Programming for Essential Digital Skills, Part 2

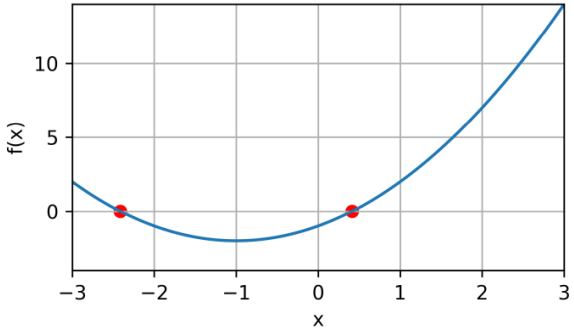
Pieter Kleer

2024

Chapter 8 - Mathematics and plotting

Root finding

Consider $f(x) = x^2 + 2x - 1$. A root x of the function f is a point that satisfies $f(x) = 0$.



SciPy: Scientific Computing with Python

Subpackages

SciPy is organized into subpackages covering different scientific computing domains. These are summarized in the following table:

Subpackage	Description
`cluster`	Clustering algorithms
`constants`	Physical and mathematical constants
`fft`	Discrete Fourier transforms
`fftpack`	Fast Fourier Transform routines (legacy)
`integrate`	Integration and ordinary differential equation solvers
`interpolate`	Interpolation and smoothing splines
`io`	Input and Output
`linalg`	Linear algebra
`ndimage`	N-dimensional image processing
`odr`	Orthogonal distance regression
`optimize`	Optimization and root-finding routines

Solving the equation $f(x) = 0$ using `fsolve()`

Solving the equation $f(x) = 0$ using `fsolve()`

```
import scipy.optimize as optimize

def f(x):
    return x**2 + 2*x - 1

guess = 3
f_zero = optimize.fsolve(f,guess)[0]

print("A root of the function f is given by", f_zero)
```

```
A root of the function f is given by 0.41421356237309503
```

Solving the equation $f(x) = 3$

Suppose we want to solve $f(x) = 3$. How to do this with root finding?

Solving the equation $f(x) = 3$

Suppose we want to solve $f(x) = 3$. How to do this with root finding?

- If we define $g(x) = f(x) - 3$, then $g(x) = 0$ if and only if $f(x) = 3$.

Solving the equation $f(x) = 3$

Suppose we want to solve $f(x) = 3$. How to do this with root finding?

- If we define $g(x) = f(x) - 3$, then $g(x) = 0$ if and only if $f(x) = 3$.

```
def g(x):
    return f(x) - 3

guess = 4
f_zero = optimize.fsolve(g,guess)[0]

print("A number x satisfying f(x) = 3, is given by", f_zero)
```

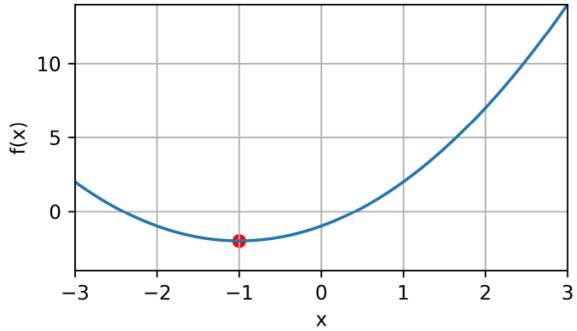
```
A number x satisfying f(x) = 3, is given by 1.2360679774998171
```

Solving the equation $f(x) = c$

```
def solve_eq(f,c,guess):
    # This function returns the solution to f(x) = c using
    # fsolve() on the function g(x) = f(x) - c

    def g(x):
        return f(x) - c

    x = optimize.fsolve(g,guess)[0]
    return x
```

Minimizing a function f

Consider $f(x) = x^2 + 2x - 1$. Minimum of f is a point x for which $f(x)$ is smallest.



Computing a minimum of f using `fmin()`

```
import scipy.optimize as optimize

def f(x):
    return x**2 + 2*x - 1

guess = 1
minimum = optimize.fmin(f,guess)
```

```
Optimization terminated successfully.
         Current function value: -2.000000
         Iterations: 19
         Function evaluations: 38
```

```
print('The minimum of the function f is attained at x = ', minimum)
```

```
The minimum of the function f is attained at x =  [-1.]
```

Computing a minimum of f using `fmin()`

```
import scipy.optimize as optimize

def f(x):
    return x**2 + 2*x - 1

guess = 1
minimum = optimize.fmin(f,guess,disp=False)[0]

print('The minimum of the function f is attained at x = ', minimum)
```

```
The minimum of the function f is attained at x =  -1.0000000000000018
```

Computing a minimum of f using `fmin()`

```
import scipy.optimize as optimize

def f(x):
    return x**2 + 2*x - 1

guess = 1
minimum = optimize.fmin(f,guess,disp=False)[0]

print('The minimum of the function f is attained at x = ', minimum)
```

```
The minimum of the function f is attained at x =  -1.0000000000000018
```

Note: `fmin()` might return a "local" minimum, which is not the true minimum of the function (Classroom Exercise 1).

Matplotlib: Data visualization

Matplotlib is a package that can be used for data visualization

- For this we use the `matplotlib.pyplot` (sub)package ...
- ... which we usually import under the name `plt`

How are functions plotted in Python?

1. Create a vector of x-values, e.g.,

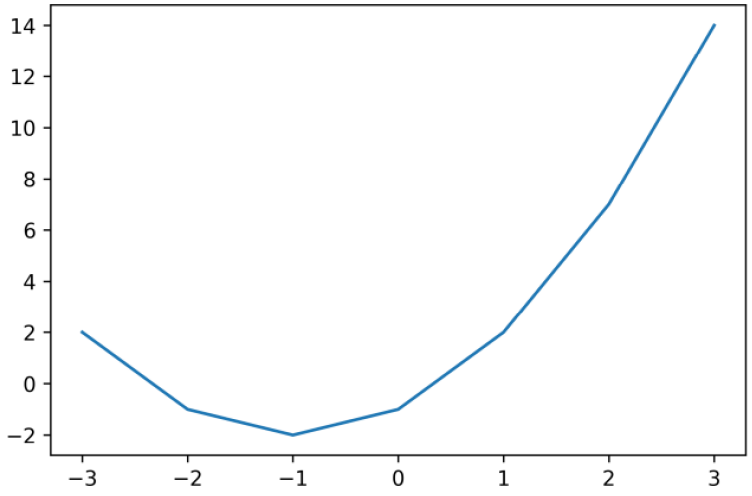
$$x = [-3, -, 2, -1, 0, 1, 2, 3].$$

2. Compute the function values

$$[f(-3), f(-2), f(-1), f(0), f(1), f(2), f(3)] = [2, -1, -2, -1, 2, 7, 14].$$

3. Draw the points $(x_i, f(x_i))$ and connect them with line segments.

Resulting Python plot



Plotting a "smooth" line

Increase the number of points in x to get a smoother line using `np.linspace()`.

- Command `np.linspace(a,b,k)` plots k evenly spaced points in interval $[a, b]$.

Plotting a "smooth" line

Increase the number of points in x to get a smoother line using `np.linspace()`.

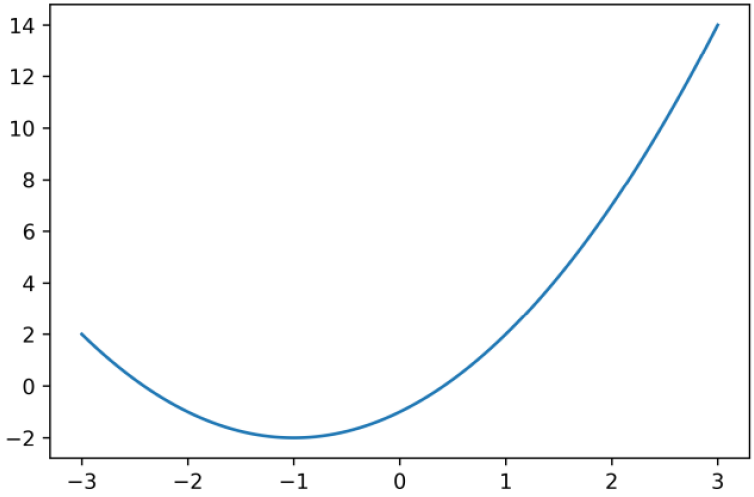
- Command `np.linspace(a,b,k)` plots k evenly spaced points in interval $[a, b]$.

```
import numpy as np

a = 0
b = 1
k = 11

x = np.linspace(a,b,k)
print(x)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

Resulting “smoothed” Python plot

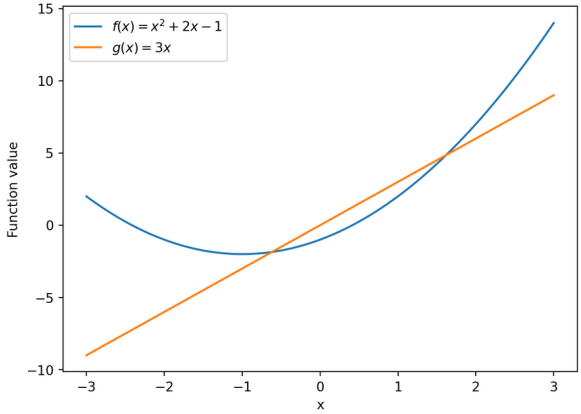
Using `x = np.linspace(-3,3,600)`



Adding legend to plot

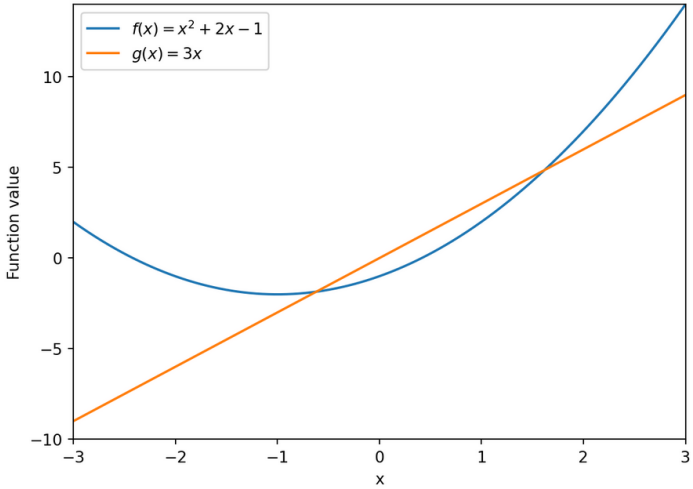
Use label-argument in `plt.plot()` in combination with `plt.legend()` at the end …

- … and `plt.xlabel('x')` and `plt.ylabel('Function value')` for axis labels.



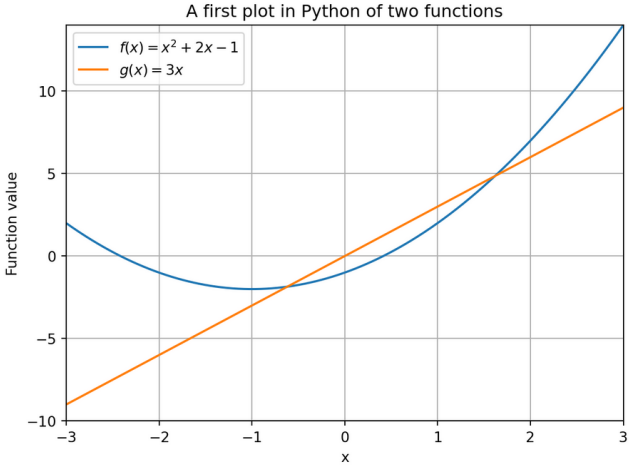
Fixing axes ranges

Use `plt.xlim(-3,3)` and `plt.ylim(-10,14)` to fix range of horizontal/vertical axis, resp.



Adding title and grid

- Use `plt.title('A first plot of two functions')` to add title
- Use `plt.grid()` to add grid.



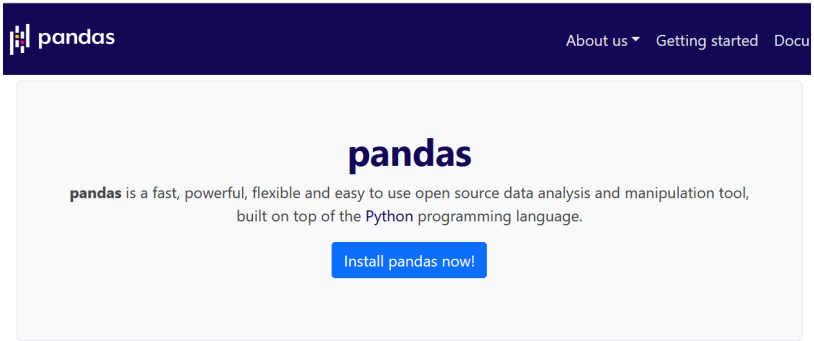
Classroom Exercise 1

Consider the function $f(x) = \frac{9}{10}x^4 - 3x^3 - \frac{7}{2}x^2 + 12x + 3$.

- Plot this function with horizontal axis range $[-6, 6]$, and vertical axis range $[-15, 15]$.
- Find four roots of this function with `fsolve()` by trying out different initial guesses.
- Find a minimum of this function with `fmin()` by using initial guesses -1 and 2. Are both solutions actual minima of the function?

Chapter 9 - Data handling with Pandas

Pandas



Import the package under alias `pd`

```
import pandas as pd
```

Input data

Data can come form many sources. We look at two possibilities:

- Python dictionary (data contained in Python script)
- Matrix, i.e., list of lists (data contained in Python script)
- Comma-separated values (CSV) file (data loaded into Python from another file)

Data in dictionary

```
# Dictionary with data
dataset = {
 'name' : ["Aiden", "Bella", "Carlos", "Dalia", "Elena", "Farhan"],
 'height (cm)' : [185, 155, 190, 185, 160, 170],
 'weight (kg)' : [80, 60, 100, 85, 62, 75],
 'age (years)' : [23, 23, 23, 21, 19, 25],
 'dietary preference' : ['Veggie','Veggie','None','None','Vegan','None']
}
```

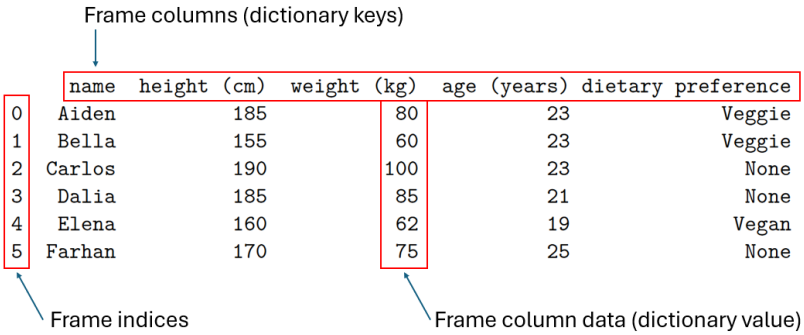
Pandas data frame (created from dictionary)

We load data into a data frame using `DataFrame()` whose input argument is our dictionary.

```
data_frame = pd.DataFrame(dataset)

print(data_frame)
```

```
     name  height (cm)  weight (kg)  age (years) dietary preference
0   Aiden          185           80           23              Veggie
1   Bella          155           60           23              Veggie
2  Carlos          190          100           23                None
3   Dalia          185           85           21                None
4   Elena          160           62           19               Vegan
5  Farhan          170           75           25                None
```

Frame info


Frame columns (dictionary keys)
name height (cm) weight (kg) age (years) dietary preference
0 Aiden 185 80 23 Veggie
1 Bella 155 60 23 Veggie
2 Carlos 190 100 23 None
3 Dalia 185 85 21 None
4 Elena 160 62 19 Vegan
5 Farhan 170 75 25 None
Frame indices
Frame column data (dictionary value)

Accessing data from Pandas frame

Suppose we have a frame named frame_name.

Command	**Output**
`frame_name.head(k)`	First k rows of frame
`frame_name.tail(k)`	Last k rows of frame
`frame_name.loc[i]`	Row with index i
`frame_name.loc[i,col_name]`	Element on row i in column `col_name`
`frame_name.loc[:,col_name]`	Column `col_name`
or `frame_name[col_name]`	

Accessing blocks of the frame

With `frame_name.loc[[i1,i2,...,iq],[col_1,...,col_r]]`:

- Block formed by row indices $i_1, \ldots, i_q$ and columns with name $\text{col}_1, \ldots, \text{col}_r$

```
# Extract block with rows 2,4 and columns name, height and age.
x = data_frame.loc[[2,4],['name','height (cm)', 'age (years)']]
print(x)
```

```
     name  height (cm)  age (years)
2  Carlos          190           23
4   Elena          160           19
```

With `frame_name.loc[[i1,i2,...,iq]]`:

- Block formed by row indices $i_1, \ldots, i_q$ and all columns.

Accessing rows using Boolean list

For list x containing values `True` or `False`:

- Access True-rows with `frame_name.loc[x]`.

```
x = [False, False, True, False, True, False]
print(data_frame.loc[x])
```

```
     name  height (cm)  weight (kg)  age (years) dietary preference
2  Carlos          190          100           23                None
4   Elena          160           62           19               Vegan
```

Accessing rows based on conditional statement

Suppose we only want the people whose dietary preference is `'None'`.

- Conditional statement `data_frame.loc[:,'dietary preference'] == 'None'`

```
x = data_frame.loc[:,'dietary preference'] == 'None'
print(x)
```

```
0    False
1    False
2     True
3     True
4    False
5     True
Name: dietary preference, dtype: bool
```

Accessing rows based on conditional statement (cont'd)

```
x = data_frame.loc[:,'dietary preference'] == 'None'
print(data_frame.loc[x])
```

```
     name  height (cm)  weight (kg)  age (years) dietary preference
2  Carlos          190          100           23                None
3   Dalia          185           85           21                None
5  Farhan          170           75           25                None
```

Editing frame data

```
data = [
[2,4,-1,2],
[5,1,2,9],
[3,7,8,9]
]

frame = pd.DataFrame(data)

print(frame)
```

```
   0  1  2  3
0  2  4 -1  2
1  5  1  2  9
2  3  7  8  9
```

Editing row names

Row and column names are stored in `frame.index` and `frame.columns`, respectively.

```
print(frame.index)
```

```
RangeIndex(start=0, stop=3, step=1)
```

```
frame.index = ['Row0','Row1','Row2']
print(frame)
```

```
      0  1  2  3
Row0  2  4 -1  2
Row1  5  1  2  9
Row2  3  7  8  9
```

Editing column names

```
frame.columns = ['Col0','Col1','Col2','Col3']

print(frame)
```

```
      Col0  Col1  Col2  Col3
Row0     2     4    -1     2
Row1     5     1     2     9
Row2     3     7     8     9
```

Editing entries

Editing entries can be done with `frame.loc[row_name,col_name] = new_value`

```
# Edit entry on row 1, column 2
frame.loc['Row1','Col2'] = 10

print(frame)
```

```
      Col0  Col1  Col2  Col3
Row0     2     4    -1     2
Row1     5     1    10     9
Row2     3     7     8     9
```

Editing whole row

Replace row with list y: `frame.loc[row_name,:] = y`

```
# Replace row 2
y = [-2,-2,-2,-2]
frame.loc['Row2'] = y #frame.loc['Row2',:] = y also works

print(frame)
```

```
      Col0  Col1  Col2  Col3
Row0     2     4    -1     2
Row1     5     1    10     9
Row2    -2    -2    -2    -2
```

Editing whole column

Replace column with list y: `frame.loc[:,col_name] = y`

```
# Replace column 2
y = [-1,-1,-1]
frame.loc[:,'Col2'] = y

print(frame)
```

```
      Col0  Col1  Col2  Col3
Row0     2     4    -1     2
Row1     5     1    -1     9
Row2    -2    -2    -1    -2
```

Editing whole column according to a mathematical function

Suppose we want to square every number in the column `'Col1'`.

```
def f(x):
    return x**2
```

Use `apply()` function and overwrite entries in `'Col1'` (don't use `frame.loc['Col1']`!)

```
frame['Col1'] = frame['Col1'].apply(f)

print(frame)
```

```
      Col0  Col1  Col2  Col3
Row0     2   16    -1     2
Row1     5    1    -1     9
Row2    -2    4    -1    -2
```

Adding new row

New rows appear at the bottom of the frame.

```
# Add a row
frame.loc['New row'] = [5,5,3,1]
print(frame)
```

```
         Col0  Col1  Col2  Col3
Row0        2    16    -1     2
Row1        5     1    -1     9
Row2       -2     4    -1    -2
New row     5     5     3     1
```

Adding new column

New columns appear at the right of the frame.

```
frame.loc[:,'New column'] = [1,1,1,1]
print(frame)
```

```
          Col0  Col1  Col2  Col3  New column
Row0         2    16    -1     2           1
Row1         5     1    -1     9           1
Row2        -2     4    -1    -2           1
New row      5     5     3     1           1
```

Inserting a new column

Can also insert column at specified location using `frame.insert()`

- Takes as input insertion position, column name and column data.

```
# Insert column with name 'New column' and data [10,10,10,10] at position 2.
frame.insert(2,'Inserted column', [10,10,10,10])

print(frame)
```

```
         Col0  Col1  Inserted column  Col2  Col3  New column
Row0        2    16               10    -1     2           1
Row1        5     1               10    -1     9           1
Row2       -2     4               10    -1    -2           1
New row     5     5               10     3     1           1
```

Computing statistics of column data

Can compute properties of data like minimum, maximum, mean, etc.

```
print(frame)
```

```
          Col0  Col1  Inserted column  Col2  Col3  New column
Row0         2    16               10    -1     2           1
Row1         5     1               10    -1     9           1
Row2        -2     4               10    -1    -2           1
New row      5     5               10     3     1           1
```

```
# Minimum of the first column
min_col1 = frame.loc[:,'Col1'].min() #Use .max()/mean() for maximum/mean
print(min_col1)
```

```
1
```

Classroom exercise

```
data = [
[2,4,-1,2],
[5,1,2,9],
[3,7,8,9]
]

frame = pd.DataFrame(data)
frame.index = ['Row0','Row1','Row2']
frame.columns = ['Col0','Col1','Col2','Col3']
```

Create the following Pandas data frame that has an extra row with the maximum per column

	Col0	Col1	Col2	Col3
Row0	2	4	-1	2
Row1	5	1	2	9
Row2	3	7	8	9
Maximum	5	7	8	9

Importing data into Python

Data from, e.g., comma-separated values (CSV) file can be imported into Python using `read_csv()`.

```
csv_to_frame = pd.read_csv('dataset.csv')

print(csv_to_frame)
```

```
     name   height (cm)   weight (kg)   age (years)  dietary preference
0   Aiden           185            80            23              Veggie
1   Bella           155            60            23              Veggie
2  Carlos           190           100            23                None
3   Dalia           185            85            21                None
4   Elena           160            62            19               Vegan
5  Farhan           170            75            25                None
6   Geert           178            80            25              Veggie
```

Data header

Python interprets first line of .csv file as header with column names for the Pandas frame.

- No header present? Set `header=None` as an additional argument in `read_csv()`.

```
csv_to_frame = pd.read_csv('dataset.csv', header=None)
print(csv_to_frame)
```

```
        0              1              2              3                    4
0    name   height (cm)   weight (kg)   age (years)   dietary preference
1   Aiden            185             80            23               Veggie
2   Bella            155             60            23               Veggie
3  Carlos            190            100            23                 None
4   Dalia            185             85            21                 None
5   Elena            160             62            19                Vegan
6  Farhan            170             75            25                 None
7   Geert            178             80            25               Veggie
```

Exporting data out of Python

Can export Pandas frame to a .csv file using `to_csv()`.

```
frame.to_csv('new_dataset.csv')
```

This creates new file in same folder as Python script with the given name.

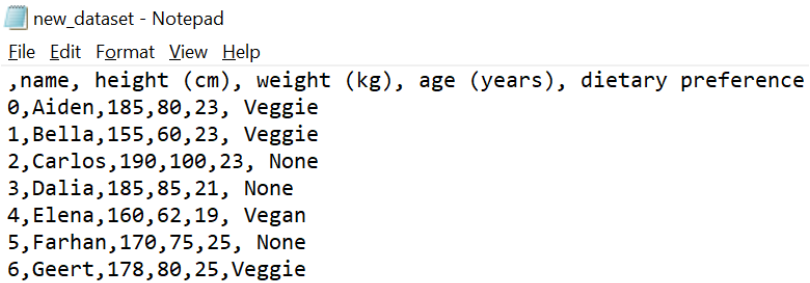


Figure 1: Exported data in .csv file (with row indices)

Suppressing row indices in exported file

By default, Python includes the row indices in the exported .csv file.

- Suppress row indices in exported file with `index=False` in `to_csv()`.

```
frame.to_csv('new_dataset.csv', index=False)
```

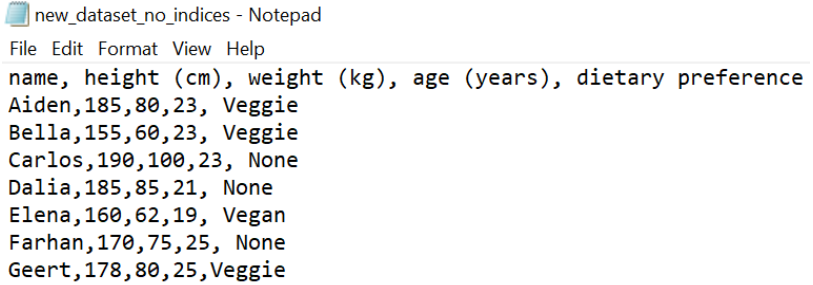


Figure 2: Exported data in .csv file (without row indices)

Chapter 10 - Object oriented programming

Object oriented programming

Python is structered around objects that contain data and on which functions can be performed.

For example,

- List: Can edit or add elements
- Data frame: Can edit or add rows/columns

Class

A class is a blueprint for objects of a certain type.

- Determines the attributes (properties) that an object has and the methods (functions) that can be performed on the objects.

Class

A class is a blueprint for objects of a certain type.

- Determines the attributes (properties) that an object has and the methods (functions) that can be performed on the objects.

Data frame example

- Attributes: Column names, row names, (initial) data
- Methods: Add row, add column, rename row, rename column, compute summary statistics

Class

A class is a blueprint for objects of a certain type.

- Determines the attributes (properties) that an object has and the methods (functions) that can be performed on the objects.

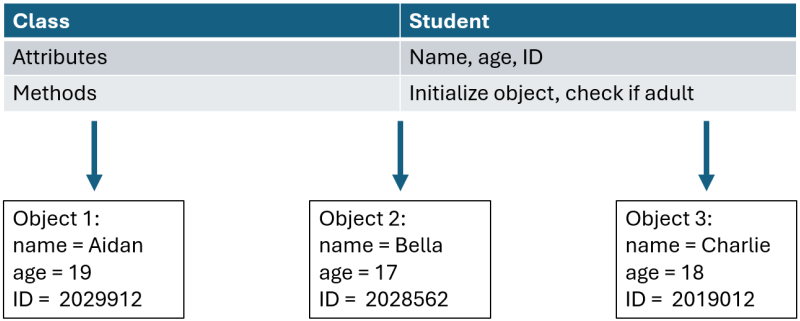
Data frame example

- Attributes: Column names, row names, (initial) data
- Methods: Add row, add column, rename row, rename column, compute summary statistics

Methods either change the attributes or yields additional information of an object.

Student example

We will create a class `Student` whose objects are students with attributes: name, age, (student) ID.



We will write a method that checks if student is an adult (≥ 18) or not.

Initializing objects

Initialization is always done with `__init__()` function.

```
class Student:
    # This function initializes an object of the class Student
    # by setting the attributes (name, age and ID) of an object.
    def __init__(self,name,age,student_number):
        self.name = name
        self.age = age
        self.ID = student_number
```

The argument `self` should be thought of as the object that we want to create.

- The use of `self` as a variable name for this is standard in Python

Methods

Methods are Python functions in a class. `__init__()` is also a method of the class.

```
class Student:
    def __init__(self,name,age,student_number):
        self.name = name
        self.age = age
        self.ID = student_number

    # Check if student is adult
    def adult(self):
        if self.age >= 18:
            return print(self.name," is an adult")
        else:
            return print(self.name,"is not an adult")
```

- Every method has first input argument `self`.
- Attribute `attribute_name` can be accessed in method using `self.attribute_name`.

Additional inputs and changing attributes

Methods can require additional input arguments beside `self`.

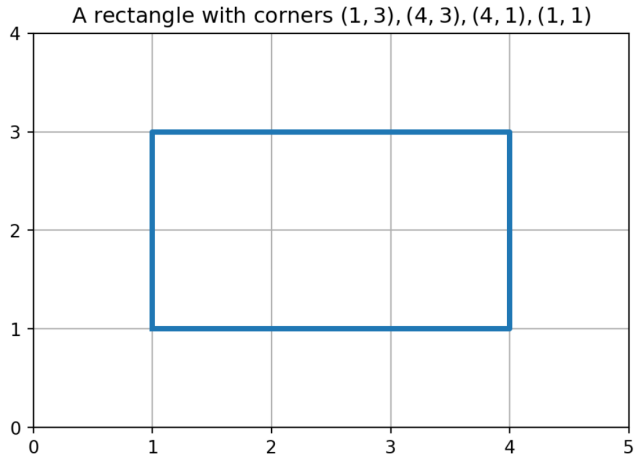
- See `reg_check()` example in course document.

Methods can be used to manipulate attributes.

- See `addCourse()`/`delCourse` example in course document.

Mathematical example: Rectangles

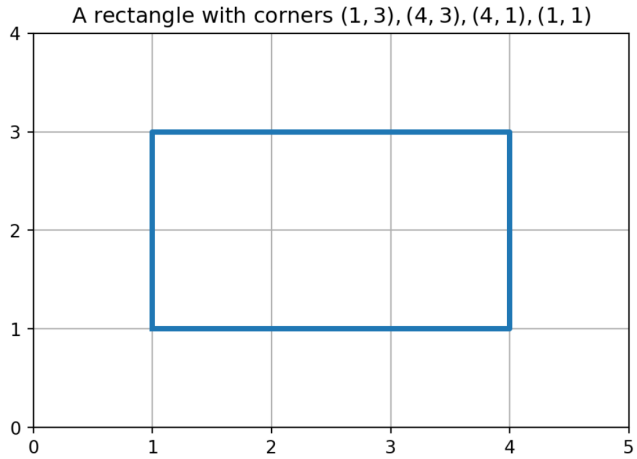
We create a class `Rectangle` whose objects are rectangles in a two-dimensional plane.



How to model a rectangle? What should the attributes be?

Mathematical example: Rectangles

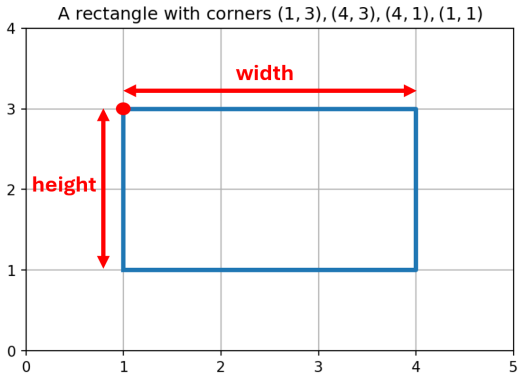
We create a class `Rectangle` whose objects are rectangles in a two-dimensional plane.



How to model a rectangle? What should the attributes be? Depends on desired methods...

Rectangle attributes

To uniquely determine a rectangle, it suffices to know: Upper-left corner point, width and height.



Rectangle __init__() method

```
class Rectangle:
    # Here corner is the upper-left corner point which
    # should be a list containing the x- and y-coordinate.
    def __init__(self, corner, height, width):
        self.corner= corner
        self.height = height
        self.width = width

rectangle1 = Rectangle([1,3],2,3)

# Print upper left corner of the rectangle
print(rectangle1.corner)
```

```
[1, 3]
```

Rectangle methods: Area and circumference

```
class Rectangle:
    # Initialize rectangle by providing upper-left corner, width and height
    def __init__(self,corner,width,height):
        self.corner = corner
        self.height = height
        self.width = width

    # Compute area = width*height of rectangle
    def area(self):
        return self.height*self.width

    # Compute circumference = 2*width + 2*height of rectangle
    def circumference(self):
        return 2*self.width + 2*self.height
```

Rectangle methods: Compute all corner points

```
class Rectangle:
    # Initialize rectangle by providing upper-left corner, width and height
    def __init__(self, corner, height, width):
        self.corner= corner
        self.height = height
        self.width = width

    # Compute corner points: the output has the points in the
    # order [upper-left, upper-right, lower-right, lower-left]
    def corners(self):
        up_right = [self.corner[0] + self.width, self.corner[1]]
        low_right=[self.corner[0]+self.width,self.corner[1]-self.height]
        low_left = [self.corner[0],  self.corner[1] - self.height]
        return [self.corner, up_right, low_right, low_left]
```

Rectangle methods: Plotting

You can find the `plotting()` method in the course document. The idea is as follows:

- Determine all corner points using the `corners()` method.
- Plot the points in order: upper-left, upper-right, lower-right, lower-left, upper-left.

Rectangle methods: Plotting

You can find the `plotting()` method in the course document. The idea is as follows:

- Determine all corner points using the `corners()` method.
- Plot the points in order: upper-left, upper-right, lower-right, lower-left, upper-left.

The `plot()` function plots these five points and connects them with a line segment (resulting in a rectangle shape).

- Plot list with all x-coordinates against list with all y-coordinates of the five points.