TPIb_Tutorium04_coding_SS25

May 8, 2025

1 Some Python code to get started

Philipp Hoevel, Philipp Hoevel, Saarland University
Date: **May 08, 2025**
Inspired by Hans Petter Langtangen, RIP

1.1 Why programming?

> *Everybody in this country should learn how to program a computer... because it teaches you how to think*. Steve Jobs, 1955-2011.

1.2 The learning strategy is about doing exercises

- Study and try to understand examples
- Program a lot!
- Try to solve all excercise.

1.3 This introduction 1 is about getting started

- printing text and numbers
- playing with matrices
- testing data types

```
In [44]: print('Hello world!')
```

```
Hello world!
```

1.4 Question 1

(a) Define a vector x composed of 101 entries equally spaced between 1.5 and -3.5 (where 1.5 and -3.5 are the first and last entries of x , respectively).

```
In [1]: import numpy as np
```

```
In [2]: x = np.linspace(1.5, -3.5, 101)
        x
```

```
Out[2]: array([ 1.5 ,  1.45,  1.4 ,  1.35,  1.3 ,  1.25,  1.2 ,  1.15,  1.1 ,
                1.05,  1.  ,  0.95,  0.9 ,  0.85,  0.8 ,  0.75,  0.7 ,  0.65,
                0.6 ,  0.55,  0.5 ,  0.45,  0.4 ,  0.35,  0.3 ,  0.25,  0.2 ,
                0.15,  0.1 ,  0.05,  0.  , -0.05, -0.1 , -0.15, -0.2 , -0.25,
               -0.3 , -0.35, -0.4 , -0.45, -0.5 , -0.55, -0.6 , -0.65, -0.7 ,
               -0.75, -0.8 , -0.85, -0.9 , -0.95, -1.  , -1.05, -1.1 , -1.15,
               -1.2 , -1.25, -1.3 , -1.35, -1.4 , -1.45, -1.5 , -1.55, -1.6 ,
               -1.65, -1.7 , -1.75, -1.8 , -1.85, -1.9 , -1.95, -2.  , -2.05,
               -2.1 , -2.15, -2.2 , -2.25, -2.3 , -2.35, -2.4 , -2.45, -2.5 ,
               -2.55, -2.6 , -2.65, -2.7 , -2.75, -2.8 , -2.85, -2.9 , -2.95,
               -3.  , -3.05, -3.1 , -3.15, -3.2 , -3.25, -3.3 , -3.35, -3.4 ,
               -3.45, -3.5 ])
```

(b) Define a vector y corresponding to vector x from which all entries indexed by an odd integer have been discarded. In other words, y should be a vector whose entries correspond to the entries of x that have an even index (or position) in x.

```
In [3]:  y = x[1::2]

In [4]: y

Out[4]: array([ 1.45,  1.35,  1.25,  1.15,  1.05,  0.95,  0.85,  0.75,  0.65,
                0.55,  0.45,  0.35,  0.25,  0.15,  0.05, -0.05, -0.15, -0.25,
               -0.35, -0.45, -0.55, -0.65, -0.75, -0.85, -0.95, -1.05, -1.15,
               -1.25, -1.35, -1.45, -1.55, -1.65, -1.75, -1.85, -1.95, -2.05,
               -2.15, -2.25, -2.35, -2.45, -2.55, -2.65, -2.75, -2.85, -2.95,
               -3.05, -3.15, -3.25, -3.35, -3.45])
```

(c) Print the values of the last 10 entries of vector y in the command window.

```
In [5]: y[-10:]

Out[5]: array([-2.55, -2.65, -2.75, -2.85, -2.95, -3.05, -3.15, -3.25, -3.35,
               -3.45])

In [6]: y[0], y[1]

Out[6]: (1.45, 1.35)
```

(d) Assign the value π to all entries of vector x.

```
In [7]: x[:] = np.pi

In [8]: x

Out[8]: array([3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
               3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
               3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
               3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
               3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
```

```
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265, 3.14159265, 3.14159265, 3.14159265, 3.14159265,
        3.14159265])
```

1.5 Question 2:

(a) Using functions such as ones and eye wisely, create the following (6×6) matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 2 & 2 \\ 0 & 1 & 0 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 2 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}.$$

```
In [9]: n = 3;
        A = np.block([[np.eye(n), 2 * np.ones((n,n))],
             [np.zeros((n,n)), - np.eye(n)]])
        A
```

```
Out[9]: array([[ 1.,  0.,  0.,  2.,  2.,  2.],
               [ 0.,  1.,  0.,  2.,  2.,  2.],
               [ 0.,  0.,  1.,  2.,  2.,  2.],
               [ 0.,  0.,  0., -1., -0., -0.],
               [ 0.,  0.,  0., -0., -1., -0.],
               [ 0.,  0.,  0., -0., -0., -1.]])
```

```
In [10]: A.shape
```

```
Out[10]: (6, 6)
```

(b) Using the kron function, create the 3×12 matrix B formed (column-wise) by reading the entries of matrix A column-wise.

```
In [11]: B = np.ones((3,12))
         B
```

```
Out[11]: array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])

In [12]: C = np.kron(B, A) # Kronecker product
         C.view()

Out[12]: array([[ 1.,  0.,  0., ...,  2.,  2.,  2.],
                [ 0.,  1.,  0., ...,  2.,  2.,  2.],
                [ 0.,  0.,  1., ...,  2.,  2.,  2.],
                ...,
                [ 0.,  0.,  0., ..., -1., -0., -0.],
                [ 0.,  0.,  0., ..., -0., -1., -0.],
                [ 0.,  0.,  0., ..., -0., -0., -1.]])

In [13]: C.shape

Out[13]: (18, 72)
```

1.6 Question 3:

(a) Evaluate the expression 10^{1000}.

```
In [14]: a = 10**1000
         a

Out[14]: 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

(b) Can you compute with this number?

```
In [15]: a / 2


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

         OverflowError                             Traceback (most recent call last)

         /tmp/ipykernel_11496/3207491792.py in <module>
         ----> 1 a / 2


         OverflowError: integer division result too large for a float
```

Everything in Python is an object
Variables refer to objects:

```
In [16]: a = 5       # a refers to an integer (int) object
         b = 9       # b refers to an integer (int) object
         c = 9.0     # c refers to a real number (float) object
```

```
d = b//a     # d refers to an int//int => int object
e = b/a      # e refers to an int/int => float object
f = c/a      # f refers to float/int => float object
s = 'b/a=%g' % (b/a)  # s is a string/text (str) object
d, e, f, s
```

```
Out[16]: (1, 1.8, 1.8, 'b/a=1.8')
```

We can convert between object types:

```
In [17]: a = 3               # a is int
         b = float(a)        # b is float 3.0
         c = 3.9             # c is float
         d = int(c)          # d is int 3
         d = round(c)        # d is float 4.0
         d = int(round(c))   # d is int 4
         d = str(c)          # d is str '3.9'
         e = '-4.2'          # e is str
         f = float(e)        # f is float -4.2
```

(c) Knowing how Inf , -Inf , and NaN behave in arithmetic operations is useful to trace back their first occurence. To get more familiar with those special floating-point numbers, evaluate the following expressions in the Command Window and check that the results make sense to you:

(d) 0==(-0)

(ii) np.Inf+1

(iii) np.Inf/0

(iv) np.Inf/(-0)

(v) np.Inf/np.Inf

(vi) np.Inf-np.Inf

(vii) 0*np.Inf

(viii) 0/0

```
In [18]:  0==(-0)
```

```
Out[18]: True
```

```
In [19]: np.Inf + 1
```

```
Out[19]: inf
```

```
In [20]: np.Inf / 0
```

```
        ---------------------------------------------------------------------------
        ZeroDivisionError                         Traceback (most recent call last)

        /tmp/ipykernel_11496/1685272088.py in <module>
    ----> 1 np.Inf / 0


        ZeroDivisionError: float division by zero

In [21]: np.Inf / (-0)

        ---------------------------------------------------------------------------
        ZeroDivisionError                         Traceback (most recent call last)

        /tmp/ipykernel_11496/3779616040.py in <module>
    ----> 1 np.Inf / (-0)


        ZeroDivisionError: float division by zero

In [22]: np.Inf/np.Inf

Out[22]: nan

In [23]: np.Inf-np.Inf

Out[23]: nan

In [24]: 0 * np.Inf

Out[24]: nan

In [25]: 0 / 0

        ---------------------------------------------------------------------------
        ZeroDivisionError                         Traceback (most recent call last)

        /tmp/ipykernel_11496/4154377607.py in <module>
    ----> 1 0 / 0


        ZeroDivisionError: division by zero
```

1.7 Question 4:

64-bit binary numbers are represented in a computer by 64 bits, as the name suggests. 1 bit for the sign, 11 bits for the characteristics, 52 bits for the man- tissa. 64-bit representation refers to type double or double precision. 32-bit representation refers to type float or single precision.

The precision is set by the smallest difference that allows to distinguish between to subsequent machine numbers and can be understood as an upper bound for roundoff errors. The precision can be tested by halfing the distance to the number 1, until the machine representation is indistinguishable from 1.

(a) Give a Python implementation for finding the machine precision for single and double type.

```
In [26]: def precision(EPS):
             prev_epsilon = 0
             while ((1 + EPS ) != 1):
                 prev_epsilon = EPS
                 EPS = EPS / 2
             print(prev_epsilon)

In [27]: precision(np.single(0.5))

2.220446049250313e-16


In [28]: precision(np.double(0.5))

2.220446049250313e-16


In [29]: def machineEpsilon(func=float):
             machine_epsilon = func(1)
             while func(1)+func(machine_epsilon) != func(1):
                 machine_epsilon_last = machine_epsilon
                 machine_epsilon = func(machine_epsilon) / func(2)
             return machine_epsilon_last

In [30]: machineEpsilon(np.single)

Out[30]: 1.1920929e-07

In [31]: machineEpsilon(np.double)

Out[31]: 2.220446049250313e-16

In [32]: 7./3 - 4./3 -1 # a short cut to machine precision

Out[32]: 2.220446049250313e-16
```

(b) Compare your result with the commands: np.finfo(...).eps

```
In [33]: np.finfo(np.single).eps
```

```
Out[33]: 1.1920929e-07

In [34]: np.finfo(np.float32).eps

Out[34]: 1.1920929e-07

In [35]: np.finfo(np.double).eps

Out[35]: 2.220446049250313e-16

In [36]: np.finfo(np.float).eps

/tmp/ipykernel_11496/386457714.py:1: DeprecationWarning: `np.float` is a deprecated
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
  np.finfo(np.float).eps


Out[36]: 2.220446049250313e-16

In [37]: np.finfo(np.float128).eps # might not work on all installations (if floati

Out[37]: 1.084202172485504434e-19
```

(c) What are the largest and smallest numbers for float, float32, float128 type? How could you extract that information or compute the values?

```
In [38]: np.finfo(np.single).max # single

Out[38]: 3.4028235e+38

In [39]: np.finfo(np.double).max # float

Out[39]: 1.7976931348623157e+308

In [40]: np.finfo(np.float128).max # might not work on all installations (if floati

Out[40]: 1.189731495357231765e+4932

In [41]: np.finfo(np.single).min # single

Out[41]: -3.4028235e+38

In [42]: np.finfo(np.double).min # float

Out[42]: -1.7976931348623157e+308

In [43]: np.finfo(np.float128).min # might not work on all installations (if floati

Out[43]: -1.189731495357231765e+4932
```

1.8 Let's try some turtle graphics

```
In [54]: import turtle

In [55]: turtle.down()

In [47]: turtle.forward(100)

In [48]: turtle.right(90)
         turtle.forward(100)
         turtle.right(90)
         turtle.forward(100)
         turtle.right(90)
         turtle.forward(100)
         turtle.right(90)

In [49]: for side in range(12):
             turtle.forward(100)
             turtle.right(30)

In [50]: turtle.bye()

In [56]: turtle.down()
         for steps in range(100):
             for c in ('blue', 'red', 'green'):
                 turtle.color(c)
                 turtle.forward(steps)
                 turtle.right(30)

In [57]: turtle.home()

In [58]: turtle.bye()

In [ ]:
```