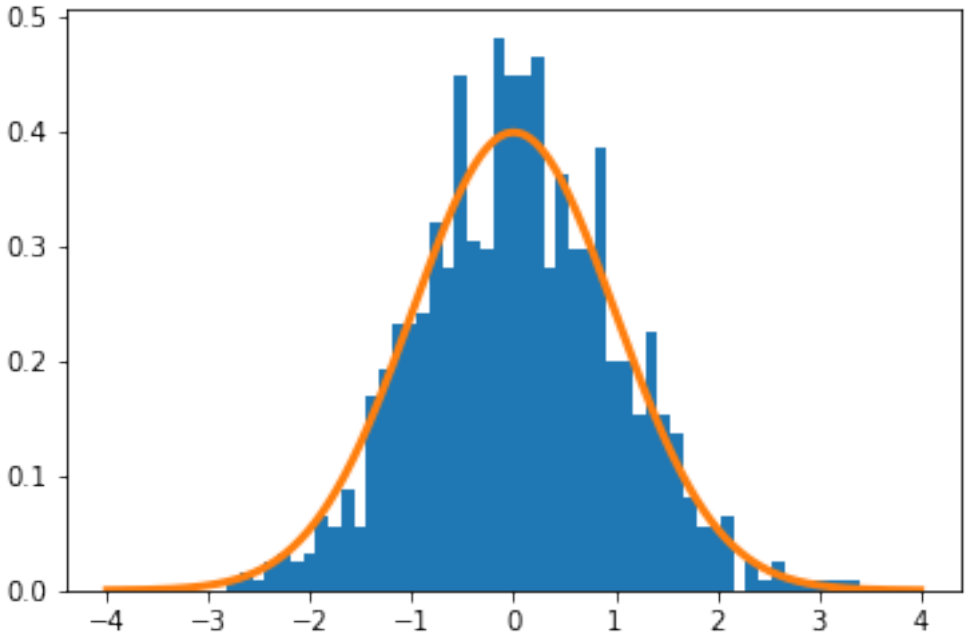
```
In [41]: data = np.random.normal(0, 1, 1000)

         # Histogramm plotten
         plt.hist(data, bins=50, normed=True)
```

```
# Kurve der Standardnormalverteilung plotten
x_grid = np.linspace(-4, 4, 1000)
y = [1/np.sqrt(2*np.pi)*np.exp(-x**2/2.0) for x in x_grid]

plt.plot(x_grid, y, linewidth=3)

plt.show()
```



1.16.3 Pandas

Pandas ist eine Bibliothek für die Datenanalyse. Hier finden Sie ein gutes Tutorial. Zitat der Pandas-Website:

> "Python has long been great for data munging and preparation, but less so for data analysis and modeling. pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R."

Zitat: http://pandas.pydata.org

Der für uns entscheidende Teil aus Pandas ist das `DataFrame`. Angenommen wir öffnen einen Datensatz aus einer CSV-Datei:

```
In [42]: # non-pandas code
         with open("baseball.csv", "r") as f:
             for _ in range(3):
```

```
        print(f.readline())

    # Quelle der Datei: http://seanlahman.com/baseball-archive/statistics/
```

```
playerID,birthYear,birthMonth,birthDay,birthCountry,birthState,birthCity,deathYear,deathMonth,d

aardsda01,1981,12,27,USA,CO,Denver,,,,,,,David,Aardsma,David Allan,215,75,R,R,2004-04-06,2015-0

aaronha01,1934,2,5,USA,AL,Mobile,,,,,,,Hank,Aaron,Henry Louis,180,72,R,R,1954-04-13,1976-10-03
```

```
In [44]: # pandas code
         import pandas as pd

         df = pd.read_csv("baseball.csv")
         df.tail(5)
```

Out[44]:

	playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	\
19100	zupofr01	1939.0	8.0	29.0	USA	CA	
19101	zuvelpa01	1958.0	10.0	31.0	USA	CA	
19102	zuverge01	1924.0	8.0	20.0	USA	MI	
19103	zwilldu01	1888.0	11.0	2.0	USA	MO	
19104	zychto01	1990.0	8.0	7.0	USA	IL	

	birthCity	deathYear	deathMonth	deathDay	...	nameLast	\
19100	San Francisco	2005.0	3.0	25.0	...	Zupo	
19101	San Mateo	NaN	NaN	NaN	...	Zuvella	
19102	Holland	2014.0	9.0	8.0	...	Zuverink	
19103	St. Louis	1978.0	3.0	27.0	...	Zwilling	
19104	Monee	NaN	NaN	NaN	...	Zych	

	nameGiven	weight	height	bats	throws	debut	finalGame	\
19100	Frank Joseph	182.0	71.0	L	R	1957-07-01	1961-05-09	
19101	Paul	173.0	72.0	R	R	1982-09-04	1991-05-02	
19102	George	195.0	76.0	R	R	1951-04-21	1959-06-15	
19103	Edward Harrison	160.0	66.0	L	L	1910-08-14	1916-07-12	
19104	Anthony Aaron	190.0	75.0	R	R	2015-09-04	2016-08-24	

	retroID	bbrefID
19100	zupof101	zupofr01
19101	zuvep001	zuvelpa01
19102	zuveg101	zuverge01
19103	zwild101	zwilldu01
19104	zycht001	zychto01

[5 rows x 24 columns]

Auch komplexere Anfragen sind möglich:

```
In [45]: df.loc[(df.birthState=="CA") & (df.birthYear>1990) & (df.throws=="L")][["nameGiven", "
```

```
Out[45]:              nameGiven   nameLast
         1815     Steven Joseph     Brault
         10492   Gregory Norman      Mahle
         12798  Steven Chandler      Okert
         13021       Henry Cole      Owens
         13295      Joc Russell   Pederson
         15908     Jonathan Lee  Singleton
         15933      Tyler Wayne     Skaggs
```

1.16.4 Scikit-Learn

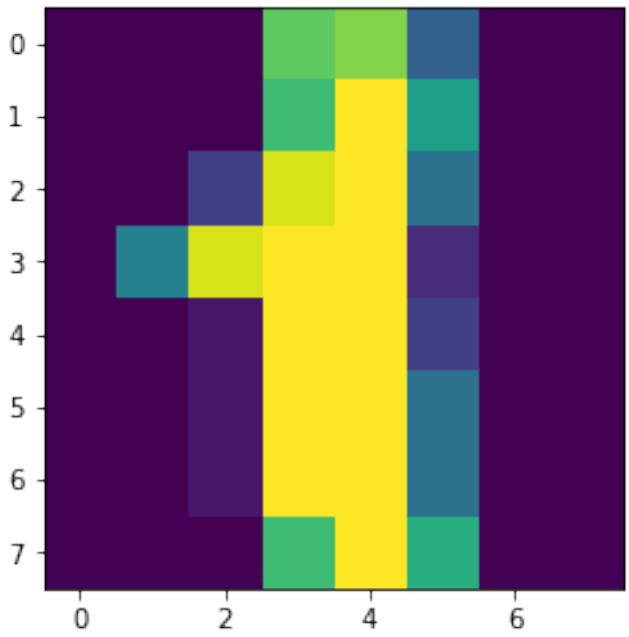
Sklearn ist die Standard-Python-Bibliothek für Machine Learning Anwendungen. Es enthält Methoden für: * Klassifikation * Clustering * Dimensionsreduzierung * Regression * Neuronale Netzwerke * ...

Beispiel: Handschrifterkennung

```
In [49]: # load dataset:
         from sklearn.datasets import load_digits

         digits = load_digits()

         #print(digits.data[:2])
         plt.imshow(digits.images[1])
         plt.show()
```



```
In [50]: # split dataset
         from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_s

         print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(179, 64) (1618, 64) (179,) (1618,)
```

```
In [51]: # train classifier
         from sklearn.ensemble import RandomForestClassifier

         model_rf = RandomForestClassifier(n_estimators=10, n_jobs=4, random_state=0)

         model_rf.fit(X_train, y_train)
```

```
Out[51]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=4,
                     oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
In [52]: # evaluate classifier
         from sklearn.metrics import accuracy_score

         y_pred = model_rf.predict(X_test)

         print("Accuracy score = ", accuracy_score(y_test, y_pred))
```

```
Accuracy score =  0.792954264524
```

2 Und jetzt: ausprobieren!

Vielen Dank an dieser Stelle an Timothy Gebhard für seine Hilfe bei diesem Tutorial.