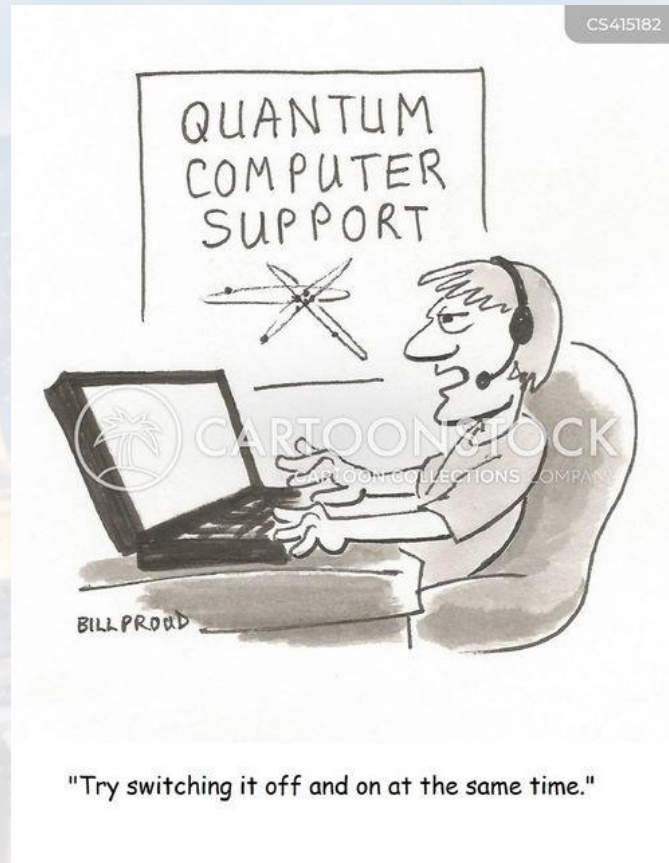


M. Hohle:

Physics 77: Introduction to Computational Techniques in Physics



syllabus:

- Introduction to Unix & Python (week 1 - 2)
- Functions, Loops, Lists and Arrays (week 3 - 4)
- Visualization (week 5)
- Parsing, Data Processing and File I/O (week 6)
- Statistics and Probability, Interpreting Measurements (week 7 - 8)
- Random Numbers, Simulation (week 9)
- Numerical Integration and Differentiation (week 10)
- Root Finding, Interpolation (week 11)
- Systems of Linear Equations (week 12)
- Ordinary Differential Equations (week 13)
- Fourier Transformation and Signal Processing (week 14)
- Capstone Project Presentations (week 15)

Week	Dates	Topics	Reading	Lecture Slides	Workshop	Homework
1 - 2	Aug 28th / Sep 4th	Introduction to Unix and Python	K&N Ch. 1			
3 - 4	Sep 11th / Sep 18th	Functions, Loops, Lists, Arrays	K&N Ch.2-3			
5	Sep 25th	Visualization	K&N Ch. 4, K&N Ch. 6.3-6.4			
6	Oct 2nd	Parsing, Data Processing, and File I/O	K&N Ch. 4			
7 - 8	Oct 9th / Oct 16th	Statistics and Probability, Interpreting Measurements	Hughes			



Week	Dates	Topics	Reading	Lecture Slides	Workshop	Homework
9 - 10	Oct 23rd / Oct 30th	Random Numbers, Simulation	K&N Ch. 6, Newman Ch. 10			
11 - 12	Nov 6th / Nov 13th	Numerical Integration and Differentiation	K&N Ch. 6 Newman Ch. 5			
13	Nov 20th	Root finding, Interpolation	K&N Ch. 6.5 Newman Ch. 6			
14	Nov 27th	Systems of Linear Equations	Newman Ch. 6 K&N Ch. 6.6			
15	Dec 4th	Ordinary Differential Equations	K&N Ch. 6.8-6.9 Newman Ch. 8			
16	Dec 11th	Fourier Transforms, Signal Processing	Newman Ch. 7			
17	Dec 18th	Capstone Project Presentations				

Lecture: Wednesdays, 2 - 4pm, Physics Building 251

Workshops: Fridays, 2 - 4pm (section 1) Social Science, 170 and
Fridays, 4 - 6pm (section 2) Social Science, 140

Office Hours: Wednesdays, 4 – 6pm, 397 Physics North (17)

Instruction begins: August 28, 2024

Instruction ends: Friday, December 13, 2024

my contact: markus.hohle@berkeley.edu

TA: Krish Desai krish.desai@berkeley.edu

check out: [Google Colab](#)
[bcourse](#)
[datahub.berkeley](#)

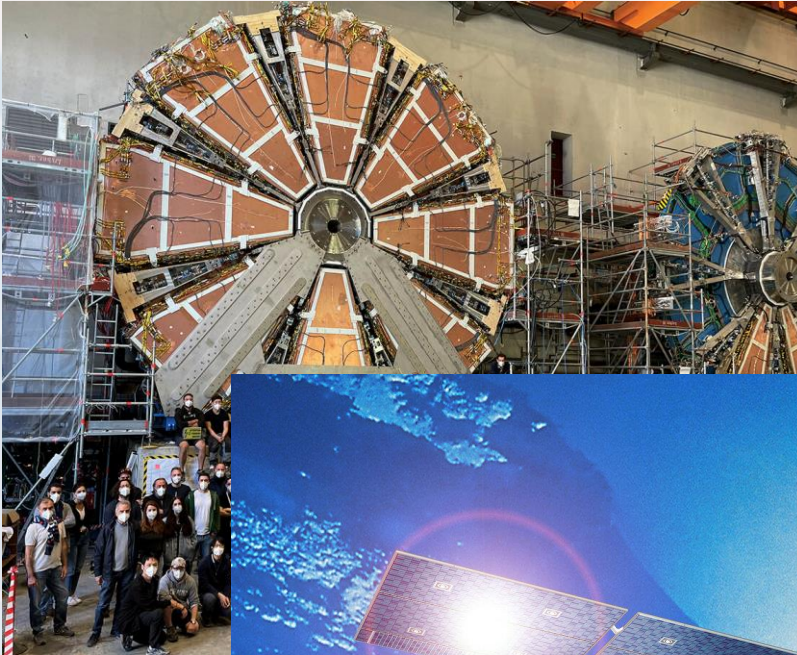
motivation: The good old days:



Salar de Uyuni,
Bolivia, Mar 2003



motivation:



source: CERN



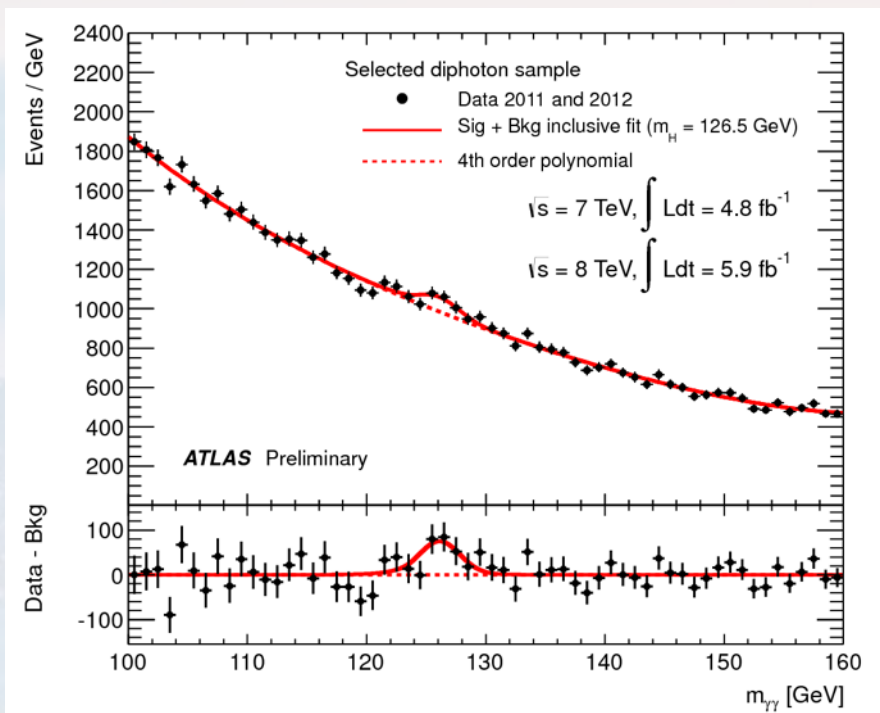
source: LIGO collaboration



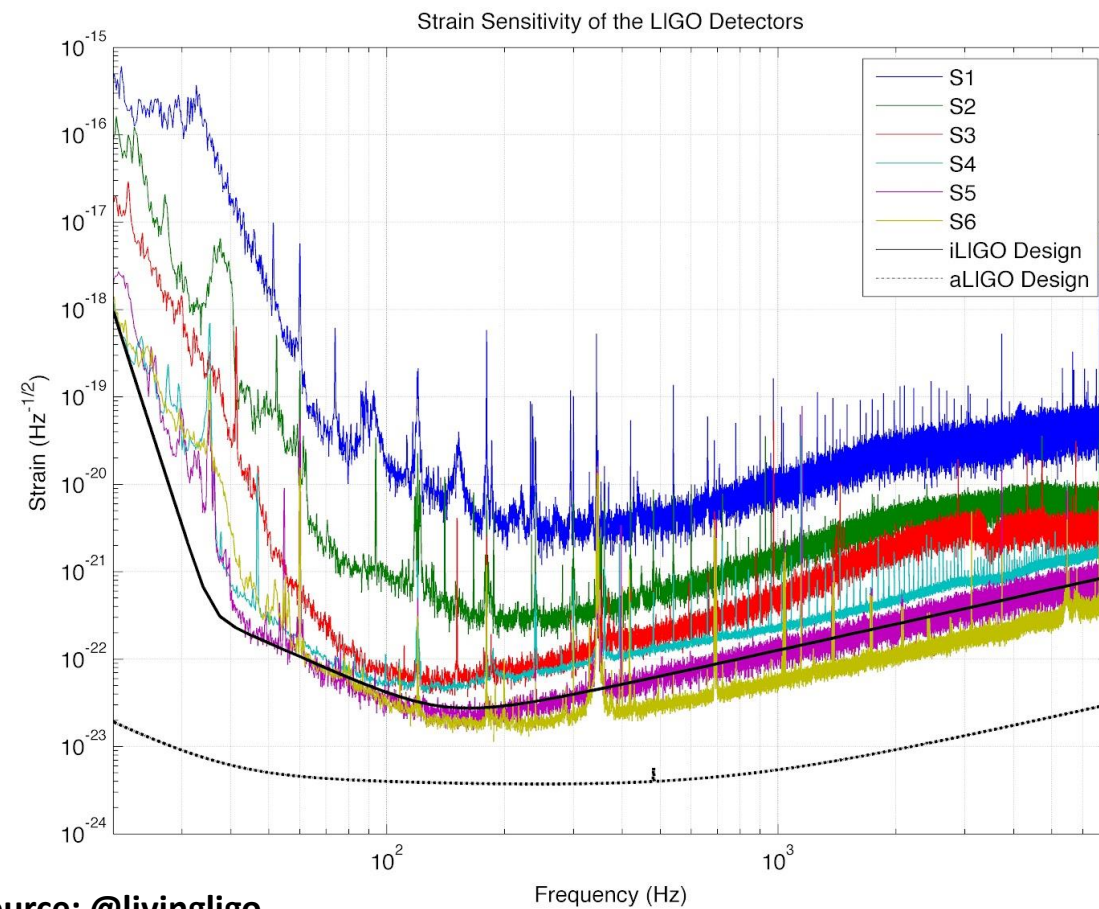
source: ESA



motivation:



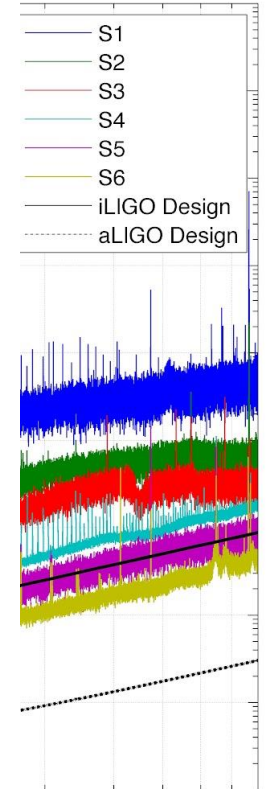
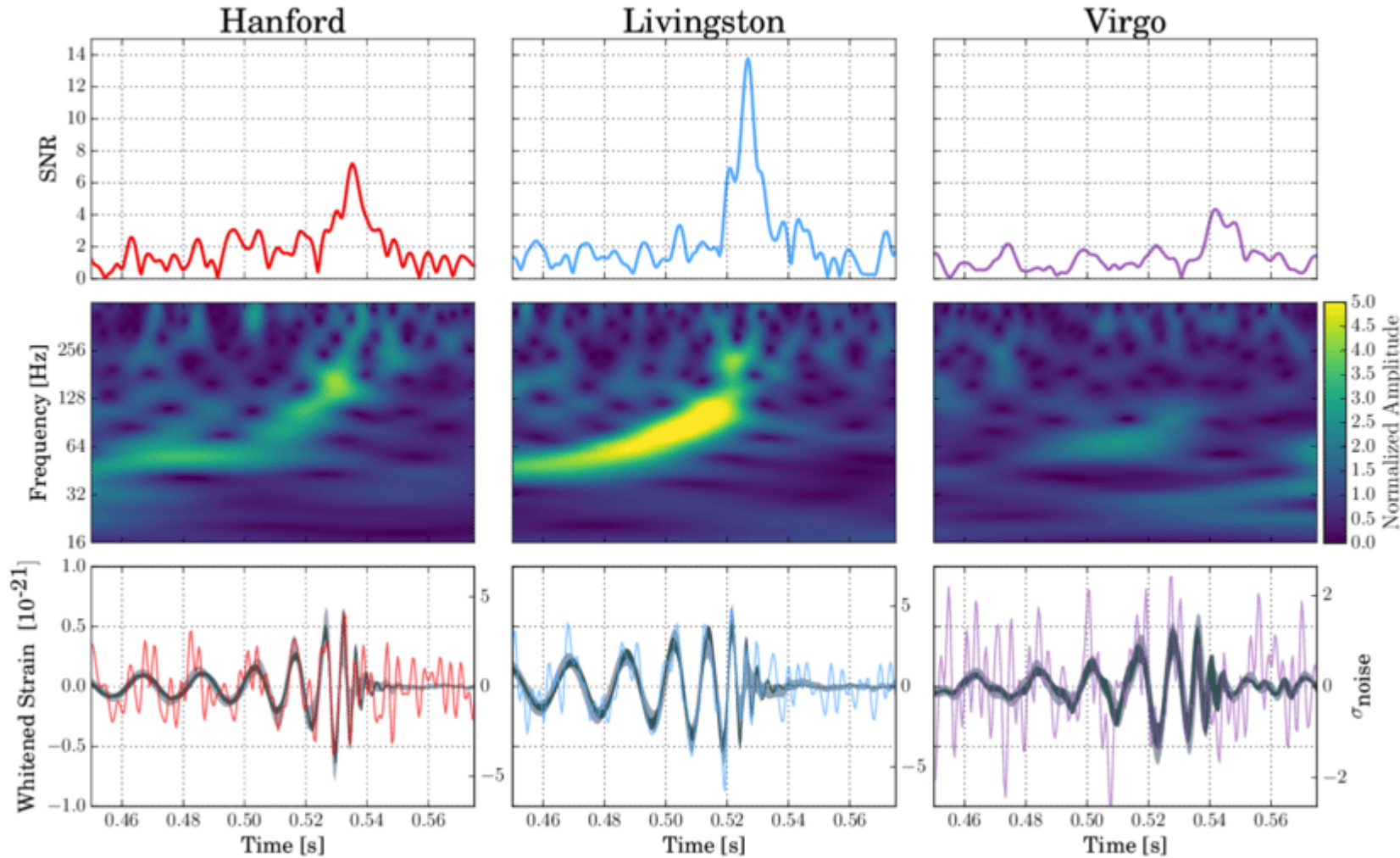
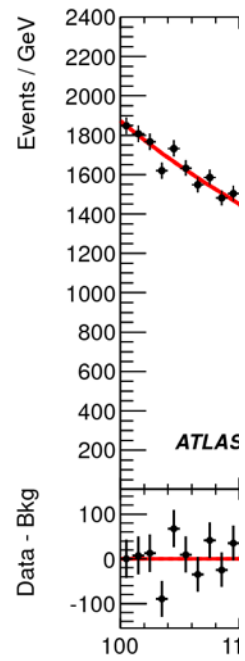
Source: CERN/ATLAS



Source: @livingligo



motivation:





goal: being able to write code like this...

```
@my_timer
```

```
class SignalDetect():
```

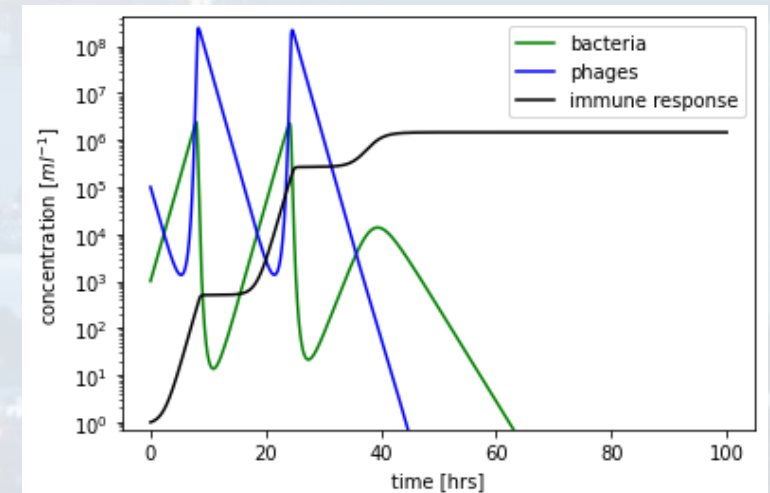
```
    def __init__(self, T, Range_phi = [0, 2*np.pi], dphi = 0.01,\n                  MaxM = 20, **Opts):
```

```
        L = listdir(getcwd())
```

```
        [remove(i) for i in L if '.npy' in i]
```

```
        ...
```

...and simulating systems like that



syllabus:

- Introduction to Unix & Python (week 1 - 2)
- Functions, Loops, Lists and Arrays (week 3 - 4)
- Visualization (week 5)
- Parsing, Data Processing and File I/O (week 6)
- Statistics and Probability, Interpreting Measurements (week 7 - 8)
- Random Numbers, Simulation (week 9)
- Numerical Integration and Differentiation (week 10)
- Root Finding, Interpolation (week 11)
- Systems of Linear Equations (week 12)
- Ordinary Differential Equations (week 13)
- Fourier Transformation and Signal Processing (week 14)
- Capstone Project Presentations (week 15)

syllabus:

- Introduction to Unix & Python (week 1 - 2)

- Functions, Loops, Lists and Arrays (week 3 - 4)

- Visualization (week 5)

- Parsing, Data Processing and File I/O (week 6)

- Statistics and Probability, Interpreting Measurements (week 7 - 8)

- Random Numbers, Simulation (week 9)

- Numerical Integration and Differentiation (week 10)

- Root Finding, Interpolation (week 11)

- Systems of Linear Equations (week 12)

- Ordinary Differential Equations (week 13)

- Fourier Transformation and Signal Processing (week 14)

- Capstone Project Presentations (week 15)



Operating **S**ystem: interface between human and computer

→ **G**raphical **U**ser **I**nterface: Windows, OS X, iOS, Android

→ text based: MS-DOS

→ mixed: Unix

Unix:

- Bell Labs, early 70s
- Written in C and Assembly

Linux:

- as a an alternative to Unix
- Linus Torvalds (1991)
- Ubuntu, SUSE, ... **Scientific Linux**

efficient, fast, robust
optimized, but...

... not suitable for
the *standard* user



Programming Languages: → translate human instructions to a form understandable by a computer

the “*style*”
of programming

procedural: functions/ routines that call each other
(Fortran, ALGOL, COBOL, BASIC, Pascal, C)

object oriented (OOP): creating objects/types of different properties (see later)
C++, Fortran 2003, Java, MATLAB, **Python**, Ruby, ...

how a programming
language “*talks*” to your
CPU or GPU

compiled language: close to the resulting machine code, **fast**
Fortran, C, C++, Java, Cobol, Pascal

interpreted language: an interpreter translates between source code and
machine code. **Slower, but simpler syntax**
Perl, Raku, **Python**, MATLAB

source: wikipedia

```
A 002000 C2 30 REP #30
A 002002 18 CLC
A 002003 F8 SED
A 002004 A9 34 12 LDA #1234
A 002007 69 21 43 ADC #4321
A 00200A 8F 03 7F 01 STA $017F03
A 00200E D8 CLD
A 00200F E2 30 SEP #30
A 002011 00 BRK
A 2012

PB PC NUWxDIZC .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
S 2000

BREAK

PB PC NUWxDIZC .A .X .Y SP DP DB
; 00 2013 00110000 5555 0000 0002 CFFF 0000 00
m 7F03 7F03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```




Bits & Bytes: We need only **two** states: **on** and **off**

How many different states can I create with 8 switches?



$n = 8$

$$2 \times 2 \times 2 \dots = 2^n$$

smallest memory cell:

8bits = 1byte

bit stands for **binary digit**

For some reason humans use **ten** states $a = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$$N_{dec} = \sum_{i=0} a_{ji} 10^i$$

0	00000000
1	0000000 1
2	000000 10
3	000000 11
4	00000 100
5	00000 101
6	00000 110
7	00000 111
8	0000 1000
9	0000 1001
10	0000 1010
11	0000 1011
12	0000 1100
13	0000 1101
14	0000 1110
15	0000 1111
16	000 10000
...	



Bits & Bytes: We need only **two** states: **on** and **off**

For some reason humans use **ten** states $a = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$$N_{dec} = \sum_{i=0} a_{ji} 10^i$$

vs **two** states $a = \{0, 1\}$

$$N_{bin} = \sum_{i=0} a_{ji} 2^i$$

Let's write the number 15 in: *decimal*, *binary* and to base *three*:

0	00000000
1	0000000 1
2	000000 10
3	000000 11
4	00000 100
5	00000 101
6	00000 110
7	00000 111
8	0000 1000
9	0000 1001
10	0000 1010
11	0000 1011
12	0000 1100
13	0000 1101
14	0000 1110
15	0000 1111
16	000 10000
...	



Bits & Bytes: We need only **two** states: **on** and **off**



8 bits = 1 byte (B)

1 kB = 1024 B

1 MB = 1024 kB etc

So, we are fine with natural numbers...

... what is with negative numbers...

...or fractions...

...or π and e

→ float and double numbers need three *fields*:

- sign
- exponent
- fraction

more [here](#)

0	00000000
1	0000000 1
2	000000 10
3	000000 11
4	00000 100
5	00000 101
6	00000 110
7	00000 111
8	0000 1000
9	0000 1001
10	0000 1010
11	0000 1011
12	0000 1100
13	0000 1101
14	0000 1110
15	0000 1111
16	000 10000
...	



Bits & Bytes:

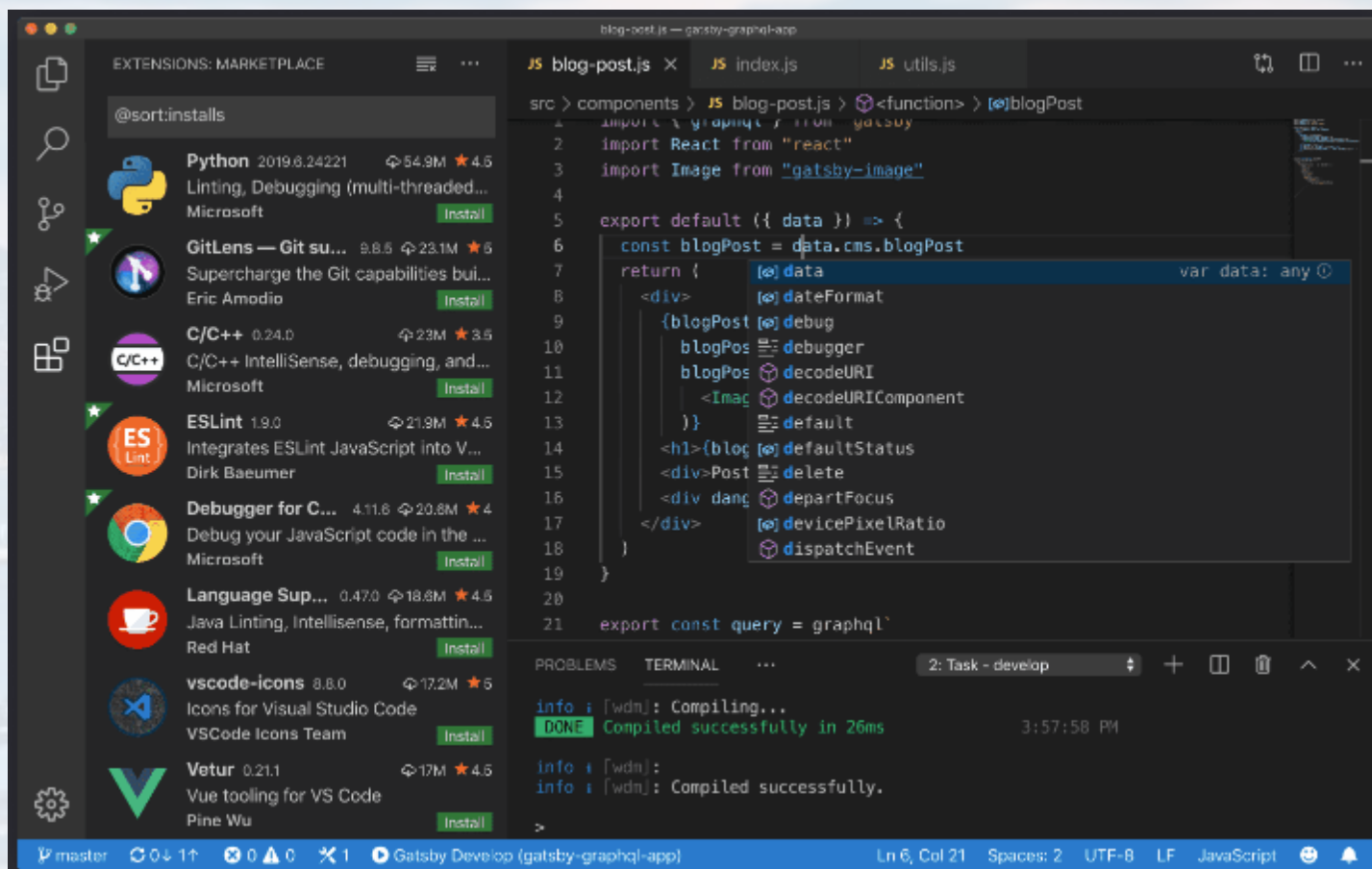
	rel. approx. error (ϵ)	range
16 bit int		-32768 ... 32767
32 bit int		$\approx -10^9 \dots 10^9$
32 bit float	$\approx 10^{-8}$	$\approx 10^{-38} \dots 10^{38}$
64 bit double	$\approx 10^{-16}$	$\approx 10^{-308} \dots 10^{308}$

0 00000000
1 0000000**1**
2 000000**10**
3 000000**11**
4 00000**100**
5 00000**101**
6 00000**110**
7 00000**111**
8 0000**1000**
9 0000**1001**
10 0000**1010**
11 0000**1011**
12 0000**1100**
13 0000**1101**
14 0000**1110**
15 0000**1111**
16 000**1**0000
...

Luckily, Python will tell us when an operation doesn't makes sense based on precision, but still: **be cautious!**



build your environment



<https://themanifest.com/software-development/blog/programming-engineer>



1) if your OS is Windows

→ install Windows Subsystem for Linux (WSL)

→ follow the instructions [here](#)



2) if your OS is Unix/Linux

→ we can start right away!

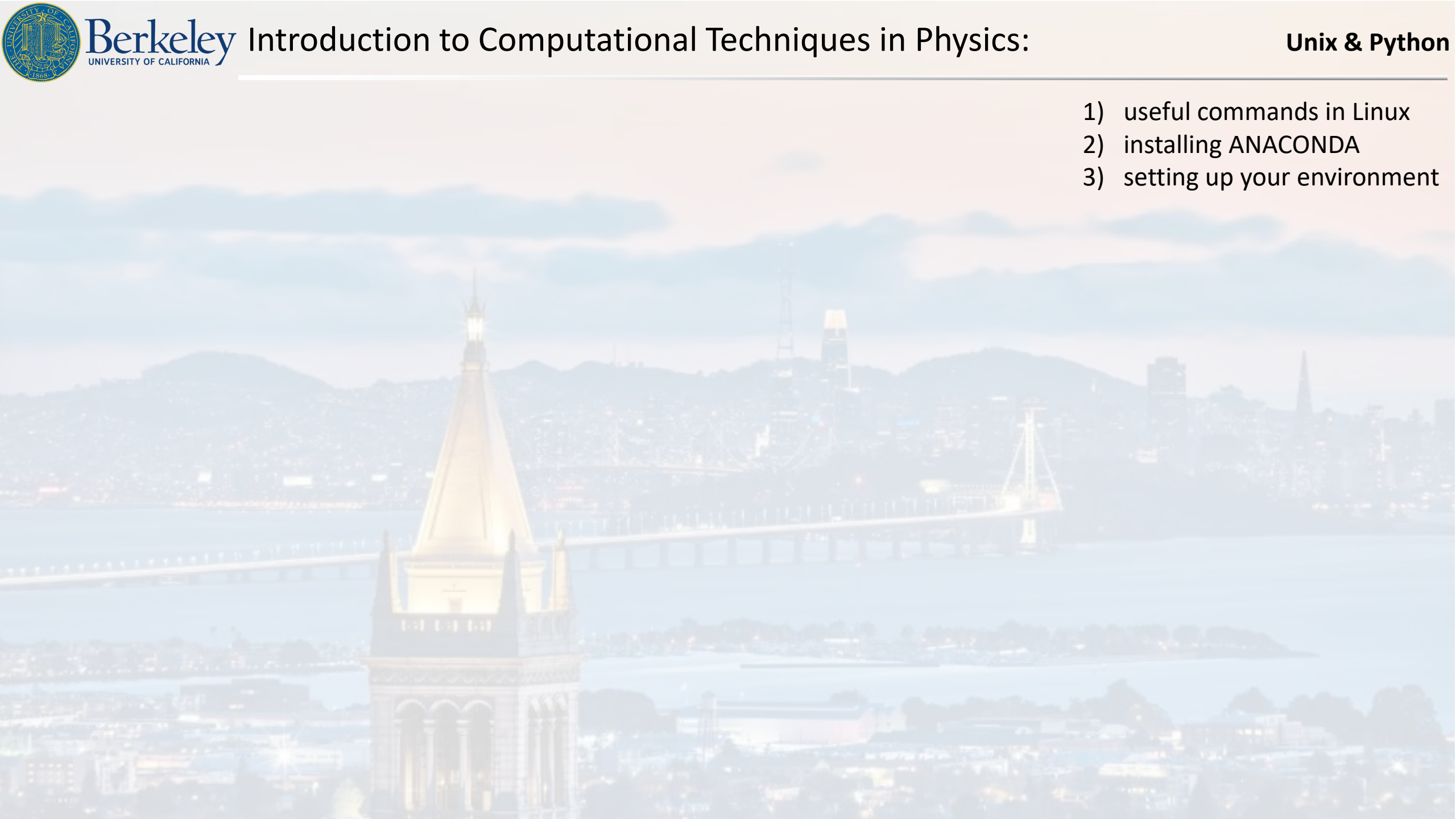
3) optional

→ once you have Linux, there are lot's of [useful tools](#) you can install and link together

- Google Chrome
- Visual Studio Code (aka VS Code)
- NodeJS
- Docker



- 1) useful commands in Linux
- 2) installing ANACONDA
- 3) setting up your environment





list your files and folders

ls

- 1) **useful commands in Linux**
- 2) installing ANACONDA
- 3) setting up your environment

```
(base) mmh_user@DESKTOP-PPSA666:~$ ls
Anaconda3-2023.09-0-Linux-x86_64.sh  AppAcademy  Untitled.ipynb  snap
Anaconda3-2024.06-1-Linux-x86_64.sh Downloads
```

files

folder

shell scripts
→ like executables



list your files and folders

ls

-lrt

1) **useful commands in Linux**

2) installing ANACONDA

3) setting up your environment

called *flags*

- **l** stands for *long*: shows a file with all permissions and properties, each per line
- **r** stands for *reverse* (oldest first)
- **t** stands for the *time* flag (as criterium for r)

```
(base) mmh_user@DESKTOP-PPSA666:~$ ls -lrt
total 2158464
drwx----- 2 mmh_user mmh_user      4096 Apr  8 20:03 Downloads
drwx----- 3 mmh_user mmh_user      4096 Apr 10 19:16 snap
drwxr-xr-x  5 mmh_user mmh_user      4096 Apr 24 18:29 AppAcademy
-rwxr-xr-x  1 mmh_user mmh_user 1153404010 Jul 24 12:36 Anaconda3-2023.09-0-Linux-x86_64.sh
-rwxr-xr-x  1 mmh_user mmh_user 1056829859 Aug  5 18:53 Anaconda3-2024.06-1-Linux-x86_64.sh
drwxr-xr-x 31 mmh_user mmh_user      4096 Aug  5 19:22 anaconda3
-rw-r--r--  1 mmh_user mmh_user      616 Aug  5 19:30 Untitled.ipynb
```



list your files and folders

```
ls -lrt
```

but these are only those files you see.
try:

```
ls -la
```

- 1) **useful commands in Linux**
- 2) installing ANACONDA
- 3) setting up your environment

will be
important
(see later)

pointer/
links/aliases

```
(base) mmh_user@DESKTOP-PPSA666:~$ ls -la
total 2158640
drwxr-x--- 23 mmh_user mmh_user    4096 Aug  5 19:41 .
drwxr-xr-x  3 root      root        4096 Apr  2 21:35 ..
drwxr-xr-x  4 mmh_user mmh_user    4096 Apr  3 04:49 .aa-setup-checker
drwxr-xr-x  3 mmh_user mmh_user    4096 Aug  4 13:14 .anaconda
lrwxrwxrwx  1 mmh_user mmh_user      21 Apr  3 04:31 .aws -> /mnt/c/Users/MMH/.aws
lrwxrwxrwx  1 mmh_user mmh_user      23 Apr  3 04:31 .azure -> /mnt/c/Users/MMH/.azure
-rw-----  1 mmh_user mmh_user   26898 Aug  5 23:07 .bash_history
-rw-r--r--  1 mmh_user mmh_user    220 Apr  2 21:35 .bash_logout
-rw-r--r--  1 mmh_user mmh_user    4582 Jul 24 12:40 .bashrc
drwx----- 11 mmh_user mmh_user    4096 Aug  5 19:42 .cache
drwxr-xr-x  2 mmh_user mmh_user    4096 Jul 24 12:42 .conda
-rw-r--r--  1 mmh_user mmh_user      25 Aug  5 19:03 .condarc
drwx-----  6 mmh_user mmh_user    4096 Aug  5 19:45 .config
drwxr-xr-x  5 mmh_user mmh_user    4096 Apr  3 04:31 .docker
```




list your files and folders

```
ls -lrt
```

```
ls -la
```

You can search for files/folders with particular substrings using a “wildcard”

```
ls *.py
```

```
ls *py*
```

- 1) **useful commands in Linux**
- 2) installing ANACONDA
- 3) setting up your environment

```
(base) mmh_user@DESKTOP-PPSA666:~$ ls -lrt *A*  
-rwxr-xr-x 1 mmh_user mmh_user 1153404010 Jul 24 12:36 Anaconda3-2023.09-0-Linux-x86_64.sh  
-rwxr-xr-x 1 mmh_user mmh_user 1056829859 Aug 5 18:53 Anaconda3-2024.06-1-Linux-x86_64.sh
```

```
AppAcademy:  
total 12  
drwxr-xr-x 13 mmh_user mmh_user 4096 Apr 17 18:34 HTMLExercises  
drwxr-xr-x 4 mmh_user mmh_user 4096 Apr 23 21:13 GitExercises  
drwxr-xr-x 7 mmh_user mmh_user 4096 May 3 23:36 JavaExercises
```



changing your directory

`cd` always leads back to the home directory

`cd ../` one level up

`cd ../my_dir` one level up, down to `my_dir`

`cd another/dir` one level down to `dir`

`mkdir test` creating the new directory *test*

`rm -r test` removing the directory using the flag *r* (here: recursively)

`rm any_file` when removing a file, no flag is needed

1) **useful commands in Linux**

2) installing ANACONDA

3) setting up your environment

Danger: Be careful with `rm`! There is no undelete command in Unix!



copying files and folders

- 1) useful commands in Linux**
- 2) installing ANACONDA
- 3) setting up your environment

```
cp my_file ../somewhere/else
```

```
cp my_file_original my_file_copy
```

```
cp my_file ../somewhere/else/my_file_copy
```

```
cp -r /entireDirectory somewhere/else/to/new_destination
```

note: the flag *r* is needed to copy the directory with all the sub directories

note: there are way more commands and flags we will be learning soon :)



You can install ANACONDA Navigator for **Windows**

[here](#)

- 1) useful commands in Linux
- 2) installing ANACONDA
- 3) setting up your environment

The screenshot shows the Anaconda Navigator application window. The top navigation bar includes links for Products, Solutions, Resources, Partners, and Company, along with a Sign Up button. The main heading is "Download Now" for Anaconda Navigator. Below this, there's a section for "Download Distributors" with a "Download" button. The left sidebar contains navigation links: Home, Environments, Learning, and Community. The main content area displays a grid of application tiles, each with an icon, name, version, description, and an "Install" or "Launch" button. The tiles include:

- PyCharm Professional**: The Python IDE for data science. It combines the interactivity of Jupyter notebooks with intelligent Python coding assistance, Anaconda support, and scientific libraries. (Install button)
- Anaconda AI Navigator**: Access various large language models (LLMs) curated by Anaconda, and start leveraging secure local AI today. (Install button)
- Anaconda Toolbox** 4.0.15: Anaconda Assistant. JupyterLab supercharged with a suite of Anaconda extensions, starting with the Anaconda Assistant AI chatbot. (Launch button)
- Anaconda Cloud Notebooks**: Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage. (Launch button)
- anaconda_prompt** 1.0.0: Opens a terminal instance with conda activated (requires menuinst 2.1.0 or greater). (Launch button)
- CMD.exe Prompt** 0.1.1: Run a cmd.exe terminal with your current environment from Navigator activated. (Launch button)
- console_shortcut_miniconda** 0.1.1: Anaconda Powershell Prompt. (Launch button)
- JupyterLab** 4.0.11: An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture. (Launch button)
- Jupyter Notebook** 7.0.8: Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis. (Launch button)
- Powershell Prompt** 0.0.1: Run a Powershell terminal with your current environment from Navigator activated. (Launch button)
- Qt Console** 5.5.1: PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more. (Launch button)
- Spyder** 5.5.1: Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features. (Launch button)
- watsonx**: IBM watsonx. IBM watsonx is an enterprise-ready AI platform including a data science model platform to build, train, manage, and deploy. (Launch button)
- ORACLE Cloud Infrastructure**: Oracle Data Science Service. OCI Data Science offers a machine learning platform to build, train, manage, and deploy. (Launch button)
- Glueviz** 1.2.4: Multidimensional data visualization across files. Explore relationships within and among. (Launch button)
- Orange 3** 3.36.2: Component based data mining framework. Data visualization and data analysis for. (Launch button)
- jowershell_shortcut_minicondi** 0.0.1: Anaconda Powershell Prompt. (Launch button)
- RStudio** 1.1.456: A set of integrated tools designed to help you be more productive with R. Includes R. (Launch button)



You can install ANACONDA Navigator **for Linux**

1) we move into the home directory (cd) and run the curl (Client URL) command in order to download the ANACONDA installer from [here](#)

```
curl -O https://repo.anaconda.com/archive/Anaconda3-2024.06-1-Linux-x86_64.sh
```

2) running the installer

```
bash ~/Downloads/Anaconda3-2024.06-1-Linux-x86_64.sh
```

3) it might give us an error message “permission denied”
→ turning the installer shell script into an “executable” using chmod

```
chmod +x Anaconda3-2024.06-1-Linux-x86_64.sh
```

4) run the installer manually

```
./Anaconda3-2024.06-1-Linux-x86_64.sh
```

- 1) useful commands in Linux
- 2) installing ANACONDA**
- 3) setting up your environment



3) it might give us an error message “permission denied”

```
chmod +x Anaconda3-2024.06-1-Linux-x86_64.sh
```

4) run the installer manually

```
./Anaconda3-2024.06-1-Linux-x86_64.sh
```

5) confirm license terms by typing *yes* and press *enter*

6) you might need to refresh your terminal by running

```
source ~/.bashrc
```

Now Anaconda should be working. Type

```
conda install python
```

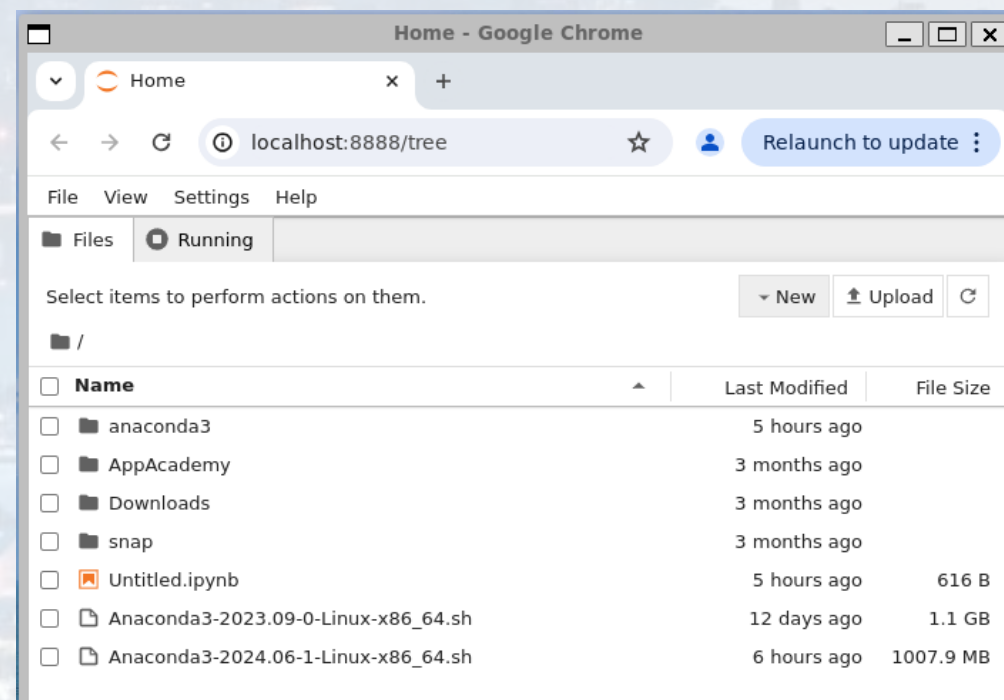
finally, run

```
jupyter-notebook &
```

1) useful commands in Linux

2) installing ANACONDA

3) setting up your environment





Note: conda on unix doesn't work well with Spyder

- 1) useful commands in Linux
- 2) installing ANACONDA
- 3) **setting up your environment**

```
(base) mmh_user@DESKTOP-PPSA666:~$ spyder &
[1] 52032
(base) mmh_user@DESKTOP-PPSA666:~$ Could not load the Qt platform plugin "xcb" in "" even though it was found.
This application failed to start because no Qt platform plugin could be initialized. Reinstalling the application may fi
x this problem.

Available platform plugins are: eglfs, minimal, minimalegl, offscreen, vnc, webgl, xcb.

[1]+  Aborted                  spyder
(base) mmh_user@DESKTOP-PPSA666:~$
```

creating a conda environment for the Spyder IDE:

```
conda create -n spyder-env -c conda-forge python=3.11 spyder
```

activating the environment:

```
conda activate spyder-env
```



Note: conda on unix doesn't work well with Spyder

- 1) useful commands in Linux
- 2) installing ANACONDA
- 3) **setting up your environment**

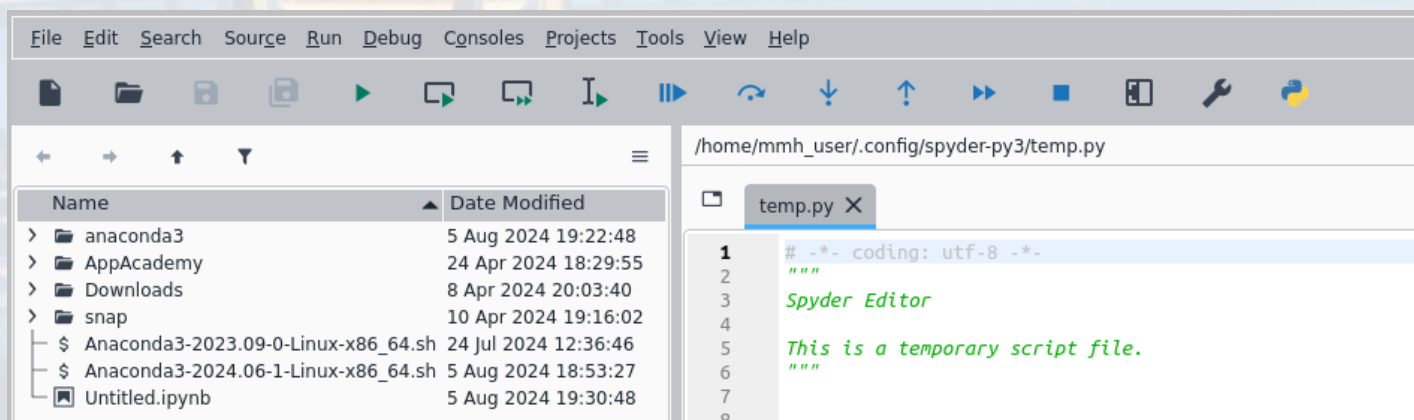
creating a conda environment for the Spyder IDE:

```
conda create -n spyder-env -c conda-forge python=3.11 spyder
```

activating the environment:

```
conda activate spyder-env
```

```
(base) mmh_user@DESKTOP-PPSA666:~$ conda activate spyder-env  
(spyder-env) mmh_user@DESKTOP-PPSA666:~$ spyder &  
[1] 53245  
(spyder-env) mmh_user@DESKTOP-PPSA666:~$ fromIccProfile: failed minimal tag size sanity
```





depending on your project, you can create many different conda environments

- 1) useful commands in Linux
- 2) installing ANACONDA
- 3) setting up your environment**

```
conda create -n <MyEnv> python=3.10 scipy=0.17.3
```

showing all libraries:

```
conda list
```

showing all environments

```
conda info --envs
```

```
(base) mmh_user@DESKTOP-PPSA666:~$ conda info --envs
# conda environments:
#
base                * /home/mmh_user/anaconda3
spyder-env           /home/mmh_user/anaconda3/envs/spyder-env
```



Now it is time for python!

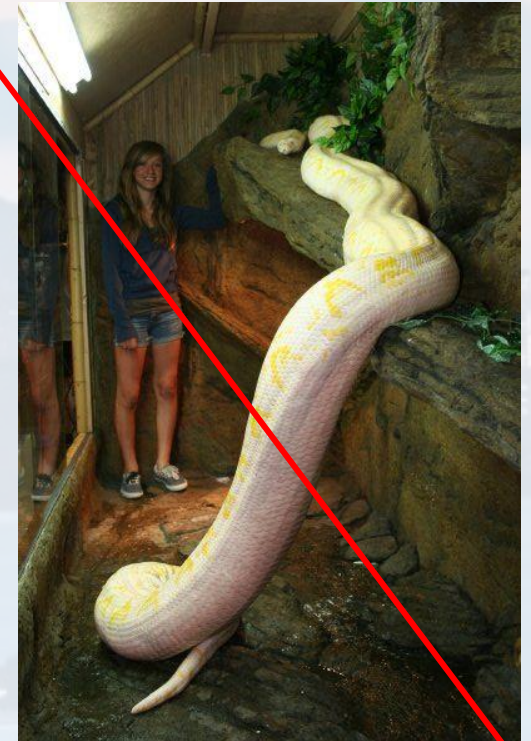
- 1990, von Guido van Rossum



- named after “**Monty Python**”, not the serpent



















- idea: flexible, simple and compact syntax, extendable







TIOBE index July 2024

Jul 2024	Jul 2023	Change	Programming Language		Ratings	Change
1	1			Python	16.12%	+2.70%
2	3	▲		C++	10.34%	-0.46%
3	2	▼		C	9.48%	-2.08%
4	4			Java	8.59%	-1.91%
5	5			C#	6.72%	-0.15%
6	6			JavaScript	3.79%	+0.68%
7	13	▲▲		Go	2.19%	+1.12%
8	7	▼		Visual Basic	2.08%	-0.82%
9	11	▲		Fortran	2.05%	+0.80%
10	8	▼		SQL	2.04%	+0.57%
11	15	▲▲		Delphi/Object Pascal	1.89%	+0.91%
12	10	▼		MATLAB	1.34%	+0.08%
13	17	▲▲		Rust	1.18%	+0.29%
14	16	▲		Ruby	1.16%	+0.25%
15	12	▼		Scratch	1.15%	+0.08%
16	9	▼▼		PHP	1.15%	-0.27%



 **ANACONDA** Products ▾ Pricing Solutions ▾ Resources ▾ Blog Company ▾ [Get Started](#)


<https://www.anaconda.com/>

 **ANACONDA**

Data science technology for
human sensemaking.

A movement that brings together millions of data science practitioners,
data-driven enterprises, and the open source community.

Help

 **ANACONDA NAVIGATOR**











Home

Environments

Learning

Community

Applications on: base (root) Channels

 CMD.exe Prompt 0.1.1 Run a cmd.exe terminal with your current environment from Navigator activated. Launch	 JupyterLab 1.2.6 An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture. Launch	 Jupyter Notebook 6.0.3 Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis. Launch	 Powershell Prompt 0.0.1 Run a Powershell terminal with your current environment from Navigator activated. Launch	 Qt Console 4.6.0 PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more. Launch	 Spyder 4.0.1 Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features. Launch
 VS Code 1.45.1 Streamlined code editor with support for development operations like debugging, task running and version control. Launch	 Glueviz 0.15.2 Multidimensional data visualization across files. Explore relationships within and among related datasets. Install	 Orange 3 3.23.1 Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox. Install	 RStudio 1.1.456 A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks. Install		



Spyder

folder navigator

The screenshot shows the Spyder Python IDE interface. The left sidebar contains a 'folder navigator' showing the file structure of the current project. The main editor displays a 'py script' (a Python file named 'howToImplConvLayerMaxpoolFlatt.py') which is currently untitled. The right sidebar shows the 'workspace' area, which displays current plots or variables. At the bottom, the 'console' area is visible, showing the Python interpreter output and the IPython prompt. Annotations with boxes and arrows point to these specific components: 'content of current folder' points to the file explorer, 'py script: yet untitled, we are going to need later' points to the code editor, 'workspace: displays current plots or variables' points to the variable explorer, and 'console: typing commands & executing scripts' points to the IPython console.

content of current folder

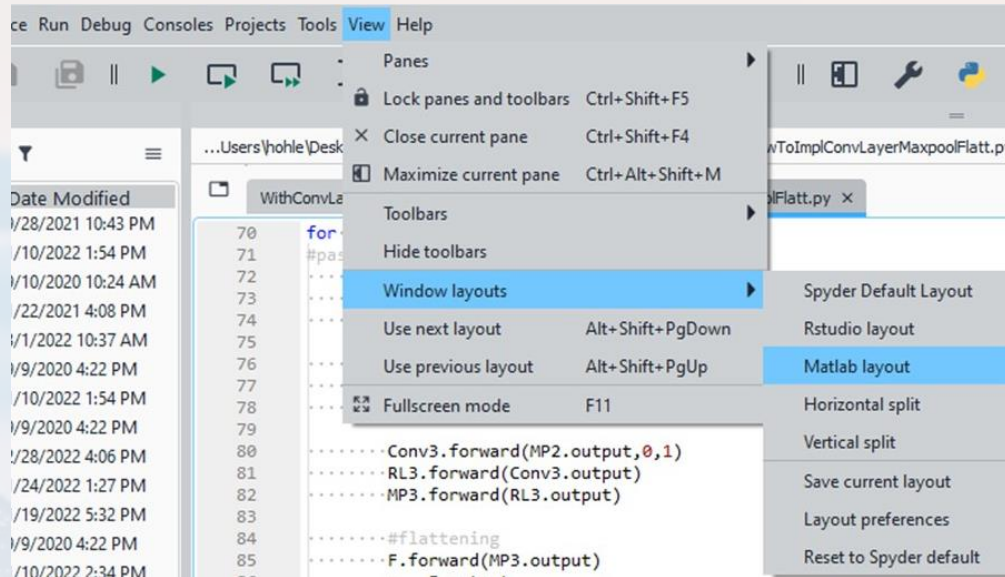
py script: yet untitled, we are going to need later

workspace: displays current plots or variables

console: typing commands & executing scripts



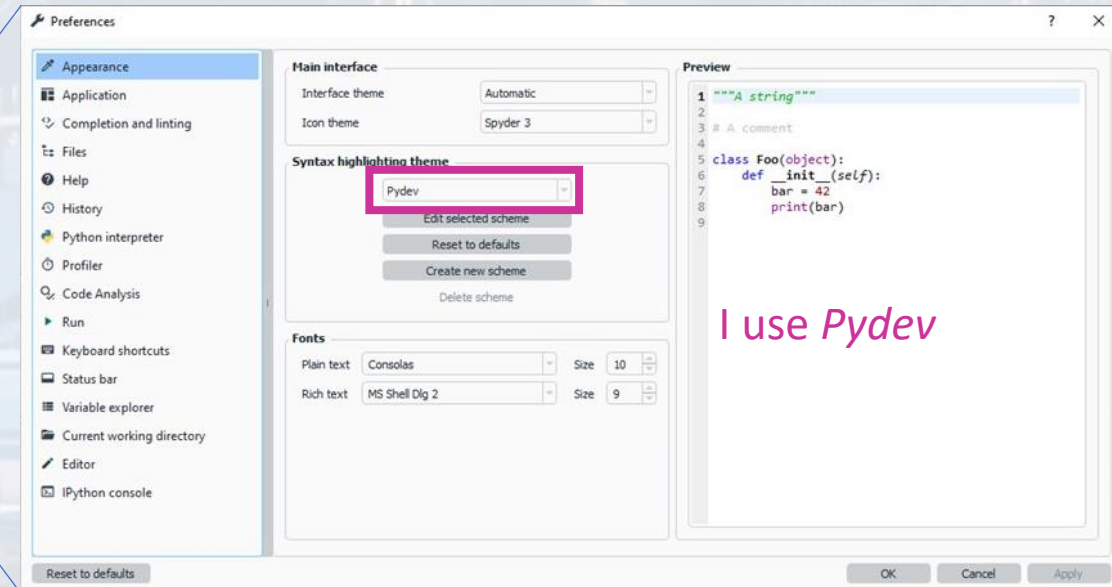
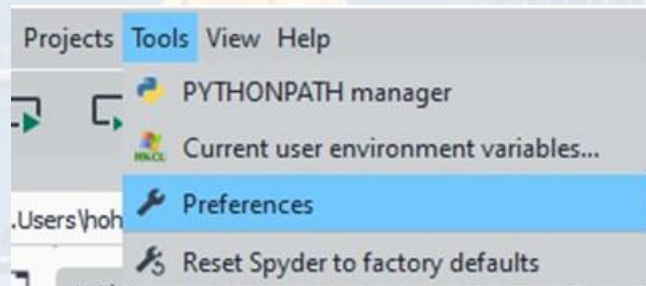
Spyder



settings:

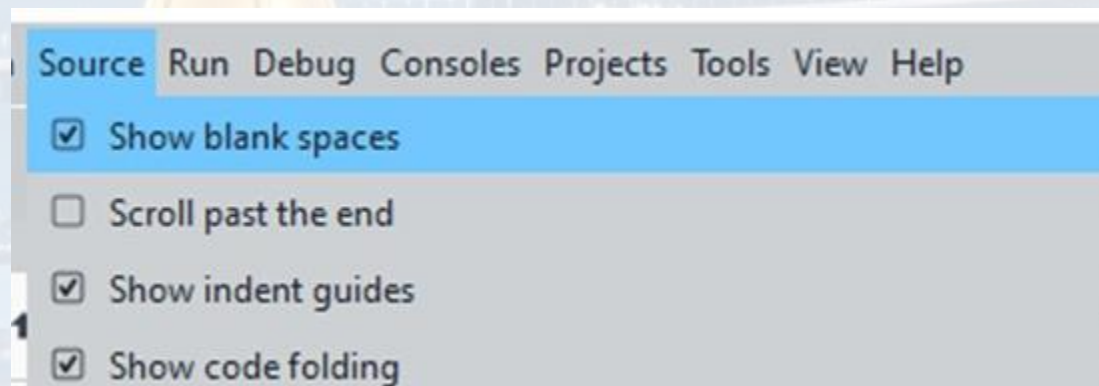
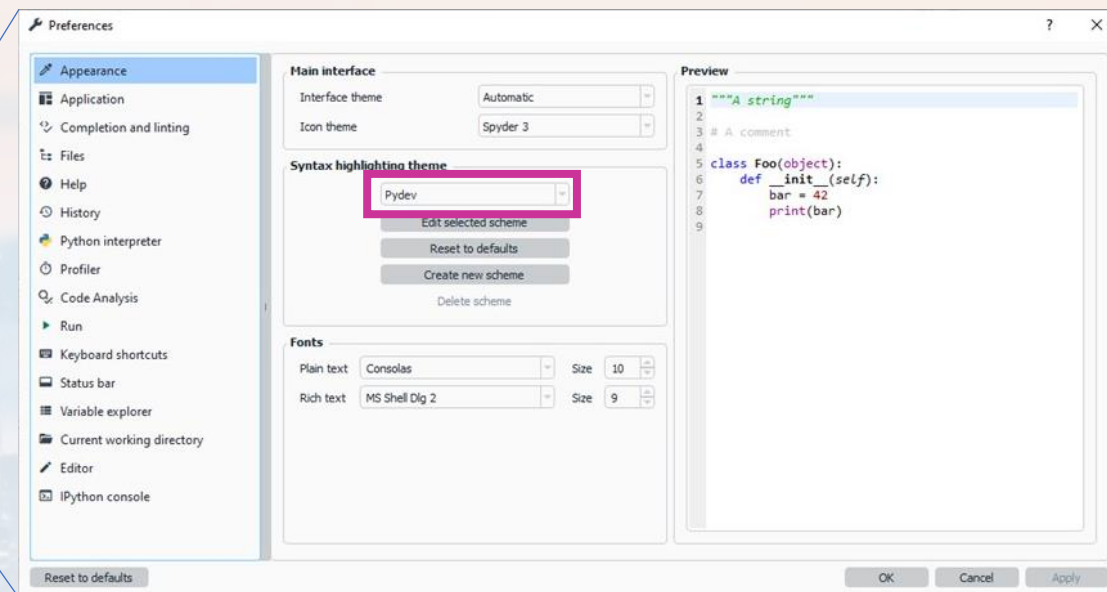
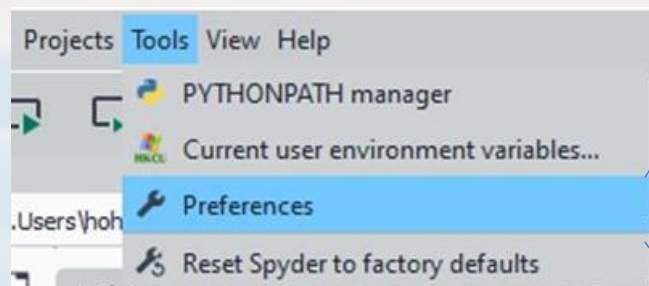
toolbar: *View* → *Window layouts*

e. g. Matlab





Spyder



blanks are relevant for synthax!



Jupyter

lofi hip hop radio - beats to
PLAYING

Home Page - Select or create a

localhost:8888/tree

jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them.

0 /

3D Objects

anaconda3

Contacts

Desktop

Documents

Downloads

Favorites

Jedi

Links

Music

OneDrive

Pictures

Saved Games

Searches

localhost:8888/tree#

Upload New

Notebook:

Python 3 (ipykernel)

Create a new notebook with Python 3 (ipykernel)

Text File

Folder

Terminal

2 months ago

5 days ago

a year ago

a year ago

a year ago

a year ago

a year ago

a year ago

a year ago

Type here to search

11:33
04/12/2023



Jupyter

lofi hip hop radio - beats to
PLAYING

Home Page - Select or create a

Untitled1 - Jupyter Notebook

localhost:8888/notebooks/Untitled1.ipynb?kernel_name=python3

jupyter Untitled1 Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

In []: |

jupyter Untitled1 Last Checkpoint: 7 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Save Add Delete Copy Paste Up Down Run Stop Restart Code

In []: `print("test")`

jupyter Untitled1 Last Checkpoint: 9 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

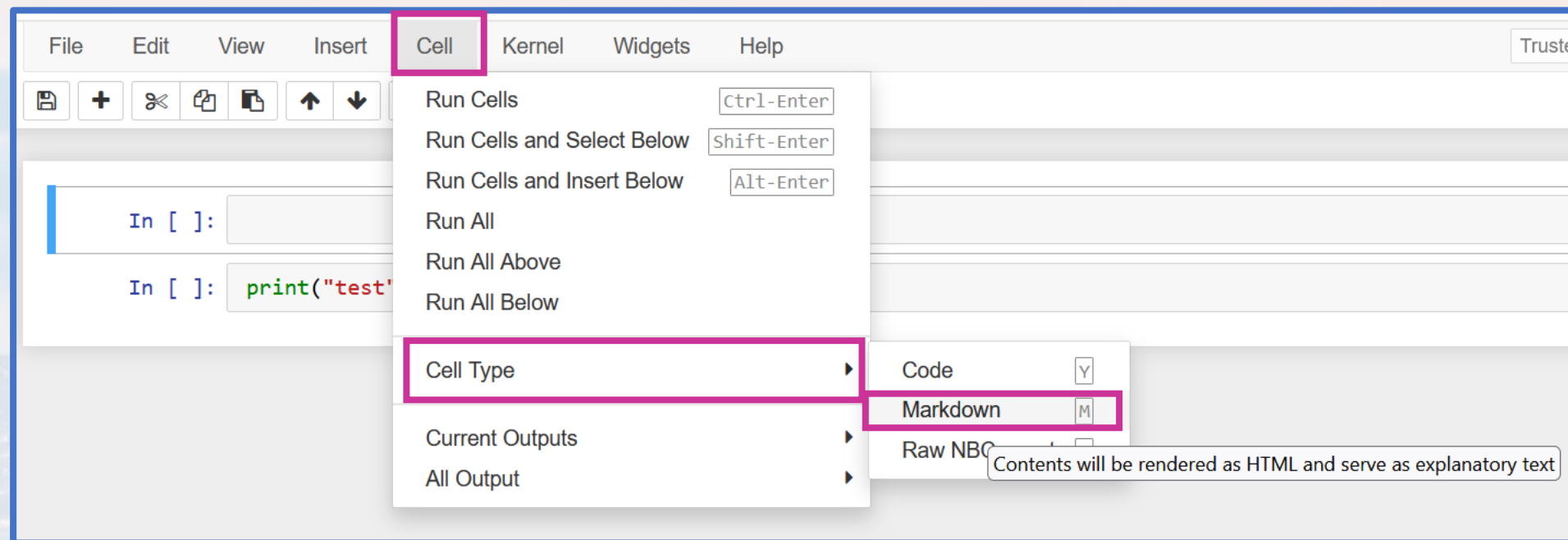
Save Add Delete Copy Paste Up Down Run Stop Restart Code

In []: `print("test")`

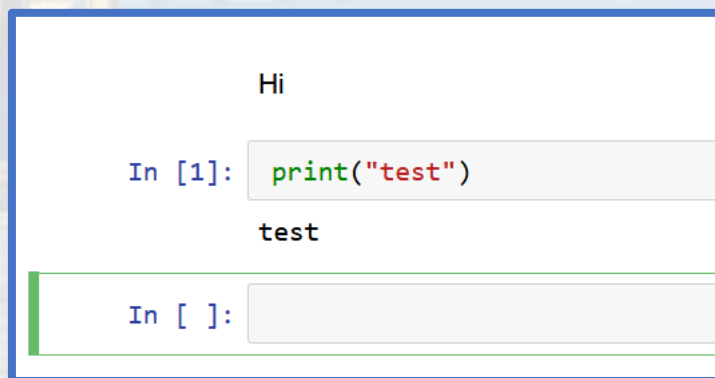
Insert Cell Above A
Insert Cell Below B
Insert an empty Code cell above the currently active cell



Jupyter



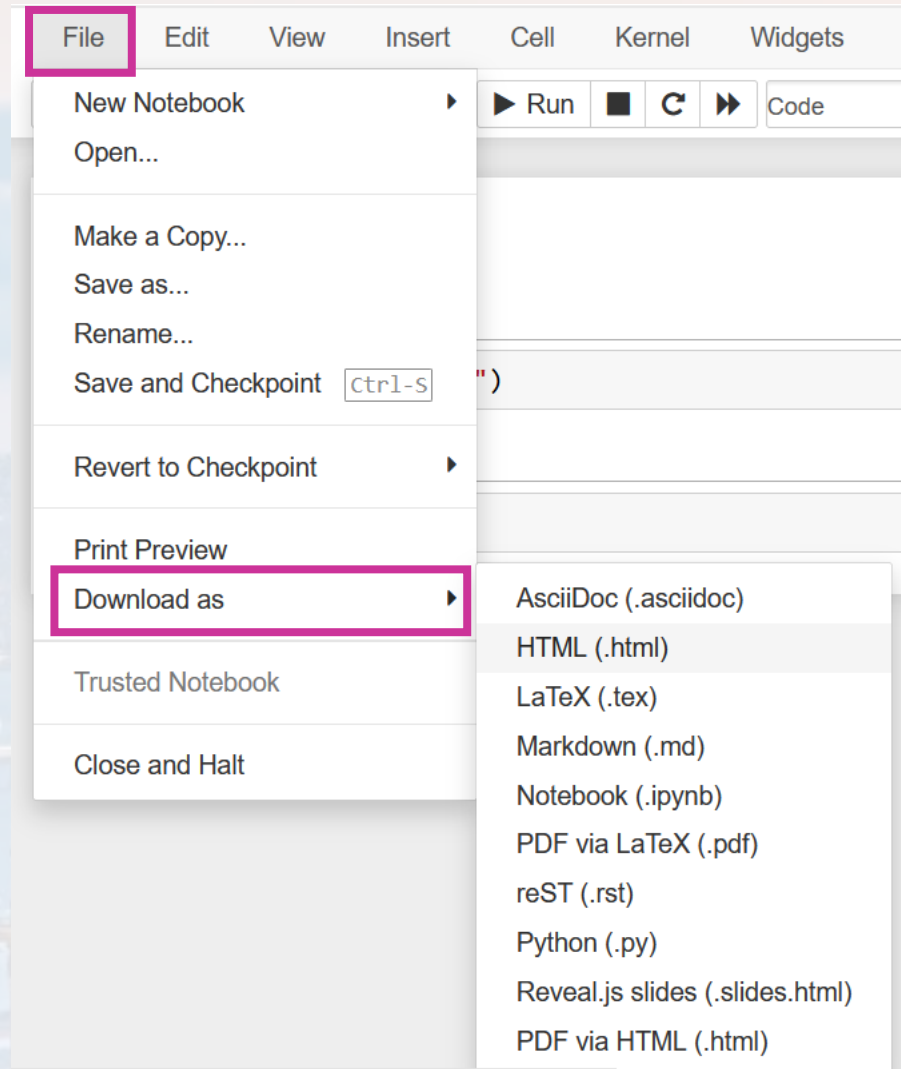
type something → click **run**



web: Jupyter → markdown styles
(including LaTeX)



Jupyter





basic types:

```
String = 'Hello ' + 'World'
```

```
List = [1, 2, 3, 5, 'World'] #default
```

```
Tuple = (1, 2)
```

```
Dict = {'A': 1, 'B': 2}
```

```
Array = np.array([1, 2, 3, 5])
```

```
pd.DataFrame
```

other objects:

```
class
```

```
def #function/method
```




- labels and titles of plots
- paths and file names
- error messages

```
String    = 'Hello ' + 'World'  
List      = [1, 2, 3, 5, 'World']  
Tuple     = (1, 2)  
Dict      = {'A': 1, 'B': 2}  
Array     = np.array([1, 2, 3, 5])  
pd.DataFrame
```

```
string1 = 'Hello Students'
```

```
string2 = ', how are you'
```

```
string12 = string1 + string2
```

```
Out[1]: 'Hello Students, how are you'
```

concatenating is incredibly easy!

```
S = 'abc'
```

```
3*S
```

```
Out[2]: 'abcabcabc'
```

```
string12[2:6]
```

slicing

```
[ 1, 5, 0, -3]
```

slices: 0 1 2 3 4



- labels and titles of plots
- paths and file names
- error messages

string12[2:6]

slicing

```
String    = 'Hello ' + 'World'  
List      = [1, 2, 3, 5, 'World']  
Tuple     = (1, 2)  
Dict      = {'A': 1, 'B': 2}  
Array     = np.array([1, 2, 3, 5])  
pd.DataFrame
```

slices: 0 1 2 3 4
 [1, 5, 0, -3]

index: 0 1 2 3
 [1, 5, 0, -3]

indexing

index: -4 -3 -2 -1

string12[-1]

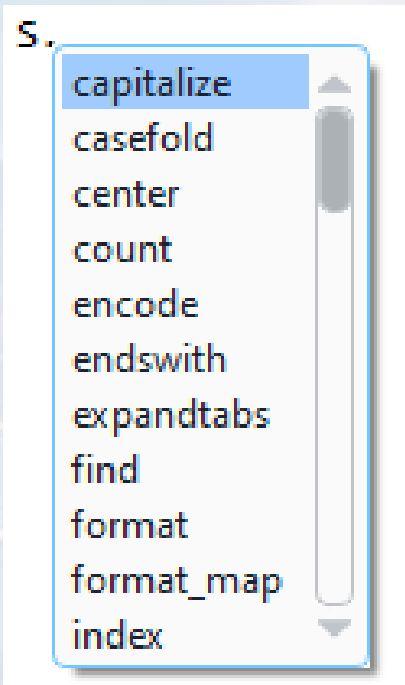
string12[1:]

string12[:-1]



- labels and titles of plots
- paths and file names
- error messages

```
String    = 'Hello ' + 'World'  
List      = [1, 2, 3, 5, 'World']  
Tuple     = (1, 2)  
Dict      = {'A': 1, 'B': 2}  
Array     = np.array([1, 2, 3, 5])  
pd.DataFrame
```



try some of the functions like

```
S.count()  
S.find()
```



default type in python

```
L = [1, 2, 3, -2]
```

```
2*L
```

```
Out[3]: [1, 2, 3, -2, 1, 2, 3, -2]
```

```
type(L)
```

```
Out[4]: list
```

```
String    = 'Hello ' + 'World'  
List      = [1, 2, 3, 5, 'World']  
Tuple     = (1, 2)  
Dict      = {'A': 1, 'B': 2}  
Array     = np.array([1, 2, 3, 5])  
pd.DataFrame
```




default type in python

be careful:

```
L1 = [1, 2]
```

```
L2 = L1
```

```
L1[0] = 5
```

```
print(L2[0])
```

What is the result? What happens if you do the same with an **int**?

```
L1 = [1, 2]
```

```
L2 = L1.copy()
```

```
L1[0] = 5
```

```
print(L2[0])
```

```
String = 'Hello ' + 'World'  
List = [1, 2, 3, 5, 'World']  
Tuple = (1, 2)  
Dict = {'A': 1, 'B': 2}  
Array = np.array([1, 2, 3, 5])  
pd.DataFrame
```

lists, dictionaries, sets
and bytearrays are **mutable**.



```
L1 = [1, 2, 4]
```

```
L2 = ['a', 'b', 'c']
```

```
T = (L1, L2)
```

```
len(T)
```

```
Out[5]: 2
```

```
T[1]
```

```
Out[6]: ['a', 'b', 'c']
```

```
type(T)
```

```
Out[7]: tuple
```

```
(out1, out2) = T
```

```
out1
```

```
Out[8]: [1, 2, 4]
```

```
String    = 'Hello ' + 'World'  
List      = [1, 2, 3, 5, 'World']  
Tuple     = (1, 2)  
Dict      = {'A': 1, 'B': 2}  
Array     = np.array([1, 2, 3, 5])  
pd.DataFrame
```




key value

```
Dict = {'A': [1, 0, 0, 0], 'C': [0, 1, 0, 0], 'G': [0, 0, 1, 0], 'T': [0, 0, 0, 1]}
```

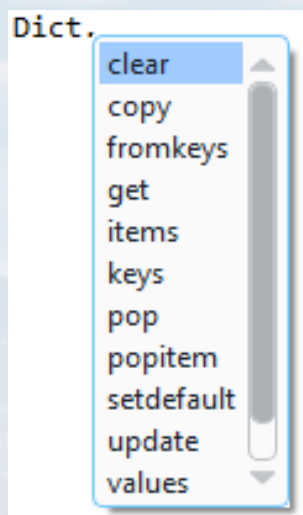
```
Dict['A']  
Out[9]: [1, 0, 0, 0]
```

```
type(Dict)  
Out[21]: dict
```

```
String = 'Hello ' + 'World'  
List = [1, 2, 3, 5, 'World']  
Tuple = (1, 2)  
Dict = {'A': 1, 'B': 2}  
Array = np.array([1, 2, 3, 5])  
pd.DataFrame
```



check out



```
String    = 'Hello ' + 'World'  
List      = [1, 2, 3, 5, 'World']  
Tuple     = (1, 2)  
Dict      = {'A': 1, 'B': 2}  
Array     = np.array([1, 2, 3, 5])  
pd.DataFrame
```

try some of the functions like

```
Dict.values()
```

```
Dict.pop('A')
```

```
Dict.update({'U': [2, 0, 0, 0]})
```




syntax:

[] → **arrays** (lists, np.array, data frames → see next week)

() → functions or **tuple**

T = (a, v)

(a_new, v_new) = T

{ } → **Dictionaries**

```
Dict = { 'A': [1,0,0,0], 'C': [0,1,0,0], 'G': [0,0,1,0], 'T': [0,0,0,1]}
```

```
Dict['A']
```

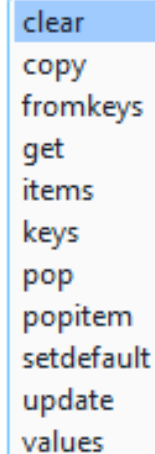
```
Out[8]: [1, 0, 0, 0]
```

```
String = 'Hello ' + 'World'  
List   = [1, 2, 3, 5, 'World']  
Tuble  = (1, 2)  
Dict    = {'A': 1, 'B': 2}  
Array   = np.array([1, 2, 3, 5])  
pd.DataFrame
```



syntax:

Dict.



A screenshot of a Python dictionary's methods menu. The menu is titled 'Dict.' and lists the following methods: clear, copy, fromkeys, get, items, keys, pop, popitem, setdefault, update, and values. The 'clear' method is highlighted at the top of the list.

type:

`dir(dict)`

```
['__class_getitem__',  
 '__contains__',  
 '__delattr__',  
 '__delitem__',  
 '__dir__',  
 '__doc__',  
 '__eq__',  
 '__format__',  
 '__ge__',  
 '__getattr__',  
 '__getitem__',  
 '__getstate__',  
 '__gt__',  
 '__hash__',  
 '__init__',  
 '__init_subclass__',  
 '__ior__',  
 '__iter__',  
 '__le__',  
 '__len__',  
 '__lt__',  
 '__ne__',  
 '__new__',  
 '__or__',  
 '__reduce__',  
 '__reduce_ex__',  
 '__repr__',  
 '__reversed__',  
 '__ror__',  
 '__setattr__',  
 '__setitem__',  
 '__sizeof__',  
 '__str__',  
 '__subclasshook__',  
 'clear',  
 'copy',  
 'fromkeys',  
 'get',  
 'items',  
 'keys',  
 'pop',  
 'popitem',  
 'setdefault',  
 'update',  
 'values']
```

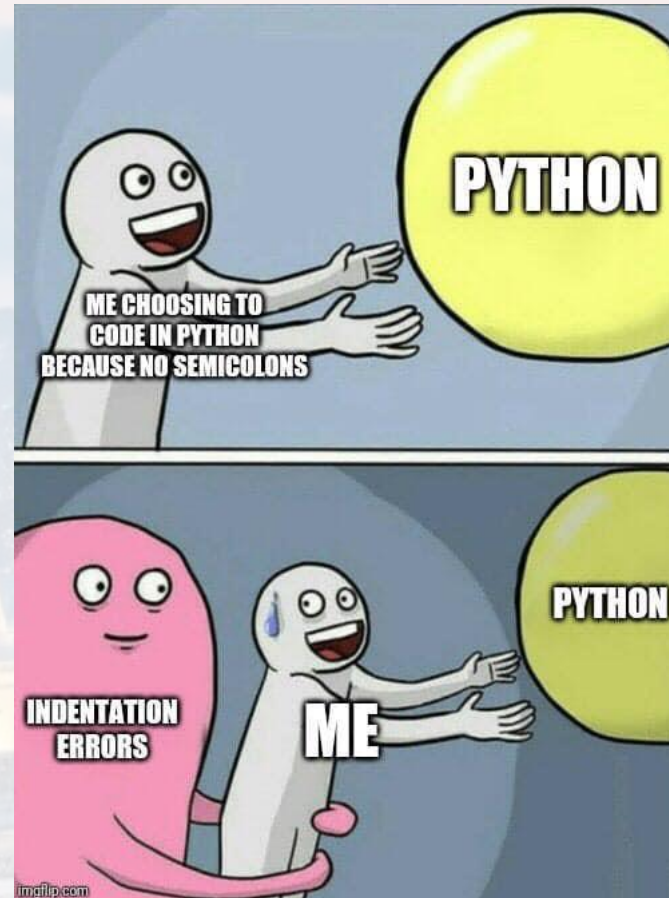
attributes

functions aka methods

```
String    = 'Hello ' + 'World'  
List      = [1, 2, 3, 5, 'World']  
Tuble     = (1, 2)  
Dict      = {'A': 1, 'B': 2}  
Array     = np.array([1, 2, 3, 5])  
pd.DataFrame
```




Introduction to Unix & Python



Thank you for your attention!