# Chapter 1: Python introduction

Werner Bauer

UNIVERSITY OF SURREY

# Outline

# Python installation

- Python is an open source (freely available) programming language. There are many ways to install and use Python. One possible way is:
  - **Anaconda**: a Python distribution for scientific computing,
  - **Spyder**: a Python IDE, particularly suited for Matlab users.
- Using Anaconda, you can install Spyder as follows:

```
conda install spyder # execute this in a shell
```

- Install also already NumPy, SciPy, and Matplotlib:

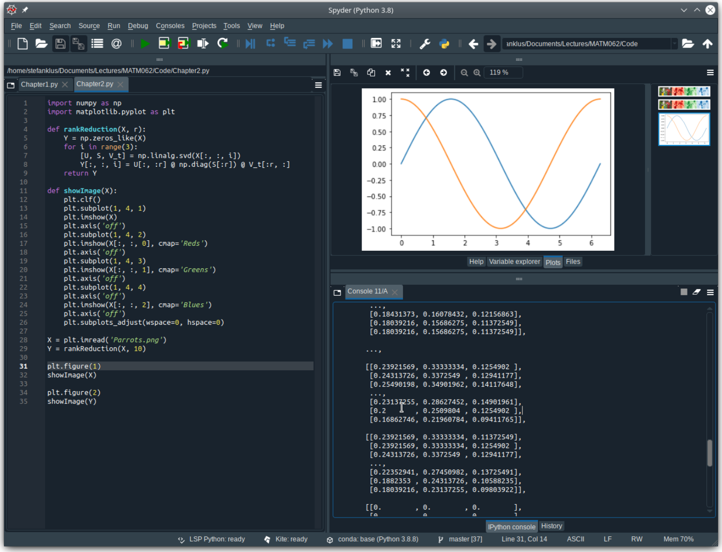
```
conda install numpy scipy matplotlib
```

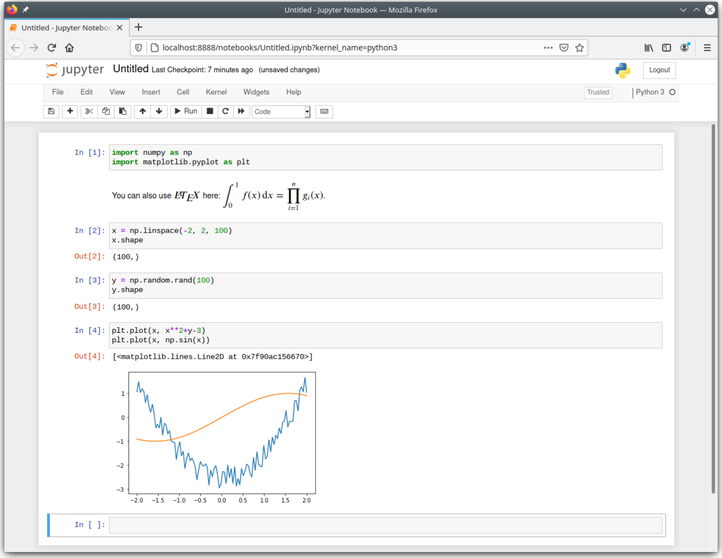
- If you want to use Jupyter notebooks:

```
conda install jupyter
```

- The obligatory “Hello, World!”program:

```
print("Hello, World!")
```


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/stefanklus/Documents/Lectures/MATM062/Code/Chapter2.py
Chapter1.py
Chapter2.py
import numpy as np
import matplotlib.pyplot as plt
def rankReduction(X, r):
Y = np.zeros_like(X)
for i in range(3):
[U, S, V_t] = np.linalg.svd(X[:, :, i])
Y[:, :, i] = U[:, :r] @ np.diag(S[:r]) @ V_t[:r, :]
return Y
def showImage(X):
plt.clf()
plt.subplot(1, 4, 1)
plt.imshow(X)
plt.axis('off')
plt.subplot(1, 4, 2)
plt.imshow(X[:, :, 0], cmap='Reds')
plt.axis('off')
plt.subplot(1, 4, 3)
plt.imshow(X[:, :, 1], cmap='Greens')
plt.axis('off')
plt.subplot(1, 4, 4)
plt.imshow(X[:, :, 2], cmap='Blues')
plt.axis('off')
plt.subplots_adjust(wspace=0, hspace=0)
X = plt.imread('Parrots.png')
Y = rankReduction(X, 10)
plt.figure(1)
showImage(X)
plt.figure(2)
showImage(Y)
119 %
Help Variable explorer Plots Files
Console 11/A
IPython console History
LSP Python: ready
Kite: ready
conda: base (Python 3.8.8)
master [37]
Line 31, Col 14
ASCII LF RW Mem 70%


Untitled - Jupyter Notebook — Mozilla Firefox
Untitled - Jupyter Noteboo
localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3
jupyter
Untitled Last Checkpoint: 7 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [1]:
import numpy as np
import matplotlib.pyplot as plt
You can also use $\LaTeX$ here: $\int_0^1 f(x)\,\mathrm{d}x = \prod_{i=1}^n g_i(x)$.
In [2]:
x = np.linspace(-2, 2, 100)
x.shape
Out[2]:
(100,)
In [3]:
y = np.random.rand(100)
y.shape
Out[3]:
(100,)
In [4]:
plt.plot(x, x**2+y-3)
plt.plot(x, np.sin(x))
Out[4]:
[<matplotlib.lines.Line2D at 0x7f90ac156670>]
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
In [ ]:

- Variable names must start with a letter or an underscore:

```
b = True    # boolean
n = 3       # integer
x = 1.41    # floating point number
z = 1 + 2j  # complex number
s = 'text'  # string
t = None    # empty value
```

- Print the variable:

```
print(b) # True
print(n) # 3
print(x) # 1.41
print(z) # (1+2j)
print(s) # text
print(t) # None
```

- Python automatically assigns a data type:

```
print(type(b)) # <class 'bool'>
print(type(n)) # <class 'int'>
print(type(x)) # <class 'float'>
print(type(z)) # <class 'complex'>
print(type(s)) # <class 'str'>
print(type(t)) # <class 'NoneType'>
```

# Boolean variables

```python
b = [False, True]

for i in b:
    print("!{} = {}".format(i, not i))

# !False = True
# !True = False

for i in b:
    for j in b:
        print("{} & {} = {}".format(i, j, i and j))

# False & False = False
# False & True = False
# True & False = False
# True & True = True

for i in b:
    for j in b:
        print("{} | {} = {}".format(i, j, i or j))

# False | False = False
# False | True = True
# True | False = True
# True | True = True
```

| $A$ | $\neg A$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

| $A$ | $B$ | $A \wedge B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $A$ | $B$ | $A \vee B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# Tuple, list, set, and dict

- Note that Python uses 0-based indexing (unlike Matlab).
- Tuples are ordered and cannot be modified:

```
t = (2, 'a', 11.2, (3, 4), True)
print(t[2]) # 11.2
```

- Lists are ordered and can be modified:

```
l = [5, 4, 3]
l.append(2)    # add 2 to the list
print(l)       # [5, 4, 3, 2]
print(len(l))  # 4

l = l[::-1]    # revert list
print(l)       # [2, 3, 4, 5]

l.insert(1, 7) # insert 7 at the first index
print(l)       # [2, 7, 3, 4, 5]

l.pop()        # remove last item
print(l)       # [2, 7, 3, 4]
```

- Access the last elements of a list using negative indices:

```
print(l[-1]) # 4
print(l[-2]) # 3
print(l[-3]) # 7
```

# Tuple, list, set, and dict

- Sets are unordered collections of variables:

```
s = {'a', 'b', 'c'}

print(s == {'b', 'a', 'c', 'a'}) # True

s.add('d') # add an element to the set
print(s)   # {'c', 'b', 'a', 'd'}

print('a' in s) # True
print('z' in s) # False
```

- Dictionaries are unordered collections of key-value pairs:

```
d = {'red':  [1, 0, 0],
     'green':[0, 1, 0],
     'blue': [0, 0, 1]}

print(d['red'])   # [1, 0, 0]
print(d.keys())   # dict_keys(['red', 'green', 'blue'])
print(d.values()) # dict_values([[1, 0, 0], [0, 1, 0], [0, 0, 1]])

print(d['yellow']) # KeyError: 'yellow'
```

- Use `defaultdict` if you want to provide a default value.

# Module import

- Additional modules can be imported as follows:

```
import math
print(math.factorial(5)) # 120
dir(math) # prints a list of functions contained in math
```

- You can also write your own modules and import them.
- The imported modules need to be in the same directory or contained in the search path for module files.
- In Spyder, open the path manager or use:

```
sys.path.insert(0, "/home/xyz/FolderName")
```

- Import only specific functions:

```
from math import factorial
factorial(5) # note that we can now write factorial instead of math.factorial
```

- It is also possible to give imported modules a new name:

```
import numpy as np                # these are common aliases
import scipy as sp                # which will often be used
import matplotlib.pyplot as plt   # in what follows
```

# For loops, while loops, and range

Python:

```python
for i in range(10):
    print(i)

for i in range(10, 20):
    print(i)

for i in range(20, 30, 2):
    print(i)
```

Matlab:

```matlab
for i = 0:9
    fprintf('%d\n', i);
end
for i = 10:19
    fprintf('%d\n', i);
end
for i = 20:2:28
    fprintf('%d\n', i);
end
```

Loop 1: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$
Loop 2: $10, 11, 12, 13, 14, 15, 16, 17, 18, 19$
Loop 3: $20, 22, 24, 26, 28$

Python:

```python
d = 256
while d > 1:
    d = d//2
    print(d)
```

Matlab:

```matlab
d = 256;
while d > 1
    d = d/2;
    fprintf('%d\n', d);
end
```

Output: $128, 64, 32, 16, 8, 4, 2, 1$

Use / for floating point division and // for integer division.
That is, $5/2 = 2.5$ and $5//2 = 2$.

# Break and continue

Python:

```python
s = 0
for i in range(10):
    s = s + i**2
    if s > 50:
        break
print(s) # 55
```

Matlab:

```matlab
s = 0;
for i = 0:9
    s = s + i^2;
    if s > 50
        break
    end
end
disp(s); % 55
```

The break statement terminates the current loop.

Python:

```python
for i in range(10):
    if i % 2 == 0:
        continue
    print(i) # 1, 3, 5, 7, 9
```

Matlab:

```matlab
for i = 0:9
    if mod(i, 2) == 0
        continue;
    end
    disp(i); % 1, 3, 5, 7, 9
end
```

The continue statement skips all the remaining statements and returns the control to the beginning of the loop.

Python:

```python
def fibonacci():
    x, y = 0, 1
    while True:
        x, y = y, x+y
        yield y

g = fibonacci()

print(next(g)) # 1
# ...
print(next(g)) # 2
# ...
l = [next(g) for i in range(10)] # [3, 5, 8, 13, 21, 34, 55, 89, 144, 233]
```

- If a function contains at least one yield statement, it becomes a generator function.
- Yield pauses the function, saves its current state, and continues when the function is called again.
- There is no comparable Matlab feature.

# If/else statements

Python:

```python
for i in range(1, 5):
    if i % 2 == 0:
        print('even')
    else:
        print('odd')
```

Matlab:

```matlab
for i = 1:4
    if mod(i, 2) == 0
        fprintf('even\n')
    else
        fprintf('odd\n')
    end
end
```

Output: odd, even, odd, even

Python:

```python
if expression:
    statements
elif expression:
    statements
else:
    statements
```

Matlab:

```matlab
if expression
    statements
elseif expression
    statements
else
    statements
end
```

## Indentation

Note that Python relies on the indentation to define a block of code. Other programming languages often use `begin` and `end` or { and } for this purpose.

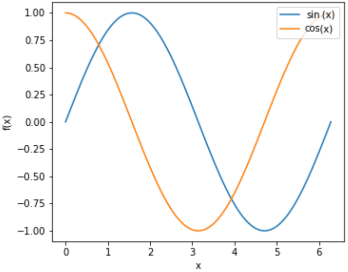
# Lambda functions and anonymous functions

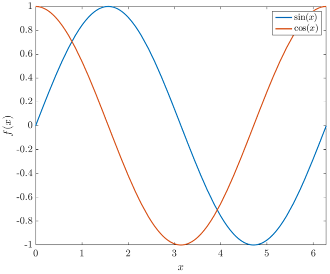
Python:

```python
f = lambda x: math.sin(x)
g = lambda x: math.cos(x)
h = lambda x, y: x**2 + y**2
f(math.pi)
h(2, 3)
x = np.linspace(0, 2*math.pi, 100)
plt.plot(x, np.vectorize(f)(x))
plt.plot(x, np.vectorize(g)(x))
```

Matlab:

```matlab
f = @(x) sin(x);
g = @(x) cos(x);
h = @(x, y) x^2 + y^2;
f(pi)
h(2, 3)
figure; hold on;
fplot(f, [0, 2*pi]);
fplot(g, [0, 2*pi]);
```



# Map, reduce, and filter

- Apply `len` to all items of the list:

```python
tuple_lengths = list(map(len, [(1,), (1, 2), (1, 2, 3)])) # tuple_lengths = [1, 2, 3]
```

- Apply lambda function sequentially to pairs of values:

```python
from functools import reduce
r = reduce((lambda x, y: x * y), [1, 2, 3, 4]) # r = 24
```

- This corresponds to:

```python
r = 1
for i in [1, 2, 3, 4]:
    r = r * i
```

- Filter values for which the lambda function returns `True`:

```python
x = [7, -1, 3, 2, -3, -2]
y = [i for i in filter(lambda x: x>0, x)] # y = [7, 3, 2]
```

- You could also write:

```python
y = [i for i in x if i > 0]
```

- Generate a list of all permutations of $[1, 2, 3]$:

```python
x = [1, 2, 3]
y = list(itertools.permutations(x)) # the generator is converted to a list

print(y) # [(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]
```

- Generate a list of all combinations of $[1, 2, 3, 4]$:

```python
import itertools

x = [1, 2, 3, 4]
y = list(itertools.combinations(x, 3))

print(y) # [(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)]
```

- Compute accumulated sums of $[1, 2, 3, 4, 5]$:

```python
import itertools

x = [1, 2, 3, 4, 5]
y = list(itertools.accumulate(x))

print(y) # [1, 3, 6, 10, 15]
```

# Function arguments

- Passing single argument:

```
def double(n):
    return 2*n
double(2) # 4
```

- Passing multiple arguments to a function:

```
def func(*args): # args is a tuple that contains all parameters
    for i in args:
        print(i)

func(1, 'a', -2)
# 1
# a
# -2
l = [1,'a',-2]
func(*l) # * unpacks the list, same output as above

func(l) # [1, 'a', -2]
```

- Passing multiple named arguments to a function:

```
def func(**kwargs): # kwargs is a dictionary that contains keyword/value pairs
    for keyword, value in kwargs.items():
        print(f'{keyword} = {value}')

func(kw1=1, kw2='a', kw3=-2)
# kw1 = 1
# kw2 = a
# kw3 = -2
```

- Can of course be combined.

# Classes

```
class Fraction(object):
    def __init__(self, a, b):
        '''Initializer or constructor, called when you create a new object.'''
        d = math.gcd(a, b) # greatest common divisor of a and b
        self.a = a // d
        self.b = b // d

    def __repr__(self):
        '''Prints fraction.'''
        return "Fraction {}/{}".format(self.a, self.b)

    def __add__(self, rhs):
        '''Adds two fractions.'''
        p = self.a * rhs.b + self.b * rhs.a
        q = self.b * rhs.b
        return Fraction(p, q)

    def __eq__(self, rhs):
        '''Checks if two fractions are equal.'''
        return self.a * rhs.b == rhs.a * self.b

x1 = Fraction(1, 3) # x1 = 1/3
x2 = Fraction(1, 6) # x2 = 1/6
x3 = x1 + x2

print(x3)                   # Fraction 1/2
print(x3 == Fraction(2, 4)) # True

x4 = x1 - x2 # not implemented yet, define __sub__ function
```

# Inheritance

```python
class Person(object):
    type = 'Person' # this is a class attribute, everything is by default public

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f'Name: {self.name}\nAge: {self.age}'

    def increaseAge(self):
        self.age += 1

class Student(Person): # inherits properties and functions from Person
    type = 'Student'

    def __init__(self, name, age, urn):
        super().__init__(name, age)
        self.urn = urn

    def __str__(self):
        return super().__str__() + f'\nURN: {self.urn}'

c1 = Person('Alice', 21)
c2 = Student('Bob', 22, 720283)
print(c1)         # Name: Alice, Age: 21
print(c1.type)    # Person
print(c2)         # Name: Bob, Age: 22, URN: 720283
print(c2.type)    # Student
c2.increaseAge() # calls the inherited function
print(c2.age)     # 23
```

# Other useful features

- Different string formatting options:

```
x = 5
print("%d^2 = %d" % (x, x**2))     # 5^2 = 25 (old C-style formatting)
print("{}^2 = {}".format(x, x**2)) # 5^2 = 25 (newer)
print(f"{x}^2 = {x**2}")           # 5^2 = 25 (even newer)
```

- Don't add a newline to the string:

```
print("Don't end the line ", end="")
print("here!")
```

- Conversion of data types:

```
a = 123    # integer
b = '456'  # string

a = str(a) # '123'
b = int(b) # 456
```

- In-place operations:

```
a, b, c = 1, 2, 3
a += 2  # 3
b *= 3  # 6
c **= 4 # 81
```

# Other useful features

- Conditional expressions:

```python
def maxVal(a, b):
    return a if a > b else b

print(maxVal(2, 5)) # 5
print(maxVal(7, 5)) # 7
```

- This is similar to the ternary operator in C++:

```cpp
#include <iostream>

template<typename T>
T maxVal(T a, T b)
{
    return a > b ? a : b;
}

int main()
{
    std::cout << maxVal(2, 5) << "\n" << maxVal(7, 5) << std::endl;
}
```

- List comprehension:

```python
l1 = [1, 3, 2, 5, 4, 6]
l2 = [i**2 for i in l1]                  # [1, 9, 4, 25, 16, 36]
l3 = [2*i for i in l2 if i % 2 == 0] # [8, 32, 72]
```

# Other useful features

- Use the keyword `pass` as a placeholder for future code:

```
def toBeImplemented():
    pass
```

- Enumerate items:

```
for index, item in enumerate(['th', 'st', 'nd', 'rd', 'th']):
    print('{}{}'.format(index, item)) # 0th, 1st, 2nd, 3rd, 4th
```

- Create dictionaries:

```
d = dict(x=11, y=22, z=33)
print(d['y']) # 22
```

- Document your functions and classes with docstrings:

```
def sqr(x):
    '''This function returns x squared.'''
    return x**2 # note that in Matlab we would write x^2

print(sqr.__doc__) # This function returns x squared.
```

# Other useful features

- Check whether objects are identical:

```python
a = [1, 2, 3]
b = a.copy()
print(a == b) # True  (compares the value of a and b)
print(a is b) # False (compares the identities of a and b)

s = 'text'
t = s
print(s == t) # True (compares the value of s and t)
print(s is t) # True (compares the identities of s and t)
```

- Note that Python uses reference counting:

```python
from sys import getrefcount

s = 'some random text'
print(getrefcount(s)) # 3
t = s
print(getrefcount(s)) # 4
t = 'new text'
print(getrefcount(s)) # 3
```

- You can also compare the addresses:

```python
s = 'some random text'
t = s
print(f"Address of s: {id(s)}") # 140512670168288
print(f"Address of t: {id(t)}") # 140512670168288
```