Computer Programming I

Lecture 4: Dictionaries and CSV-files

Hans Karlsson

Dictionary

Dictionary (type **dict**)

A collection of *key-value* pairs.

Dictionary

Dictionary (type **dict**)

A collection of *key-value* pairs.

Ex 1:	Ex 2:	Ex 3:		Ex 4:
Anna 070-1122334	A .-	Sverige	SEK	Eva 23
Pelle 0732255889	B -…	UK	£	Carl 19
Olle 0708877441	C -.-	USA	$	Svea 19

Dictionary

Dictionary (type **dict**)

A collection of *key-value* pairs.

Ex 1:	Ex 2:	Ex 3:	Ex 4:
Anna 070-1122334	A .-	Sverige SEK	Eva 23
Pelle 0732255889	B -…	UK £	Carl 19
Olle 0708877441	C -.-	USA $	Svea 19

Syntax Python: Curly brackets { and }

friends = {'Anna': '070-1122334', 'Pelle': '0732255889', 'Olle': '0708877441'}

data = {'Eva':23, 'koordinater': (2,3), 44:[1,9,3]}

{key:value, key:value, …} search key coupled to value

Key must be unique and cannot be changed (immutable).

Value can be anything and can be changed

Dictionary

```
print(friends)  # {’Anna’: ’070-1122334’, ’Pelle’: ’0732255889’, ’Olle’: ’0708877441’}

print(friends[’Olle’])  # 0708877441
print(friends.get('Anna’)) # 070-1122334

friends['Olle']='0739966551’ # Change a value
friends['Nisse'] = '0704455772’  # Add a post

del friends[’Anna’] # remove Anna
print(friends.pop('Pelle’))  # remove Pelle, returns the value 0732255889

if 'Gun' in friends:
    print('Gun is my friend')
else:
    friends['Gun']='0705566773’ # add Gun
list_of_friends = list(friends) #
```

Dictionary

```
#Print the list
print('\nPrint the keys:’) # or:
for x in friends.keys():     # for i in friends:
    print(x)                 #       print(i)

print('\nPrint the values:')
for z in friends.values():
    print(z)

print('\nPrint the pairs:’)   # or:
for a, b in friends.items():  # for a in friends.items():
    print(a, b)               #     print(a)
```

What is printed?

```
code = {chr(i):i  for  i  in range(65,91)}
print(code)
```

Collections and sequences in Python

Certain types of objects can be itererated over (looped), ex 1:

```
total = 0
for tal in [3, -4, 7, 0 ,1]:
        total = total + tal
```

The sequence after ”*in*” must be a *iterable*, i.e. to be able to loop over and that returns one element.

Collections and sequences in Python

Collections are data structures with related elements.
Pythons built in collections and sequences makes it possible to store and access data in a simple way. Sequences are iterable.

Ordered collections, sequences: you can reach the elements via the **index**.

Lists:

li = [3, -4, 7, 0, 1]

Strings: (?)

namn = ’Anna-Stina, 21’

Tuples:

min_t = (15, ’Abba’, 21, ’Bethoven’)

```
li = [3, -4, 7, 0, 1]
print(li[1]) # -4
li[0] = 8
print(li)    # [8, -4, 7, 0, 1]
```

Collections are ”iterables” in Python

Exemple of Pythons built in non-sequence collections:

dictionary

```
pris = {'äpplen': 12, 'bananer': 14, 'citroner': 20}
# print(pris[0]) dosen’t work! Dictionaries are unordered.
print(pris[’äpplen’]) # 12
for e in pris.items(): # e will succesivey take the value of each key-value pairs
    print(e)
```

”Python’s dictionary is a shining star among its data structures; it is compact, fast, versatile, and extremely useful. It can be used to create a wide variety of mappings.”

You must differ between:

Mutable (can change):

Lists:

li = [3, -4, 7, 0, 1]

li[0] = 8 # [8, -4, 7, 0, 1]

Dictionaries:

pris = {'äpplen': 12, 'bananer': 14, 'citroner': 20}

pris['bananer'] = 15

Immutable (cannot change):

Strings:

namn = ’Anna-Stina, 21’

Tuples:

min_t = (15, ’Abba’, 21, ’Bethoven’)

Note! Key immutable; value mutable

Text in Python

- Strings are ’imutable’, they cannot change

  Ex: namn[1]=’a’ Wrong!
- The + and *operators creates *new* strings in the memory.
- Compare: a new string is created and written out:

```
print(namn.replace('a', 'i’)) # Svei Lundmirk–Grinström
print(namn) # Svea Lundmark–Granström
```

- Compare: a new string is created , namn now point to the new string:

```
namn = namn.replace(‘a’, ‘i’)
print(namn) # Svei Lundmirk–Grinström
```

The inbuilt function enumerate()

Gives both the index and value when you iterate through a sequence

```
town = ’Hjo’ #stäng
print('Index   Letter')
for i, v in enumerate(town):
    print(f'{i}{v:>7}')
```

Ger:

Index	Letter
0	H
1	j
2	o

Two variables.
First: a counter
Second element
Enumerate adds to the counter

```
animals = ('dog','cat','bird’) #tuple
print(list(enumerate(animals,10)))
```

Gives:

[(10, 'dog'), (11, 'cat'), (12, 'bird’)]

```
for i in enumerate(animals):
    print(i)
```

Gives:

(0, 'dog')
(1, 'cat')
(2, 'bird')

Data handling

So far : Input from the keyboard and printing on the screen:

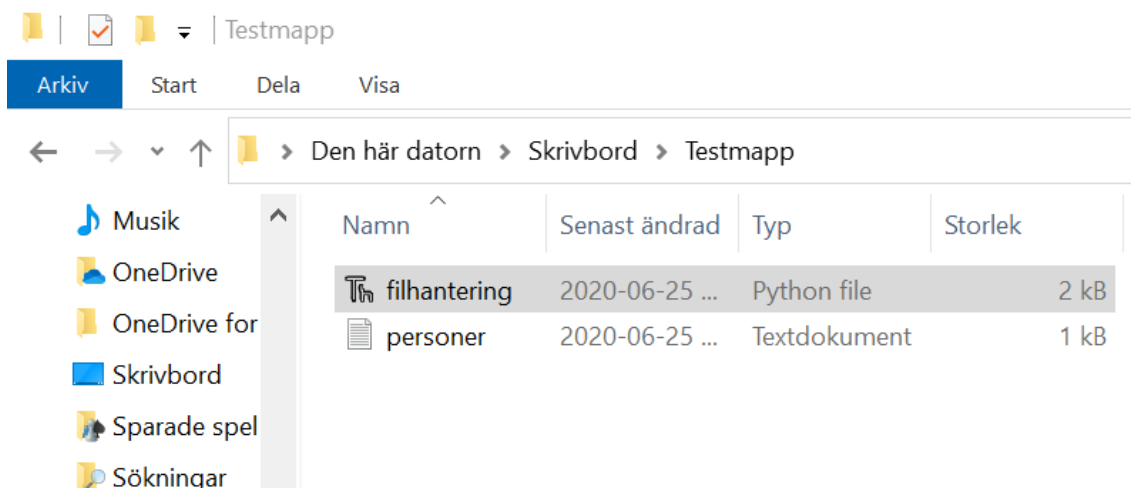
```
li = []
for i in range(4):
    li.append(int(input('Your text: ')))
```

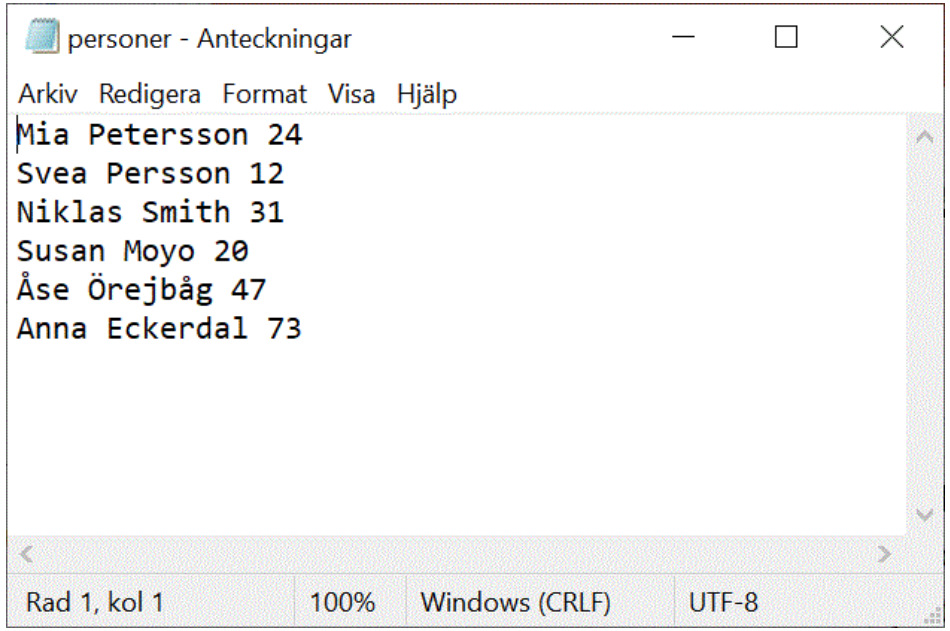
- We want the computer to read and process large amounts of data.

Data handling

Read from text-fil

Suppose there is a file with text, t ex personer.txt, in the same folder as your Python-program. Filformate textfile.





File handling

Write a Python programme that reads from the text file and prints out the content in the file.

Write a Python code that:

– Creates a file object

– Asks the file object to read from the file

- One row at a time

- The whole file

– Prints the content on the screen

File handling

```
filnamn = input(’Give the name of the filen: ’) # t ex personer.txt
fi = open(filnamn, 'r', encoding='utf-8’) # fi is an object. 'r’: Open for reading,
                                          # encoding  utf-8 (eller 'latin-1’) for åäö.
for rad in fi: # file objects are iterable. Read one row at a time.
    print(rad, end='') # Without end=’’ there will be 2 new lines
# or: print(fil.read()) #Reads the whole file, returns one string
fi.close() # Remeber to close the file

# open(), print() och input() built in functions
```

Files

Write to file

```
utfil = input(Name of the file to be created: ’)
fu = open(utfil, 'w’) #fu referers to utfil. ’w’: write over existing file or create new file
# fu = open(utfil, ’a’) Append to existing file (a för append)
# fu = open(utfil, ’x’) Create new file . Gives warnin gif the file already exists
print(’Add text, finnish with 0:’)
text = input(’Write a row, finnish with 0:’)
while text.casefold() !=’stop':
    fu.write(text+'\n')
    text = input(’Add an extra line:')
fi.close()
```

Files

Better to use ’with’, *context manager.* Opened files are automatically closed.

with open(n1, mode) as f1, open(n1, mode) as f2, …

```
readfile = input(’Give name of file: ')
writefile = input(’Give name of file to write to: ’)
with open(readfile, 'r') as f1, open(writefile, 'w') as f2:
    for person in f1: #Read one row at a time from the file that f1 refers to
        if 'Niklas' in person:
            continue # Skip Niklas
        f2.write(person) # Write on the row on the file that f2 refers to
```

Files

for r in f1: # Reads one row at a time to r

f1.read() # Reads the whole file and returns a string

f1.readline() # Reads one row. Returns ’ ’ at the end of the file

f1.read(t) # Reads t entries. Returns ’ ’ at the end of the file

f1.readlines() # Reads all the rows. Gives a lit of rows.

f2.write() # Writes a string to a file

f2.writelines() # Writes a list with one element per row.

csv-data

- Python has the module csv to read and write csv-data, *comma separated values*. The csv-format is commonly used for spreadsheets and data bases.

Example: the file people.csv:

Content:

```
No, Name, country
1, Alex, USA
2, Erik, Sweden
3, Cheng, China
```

Python code reades the csv file and prints the content:

=>

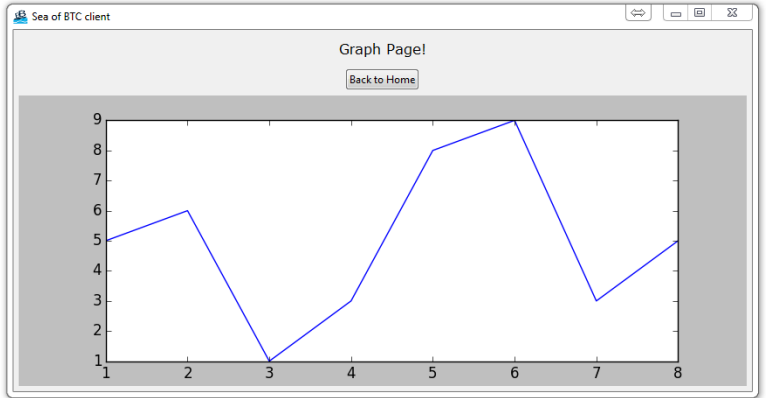
```
['No', ' Name', ' country']
['1', ' Alex', ' USA']
['2', ' Erik', ' Sweden']
['3', ' Cheng', ' China']
```

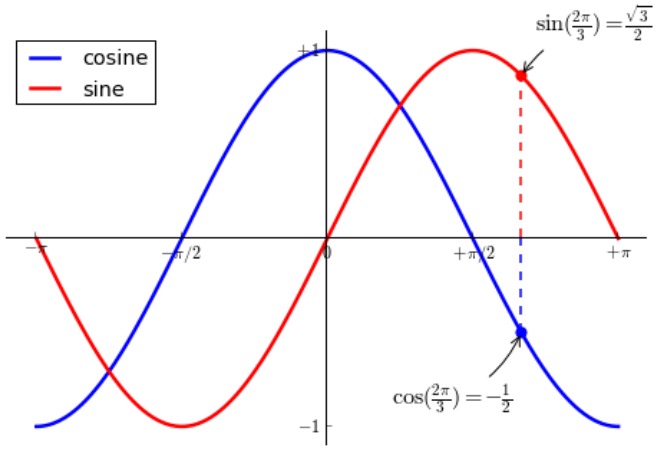
```
import csv

with open('people.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        print(row)
```

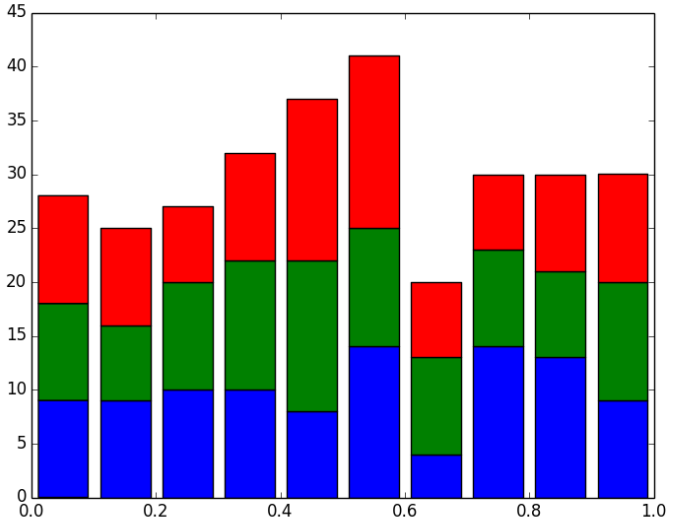
```
# read the file as a csv file.
# the reader-object itererats over each
# row in the file, and  returns ***lists***, one per
# row, where the elements are strings.
# It's the comma that separates the data!
```

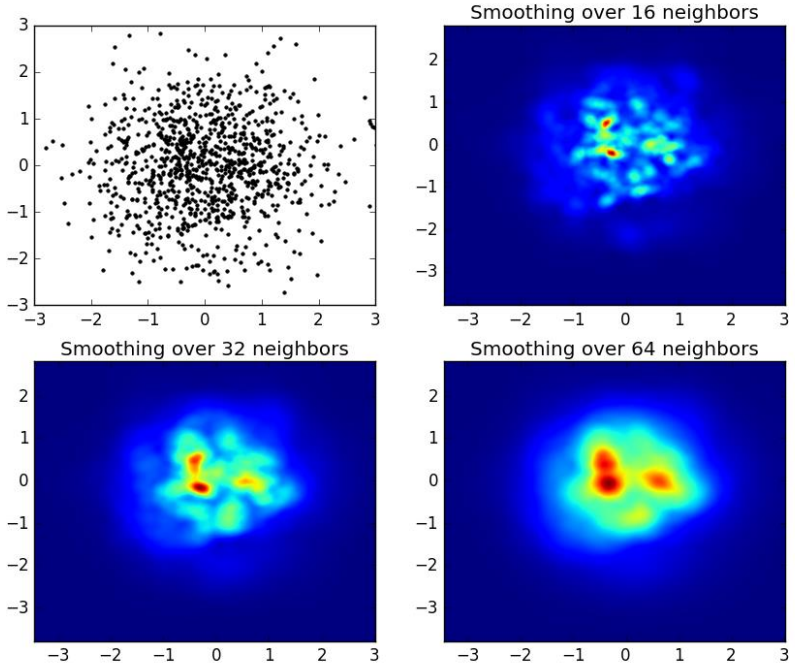
Datavisualisation with Python: matplotlib

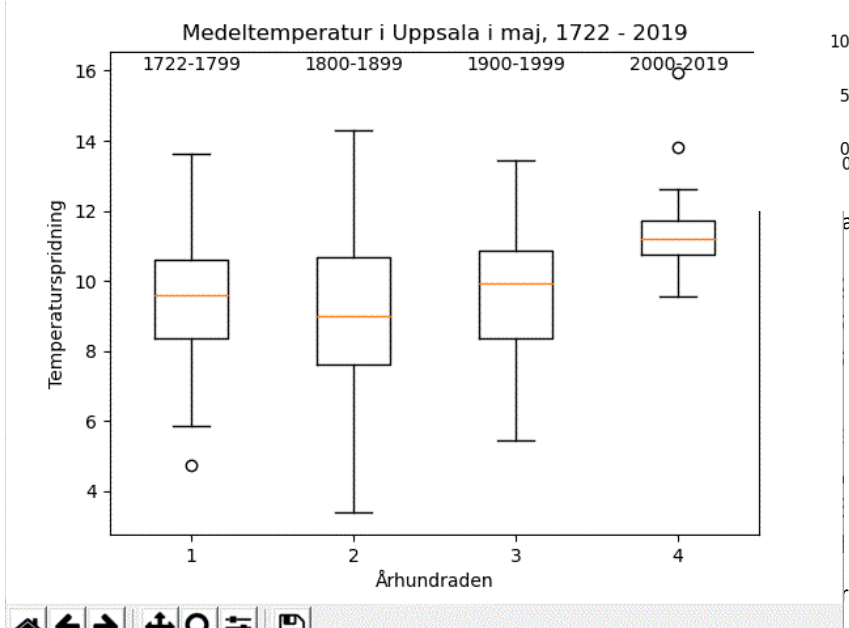
"Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python." https://matplotlib.org/











are licensieras enligt CC BY-SA

ras enligt CC BY-SA-NC

Datavisualisation with matplotlib

import matplotlib.pyplot as plt # pyplot is a module in the package matplotlib

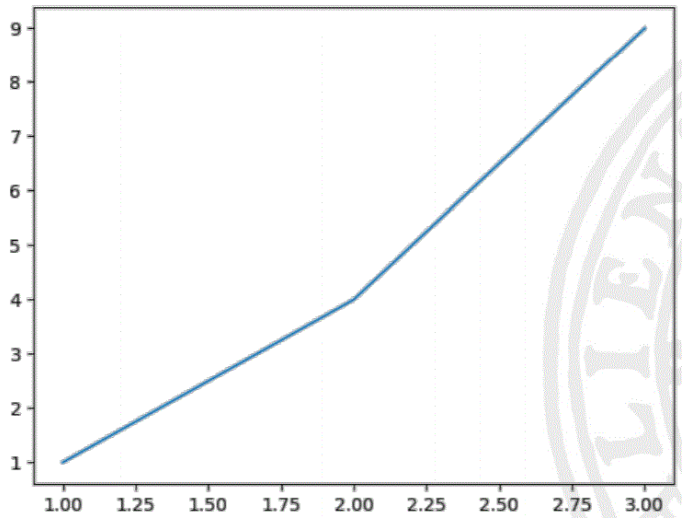
#Draw lines and points:

plt.plot([1,2,3],[1,4,9],"-") # a) line between (1,1), (2,4), (3,9)

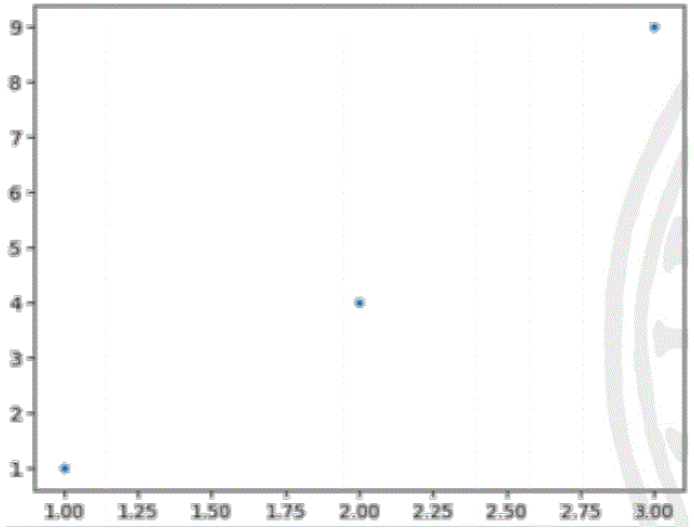
plt.plot([1,2,3],[1,4,9],".") # b) draw points

plt.show()

a)

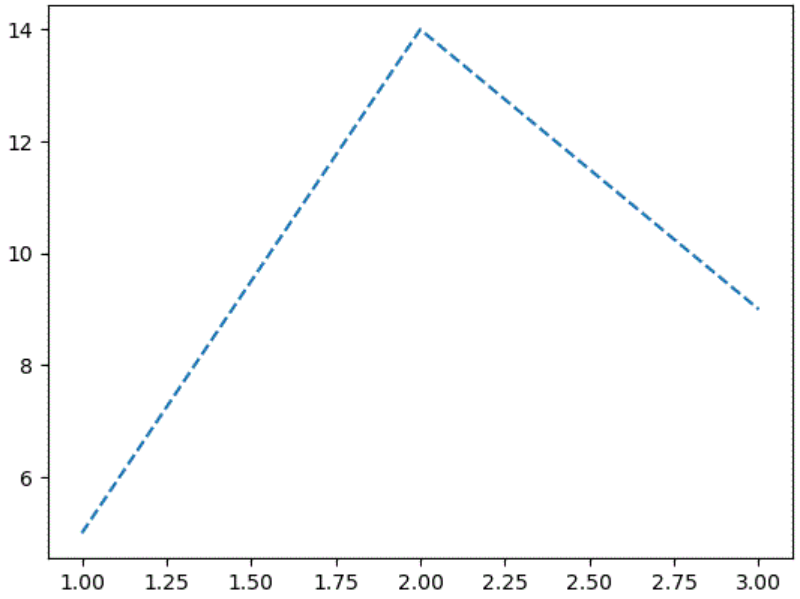


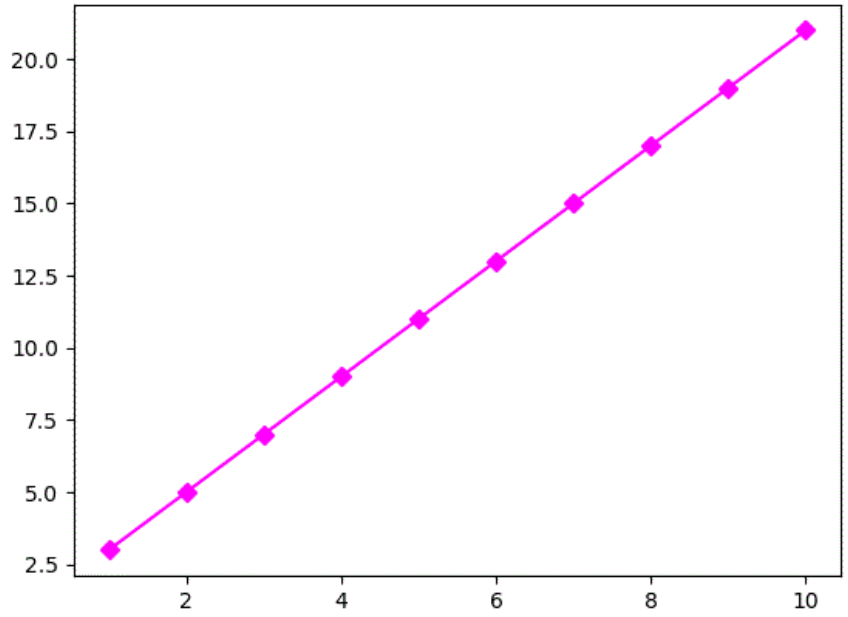
b)

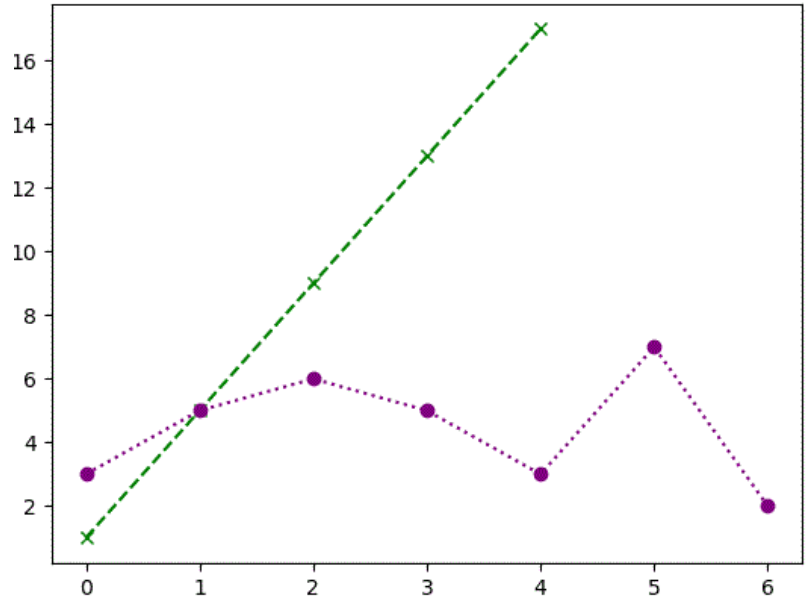


Datavisualisation with matplotlib

Ex1:

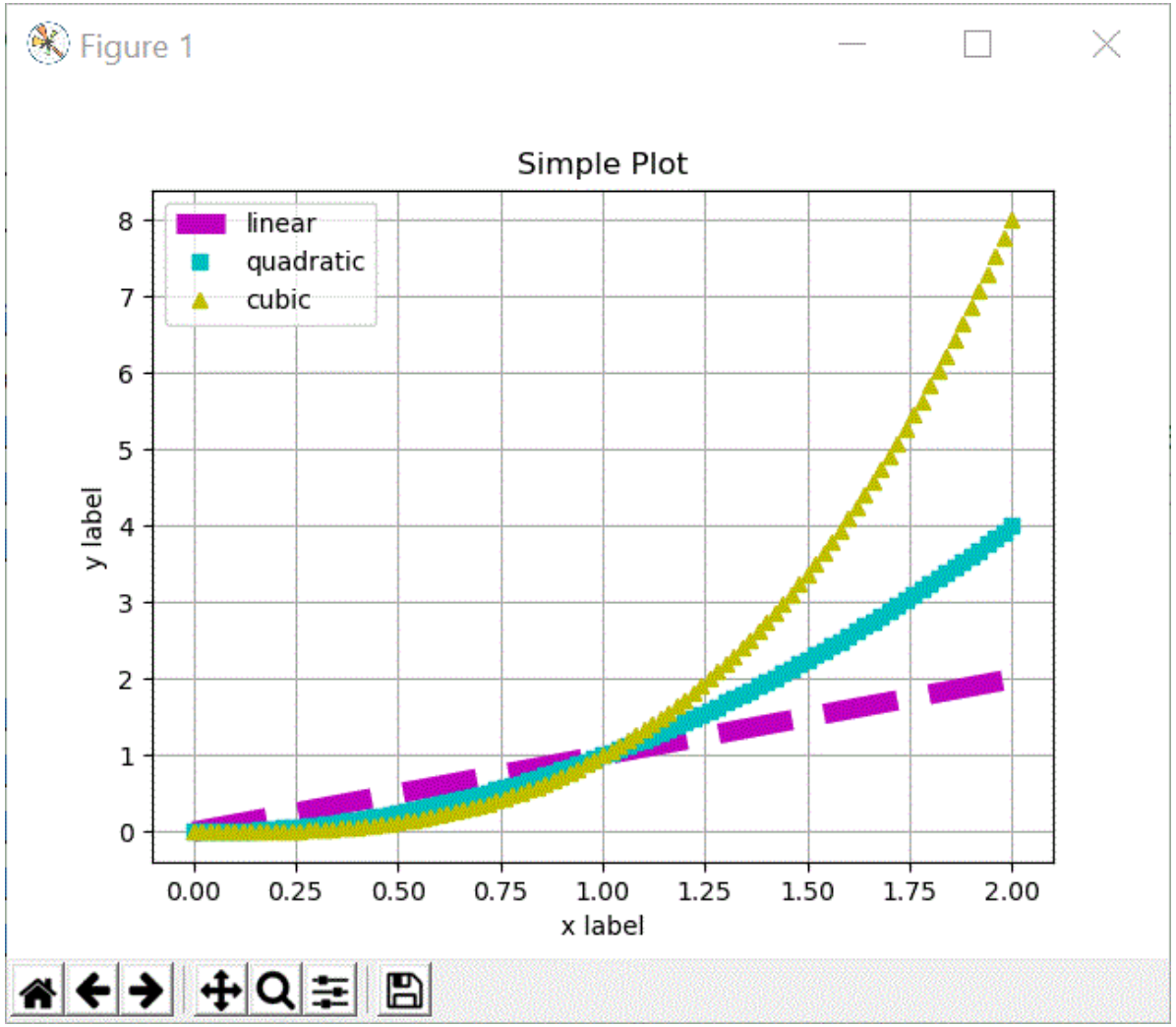






Datavisualisation with matplotlib

Ex2:



Datavisualisation with matplotlib

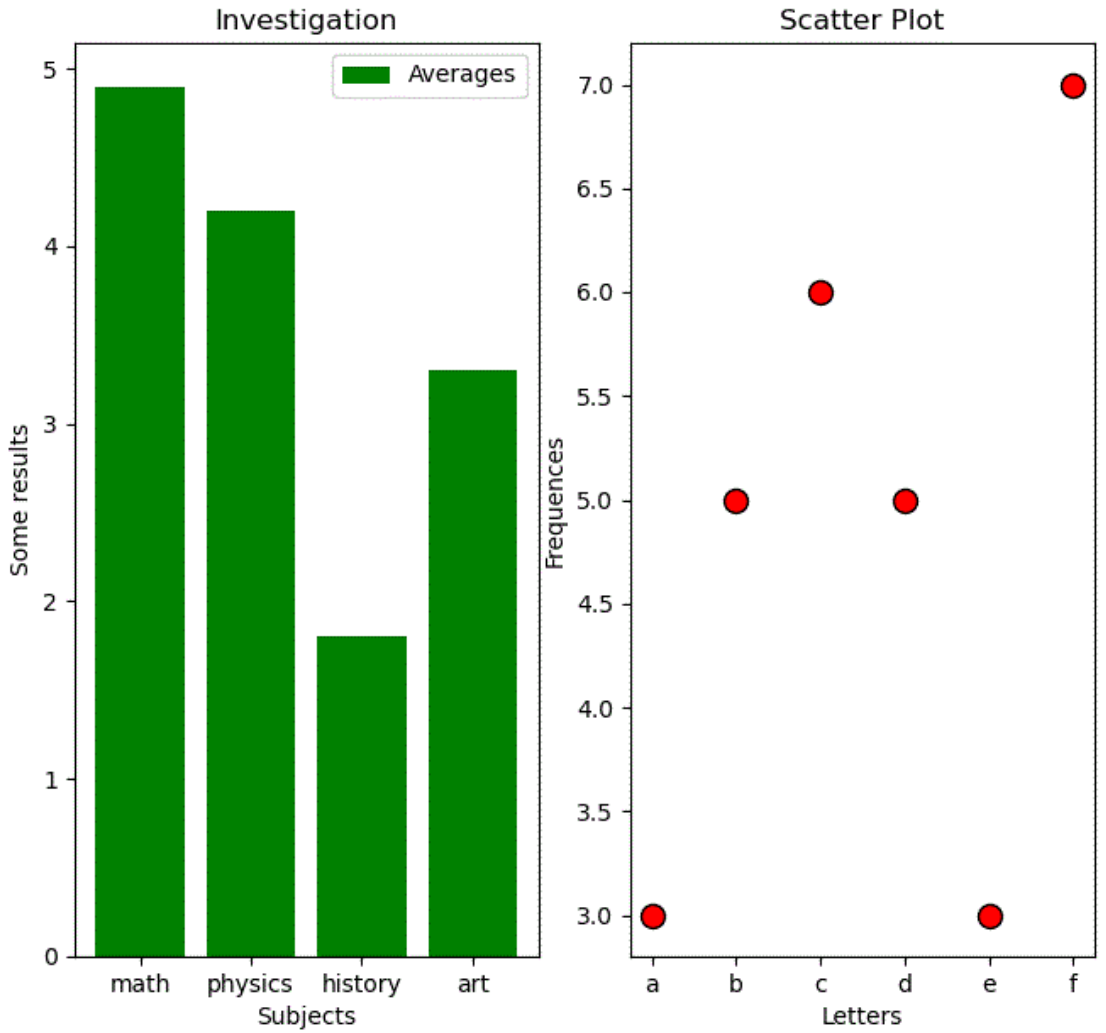
import matplotlib.pyplot as plt

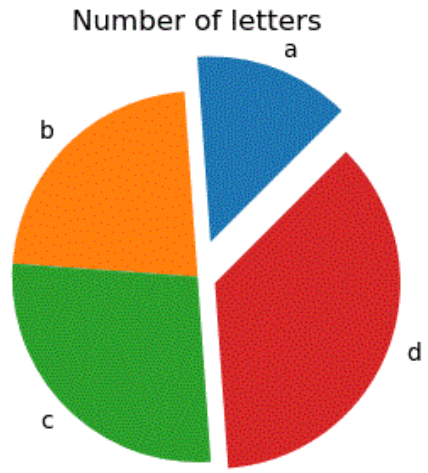
Plot a line:

- Number of x-values == number of y-values
- Linestyle (ls) t ex '-', '--', '-.', ':', 'None', 'solid', 'dashed', 'dashdot', 'dotted’, …
- Marker t ex ^(triangle), o (circle), x, s (square), p (pentagon)
- Color t ex 'y','m','c','r','g','b','w' and ‘k’
- Linewidth
- …

Datavisualisation with matplotlib

Ex3:





Datavisualisation with matplotlib

- plot(): plots lines or points
  - label: Give a name to the plot
- xlabel()/ylabel(): Labels on the x and y axis.
- title(): Title.
- annotate(): Text at specific points
- legend(): Show the lables.
- grid(): Gridlines
- show(): Show the whole graph.

Datavisualisation with matplotlib

plt.plot(…)

plt.bar(…)

plt.scatter(…)

plt.pie(…)

plt.hist(…)

plt.boxplot()