Norges miljø- og
biovitenskapelige
universitet

Institutt for datavitenskap

Digitalisering på Ås

DAT121
Introduction to data science

Welcome to Data Science at NMBU

Welcome to Data Science at NMBU!

Faculty at the Department of Data Science (Institutt for datavitskap)

Fadi Al Machot

machine learning and formal methods

Martin Thomas Horsch

digitalization and process data technology

Jonas Kusch

numerical methods

Hans Ekkehard Plesser

computational neuroscience and HPC

Oliver Tomic

statistical methods and models

Kristin Tøndel

metamodelling and chemometrics

Habib Ullah

image recognition and digital agri-/aquaculture

Eirik Valseth

continuum mechanics

???

Data Science students' union

The data science **linjeforening**

- contributes to students' social and academic activities
- facilitates collaboration across faculties and courses of study
- establishes links between students and industry

Students can **register for membership individually**, via this link:
https://docs.google.com/forms/d/1ccQUkPIqFvKLqWDUn6hFIEwMstLrtYgGx20UfY2u3zg/

datasci@nmbu.no

http://datascinmbu.no/

Course curriculum support

REALTEK's studieveiledning (course curriculum support) from office TF3-105b:

Contact email: studieveileder-realtek@nmbu.no

Course curriculum in the 2-year data science master

Modules relevant to the two-year master in data science include:

Year 1, August	DAT121 *Introduction to data science*		
Year 1, autumn	INF201 *Advanced programming*	INF230 *Data management and analysis*	
Year 1, spring	DAT200 *Applied machine learning*	INF221 *Informatics for data scientists*	MATH280 *Applied linear algebra*
Year 2, autumn	DAT300 *Applied deep learning*	DAT390 *Data science seminar*	
Year 2, spring	M.Sc. thesis work		

Course curriculum in the 2-year data science master

Modules relevant to the two-year master in data science include:

Year 1, August

- DAT121 *Introduction to data science* **Martin Thomas Horsch**

Year 1, autumn

- INF201 *Advanced programming* **Jonas Kusch**
- INF230 *Data management and analysis* **Habib Ullah**
- *IND310* *Industrial optimization* **Habib Ullah**

Year 1, January

- *STAT200* *Regression*

Year 1, spring

- DAT200 *Applied machine learning* **Fadi Al Machot**
- *INF205* *Resource-efficient programming* **Martin Thomas Horsch**
- INF221 Informatics for data scientists
- MATH280 *Applied linear algebra*
- *INN351* Enterprise architecture

Year 1, June

- *INF203* *Advanced programming project* **Jonas Kusch**

Year 2, autumn

- DAT300 *Applied deep learning* **Fadi Al Machot**
- DAT390 *Data science seminar* **Kristin Tøndel**

Year 2, spring

- M.Sc. thesis work

DAT121 module structure

1. Python basics: This part is to make sure that everybody knows some Python, which we cannot necessarily assume from before.

2. Data and objects: Object-oriented programming and data management based on classes and relations (E-R diagrams, ontologies).

3. Regression basics: How to do linear and simple non-linear regression in Python, mainly using statsmodels, and interpret the results.

4. Good practice: There is so much bad practice in dealing with data! Here we together hopefully figure out how to do it better …

5. Multidimensionality: Discusses dimensionality analysis and decision support based on multicriteria optimization.

Grade: Passed/not passed "mappevurdering" from tutorial & submitted slides.

DAT121 module structure: Passing criterion

1. Python basics: This part is to make sure that everybody knows some Python, which we cannot necessarily assume from before.

2. Data and objects: Object-oriented programming and data management based on classes and relations (E-R diagrams, ontologies).

3. Regression basics: How to do linear and simple nonlinear regression in Python, mainly using statsmodels, and interpret the results.

4. Good practice: There is so much bad practice in dealing with data! Here we together hopefully figure out how to do it better.

5. Multidimensionality: Discusses dimensionality analysis and decision support based on multicriteria optimization.

There will be **five tutorial worksheets**, to be submitted on Canvas.

The **presentation** (slides) count as equivalent to three worksheets, so that there is a total of eight elements.

For the **pass grade**, you need to **demonstrate a substantial engagement** with some of the topic on **at least five out of eight**, and at least minimal engagement on at least six out of eight.

You are very **welcome to work in groups!** Please still submit your content individually. Where you worked as a group, **explain how you contributed** to clarify if it was substantial.

Grade: Passed/not passed “mappevurdering” from tutorial & submitted slides.

DAT121 module structure & calendar

1. Python basics (14th and 15th August): This part is to make sure that everybody knows some Python, which we cannot necessarily assume from before.

2. Data and objects (17th and 18th August): Object-oriented programming and data management based on classes and relations (E-R diagrams, ontologies).

3. Regression basics (21st and 22nd August): How to do linear and simple non-linear regression in Python, mainly using statsmodels, and interpret the results.

4. Good practice (23rd and 25th August): There is so much bad practice in dealing with data! Here we together hopefully figure out how to do it better …

5. Multidimensionality (28th and 29th August): Discusses dimensionality analysis and decision support based on multicriteria optimization.

Student presentations will be on the last two days: 31st August, 1st September.

Norges miljø- og
biovitenskapelige
universitet

Institutt for datavitenskap

Digitalisering på Ås

DAT121
Introduction to data science

1 Python basics

1.1 Python as a script language
1.2 Distinguishing features of Python

Schedule for 14th and 15th August

Monday, 14th August 2023

11.15 - 12.00	official start of DAT121 and 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

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.)*

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	official start of DAT121 and 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)*

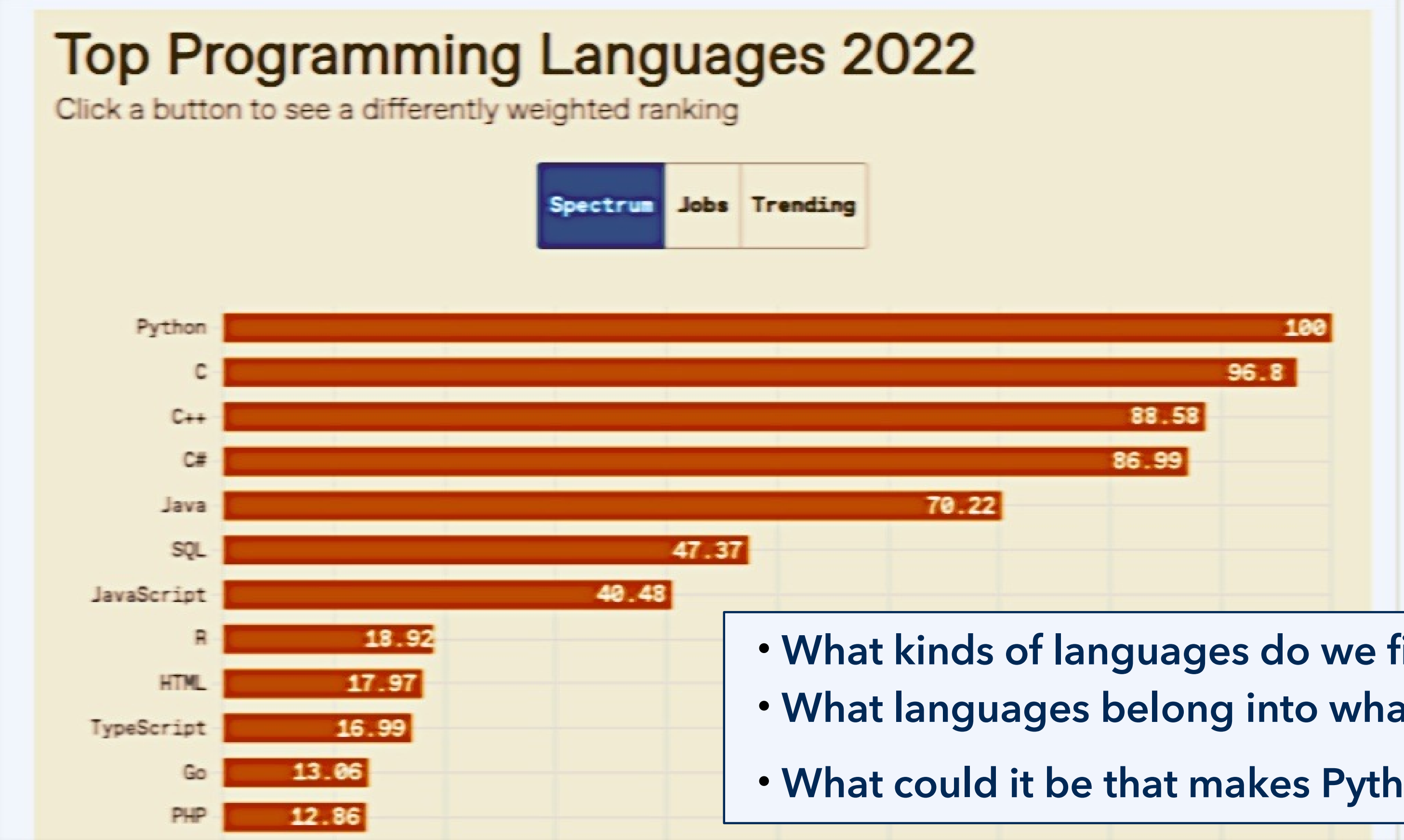
Tuesday, 15th August 2023

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, but all are most welcome to attend

Why Python?

As a discussion opener, see the IEEE Spectrum's top programming languages:[1]



- What kinds of languages do we find here?
- What languages belong into what categories?
- What could it be that makes Python so popular?

[1] S. Cass, *IEEE Spectrum*, https://spectrum.ieee.org/top-programming-languages-2022, **2022**.

Noregs miljø- og
biovitskaplege
universitet

Institutt for datavitskap

Digitalisering på Ås

1 Python basics

1.1 Python: A script language

Interpreted and compiled programming languages

A **script language** is used for writing scripts, *i.e.*, programs that require a run-time environment, such as an **interpreter** or shell, that executes the code step by step.

- In compiled languages, a compiler is used to generate an executable binary file from the code. That is not the case for a script language (interpreted language). Instead, the *code is processed by the interpreter every time* it is run.

Programming languages cannot be perfectly classified into "compiled" and "interpreted."

- Most script languages *can* be compiled, it is just not how they are typically used.
- Java uses a run-time environment, but programs need to be compiled for it.
- Logic programs (*e.g.*, in Prolog) use reasoners, sharing some features with typical interpreters, but they don't necessarily proceed step by step in a fixed order.

script language

What are the main **advantages of a script language**?

How many of the "top 12" languages from the previous slide are script languages? How many are compiled languages?

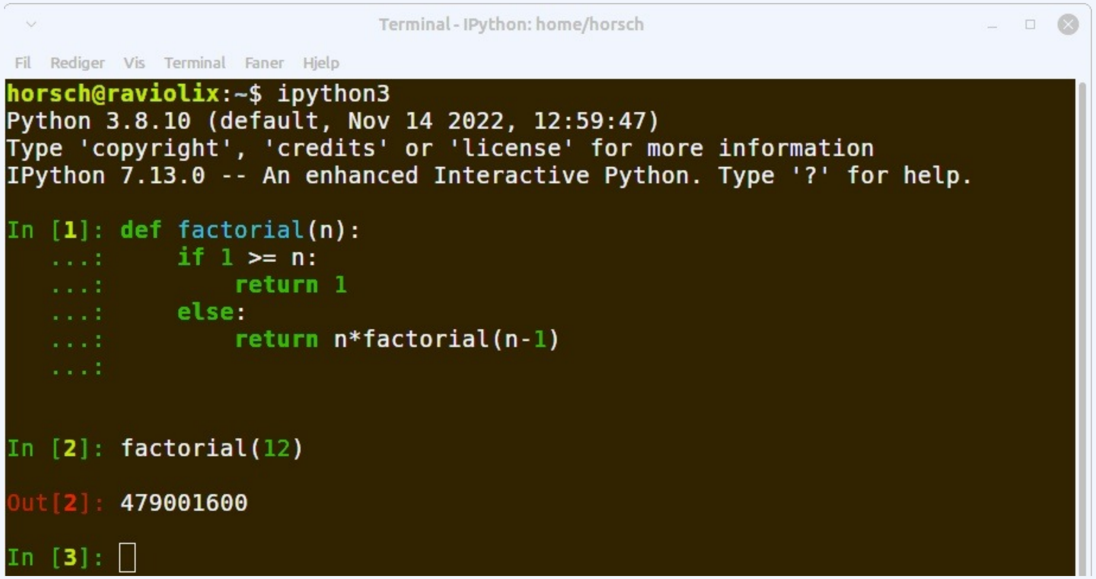
Batch and interactive interpretation

Python scripts, as **source code** files usually with a ***.py file ending**, can be run by calling the **python3** interpreter.

Or as a **shell script**:

#!/usr/bin/python3
print("Hello world!")

Python scripts can be entered line by line and run in an **interactive mode** using **ipython**.



```
horsch@raviolix:~$ ipython3
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: def factorial(n):
   ...:     if 1 >= n:
   ...:         return 1
   ...:     else:
   ...:         return n*factorial(n-1)
   ...:


In [2]: factorial(12)

Out[2]: 479001600

In [3]: 
```

However, the most popular environment for Python is **… Jupyter Notebook**.

Jupyter notebook

Python is very often run from **Jupyter Notebook**. Instead of Python code files (file ending *.py), Jupyter Notebook files (ending *.ipynb) are then used. Jupyter notebook is part of any normal installation of Python or *e.g.* Julia.

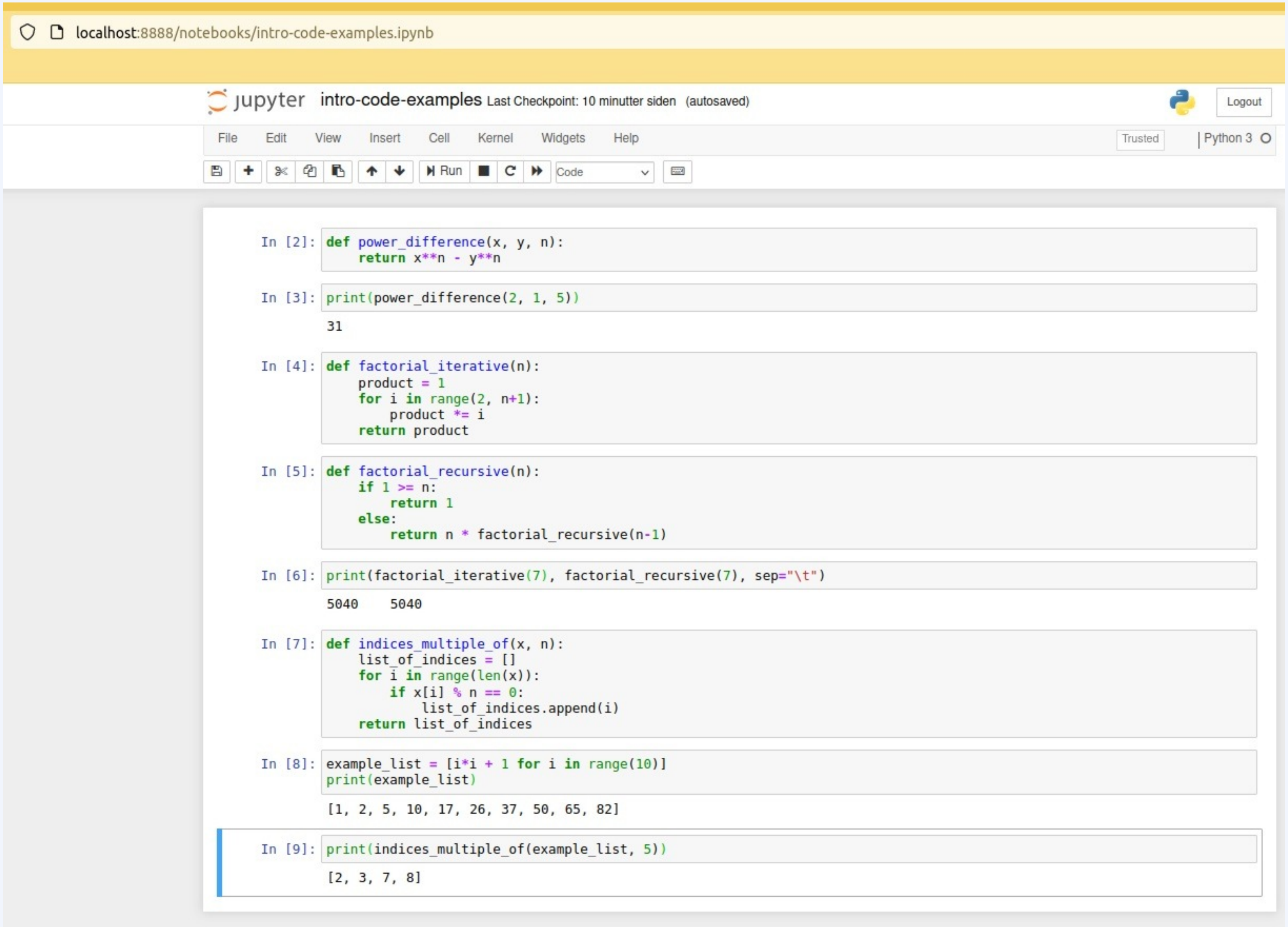
Executing **jupyter-notebook** will

- start a local web service, by default on port 8888,
- in most environments automatically open localhost:8888 in the browser.

Advantages of using Jupyter Notebook:

- "Cells" with code can be combined with cells containing text/explanations
- Easy to integrate and use code that has a visual output such as diagrams
- The user can execute a single cell, or the whole script, manually as needed
- Debugging is also made substantially easier in this way

Jupyter notebook


localhost:8888/notebooks/intro-code-examples.ipynb
jupyter
intro-code-examples Last Checkpoint: 10 minutter siden (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

```
In [2]: def power_difference(x, y, n):
            return x**n - y**n
```

```
In [3]: print(power_difference(2, 1, 5))
```

```
31
```

```
In [4]: def factorial_iterative(n):
            product = 1
            for i in range(2, n+1):
                product *= i
            return product
```

```
In [5]: def factorial_recursive(n):
            if 1 >= n:
                return 1
            else:
                return n * factorial_recursive(n-1)
```

```
In [6]: print(factorial_iterative(7), factorial_recursive(7), sep="\t")
```

```
5040    5040
```

```
In [7]: def indices_multiple_of(x, n):
            list_of_indices = []
            for i in range(len(x)):
                if x[i] % n == 0:
                    list_of_indices.append(i)
            return list_of_indices
```

```
In [8]: example_list = [i*i + 1 for i in range(10)]
        print(example_list)
```

```
[1, 2, 5, 10, 17, 26, 37, 50, 65, 82]
```

```
In [9]: print(indices_multiple_of(example_list, 5))
```

```
[2, 3, 7, 8]
```

Programming paradigms

Imperative programming

- It is stated, instruction by instruction, what the processor should do
- Control flow implemented by jumps (**goto**)

Structured programming

- Same, but with **higher-level control flow**
- Contains "instruction by instruction" code

Procedural programming

- **Functions** (procedures) as **highest-level structural unit** of code
- Still contains loops, *etc.*, for control flow within a function

Object-oriented programming (OOP)

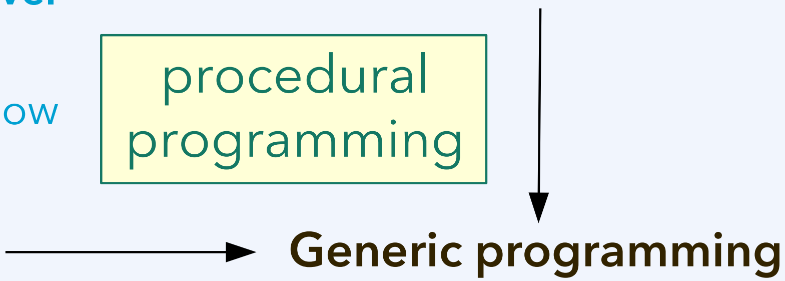
- **Classes** as **highest-level structural unit** of code; objects instantiate classes
- Still contains functions, *e.g.*, as methods

Programming paradigms based on **describing the solution** rather than computational steps:

Functional programming
(also: "declarative programming")

Constraint programming

Logic programming



procedural programming

Generic programming
(introduces ideas from declarative and logical methods into OOP)

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

1 Python basics

1.1 Python: A script language

1.2 Distinguishing features

Code examples

intro-code-examples.ipynb

```python
def power_difference(x, y, n):
    return x**n - y**n

print(power_difference(2, 1, 5))
```

```
31
```

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

print(factorial_iterative(5))
```

```
120
```

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

print(factorial_recursive(5))
```

```
120
```

Code examples

```
example_list = [i*i + 1 for i in range(10)]
print(example_list)
```

```
[1, 2, 5, 10, 17, 26, 37, 50, 65, 82]
```

```
def indices_multiple_of(x, n):
    list_of_indices = []
    for i in range(len(x)):
        if x[i] % n == 0:
            list_of_indices.append(i)
    return list_of_indices

y = indices_multiple_of(example_list, 5)
print(y)
```

```
[2, 3, 7, 8]
```

Control flow

scope

global variable

Observations about control flow in Python:

- Using Jupyter Notebook, we can execute cells at will in any order, and they share access to the same space of global variables and names.

A **global variable** is a variable that can be accessed through a name with an unrestricted **scope**. It has a name that resolves everywhere in the code.

Some say that it is *bad style* to use any *global variables* at all. The main reason is that it is *hard to debug or verify* what a code does if it relies on *write access to global variables*. Despite this, it is common practice to use global variables in script languages.

- Most control flow elements are easily recognizable from other languages: if statements, loops, functions, methods (of classes), …
- Iteration over container types (lists, dictionaries, …) is comparably simple.
- It is comparably easy to have a function return multiple objects.

Syntactic features of Python

Observations about control flow in Python:

- Using Jupyter Notebook, we can execute cells at will in any order, and they share access to the same space of global variables and names.

Discussion: Distinguishing features of Python's syntax

In what ways does the syntax of Python look different from the other programming languages that you know?

- Most control flow elements are easily recognizable from other languages: if statements, loops, functions, methods (of classes), ...
- Iteration over container types (lists, dictionaries, ...) is comparably simple.
- It is comparably easy to have a function return multiple objects.

Noregs miljø- og
biovitskaplege
universitet

Institutt for datavitskap

Digitalisering på Ås

Conclusion

Terminology: Glossary building

We are building a glossary[1] for DAT121, so that:

- … we can **discuss and agree upon**[2] **the meaning of terms** that are very important, or that were unclear or might be used in conflicting ways.
- … identify problems with concepts and **encourage critical thinking**.[2]

Proposed glossary entries related to today's lecture (be welcome to add more):

procedural programming

script language

(also: interpreted language)

dynamic typing

global variable

scope

left open for discussion

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

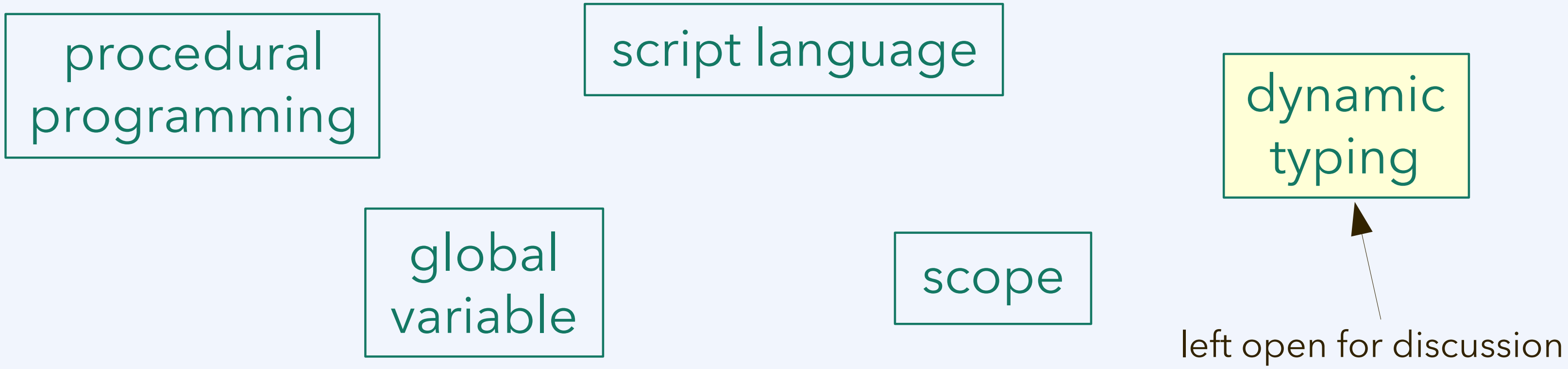
[2] Freedom and critical thinking take the priority – we need not and maybe cannot agree on all terms.

Remark: Dynamic typing

In Python, the type of a variable does not need to be declared by the programmer. A variable can even change type when it is assigned a new value:

```
x = 5
print(x)
x = "5"
print(x)
```

```
def print_twice(x):
        print(x + x)
print_twice(5)          → ???
print_twice("5")        → ???
```



What do we need to do for a similar effect in a compiled language like C++?

Norwegian disciplinary language

Goals:

- Understand both English and Norwegian academic and technical terminology, irrespective in what language the teaching is done.
- Maintain status of Norwegian as a complete language, covering all fields.

See also the Government's action plan[1] on Norwegian disciplinary language.

Discussion: What are good sources for terminology in Norwegian?

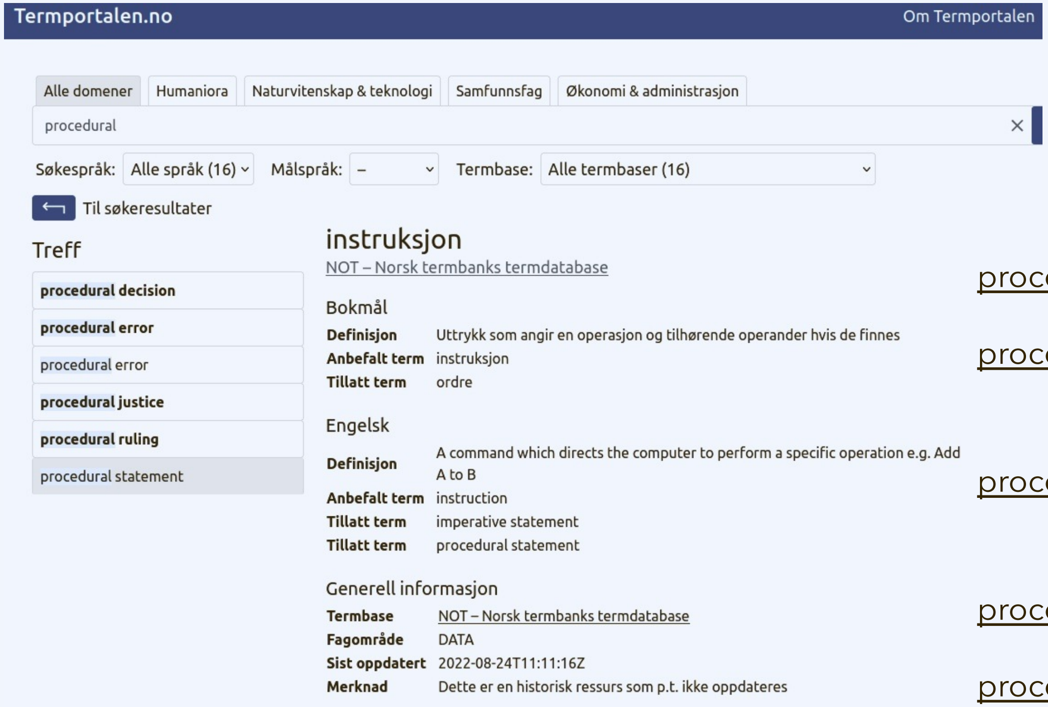
- Store Norske Leksikon (snl.no)
- Termportalen (termportalen.no)
- Entries on Wikipedia ...
- Textbooks in Norwegian ...? Are there any that you would recommend?
- ...

[1] Departments of Education and Culture, "Frå ord til handling: Handlingsplan for norsk fagspråk i akademia," **2023**.

Norwegian disciplinary language

The **termportalen.no** site is an (unfortunately sparsely populated) index of technical terms. The entries are not all appropriate for generating new terms.

Example: “Procedural,” needed for “procedural programming.”



Termportalen.no

Om Termportalen

Alle domener | Humaniora | Naturvitenskap & teknologi | Samfunnsfag | Økonomi & administrasjon

procedural ×

Søkespråk: Alle språk (16) Målspråk: – Termbase: Alle termbaser (16)

Til søkeresultater

Treff

- **procedural decision**
- **procedural error**
- procedural error
- **procedural justice**
- **procedural ruling**
- procedural statement

instruksjon

NOT – Norsk termbanks termdatabase

Bokmål

Definisjon	Uttrykk som angir en operasjon og tilhørende operander hvis de finnes
Anbefalt term	instruksjon
Tillatt term	ordre

Engelsk

Definisjon	A command which directs the computer to perform a specific operation e.g. Add A to B
Anbefalt term	instruction
Tillatt term	imperative statement
Tillatt term	procedural statement

Generell informasjon

Termbase	NOT – Norsk termbanks termdatabase
Fagområde	DATA
Sist oppdatert	2022-08-24T11:11:16Z
Merknad	Dette er en historisk ressurs som p.t. ikke oppdateres

”prosedyreprogrammering”?

procedural decision	prosessledende avgjørelse
procedural error	saksbehandlingsfeil formell feil
procedural justice	prosedyrerettferd (*nn.*) prosedyrerettferdighet (*nn.*)
procedural ruling	prosessledende kjennelse
procedural statement	

Norges miljø- og
biovitenskapelige
universitet

Institutt for datavitenskap

Digitalisering på Ås

DAT121

Introduction to data science

1 Python basics

1.1 Python as a script language

1.2 Distinguishing features of Python