Norges miljø- og biovitenskapelige universitet

Institutt for datavitenskap

Digitalisering på Ås

DAT121

Introduction to data science

1 Python basics

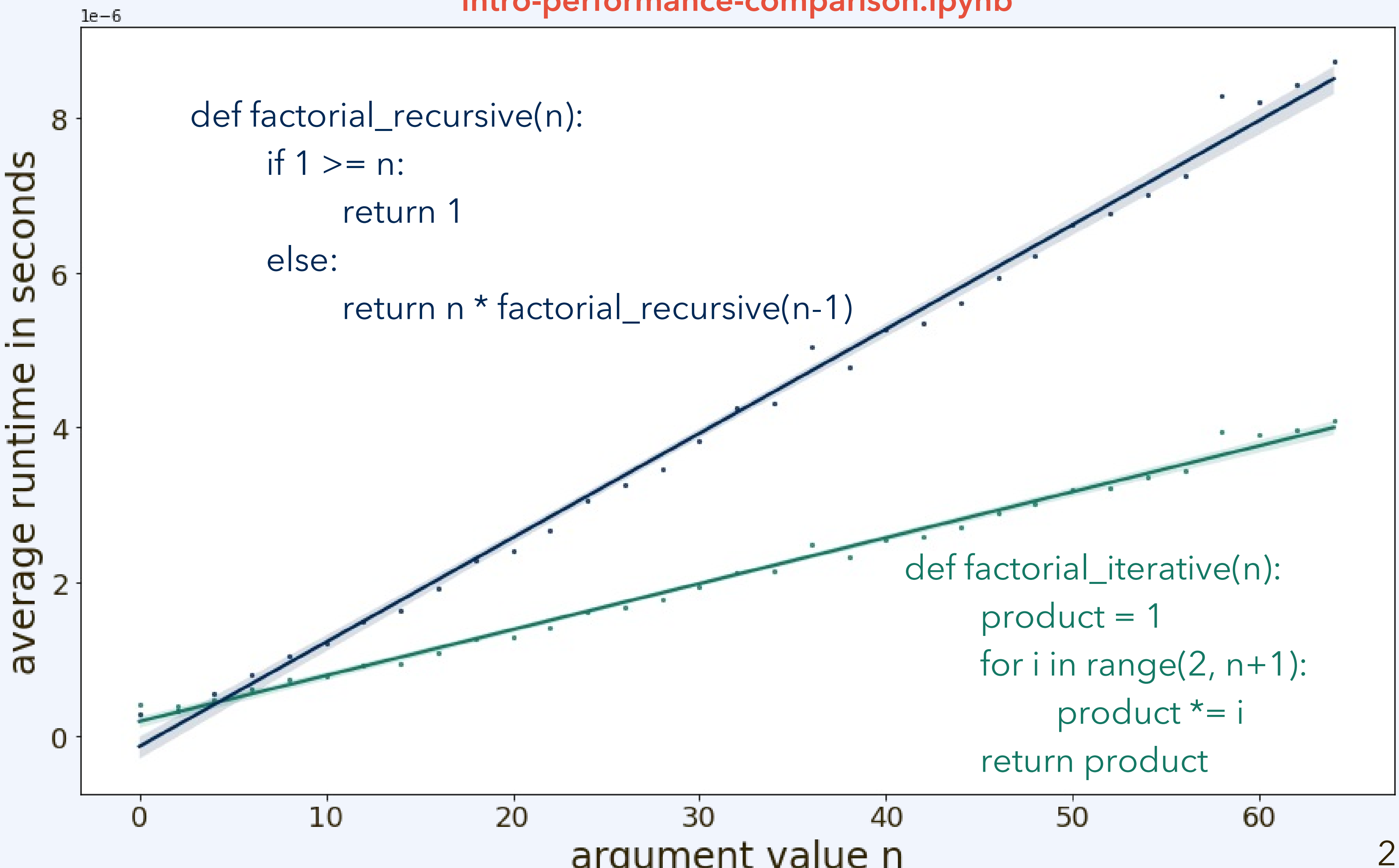
1.3 Python lists

1.4 Object references

1.5 Python libraries

Python example / performance comparison

intro-performance-comparison.ipynb



Python’s built-in container data structures

One of the ways in which Python is unusual is **what is built in**, and **what requires libraries**. The default *container data structure* is a so-called “list.”

- The **Python “list”** is not what is usually called a list when discussing data structures and algorithms in computer science, namely, a linked list. Instead, the Python list is a **dynamic array**.
  - x = [-1/4, "avit", 121, ["dat", 0], [ ]] # this is a list object
  - x[3][0] + x[1] # this is a valid way of using the list
- Python could be unique in that it does *not* natively support *static arrays*.

Python's built-in container data structures

One of the ways in which Python is unusual is **what is built in**, and **what requires libraries**. The default *container data structure* is a so-called "list."

- The **Python "list"** is not what is usually called a list when discussing data structures and algorithms in computer science, namely, a linked list. Instead, the Python list is a **dynamic array**.
  - x = [-1/4, "avit", 121, ["dat", 0], [ ]] # this is a list object
  - x[3][0] + x[1] # this is a valid way of using the list
- Python could be unique in that it does *not* natively support *static arrays*.

Lists are not the only container that is natively supported by Python:

- **Dictionaries** (what is otherwise also called a hash) map keys to values.
  - dict = {key1: val1, key2: val2}
- **Sets**, which contain each element at most once.
- **Tuples**, *i.e.*, immutable ordered collections of objects.

dictionary

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

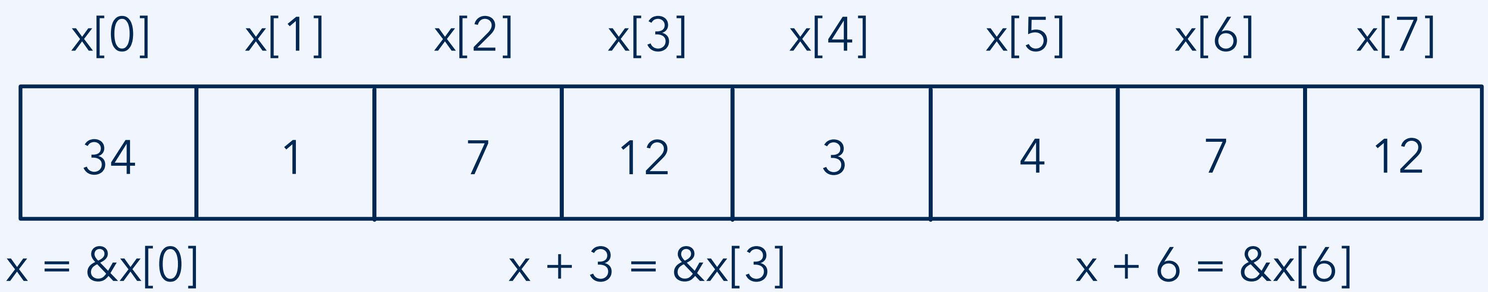
Digitalisering på Ås

1 Python basics

1.3 Python lists

Static arrays

An array contains a sequence of elements of the same type, arranged **contiguously in memory**. This supports fast access using **pointer arithmetics**. Once created, the size of a C/C++ array is fixed; we cannot append elements.



In C/C++, the type of an array such as **int[] is the same as the corresponding pointer type int***, *i.e.*, **the array actually is a pointer**. Its value is an address at which an integer is stored, namely, the memory **address of the first element**.

This is highly efficient since when x[i] is accessed, the compiler transforms this into accessing the memory address x + sizeof(int) * i.

Static arrays

An array contains a sequence of elements of the same type, arranged **contiguously in memory**. This supports fast access using **pointer arithmetics**.

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
34	1	7	12	3	4	7	12

In C/C++, the type of an array such as int[] is the same as the corresponding pointer type int*, *i.e.*, the array actually is a pointer. Its value is an address at which an integer is stored, namely, the memory address of the first element.

In **Python**, we can use the **numpy** library in order to create a static array:

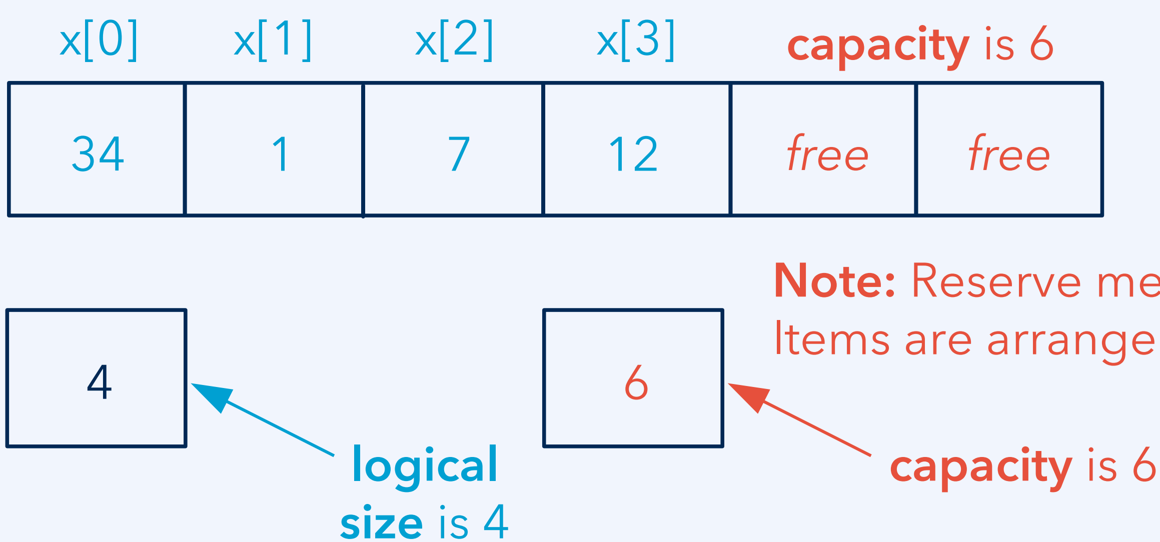
```
import numpy as np
x = np.array( [34, 1, 7, 12, 3, 4, 7, 12] )
```

static array

Lists in Python (dynamic arrays)

Conventional arrays are **static data structures**. Their size in memory is constant, and memory needs to be allocated only once, *e.g.*, at declaration time. (Details depend on programming language, compiler, flags/optimization level, *etc.*).

Dynamic data structures can change in size and/or structure at runtime. For an array, this can be implemented by **allocating reserve memory** for any elements that may be appended in the future. When the capacity of the dynamic array is exhausted, all of its contents need to be shifted to another position in memory.



in Python:

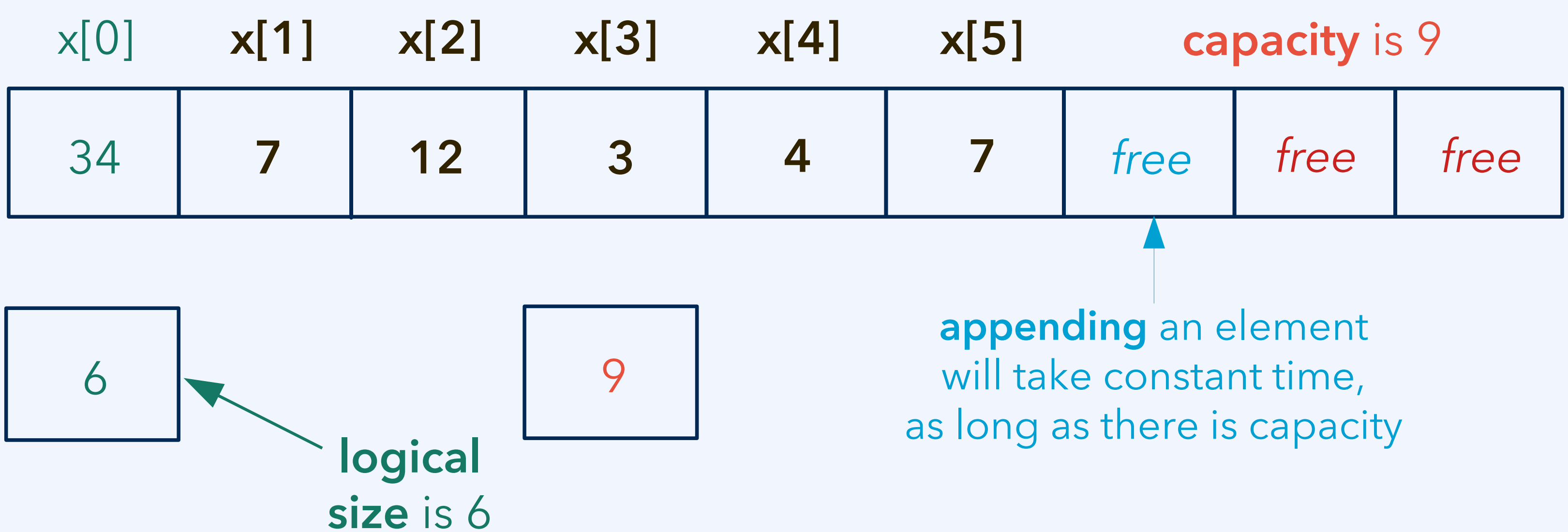
x = [34, 1, 7, 12]

Note: Reserve memory capacity is allocated. Items are arranged contiguously in memory.

Lists in Python (dynamic arrays)

dynamic array

- Read/write access to an array element: **O(1) time.**
  Address of the i-th element computable by pointer arithmetics.
- Deleting an element from the array: **O(1) at the end, O(*n*) elsewhere. All the elements with greater indices need to be shifted.**
- **Extending the array by one element?** O(1) at the end, if there is capacity. O(*n*) elsewhere, or if the capacity of the dynamic array is exhausted.



Lists in Python: Examples

Lists in Python are implemented as dynamic arrays. Their elements behave in the same way as Python variables do in general: **They are object references.**

In [1]:
```python
x = list(range(7))
y = x[2: 4]
y[0] = 7

print("x = ", x)
print ("y = ", y)
```

```
x =  [0, 1, 2, 3, 4, 5, 6]
y =  [7, 3]
```

In [3]:
```python
x = [[i] for i in range(7)]
y = x[2: 4]
y[0].pop()
y[0].append(7)

print("x = ", x)
print ("y = ", y)
```

```
x =  [[0], [1], [7], [3], [4], [5], [6]]
y =  [[7], [3]]
```

Q: What happens if these two lines are replaced with **y[0] = [7]**?

When a sublist x[i: j] is created from x, **all the sublist elements are copied**.

The behaviour above is closely tied to the use of **object references** in Python.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

1 Python basics

Argument passing (glossary)

Definition: **Argument passing** is the process of handing over data items to a procedure (*e.g.*, a function in Python) when that procedure is called.

- The *free variables of a procedure* are called **parameters**. In other words, parameters are the *names* of the variables passed to the function. **Arguments** are *concrete values* associated with the parameters at runtime.
- It is therefore more correct to speak of "argument passing" than "parameter passing," and that is also more common in English. In Norwegian, however, the expression with the word parameter ("parameteroverføring") seems to be more common.
- Most languages provide "pass by value" or "pass by reference," or both, as a mechanism. **Pass by value** means that the *value* of a data item is handed over directly, while in **pass by reference**, the procedure receives the *memory address* of that value.
- The argument passing mechanism in Python is called "**pass by object reference**," which means that *an object reference is passed by value*.

argument passing

Pass by value and pass by reference

Two major ways in which arguments can be passed to functions are “**by value**,” where the function receives the *value contained by the variable*, and “**by reference**,” where the function receives the *memory address of the variable*.

Consider the following C/C++ code:

```
int metric(int a, int b)
{
   return (a - b) * (a - b);
}

…
int c;
c = metric(2, 5);
```

a and b are passed by value.
c is assigned the return value, which
is also communicated by value.

Pass by value and pass by reference

Two major ways in which arguments can be passed to functions are “**by value**,” where the function receives the *value contained by the variable*, and “**by reference**,” where the function receives the *memory address of the variable*.

Compare the following more or less equivalent C/C++ codes:

```
int metric(int a, int b)
{
   return (a - b) * (a - b);
}

…
int c;
c = metric(2, 5);
```

a and b are passed by value.
c is assigned the return value, which is also communicated by value.

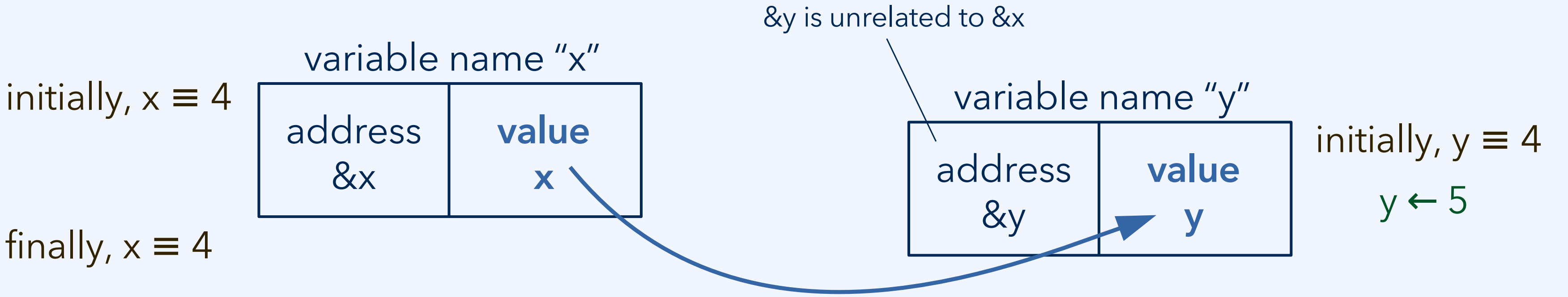
```
void metric(int a, int b, int& distance)
{
   distance = (a - b) * (a - b);
}

…
int c;
metric(2, 5, c);
```

a and b are passed by value.
c is passed by reference, and the function metric can access its memory address.

Pass by value and pass by reference

Two major ways in which arguments can be passed to functions:

Argument passing by value



Argument passing by reference

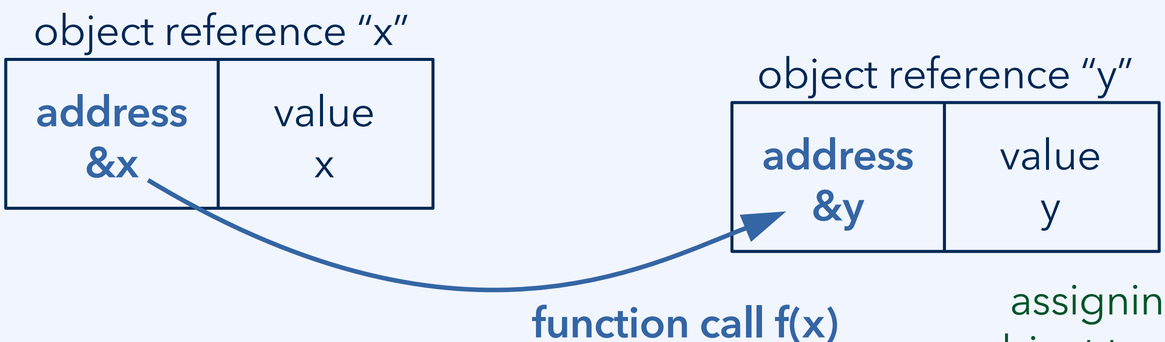
Pass by object reference

In Python, object references are passed by value (i.e., "pass by object reference"):

Argument passing by object reference in Python (similarly, in Java)

initially, x ≡ [ ]

finally, x ≡ [1]



initially, &y ≡ &x,
hence y ≡ [ ]

y.append(1)

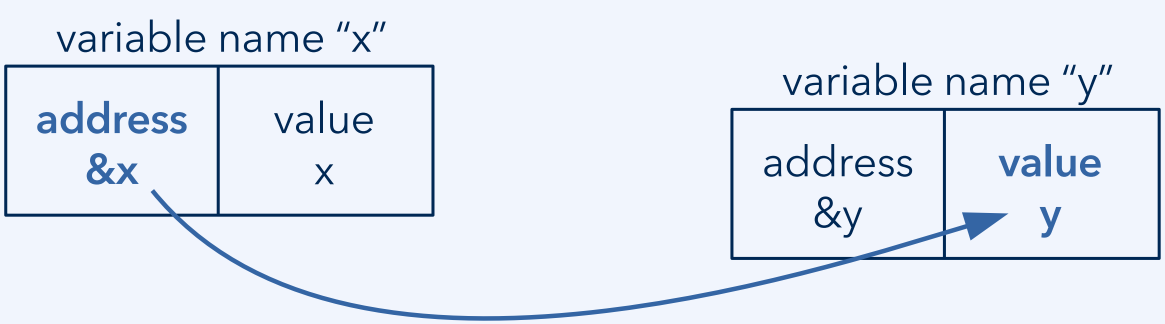
y ← [1, 2]

assigning a new object to y changes the address!

Argument passing by reference

initially, x ≡ 4

finally, x ≡ 5



initially, y ≡ &x,
hence *y ≡ 4

*y ← 5

Object reference (glossary)

object reference

Definition: A **reference** is an alias for *data stored at a certain memory address*. An **object reference** is a *reference to an object*; the memory address is hidden from the programmer, who can use the reference as if it was the object itself.

- Object-oriented programming languages usually distinguish between **classes** and **elementary data types**, and consequently between object variables (and their values, which are usually **objects**) and elementary variables (and their values, which are usually **elementary data items**).
  - Not so *in Python*, where *elementary data are objects* as well.
- In languages that use references (or pointers), there is typically a distinction between the **references/pointers**, which hold a **memory address**, and **normal variables** which directly hold a **value**.
  - Not so *in Python*, where *every variable is an object reference*.

Functions as objects

In Python it is possible to assign an *object reference to a function* to a variable.

```
def factorial_recursive(n):
    if 1 >= n:
        return 1
    else:
        return n * factorial_recursive(n-1)
```

```
def factorial_iterative(n):
    product = 1
    for i in range(2, n+1):
        product *= i
    return product
```

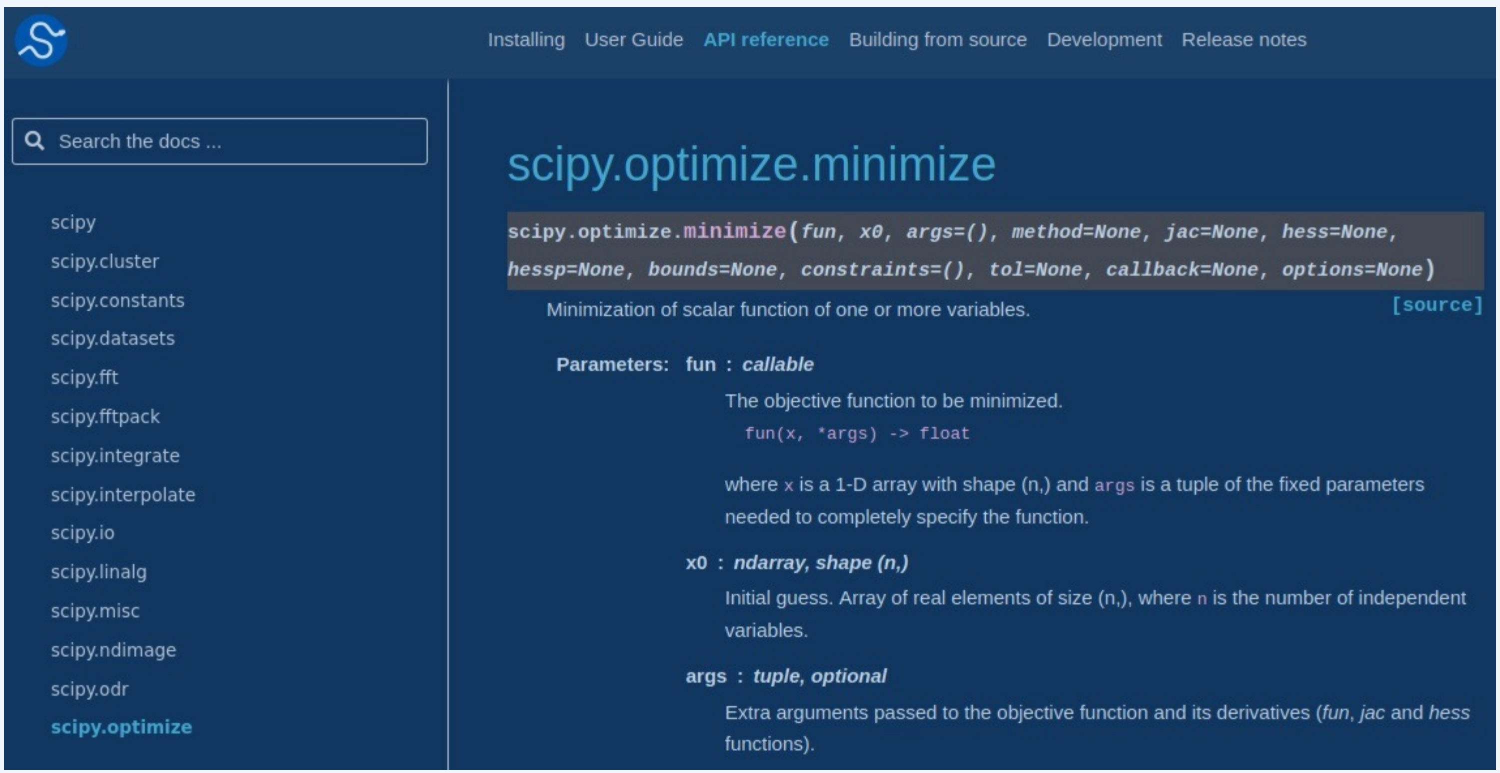
```
x = factorial_iterative
y = factorial_recursive
print( x(10) / y(8) )
```

→ 90.0

Functions as objects

In Python it is possible to assign an *object reference to a function* to a variable. This means that functions can also be passed as arguments to other functions.

One common use of this is during optimization, using scipy.opt.minimize:



Installing User Guide API reference Building from source Development Release notes

Search the docs ...

scipy
scipy.cluster
scipy.constants
scipy.datasets
scipy.fft
scipy.fftpack
scipy.integrate
scipy.interpolate
scipy.io
scipy.linalg
scipy.misc
scipy.ndimage
scipy.odr
scipy.optimize

scipy.optimize.minimize

```
scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None)
```

[source]

Minimization of scalar function of one or more variables.

Parameters: **fun** : *callable*

The objective function to be minimized.

```
fun(x, *args) -> float
```

where `x` is a 1-D array with shape (n,) and `args` is a tuple of the fixed parameters needed to completely specify the function.

x0 : *ndarray, shape (n,)*

Initial guess. Array of real elements of size (n,), where `n` is the number of independent variables.

args : *tuple, optional*

Extra arguments passed to the objective function and its derivatives (*fun*, *jac* and *hess* functions).

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

1 Python basics

Important libraries in Python

In Python, it is very rare to write code without using any libraries at all.
The following ones are among the most important:

math: Contains e.g. functions like math.factorial() and constants like math.pi.

matplotlib: Used for simple diagrams; its figure and axis objects are used by other libraries as well. Discussed in Chapter 9 of *Python for Data Analysis.*

numpy: Functionalities that turn Python into a suitable replacement for Matlab, dealing efficiently with (static) arrays and matrices. (Chapter 4 in the book.)

pandas: Deals with data that are arranged as tables. (Chapter 5 in the book.)

scipy: Scientific computing functionalities, e.g., optimization and linear algebra.

seaborn: Library of choice for some frequently used kinds of plots.

statsmodels: Linear and non-linear regression and statistical data analysis.

Important libraries in Python

The following ones are among the most important:

math: Contains e.g. functions like math.factorial() and constants like math.pi.

matplotlib: Used for simple diagrams; its figure and axis objects are used by other libraries as well. Discussed in Chapter 9 of *Python for Data Analysis*.

numpy: Functionalities that turn Python into a suitable replacement for Matlab, dealing efficiently with (static) arrays and matrices. (Chapter 4 in the book.)

pandas: Deals with data that are arranged as tables. (Chapter 5 in the book.)

scipy: Scientific computing functionalities, e.g., optimization and linear algebra.

seaborn: Library of choice for some frequently used kinds of plots.

statsmodels: Linear and non-linear regression and statistical data analysis.

Relevant Python libraries for the semantic web include **RDFlib** and **owlready2**.

numpy

We already saw that numpy can be used to create static arrays.
Frequent uses of numpy include:

- Multidimensional arrays, *e.g.*, x = np.array([[1, 2, 3], [4, 5, 6]])
  - Both for one- and multidimensional arrays, arithmetics can be carried out on an element-by-element basis, e.g., x = 1/x
  - Additionally, many numpy functions operate over arrays
- Random numbers from np.random, e.g., uniformly distributed value between 0 and 1 obtained from np.random.random()
- Python's range(init, limit, step) creates lists of equidistant integers. With np.arange(init, limit, step), an array is created instead, but more importantly, non-integer values can be used, which is often needed.

See Chapter 4 in *Python for Data Analysis* for an overview and code examples.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

Conclusion

Related teaching activities

Python programming competency is relevant cross-sectionally in data science.

INF201: Advanced programming (5 credits, autumn term – Jonas Kusch)

The module builds on solid basic programming skills, using Python as the programming language. As a student in INF201, you write and improve code on your own throughout the autumn parallel. The module discusses techniques for debugging, optimization and benchmarking, testing and documentation, and scientific data processing. It introduces good practices for writing resilient object-oriented code and developing user interfaces.

Related teaching activities

Python programming competency is relevant cross-sectionally in data science.

INF201: Advanced programming (5 credits, autumn term – Jonas Kusch)

The module builds on solid basic programming skills, using Python as the programming language. As a student in INF201, you write and improve code on your own throughout the autumn parallel. The module discusses techniques for debugging, optimization and benchmarking, testing and documentation, and scientific data processing. It introduces good practices for writing resilient object-oriented code and developing user interfaces.

INF202: Advanced programming project (5 credits, January block – Jonas Kusch)
INF203: Advanced programming project (5 credits, June block – Jonas Kusch)

Programming competency can only be developed through sustained practice. The INF201 and INF203 modules supplement the methodology-oriented modules INF201 (advanced programming) and INF205 (resource-efficient programming) by the required practice.

Related research and development activities

Python library development by NMBU data science and industrial economics:[1]

hoggorm

license BSD License | docs passing | JOSS 10.21105/joss.00980 | codecov 88% | ci-build no status | code quality ! | openssf best practices in progress 33%

hoggorm is a Python package for explorative multivariate statistics in Python. It contains the following methods:

- PCA (principal component analysis)
- PCR (principal component regression)
- PLSR (partial least squares regression)
  - PLSR1 for single variable responses
  - PLSR2 for multivariate responses
- matrix correlation coefficients RV, RV2 and SMI.

[1] https://github.com/olivertomic/hoggorm

Related research and development activities

Python library development by NMBU data science and industrial economics:[1,2]

hoggorm

license BSD License | docs passing | JOSS 10.21105/joss.00980 | codecov 88% | ci-build no status | code quality ! | openssf best practices in progress 33%

hoggorm is a Python package for explorative multivariate statistics in Python. It contains the following methods:

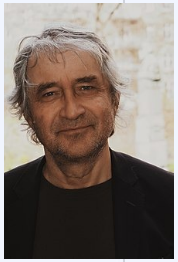
- PCA (principal component analysis)
- PCR (principal component regression)
- PLSR (partial least squares regression)
  - PLSR1 for single variable responses
  - PLSR2 for multivariate responses
- matrix correlation coefficients RV, RV2 and SMI.

Nofima

O. Tomic

T. Graff

K. H. Liland



T. Næs

Unlike scikit-learn, which is an excellent python machine learning package focusing on classification, regression, clustering and predicition, hoggorm rather aims at understanding and interpretation of the variance in the data. hoggorm also contains tools for prediction. The complementary package hoggormplot can be used for visualization of results of models trained with hoggorm.

[1] O. Tomic, T. Graff, K. H. Liland, T. Næs, *J. Open Source Softw.* **4**(39): 980, doi:10.21105/joss.00980, **2019**.

[2] https://github.com/olivertomic/hoggorm

Glossary terms

Proposed glossary[1] terms:

- How do we best define them? Is the definition controversial?
- What is the best translation into Norwegian bokmål and nynorsk?
- Are there more key concepts that would require an agreed definition?

argument passing
(specifically: pass by object reference)

dynamic array
(specifically: list data structure in Python)

dictionary
(also: hash)

left open for discussion

object reference

static array

[1] https://home.bawue.de/~horsch/teaching/dat121/glossary-en.html

Norges miljø- og biovitenskapelige universitet

Institutt for datavitenskap

Digitalisering på Ås

DAT121

Introduction to data science

1 Python basics

1.3 Python lists

1.4 Object references

1.5 Python libraries

Schedule for 14th and 15th August

Monday, 14th August 2023

09.30 - 09.45* informal meet-up *(in room TF2-323b)*

10.00 - 11.00* lecture by Swati Aggarwal* *(in room TF1-205)*

11.15 - 12.00 first lecture, welcome/Python *(in room TF1-201)*

13.15 - 15.00 Data Science semester start jointly with the 5-year Master *(in room TF1-115)*

Tuesday, 15th August 2023

09.00 - 10.00* lecture by Alexander Stasik* *(in room TF1-205)*

10.15 - 11.00 round of introductions *(in room TF1-201)*

11.15 - 12.00 second lecture on Python

13.15 - 14.00 discussion about potential DAT121 presentation topics

14.15 - 15.00 tutorial session *(ctd.)*

*not part of the official programme

Schedule for 17th and 18th August

Thursday, 17th August 2023

09.15 - 10.00 discussion and Q&A

10.15 - 11.00 first lecture on data and objects

11.15 - 12.00 problem solving and/or presentation by Kristian Liland

13.15 - 15.00 tutorial session

Friday, 18th August 2023

09.15 - 10.00 discussion and Q&A

10.15 - 11.00 second lecture on data and objects

11.15 - 12.00 Fadi Al Machot's presentation on research and Master topics

The afternoon of 18th August is reserved for the immatriculation.