Lecture 1: Introduction

Data science, machine learning, python programming

Nils Olovsson
nils.olofsson@igp.uu.se

Introduction to data science and Python programming for medical research
Uppsala University

2025

UPPSALA
UNIVERSITET

Content

Introduction

Data science

Python programming

Variables and types

Operators

Boolean

Conditional

Example

Practical

Introduction

Introduction

Introduction: Data

Types

- Physiological, biopsies
- Radiological (CT, MR, PET, ...)
- Omics (DNA, RNA, Proteins, ...)
- Forms

Sources

- Databases (Public, national, regional)
- Medical records (national, regional)
- Research projects
- In vivo/in vitro/in silico

Traits

- Numerical or cathegorical
- Dimensionallity
- Structure (*i.e.* in a graph or an image)


Atoms


Molecules


Genes


Interactions


Cells


Tissue

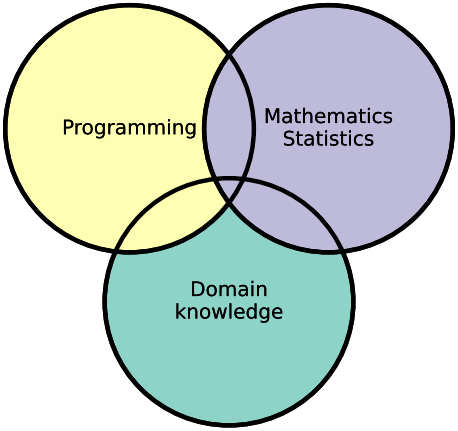

Organs


Organisms


Populations

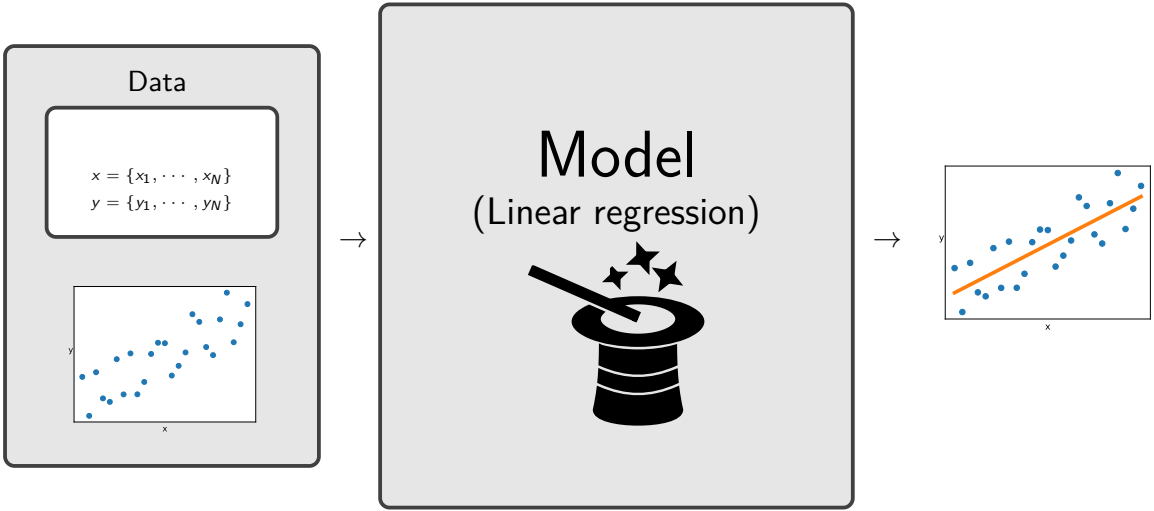
Adapted from the talk A Primer on Biomedical Data Sources by Jan Byška and Noeska Smit at the Biomedical visualization spring school 2021.

Introduction: What is data science?



Adapted from "Datascience handbook".

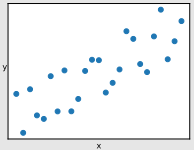
Introduction: Peak under the hood



Introduction: Peak under the hood

Data

$$x = \{x_1, \cdots, x_N\}$$
$$y = \{y_1, \cdots, y_N\}$$



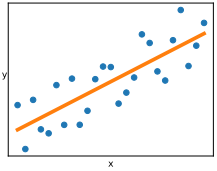
→

$$k = \frac{\sum_i y_i - \frac{N}{\sum_i x_i} \sum_i x_i y_i}{\sum_i x_i - \frac{N}{\sum_i x_i} \sum_i x_i^2}$$

```
import numpy as np
N = len(x)
sum_x = np.sum(x)
sum_y = np.sum(y)
sum_x2 = np.sum(np.power(x,2))
sum_xy = np.sum(np.multiply(x,
     y))
k = (sum_y - N/sum_x * sum_xy)
       /
    (sum_x - N/sum_x * sumx2)

```

→

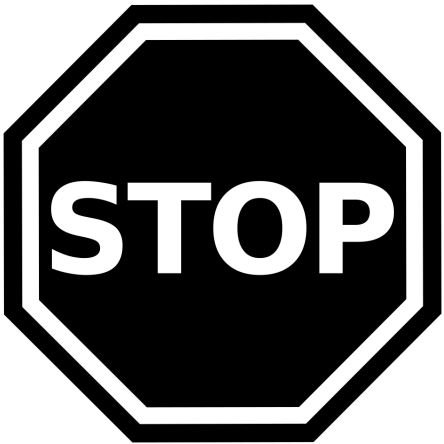


Introduction: Course plan learning outcomes

- Introductory knowledge in python programming.
- Calculate descriptive statistics and perform statistical tests.
- Plot data.
- Perform linear and non-linear regression.
- Work with high dimensional data and perform clustering.
- Dimensionality reduction with PCA, t-SNE and UMAP.
- History of AI and artificial neural networks (ANN).
- Explain how a small ANN can discriminate between two cathegories.
- Exaplain how machine learning models learn from data.
- Work with digital images.
- Use pre-trained deep learning models to solve problems in medical image analysis.

Introduction: What you will NOT learn!

- Statistics and probability
  - Suggested literature
  - Other courses
- Software development



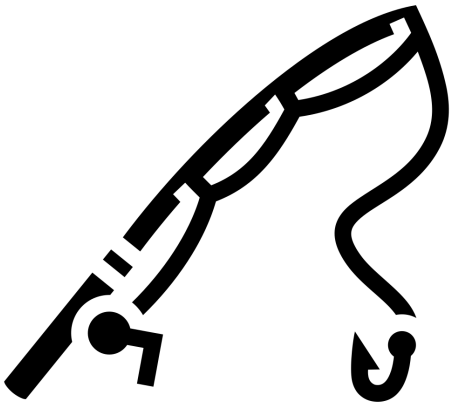
Introduction: What I hope you will actually learn

To be able to work independently using free tools without being chained to it.

Some "digital hygiene" with regards to the tools used.

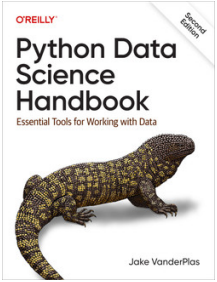
That you understand that you are allowed to try to understand things.

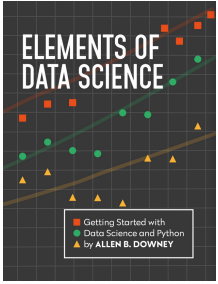
To become a creator instead of a consumer.

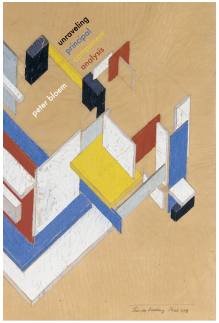


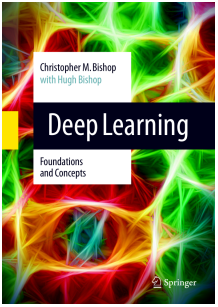
Introduction: Course literature

No mandatory course literature. Book suggestions as complement to lectures and for diving deeper into certain subjects.





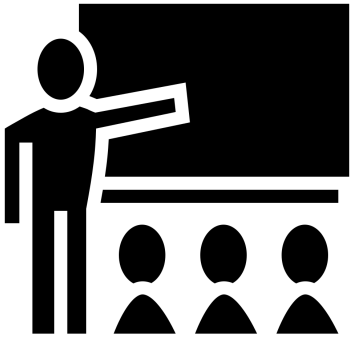




https://jakevdp.github.io/PythonDataScienceHandbook/
https://greenteapress.com/wp/elements-of-data-science/
https://peterbloem.nl/publications/unraveling-pca

Introduction: Course structure

- Lectures (15)
  - Slides will be provided
- Computer exercises (14)
  - Laptop
  - Pen and paper
  - Download lecture slides and exercise sheets.
  - Passed by showing me solution.
- Two guest lectures
  - Precision medicine
  - AI
- Written exam on theoretical parts, not code writing.
  - Pen
  - 4 hours (On the schedule)



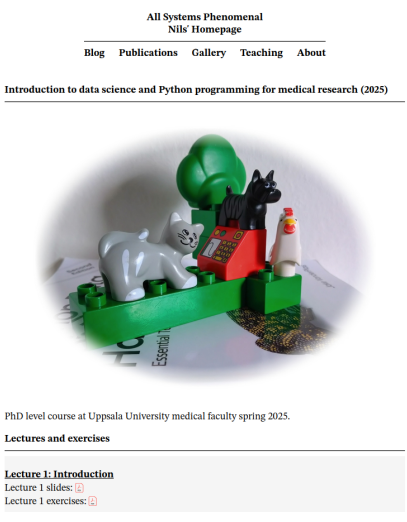
Introduction: Course homepage

All **course material** will be **continuously added** to the course homepage.

If needed I can also email the material.

Since the course will move along quite rapidly it is important that you can access the material so tell me if you have any problems!

https://nils-olovsson.se/teaching/intro-data-science-python-2025/



Introduction: Schedule

			Monday (31/3) E10:3309 BMC	Tuesday (1/4) E10:3309, BMC	Wednesday (2/4) Rita Levi-Montalcini, Rudbeck vän 3, C11/R4	Thursday (3/4) E10:3309, BMC	Friday (4/4) E10:2309, BMC
Week 14	9:00 - 10:00	Optional: Exercise supervision and QA		Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA
	10:15 - 12:00	Lectures	**M1: Course overview. Introduction to Python programing.**	**M1: The Python standard library, containers, loops, functions and classes.**	**M2: Numpy and numerical arrays. Plotting matplotlib.**	**M2: Summary statistics and statistical tests. Importing data using Pandas.**	**M2: Linear, nonlinear and logistic regression.**
	13:15 - 15:00	Supervised computer exercises	M1: Installation of Python interpreter. Writing first python scripts.	M1: Introductory programming excercises.	M2: Loading, saving, manipulating and plotting tabular and 1D data.	M2: Exercises in computing summary statistics and statistical tests.	M2: Exercises on using regression methods.
			Monday (7/4) E10:3309, BMC	**Tuesday (8/4) E10:3309, BMC**	**Wednesday (9/4) E10:2309, BMC**	**Thursday (10/4) E10:3309, BMC**	**Friday (11/4) E10:3309, BMC**
Week 15	9:00 - 10:00	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA
	10:15 - 12:00	Lectures	**M3: High dimensional data in biomedical research. Linear algebra for data analysis. Plotting 2D and 3D data with matplotlib.**	**M3: Data clustering.**	**M3: Principal component analysis (PCA) and linear dimensionality reduction.**	**M3: Manifold learning and non-linear dimensionality reduction.**	**M3: Other machine learning methods, support vector machines and random forests. Historical perspective on AI and neural networks.**
	13:15 - 15:00	Supervised computer exercises	M3: Working with and plotting higher dimensional data.	M3: Exercises in data clustering.	M3: Exercises in applying PCA.	M3: Exercises in applying manifold learning and t-SNE and UMAP.	M3: Exercises in using support vector machines and random forests for classification.

Introduction: Schedule

			Monday (14/4) E10:3309, BMC	Tuesday (15/4) E10:3309, BMC	Wednesday (16/4) E10:2309	Thursday (17/4)	Friday (18/4)
Week 16	9:00 - 10:00	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Easter	Easter
	10:15 - 12:00	Lectures	M4: Introduction to neural networks. Linear algebra for neural networks.	M4: Deep learning for classification and regression tasks.	M4: Digital image processing and analysis.		
	13:15 - 15:00	Supervised computer exercises	M4: Exercises in transforming data and implementing neural networks with pytorch.	M4: Implementing deep learning neural networks with pytorch.	M4: Loading, manipulating and plotting images using scikit-image, simpleITK and matplotlib.		
			Monday (21/4)	Tuesday (22/4) E10:2309, BMC	Wednesday (23/4) E10:3309, BMC	Thursday (24/4) Rita Levi-Montalcini, Rudbeck vån 3, C11/R4	Friday (25/4) E10:3309, BMC
Week 17	9:00 - 10:00	Optional: Exercise supervision and QA	Easter	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA
	10:15 - 12:00	Lectures		M4: Deep learning for image analysis.	M4: Other topics in deep learning.	10:00 - 11:00 Guest lecture: Precision medicine (Åsa Johansson, IGP)	10:00 - 11:00 Guest lecture: AI (Thomas Schön, IT)
	13:15 - 15:00	Supervised computer exercises		M4: Exercises on using deep neural networks to perform image analysis tasks.	Optional: Exercise supervision and QA	Solving example exam. (Barbara McClintock, Rudbeck)	QA

Introduction: Schedule

			Monday (28/4) E10:3309, BMC	Tuesday (29/4) E10:3309, BMC	Wednesday (30/4)
Week 18	9:00 - 10:00	Optional: Exercise supervision and QA	Optional: Exercise supervision and QA	(Preliminary) Exam: 4 hours	Valborg
	10:15 - 12:00	Lectures	QA		
	13:15 - 15:00	Supervised computer exercises	Optional: Exercise supervision and QA		

Introduction: Who am I?

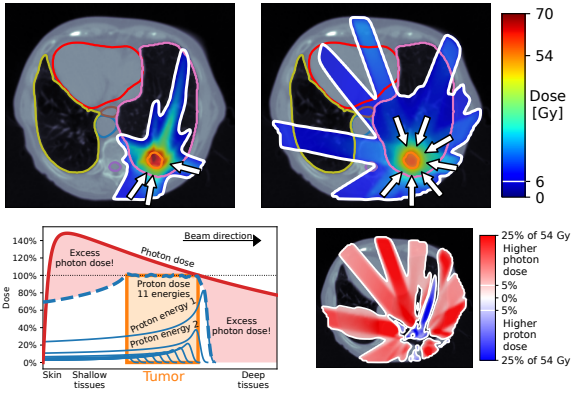
Questions

1. Who are you?
2. Background and current research subject?
3. Any programming experience?
4. Any experience working with data/ML/AI?
5. What are your expectations on this course?

Introduction: Who am I?

Questions

1. I am Nils. Last year phd student in medical radiation physics.
2. Computational engineering, Geometric modelling for a while. Current research subject is proton radiation therapy.
3. Python for $\sim$ 10 years. C++, Matlab, java, web.
4. Don't actually work with data science/ML/AI.
5. Teach you how fun programming can be and that methods that seem very complicated can be understood!



Introduction: Who are you?

List of what subjects the students are studying

Questions

1. Who are you?
2. Background and current research subject?
3. Any programming experience?
4. Any experience working with data/ML/AI?
5. What are your expectations on this course?

Data science

Data science

Data science: Machine learning

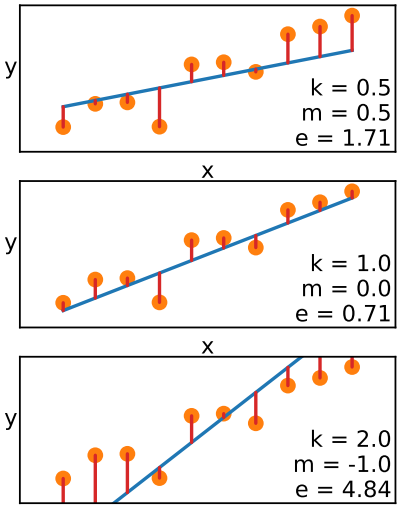
Find **parameters** of the **model** that **fits the data**.

The model does not need a mechanistic interpretation.

E.g. when fitting a line the parameters are:

1. Slope k
2. Intersection m

The fitting consists of find values of the parameters that **minimize the error** between the model and the data.



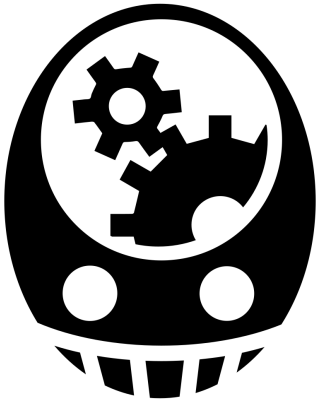
Data science: Cathegories of machine learning

Supervised

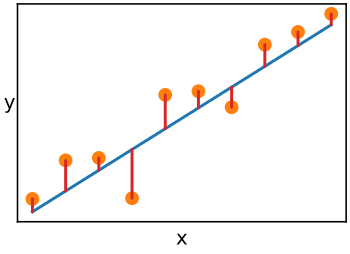
- Regression
- Classification

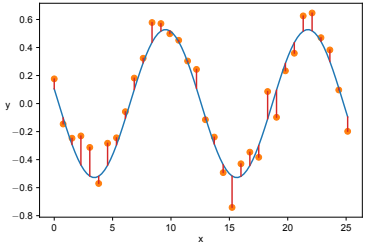
Unsupervised

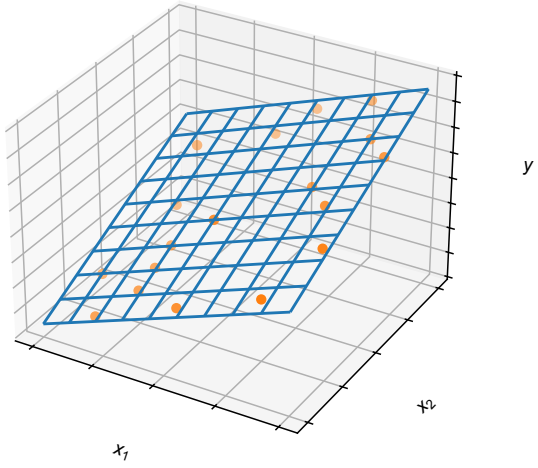
- Clustering
- Dimensionality reduction



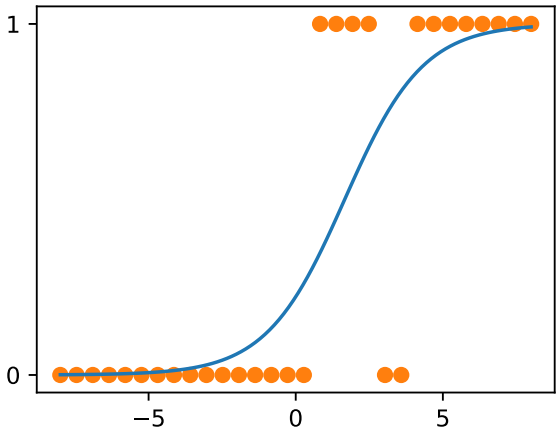
Data science: Regression

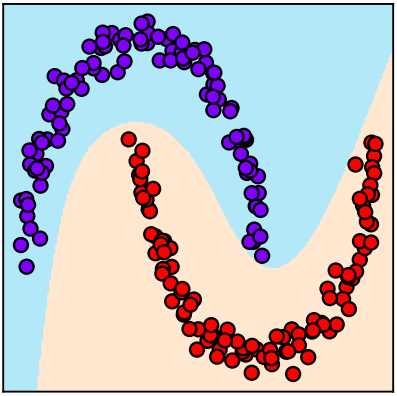






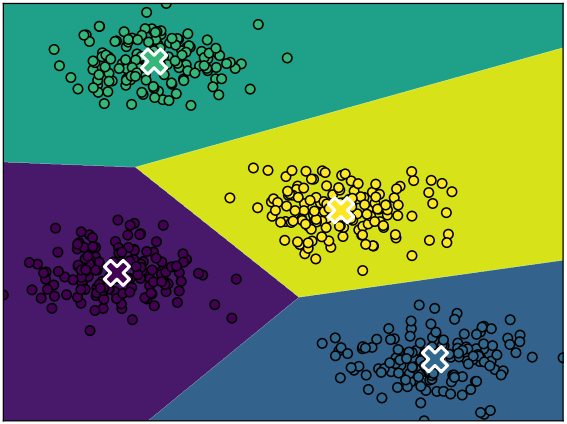
Data science: Classification

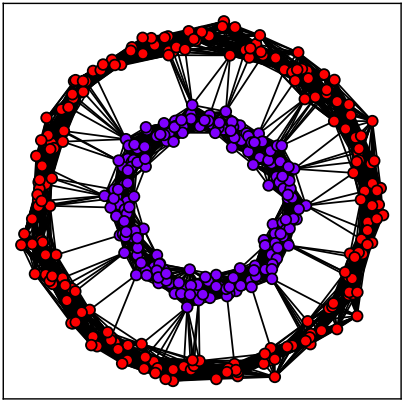




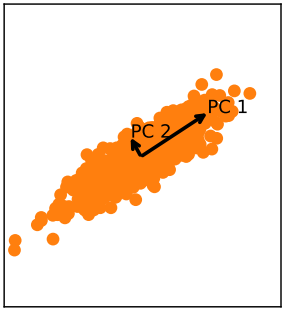
Pipelined "PolynomialFeatures" (Degree 5) and "LogisticRegression"

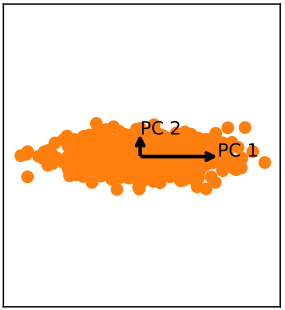
Data science: Clustering)

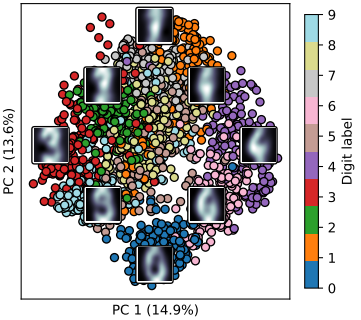




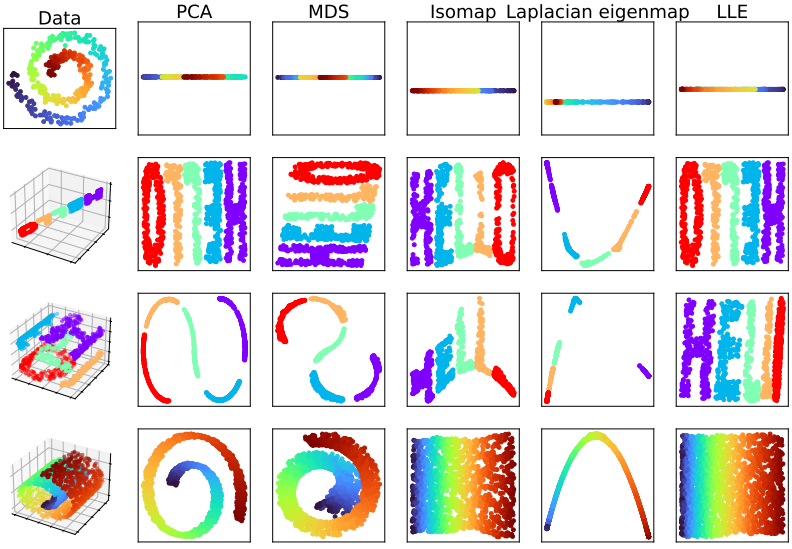
Data science: Linear dimensionality reduction with PCA



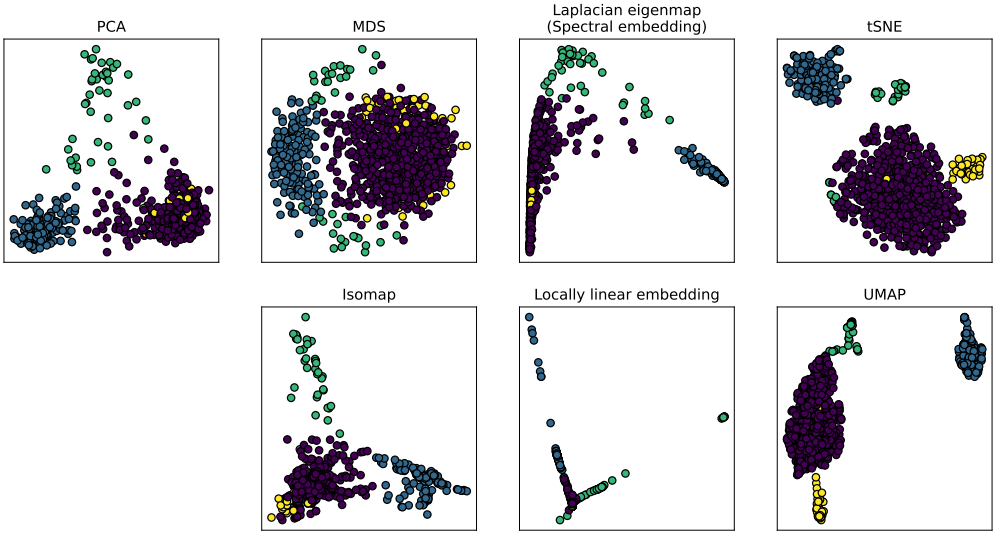




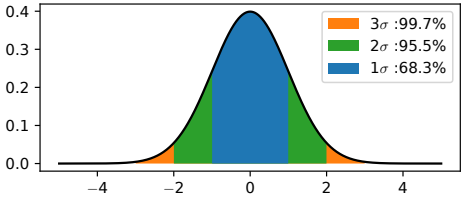
Data science: Nonlinear dimensionality reduction with manifold learning

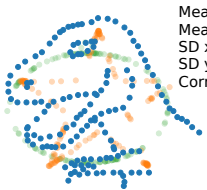


Data science: Nonlinear dimensionality reduction with manifold learning

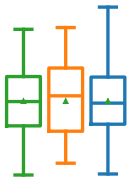


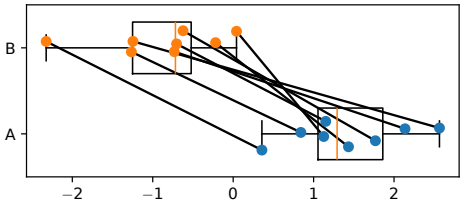
Data science: Descriptive statistics and tests



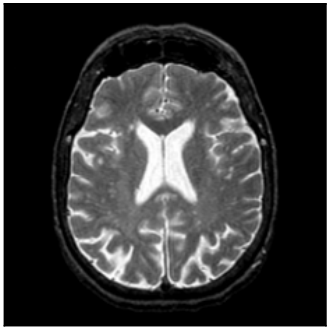


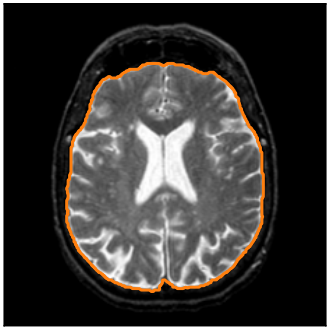
Mean x: 54.26
Mean y: 47.83
SD x: 16.71
SD y: 26.84
Correlation: -0.06

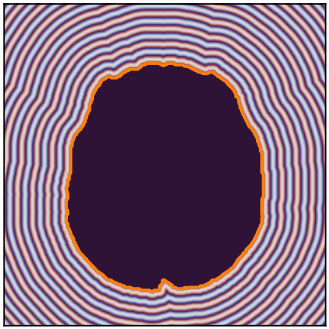




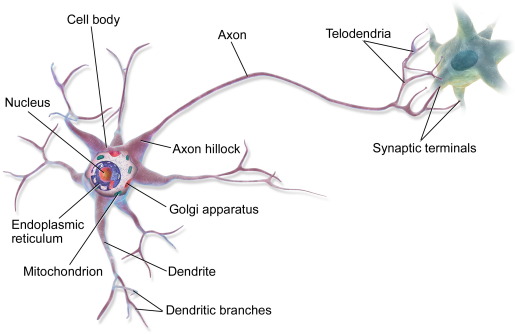
Data science: Image processing

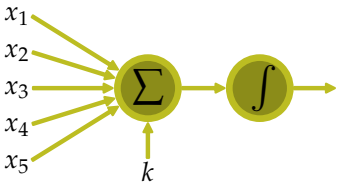




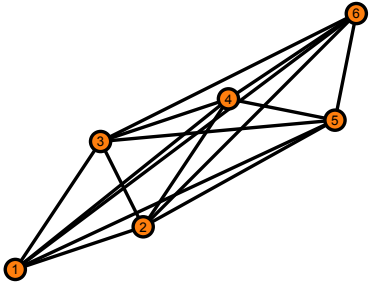


Data science: Artificial neural networks and deep learning

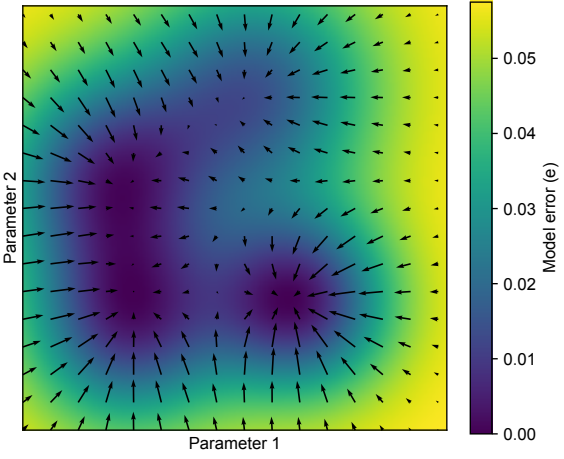




Data science: Mathematics

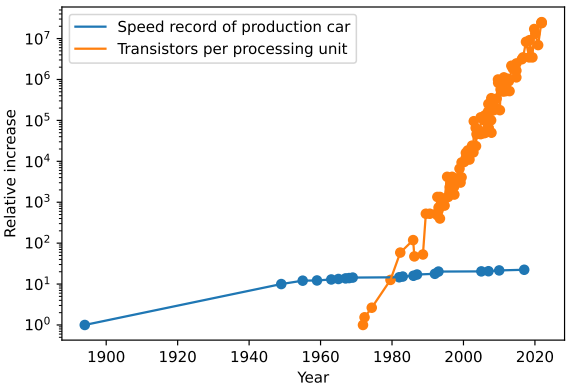


$$D = \begin{pmatrix} 0.0 & 0.6 & 0.7 & 0.0 & 0.0 & 0.0 \\ 0.6 & 0.0 & 0.4 & 0.0 & 0.0 & 0.0 \\ 0.7 & 0.4 & 0.0 & 0.6 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.6 & 0.0 & 0.5 & 0.7 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.7 & 0.5 & 0.0 \end{pmatrix}$$



Python programming

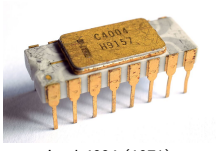
Python programming: Why computer programming?



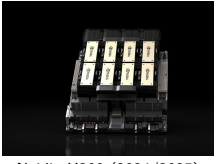
https://en.wikipedia.org/wiki/Production_car_speed_record
https://github.com/karlrupp/microprocessor-trend-data

Benz Velocipede (1894)
20 km/h

Koenigsegg Agera RS (2017)
450 km/h



Intel 4004 (1971)
2.3×10^3 transistors



Nvidia H200 (2024/2025)
8×10^{10} transistors

Python programming: Why Python?

Script languages

- **Python**
- R
- Matlab
- Julia

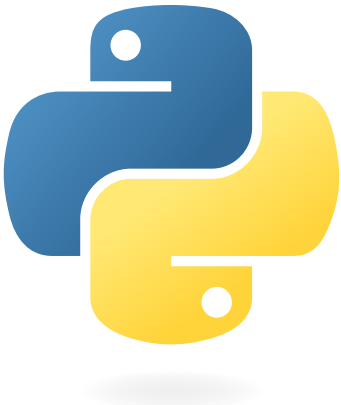
- Javascript
- Typescript
- PHP
- (HTML, CSS, ...)

Compiled languages

- C
- C++
- C#
- Java
- Rust

Python programming: What makes python special?

- Slow
+ Flexible and "easy"
+ Packages
+ Became and is popular among scientists



Python programming: Topics and packages

Course content in python packages.

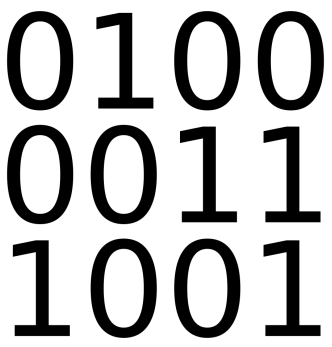
Linear algebra	numpy
Plotting	matplotlib
Tabular data	pandas
Statistics	scipy
Computing	scipy
Machine learning	sklearn
Image processing	skimage
Medical image processing	Simple ITK
Deep learning	pytorch

Python programming: What is computer programming?

Writing **instructions** to be execute by the computer.

Programming languages, like **Python**, are often written in a form of **pseudo English**.

```
for index in range(0, 5):
    if index > 2:
        print("Large index")
    else:
        print("Small index")
```



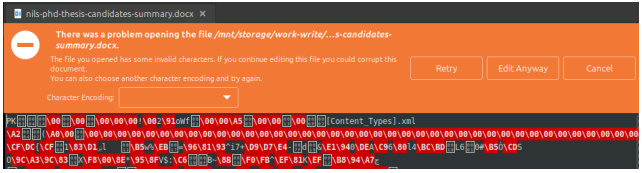
Python programming: Text files

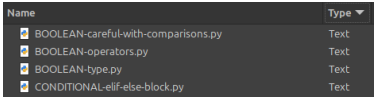
Programming code is written in **text files**.

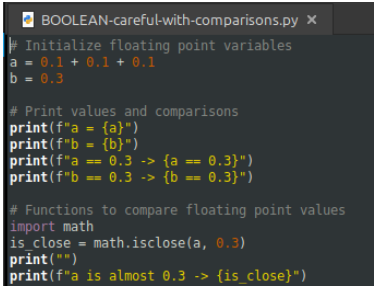
Unlike *e.g.* a word document, that also contains text, a text file does not contain any formatting or things besides text.

The **file type** is often indicated by its ending.

Python source code files should have `.py` file ending.





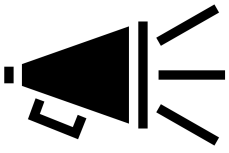


Python programming: Hello world

It is important that your program or script can **show** you some sort of **output**.

The most basic way of doing this is by using a **print** statement.

This prints text to a **terminal** were you run your code.



A Python script can be executed in a terminal by typing `python` followed by the name of the script file.

```
python hello-world.py
```

Python code (`hello-world.py`)

```
1 print("Welcome to the Python course!")
```

Terminal output

```
1 >> Welcome to the Python course!
```

Variables and types

Variables and types: Numbers

There are **two types of numbers** in Python, and in fact represented by the computer itself.

- **Integers**, `int`, representing whole **numbers without any decimals**. *E.g.* counting.
- **Floating point** numbers, `float`, **representing numbers with decimals**. *E.g.* measurements.

(Sometimes not important which type is used.) (Other times very important!!!)

Python code

```
1  # Initialize the variables
2  a = 65
3  b = 65.0
4
5  # Check the type using the 'type()' command
6  type_a = type(a)
7  type_b = type(b)
8
9  type_c = type(type_a)
10
11 # Print the variables
12 print(f"Value of variable a: {a}")
13 print(f"Value of variable b: {b}")
14 print("")
15 print(f"Type of variable a: {type_a}")
16 print(f"Type of variable b: {type_b}")
17 print(f"Type of type check???: {type_c}")
```

Terminal output

```
1  >> Value of variable a: 65
2  >> Value of variable b: 65.0
3  >>
4  >> Type of variable a: <class 'int'>
5  >> Type of variable b: <class 'float'>
6  >> Type of type check???: <class 'type'>
```

Variables and types: Strings

Strings represent **sequences of characters**. Typically for **writing**.

Can be created using **single or double quotation marks**.
Does not matter which one is used but they have to be started and terminated with same type.

Formatted strings can be used to create strings that contain values of other variables, like numbers,
(Try to get used to printing strings formatted with numbers and other strings!)

```
# Initialize the variables
a = 'Nils'
b = "Olovsson"

# Check the type using the 'type()' command
type_a = type(a)
type_b = type(b)

# Print the variables
print(f"Value of variable a: {a}")
print(f"Value of variable b: {b}")
print("")
print(f"Type of variable a: {type_a}")
print(f"Type of variable b: {type_b}")
```

```
>> Value of variable a: Nils
>> Value of variable b: Olovsson
>>
>> Type of variable a: <class 'str'>
>> Type of variable b: <class 'str'>
```

Variables and types: Allowed names and comments

Variables can be named most things. However, **some words are reserved** by the Python language itself and can not be used.

Names also **can't start with a number**.

Comments are sections of the code that **will not be executed**.
Used to **explain the code** or **temporarily cause certain lines** or sections to **not be run**.

or """

```
# This is a single line comment

""" This is a
    multiline comment
    that can be very long.
"""

# Variable names can't start with numbers or
# characters reserved for operators.
# They also can't be reserved Python
# keywords (e.g. 'if', 'else' etc)

a0 = 1.0 # Allowed (Comment after statement)
#0a = 1.0 # !!! Not allowed (SyntaxError)
#&a = 1.0 # !!! Not allowed (SyntaxError)
#if = 1.0 # !!! Not allowed (SyntaxError)

# Comments can be used to suppress statements
# or for describing the code

# Use descriptive variable names!
a = 110.0
blood_pressure_systolic_Hg = 110.0
```

Operators

Operators

Operators: Mathematics

The regular mathematical operations are available.

- `+` Add
- `-` Subtract
- `*` Multiply
- `**` Power
- `/` Divide
- `//` Integer division
- `%` Modulus

```
# Initilalize variables
a = 3.0
b = 5.0

# Do some calculations
c = a + b
d = a * b + c

# Raise to power
e = b**2

# Division
f = 12 / 5
g = 12 // 5

# Print the variables
print(f"{a:.1f} + {b:.1f} = {c:.1f}")
print(f"{a:.1f} + {b:.1f} * {c:.1f} = {d:.1f}")
print(f"{b:.1f}**2 = {e:.1f}")
print(f"12 / 5  = {f:.1f}")
print(f"12 // 5 = {g:d}")
```

```
>> 3.0 + 5.0 = 8.0
>> 3.0 + 5.0 * 8.0 = 23.0
>> 5.0**2 = 25.0
>> 12 / 5  = 2.4
>> 12 // 5 = 2
```

Operators: Order

The order of the operations are **mostly as you would expect**.

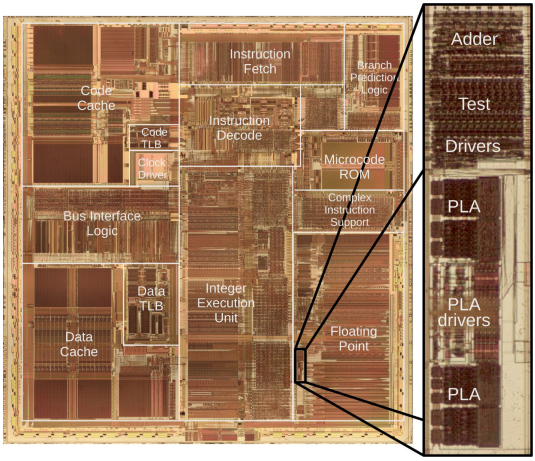
If you are unsure, put **parentheses** around expressions.

Python does not assume multiplication, as we sometimes write in mathematical notation.

All **operations have to be written explicitly!**

```
1  # Initilalize variables
2  a = 3 + 2
3  b = 4 * (3 + 2)
4
5  # We can't omit multiplication sign
6  # we do in mathematical notation!
7  #c = 4 (3 + 2) # !!! Not allowed (One error)
8  #d = (3 + 2) 4 # !!! Not allowed (Other error)
9
10 # Print the variables
11 #print(f"{a:.1f} + {b:.1f} = {c:.1f}")
12 #print(f"{a:.1f} + {b:.1f} * {c:.1f} = {d:.1f
       }")
```

Operators: Limited precision of float



Pentium processor (1993). From Ken Shirriff's blog.

https://www.righto.com/2025/01/pentium-floating-point-ROM.html

Computers are, in fact, physical machines and **not magic**.

As such they come with **limitations**.

Floating point numbers are only **approximations** of real decimal numbers.

Important to be aware of this.

```
1 a = 0.1 + 0.1 + 0.1
2 print(f"a = {a}")
```

```
1 >> a = 0.30000000000000004
```

Operators: Strings

Some operators that typically would define a mathematical operation are defined to perform **operators on strings**.

- `+` Concatenate
- `*` Repeat copies

(A string representing a number is still a string!)

```
# Strings of numbers are not numbers!
a = '3.0'
b =' 5.0'
c = a + b
print(c)

# Concatenate two strings
s0 = 'Hello '
s1 = 'class!'
sc = s0 + s1
print(sc)

# Multiply will create repeated content!
sr = 3 * s0
print(sr)
```

```
>> 3.0 5.0
>> Hello class!
>> Hello Hello Hello
```

Operators: Casting

If we want to perform mathematical operations on numbers represented by strings, we must first **cast** them to a numeric type.

Examples:

- Number to number, *e.g.* integer to float.
- String to number.
- Number to string. Check if string represents number before casting!

All types can be cast to string and as such they can be printed.

```
1  a = 3
2  b = 2.0
3
4  # Number to number
5  c = float(a)
6  d = int(b)
7
8  # String to number
9  si = str(a)
10 sf = str(b)
11
12 print(si)
13 print(sf)
14
15 # Number to string
16 print( si.isnumeric() )
17 print( sf.isdecimal() )
18 d = int(si)
19 n = float(sf)
```

```
1  >> 3
2  >> 2.0
3  >> True
4  >> False
```

Boolean

Boolean

Boolean: Type

Boolean values are logical, either `True` or `False`.

They can be the result of a comparison or assigned explicitly.

George Boole.
https://en.wikipedia.org/wiki/George_Boole

```python
# Initialize two integers to compare
x = 3
y = 5
# Perform comparisons
b0 = x == y # equal
b1 = x != y # not equal
b2 = x > y  # greater
b3 = x < y  # smaller
b4 = x >= y # great or equal
b5 = x <= y # smaller or equal
b6 = True   # assigned value
# Print the comparison results and type
print(f"{x} == {y} -> {b0} ({type(b0)})")
print(f"{x} != {y} -> {b1} ({type(b1)})")
print(f"{x} >  {y} -> {b2} ({type(b2)})")
print(f"{x} <  {y} -> {b3} ({type(b3)})")
print(f"{x} >= {y} -> {b4} ({type(b4)})")
print(f"{x} <= {y} -> {b5} ({type(b5)})")
print(f"Assigned   -> {b6} ({type(b6)})")
```

```
>> 3 == 5 -> False (<class 'bool'>)
>> 3 != 5 -> True (<class 'bool'>)
>> 3 >  5 -> False (<class 'bool'>)
>> 3 <  5 -> True (<class 'bool'>)
>> 3 >= 5 -> False (<class 'bool'>)
>> 3 <= 5 -> True (<class 'bool'>)
>> Assigned   -> True (<class 'bool'>)
```

Boolean: Operators

Boolean values can be inverted using `not`.
This flips a `True` to `False` and vice versa.

Logical statements can be chained with `and` and `or`.

Later we will see how Boolean values can also result from:

Object identity `is`, `is not`

Containment `has`, `has not`

```python
# Initialize two integers to compare
x = 3
y = 5

# Chain conditions using 'and' and 'or'
b0 = True
b1 = not b0
b2 = x > y or b0
b3 = (not x > y) and b0

# Print values and comparisons
print(f"b0: {b0}")
print(f"b1: {b1}")
print(f"b2: {b2}")
print(f"b3: {b3}")
```

```
>> b0: True
>> b1: False
>> b2: True
>> b3: True
```

(Can you follow why the values become what they are?)

Boolean: Careful with equalities

Remember that floating point values might not be *exactly* what you think they are!

However small the difference is, it matters to the computer when doing a comparison!

Use special functions that check if floating point values are *close* instead.

```python
# Initialize floating point variables
a = 0.1 + 0.1 + 0.1
b = 0.3

# Print values and comparisons
print(f"a = {a}")
print(f"b = {b}")
print(f"a == 0.3 -> {a == 0.3}")
print(f"b == 0.3 -> {b == 0.3}")

# Functions to compare floating point values
import math
is_close = math.isclose(a, 0.3)
print("")
print(f"a is almost 0.3 -> {is_close}")
```

```
>> a = 0.30000000000000004
>> b = 0.3
>> a == 0.3 -> False
>> b == 0.3 -> True
>>
>> a is almost 0.3 -> True
```

Conditional

Conditional

Conditional: `if` statements

We can execute parts of the code only `if` some Boolean value is `True`.
This is sometimes called an execution *branch*.

To do this we create an conditional `if` block.

The content inside needs to be **indented** using tab or spaces.
Four blank spaces is standard.

```
1  # Initialize two Boolean variables
2  b0 = True
3  b1 = False
4
5  # Use if-statements to only print one of these
6  # Note that the block below the if-statement
7  # is indented with four spaces compared to
8  # where 'if' begins.
9  if b0:
10     print("The b0 branch was executed!")
11     print("These branching blocks can")
12     print("be however long we want!")
13
14 if b1:
15     print("The b1 branch was executed!")
```

```
1  >> The b0 branch was executed!
2  >> These branching blocks can
3  >> be however long we want!
```

Note how the print after the second `if` was never executed!

Conditional: `elif` and `else` statements

More than one condition can be added a conditional block by adding an `elif` and an `else` statement.

```
# Initialize two Boolean variables
b0 = False
b1 = False

# Use if-statements to only print one of these
# Note that the block below the if-statement
# is indented with four spaces compared to
# where 'if' begins.
if b0:
    print("The b0 branch was executed!")
    print("These branching blocks can")
    print("be however long we want!")
elif b1:
    print("The b1 branch was executed!")
else:
    print("A default branch was executed")
```

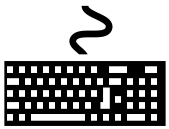
```
>> A default branch was executed
```

Example

Example

Example: User input

Even for small Python scripts it can be useful to capture some user input.



The execution can be paused and a string from the keyboard can be captured using the `input()` function.

Can be used to capture values or to pick branches of execution.

```
1  # Get a value from the user
2  s = input("Input a number and press the return
       key:\n")
3
4  # The value will be of type 'str'
5  print(f"Type of s: {type(s)}")
6
7  # Need to convert to numeric
8  v = float(s)
9  print(f"Type of v: {type(v)}")
10
11 # Print the converted value
12 print(f"User gave value: {v:.1f}")
```

```
1  >> Input a number and press the return key:
2  >> Type of s: <class 'str'>
3  >> Type of v: <class 'float'>
4  >> User gave value: 21.0
```

Example: Blood pressure

Lets write a script that can inform us whether a person has dangerously high blood pressure or not!

```
# Set the persons blood pressure
bp_s_mm_Hg = 140.0 # Systolic
bp_d_mm_Hg = 82.0 # Diastolic

# Compare against two threshold values
print(f"Blood pressure: {bp_s_mm_Hg} / {
    bp_d_mm_Hg} [mm Hg]")
if bp_s_mm_Hg >= 140.0 or bp_d_mm_Hg >= 90.0:
    print(f"Person might have hypertension!")
else:
    print("Blood pressure appears nominal.")
```

```
>> Blood pressure: 140.0 / 82.0 [mm Hg]
>> Person might have hypertension!
```

Blood pressure classifications

Categories	Systolic blood pressure, mmHg			and/or	Diastolic blood pressure, mmHg		
Method	Office	Home	24h ambulatory		Office	Home	24h ambulatory
European Society of Cardiology (2024)[10]							
Non-elevated	<120	<120	<115	and	<70	<70	<65
Elevated	120–139	120–134	115–129	and	70–89	70–84	65–79
Hypertension	≥140	≥135	≥130	or	≥90	≥85	≥80

https://en.wikipedia.org/wiki/Blood_pressure

You will expand on this with user input and full testing in the exercises!

Practical

Practical

Practical: Terminal

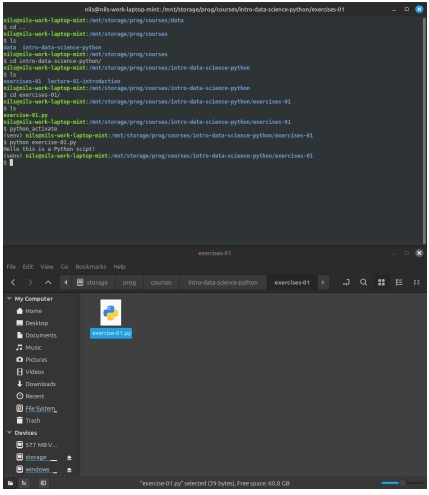
A terminal is what people used **before** we had **graphical user interfaces**.

Still **good** to be able to use **if you do computer programming**.

Important to be able to **navigate to directory where our Python scripts are**!

`cd ..`	Go one step *up*.
`cd <PATH>`	Go into subdirectory.
`ls`	Print content of directory.

Terminal emulator for windows:
https://cmder.app/



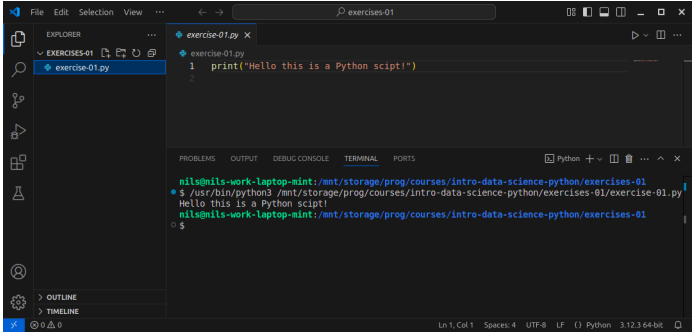
Practical: Coding environment (IDE)

An **Integrated Development Environment** (IDE) includes tools for **writing**, **running** and **testing** the code.

They also have different forms of **syntax checking**, **autocomplete** and help based on **code documentation**. More recently also contain **AI tools** for **code generation**.

- VScode
- Spyder
- (Pycharm
- Others

https://code.visualstudio.com/
https://www.spyder-ide.org/

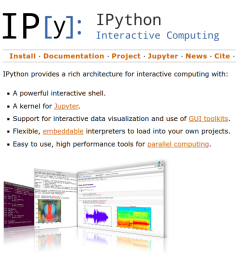


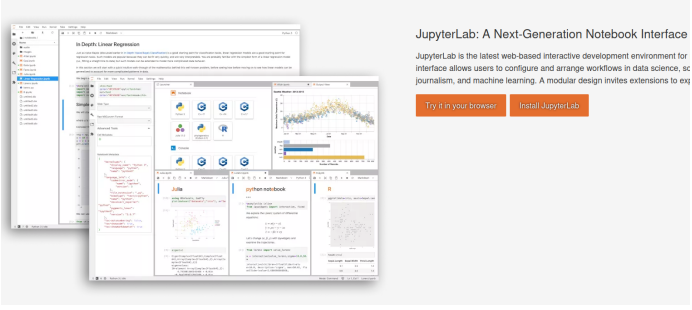
Microsoft VScode.

Practical: Python environment

The **standard Python environment** is bare bones and only **used to run scripts**.
This is primarily what we will use in the course.

iPython and Jupyter notebooks offer more **interactive environments** in which code can be changed and run and offer **graphical interfaces** and outputs.





https://ipython.org/
https://jupyter.org/

Practical: Python distributions

Python is **open source** and therefore there are **several distributors** of the Python runtime!

If you do not have Python installed I suggest that you install **miniconda**.

Last year license was changed for the regular anaconda distribution and it is no longer free for larger organizations such as universities!

Python Distributions

Aside from the official CPython distribution available from python.org, other distributions based on CPython include the following:

- ActivePython from ActiveState
- Anaconda from Continuum Analytics
- ChinesePython Project: Translation of Python's keywords, internal types and classes into Chinese. Eventually allows a programmer to write Python programs in Chinese.
- Enthought's EDM
- Win9xPython: Backport of mainline CPython 2.6/2.7 to old versions of Windows 9x/NT.
- IPython and its IPyKit variant
- PocketPython
- Portable Python: Run Python from USB device - no installation needed.
- PyIMSL Studio
- PyPy: a Python implementation in Python.
- Python(x,y): Python(x,y) is a scientific-oriented Python Distribution based on Qt, Eclipse and Spyder
- PythonForArmLinux
- PythonLabsPython: an old name for the python.org distribution
- PythonwarePython
- StacklessPython
- Tiny Python (archived link) - not to be confused with tinypy
- WinPython: Another scientific-focused Python distribution, based around Spyder

https://wiki.python.org/moin/PythonDistributions
https://www.datacamp.com/blog/navigating-anaconda-licensing
https://dev.to/kaamisan/using-miniconda-with-conda-forge-to-avoid-anaconda-licensing-issues-5hkj
https://www.anaconda.com/docs/getting-started/miniconda/install

Practical: Package manager

Much of what we will use in this course will require **additional Python packages**. Most packages are also **open source** and they also have several distributors!
(Yes, all of this is annoying and confusing.)

If you use **miniconda** there are two package managers you can easily use.

Python package index

```
pip install <PACKAGE_NAME>
```

Conda forge

```
conda install <PACKAGE_NAME>
```

https://pypi.org/

https://conda-forge.org/

Practical: Agenda

- Install Python (miniconda recomended)
- Install an IDE (VScode recomended)
- If your use Windows you can install a terminator emulator (cmder)
- In the afternoon you will work on the exercises.

https://www.anaconda.com/docs/getting-started/miniconda/install
https://code.visualstudio.com/
https://cmder.app/