Python 07

Elias Wachmann

2024

Content

1. Dictionaries
2. Recursion
3. Plotting (further topics)

Dictionaries

Dictionaries

Dictionaries are ordered data structure that map **keys** to **values**.
They are declared with curly brackets {} and the key-value pairs are separated by :

```
1 student = {'name': 'Alice', 'age': 20, '
    major': 'Computer Science', 'GPA': 3.8}
2
3 # Accessing dictionary values
4 print(student['name'])  # Output: Alice
5 print(student['GPA'])  # Output: 3.8
```

Dictionaries – operations

Indexed just like arrays but with keys

```
1 student = {'name': 'Alice', 'age': 20, '
    major': 'Computer Science', 'GPA': 3.8}
2 # Change the value of a key
3 student['name'] = 'Bob'
4 # Add a new key-value pair
5 student["courses"] = ['Math', 'CompSci']
6 # Deletes "age" - key-value pair
7 del student['age']
```

`del` is one of multiple ways to delete a key-value pair from a dictionary.

Dictionaries – keys(), values(), items()

`keys()`, `values()` and `items()` are methods that return a list of the respective elements of the dictionary.

```
1 all_keys = student.keys()
2 all_values = student.values()
3 all_items = student.items()
4 for key in student.keys():
5     print(key)
```

In this way you can iterate over the elements of a dictionary.

`items()` returns a list of tuples of the form `(key, value)`.

Dictionaries – examples

```
1 replacements = {'bad': 'good', 'ugly': '
    beautiful', 'hate': 'love'}
2 s = 'I hate bad weather. It makes me feel
    ugly.'
3 for word in s.split():
4     if word in replacements:
5         s = s.replace(word, replacements[
    word])
6 print(s)  # I love good weather. It makes me
    feel beautiful.
```

Dictionaries – examples (json)

JSON is a data format that is used to store and transmit data.
In python the json module is used to read and write json files:

```
1 import json
2 with open('slides/07/examples/example.json',
      'r') as f:
3     data = json.load(f)
4 print(data)
```

The data is returned as a dictionary.

Recursion

Recursion

- Breaking down a problem into smaller and smaller subproblems $\rightarrow$ divide and conquer.
- The solution to the original problem is then built up from the solutions to the smaller subproblems.
- Every iterative algorithm can be rewritten as in recursive form and vice versa.

Recursion – Fibonacci Example

Fibonacci numbers are defined by the following recurrence relation:

$$F_n = F_{n-1} + F_{n-2} \quad \text{with} \quad F_0 = 0, \quad F_1 = 1 \qquad (1)$$

or rather intuitively in code:

```
1 def fib_recursive(n):
2     if n < 2:
3         return n
4     return fib_recursive(n-1) +
    fib_recursive(n-2)
```

Recursion – Fibonacci Example

The same sequence can be calculated iteratively using a `for` loop:

```
1 def fib_iterative(n):
2     if n < 2:
3         return n
4     prev, curr = 0, 1
5     for i in range(n-1):
6         prev, curr = curr, prev + curr
7     return curr
```

Recursion – Remarks

While recursion is a powerful concept, which is used in many algorithms, it is not always the best choice. Especially for our toy example, the iterative version is much faster:

`fib_iterative(20)`	`fib_recursive(20)`
0.0018s	2.058s

Recusrion is also used in: Quicksort, Merge sort and Tower of Hanoi

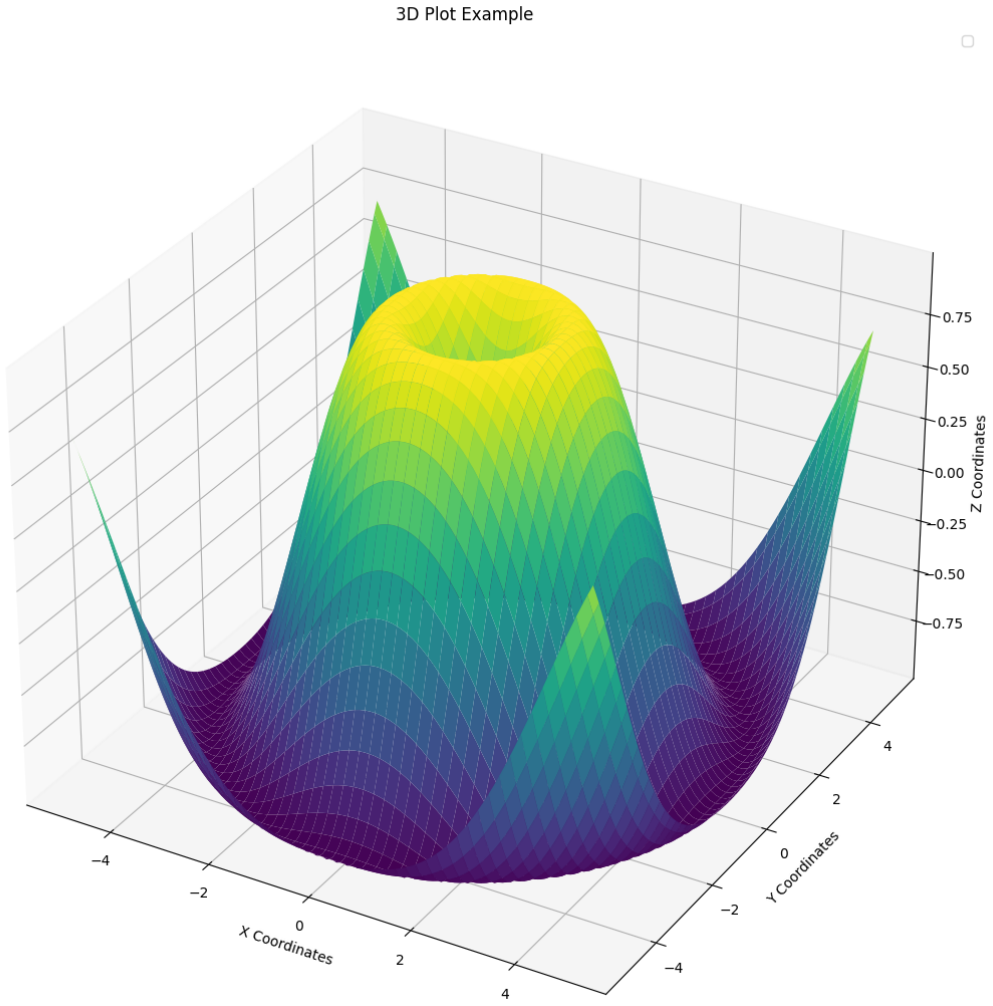
Plotting (further topics)

3D - Plotting

Matplotlib can also be used to plot 3D data.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  x = np.linspace(-5, 5, 100)
4  y = np.linspace(-5, 5, 100)
5  X, Y = np.meshgrid(x, y)
6  Z = np.sin(np.sqrt(X**2 + Y**2))
7
8  fig = plt.figure()
9  ax = fig.add_subplot(111, projection='3d')
10 ax.plot_surface(X, Y, Z, cmap='viridis')
11 # ...
```

3D - Plotting


3D Plot Example
X Coordinates
Y Coordinates
Z Coordinates
−4
−2
0
2
4
0.75
0.50
0.25
0.00
0.25
0.50
0.75

Meshgrid

Meshgrid is a function that returns coordinate matrices from coordinate vectors.

```
1 import numpy as np
2
3 x = np.linspace(1, 3, 3)
4 x, y = np.meshgrid(x, x)
5 # x = [[1, 2, 3],[1, 2, 3],[1, 2, 3]]
6 # y = [[1, 1, 1],[2, 2, 2],[3, 3, 3]]
7 x, y = np.meshgrid([1,2],[1,2,3])
8 # x = [[1, 2],[1, 2], [1, 2]]
9 # y = [[1, 1],[2, 2],[3, 3]]
```

3D - Plotting (types)

- `projection='3d'` in `add_subplot`
- `ax.view_init(elev, azim)` to set the viewing angle
- `ax.plot_surface` to plot a surface
- `ax.plot_wireframe` to plot a wireframe
- `ax.plot_trisurf` to plot a triangulated surface
- `ax.plot_trisurf` to plot a triangulated surface
- `ax.contour3D` to plot contours
- `ax.scatter3D` to plot 3D scatter plots
- `ax.bar3d` to plot 3D bars

Animations

Animations can be created using the animation module of matplotlib.

We will use the `FuncAnimation` class to create an animation of a pendulum.

We need parameters (time, fps), a way to calculate the position of the pendulum and a function to update the plot.

Animations

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import matplotlib.animation as animation
4
5  #params
6  t_max, fps = 10, 30
7  # function to call from FuncAnimation
8  def animate(i):
9      # plotting code!
10 fig, ax = plt.subplots()
11 ani = animation.FuncAnimation(fig, animate,
       frames=int(t_max * fps), interval=1000 /
       fps)
```

Animations (saving)

```
# save mp4
ani.save('pendulum.mp4', writer='ffmpeg',
    fps=fps)
# save gif
ani.save('pendulum.gif', writer='imagemagick
    ', fps=fps)
plt.show()
```

Histograms

Histograms can be created using the hist function of matplotlib.

Often histograms are visualized with additional error bars. This can be done using the errorbar function.