ETH zürich

Departement Informatik

Programming and Problem-Solving

`numpy`, `matplotlib`, `pandas`

Dennis Komm

Spring 2021 – April 22, 2021

Lists

Advanced Concepts

Listen

So far

- Initializing a list: `x = []` or `x = [1, 4, 8]`
- Initializing a list with ten zeros: `x = [0] * 10`

- Appending elements: `x.append(3)`
- Merging lists: `x = x + y` or `x = x + [5, 7, 9]`

- Accessing (and removing) the first element: `z = x.pop(0)`
- Accessing (and removing) the last element: `z = x.pop()`
- Accessing `i`th element: `z = x[i]`

List Comprehensions

Now: List Comprehensions to initialize...

- a list of the first ten natural numbers:

```
x = [i for i in range(0, 10)]
```

- a list of the first ten even numbers:

```
x = [i for i in range(0, 20, 2)]
```

- a list of the squares of the first ten natural numbers:

```
x = [i * i for i in range(0, 10)]
```

- a list of the squares of `[8, 19, 71, 101]`:

```
x = [i * i for i in [8, 19, 71, 101]]
```

List Comprehensions

[⟨Expression depending on variable `i`⟩ `for i in` ⟨list⟩]

[⟨Expression depending on variable `i`⟩ `for i in range(...)`]

Filter

[⟨Expression depending on variable `i`⟩ `for i in` ⟨list⟩ `if` ⟨Condition⟩]

- List of all numbers from `[8, 60, 3, 19, 21]` that are larger than 8:

```
x = [i for i in [8, 60, 3, 19, 21] if i > 8]
```

- List of all numbers from `[9, 6, 10, 19]` that are divisible by 5:

```
y = [9, 6, 10, 19]
x = [i for i in y if i % 5 == 0]
```


Exercise – List Comprehensions

Initialize a list that

- contains all prime numbers between 1 and 1000
- uses the function `primetest` and list comprehensions

[⟨Expression depending on variable `i`⟩ `for i in range(...) if` ⟨Condition⟩]

List Comprehensions

```
from math import sqrt

def primetest(x):
    if x < 2 or (x > 2 and x % 2 == 0):
        return False
    d = 3
    while d <= sqrt(x):
        if x % d == 0:
            return False
        d += 2
    return True

y = [i for i in range(1001) if primetest(i)]
```


Accessing Elements

- Accessing element at position 0: `x[0]`
- Accessing element at last position: `x[len(x) - 1]`
- Accessing element at last position: `x[-1]`
- Accessing sublist from positions 4 to 9: `z = x[4:10]`
- Accessing sublist from position 5 `z = x[5:]`
- Accessing sublist to position 3 `z = x[:4]`

Namespaces

Namespaces

So far

- Including own / existing modules
- Square root function from `math`

```
from math import sqrt
```

```
from math import *
```

Problem if different modules use same function names

- Use namespaces
- This gives content of module a unique name

```
import math as mymath

print(mymath.sqrt(9))
```


The Modules `numpy` and `matplotlib`

`numpy` and `matplotlib`

Two modules are frequently used in a scientific context

- `numpy` and `matplotlib`
- They allow a functionality similar to MATLAB

`numpy`

- Calculations with vectors and matrices
- Numerical methods
- Documentation: `https://numpy.org/doc/`

`matplotlib`

- Data visualization (Plots)
- Documentation: `https://matplotlib.org/contents.html`

The Module numpy

The Module numpy

numpy builds the foundation for many other scientific modules

Focus on efficient processing of large vectors and matrices

- It contains its own data structures, e.g., numpy arrays
- These work similar to Python lists
- numpy arrays are faster
- numpy arrays allow for more operations

The Module numpy

```
import numpy as np
```

- Convert Python list into numpy array

```
x = np.array([1, 3, 4])
```

- This also works for more dimensions

```
y = np.array([[1, 3, 4], [6, 8, 1], [0, 9, 4]])
```

- numpy arrays can be added and multiplied

```
print(x + y)
```

```
print(x * y)
```


The Module numpy

Large range of functions

- Linear algebra (Submodule)

```
import numpy as np
import numpy.linalg as npla

a = np.array([[5, 3, 0], [1, 2, 0], [0, 2, 11]])
b = np.array([4, 8, 1])

x = npla.solve(a, b)
```

- Statistics
- Interpolation (e.g., least square method)
- ...

The Module `matplotlib`

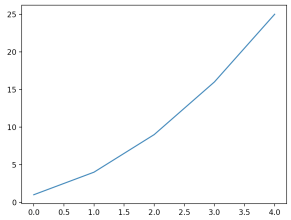
The Module `matplotlib`

Module to generate plots

- Data visualization
- Submodule `matplotlib.pyplot` allows usage analogously to MATLAB
- Data given by, e.g., Python lists or `numpy` arrays

```
import numpy as np
import matplotlib.pyplot as plt

plt.plot([1, 4, 9, 16, 25])
plt.show()
```



The Module `matplotlib` – Code-Expert

- The function `show` cannot be used in Code-Expert
- Instead, we save the plot using the function `savefig()`

```
import numpy as np
import matplotlib.pyplot as plt

plt.plot([1, 4, 9, 16, 25])
plt.savefig("cx_out/out.png")
```

- Plot has to be saved in `cx_out`
- Display under “Files”
- **Within these slides, we use `show()`**

The Module `matplotlib`

- Specifying both x- and y-values:

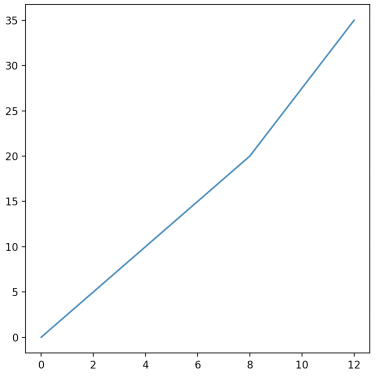
```
plt.plot([0, 4, 8, 12], [0, 10, 20, 35])
plt.show()
```

- Using `numpy` arrays:

```
x = np.array([0, 4, 8, 12])
y = np.array([0, 10, 20, 35])
plt.plot(x, y)
plt.show()
```

- Using `arange()` instead of `range()`:

```
x = np.arange(0, 13, 4)
y = np.array([0, 10, 20, 35])
plt.plot(x, y)
plt.show()
```



The Module `matplotlib`

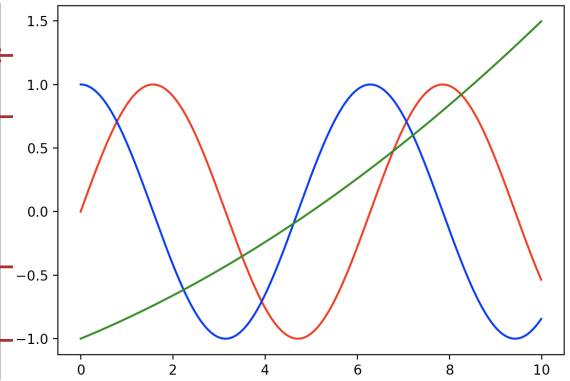
`plot(⟨x-values⟩, ⟨y-values⟩, ⟨list of options⟩)`

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10.01, 0.01)
f1 = np.sin(x)
f2 = np.cos(x)
f3 = 0.01 * x**2 + 0.15 * x - 1

plt.plot(x, f1, color="red")
plt.plot(x, f2, color="blue")
plt.plot(x, f3, color="green")

plt.show()
```



The Module `matplotlib`

Options

- color, line style, line width, dots instead of lines, . . .
- See documentation

Labeling axes

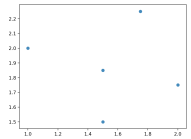
- `plt.xlabel()`
- `plt.ylabel()`

Animations

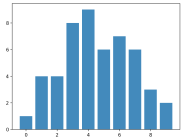
- Displaying plot only shortly with `plt.pause()` instead of `plt.plot()`
- Removing old plot with `plt.close()`
- Submodule `matplotlib.animation` for more professional animations
- Also see documentation

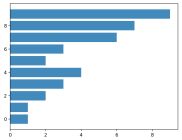
The Module `matplotlib`

```
x = np.array([1, 2, 1.5, 1.75, 1.5])
y = np.array([2, 1.75, 1.5, 2.25, 1.85])
plt.scatter(x, y)
plt.show()
```



```
x = np.arange(0, 10)
y = np.array([1, 4, 4, 8, 9, 6, 7, 6, 3, 2])
plt.bar(x, y)
plt.show()
```





```
x = np.arange(0, 10)
y = np.array([1, 1, 2, 3, 4, 2, 3, 6, 7, 9])
plt.barh(x, y)
plt.show()
```


Animated Bubblesort

```
import matplotlib.pyplot as plt

def bubblesort(data):
    n = len(data)
    x = range(len(data))
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            plt.bar(x, data)
            plt.pause(0.001)
            plt.close()

            if data[i] > data[i+1]:
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return data

print(bubblesort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```

Animated Bubblesort – Code-Expert

```
import matplotlib.pyplot as plt

def bubblesort(data):
    n = len(data)
    x = range(len(data))
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            plt.bar(x, data)
            plt.savefig("cx_out/out.png")
            input("Weiter mit beliebiger Taste")
            plt.close()
            if data[i] > data[i+1]:
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return data

print(bubblesort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```


The Module `matplotlib`

Visualizing the complexity of Bubblesort

Exercise – Complexity of Bubblesort

- Copy Bubblesort
- Use a variable `counter` to count the number of comparisons
- Return this value using `return`
- Run this algorithm on backward-sorted lists of lengths from 10 to 200
- Save these values in a list and plot it

Complexity of Bubblesort

```
def bubblesort(data):
    n = len(data)
    counter = 0
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            counter += 1if data[i]
> data[i+1]:
            if data[i] > data[i+1]:
counter += 1
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return counter
```

Count permutations

Count comparisons

```
values = []
for i in range(10, 201):
    data = np.arange(i, 0, -1)
    values.append(bubblesort(data))

plt.plot(values)
plt.show()
```

Complexity of Bubblesort

Worst Case

```
for i in range(10, 201):
    worst_data = np.arange(i, 0, -1)
    worst_values.append(bubblesort(worst_data))
```

Best Case

```
for i in range(10, 201):
    best_data = np.arange(1, i+1, 1)
    best_values.append(bubblesort(best_data))
```

Average Case

```
for i in range(10, 201):
    avg_data = np.random.randint(i, size=i)
    avg_values.append(bubblesort(avg_data))
```

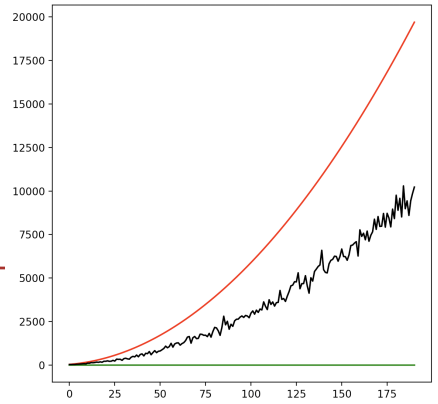

Complexity of Bubblesort

Average Case (Vertauschungen)

```
for i in range(10, 201):
    worst_data = np.arange(i, 0, -1)
    best_data = np.arange(1, i+1, 1)
    avg_data = np.random.randint(i, size=i)
    worst_values.append(bubblesort(worst_data))
    best_values.append(bubblesort(best_data))
    avg_values.append(bubblesort(avg_data))

plt.plot(worst_values, color="red")
plt.plot(best_values, color="green")
plt.plot(avg_values, color="black")

plt.show()
```



The module pandas

The Module pandas

pandas

- Processing of large sets of data
- Allows a functionality similar to Excel
- Documentation: https://pandas.pydata.org/pandas-docs/stable/

- **So far** Reading in and processing CSV file “manually”
- pandas contains data structures and functions for this

The Module pandas

- Import pandas analogously to numpy and matplotlib

```
import pandas as pd
```

- Read in CSV file and store it in a special data type pandas dataframe (instead of Python list or numpy array)

```
data = pd.read_csv("daten.csv")
```

- Files in Excel format can be read in analogously

```
data = pd.read_excel("daten.xlsx")
```


Air Measurements using pandas

Exercise – Air Measurements

Air measurements

- In Code Expert, you find a file ugz_luftqualitaetsmessungen_seit-2012.csv
- Read in the CSV file and output its content
- To this end, use read_csv() and print()

Air Measurements

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
print(data)
```

- Accessing individually cells using data.iloc
- Same functionality as lists

```
print(data.iloc[5])          Output line 5
print(data.iloc[0:10])       Output lines 0 to 9
print(data.head(3))          Output lines 0 to 2
print(data.iloc[8, 0])       Output line 8, column 0
```

Reading in and Processing CSV Files

Extract data

- Numerical data starts from line 5
- We are only interested in the first 3 columns
- We want to change column names

Selection from line 5 and columns 0 to 2

Rename columns

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
newdata = data.iloc[5:, 0:3]
newdata = newdata.rename(columns={"Zürich Stampfenbachstrasse": "SO2", \
                                  "Zürich Stampfenbachstrasse.1": "CO"})
newdata.to_csv("messungen.csv")
```


Reading in and Processing CSV Files

Accessing data using the column names

- Output all column names as list

```
print(data.columns)
```

- Output column “Datum”

```
print(data["Datum"])
```

- Output column “Zürich Stampfenbachstrasse – Kohlenmonoxid”

```
print(data["Zürich Stampfenbachstrasse.1"])
```


Reading in and Processing CSV Files

Filtering data

- Use `loc` instead of `iloc` in order to specify conditions

```
print(data.loc[data["Datum"] == "2014-12-19"])
```

- Combination of different Boolean expressions
  - Parentheses around single expressions
  - `&` instead of `and`
  - `|` instead of `or`
  - `~` instead of `not`

```
print(data.loc[(data["Datum"] == "2014-12-19") \
                                | (data["Datum"] == "2014-12-20")])
```


Reading in and Processing CSV Files

Filtering data

- Convert strings to rational numbers (`float`)

```
newdata["SO2"] = newdata["SO2"].astype(float)
newdata["CO"] = newdata["CO"].astype(float)
```

- Use relation operators to filter

```
print(newdata.loc[newdata["SO2"] > 0.1])
```

- Combine different Boolean expressions

```
print(newdata.loc[(newdata["SO2"] > 0.1) & (newdata["SO2"] < 0.4)])
```

- Choose columns with second argument

```
print(newdata.loc[newdata["SO2"] > 0.2, "Datum"])
```

Exercise – Air Measurements

Air measurements

- Extract all CO entries from `newdata` for which the SO2 value is smaller than 0.1 or larger than 0.25
- Convert the CO entries into a Python list using `list()`
- Plot the values using `matplotlib`

Reading in CSV File

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")

newdata = data.iloc[5:, 0:3]
newdata = newdata.rename(columns={"Zürich Stampfenbachstrasse": "SO2", \
                                  "Zürich Stampfenbachstrasse.1": "CO"})

newdata["SO2"] = newdata["SO2"].astype(float)
newdata["CO"] = newdata["CO"].astype(float)

newdata = newdata.loc[(newdata["SO2"] < 0.1) | (newdata["SO2"] > 0.25), "CO"]
datalist = list(newdata)
plt.plot(datalist)
plt.show()
```


Pandas

Further Functionality

Further Functionality

- Delete columns

```
del data["Column"]
```

- Add columns

```
data["Sum"] = data["Column 1"] + data["Column 2"]
```

- Sort data

```
data = data.sort_values("Column")
```

- ...