Anaconda and jupyter notebook tutorial and installation instructions

Motivation:

Up to this point, most of the programming languages that we have met in other courses in the Technion are highly standardized (C, C++, Java, ML etc').
Meaning they have a very narrow set of versions (C90, C99, C11), that holds (most of the times) 4 important properties that are sometimes taken for granted:

1) The versions are backward compatible.
2) Libraries are not using each other.
3) Libraries are implemented for every version.
4) Libraries are stable (don't have many bugs).

The meaning of backward compatibility is that code that was written using C99 syntax, can be compiled as if it is written in C11 standard (again, for most of the language common features).

Libraries are not cross dependent meaning that, for example, if you are calling a function *f* from stdlib, *f* it will not call for functions that are in stdio.

Those properties enable us to just compile the code using a specific standard, which in return will use the same version for every imported library, and that the code will be compiled together successfully.

Python unfortunately lacks all of those properties:
There are many versions of python, and those versions are not always backward compatible.
In addition to that, python encourages writing of open source libraries from unofficial entities, and those libraries are almost always using other libraries in their implementation.

This cause a mess of versions control: in order ro run a python project, in addition to the python language version, ones should also provide the versions of all of the libraries that are being used in the code, and to manually make sure that every library is using other libraries with version that are compatible with its own version (in addition to import those nested libraries).

The set of all of the libraries and their versions is called an environment.

Anaconda is basically a tool that manages environments.
For a given version of python, and for every library that is being used, anaconda calculates the dependencies between all of the components and enforce them.

In Anaconda you can create many different environments, and use each one for a specific task.

In addition to that, if you want to use a specific library with a specific version, you can tell Anaconda to do so, and it will calculate the new dependencies and enforce them as well (or tell you that it is impossible).

Installing Anaconda:

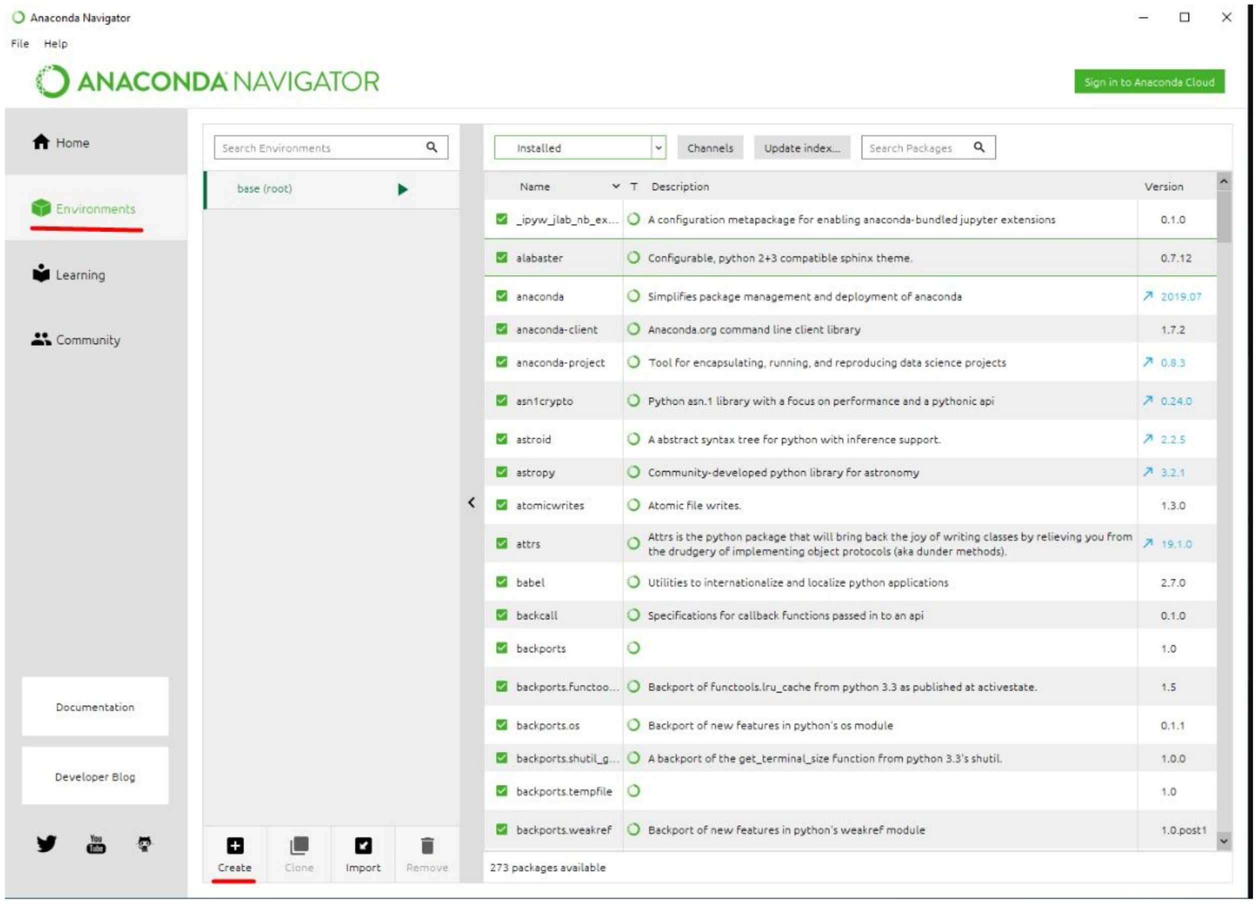
This is relevant for you pc, the server is being set up using a tool named pip, which manually install every library (and its dependent libraries).
please refer to the server instruction in the website for full instructions.

Download from the official site: https://www.anaconda.com/distribution/
Install using the instructions at the official site as well:
https://docs.anaconda.com/anaconda/install/

Installation is pretty straightforward.

Using Anaconda:

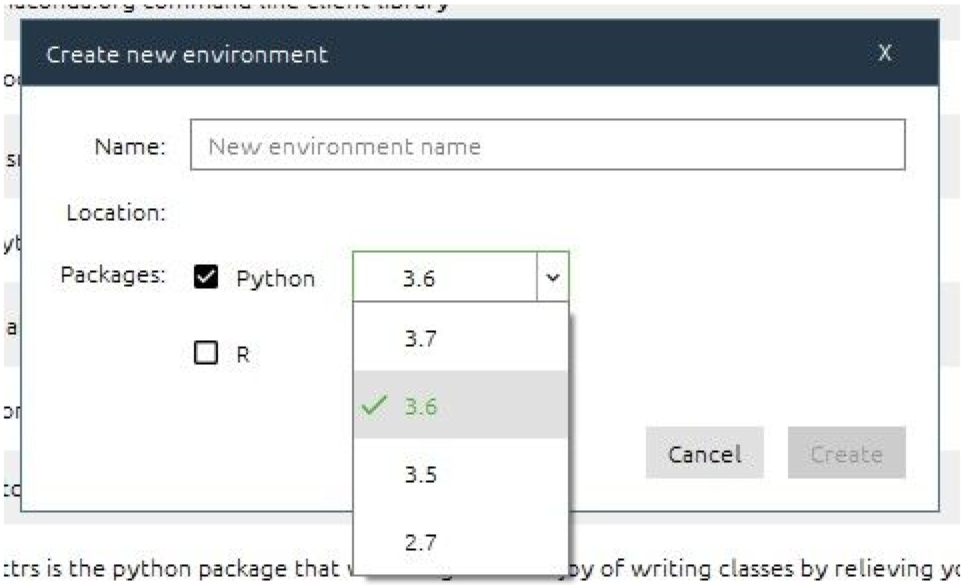
To create an environment, go to environment -> Create:



	Name	T	Description	Version
✓	_ipyw_jlab_nb_ex...	○	A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
✓	alabaster	○	Configurable, python 2+3 compatible sphinx theme.	0.7.12
✓	anaconda	○	Simplifies package management and deployment of anaconda	↗ 2019.07
✓	anaconda-client	○	Anaconda.org command line client library	1.7.2
✓	anaconda-project	○	Tool for encapsulating, running, and reproducing data science projects	↗ 0.8.3
✓	asn1crypto	○	Python asn.1 library with a focus on performance and a pythonic api	↗ 0.24.0
✓	astroid	○	A abstract syntax tree for python with inference support.	↗ 2.2.5
✓	astropy	○	Community-developed python library for astronomy	↗ 3.2.1
✓	atomicwrites	○	Atomic file writes.	1.3.0
✓	attrs	○	Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	↗ 19.1.0
✓	babel	○	Utilities to internationalize and localize python applications	2.7.0
✓	backcall	○	Specifications for callback functions passed in to an api	0.1.0
✓	backports	○		1.0
✓	backports.functoo...	○	Backport of functools.lru_cache from python 3.3 as published at activestate.	1.5
✓	backports.os	○	Backport of new features in python's os module	0.1.1
✓	backports.shutil_g...	○	A backport of the get_terminal_size function from python 3.3's shutil.	1.0.0
✓	backports.tempfile	○		1.0
✓	backports.weakref	○	Backport of new features in python's weakref module	1.0.post1

273 packages available

And choose a name and python version:


Create new environment

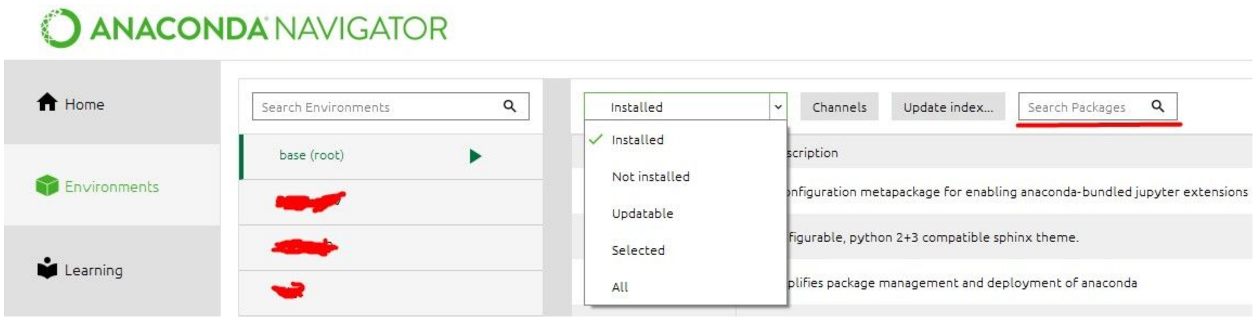
Name: New environment name

Location:

Packages: ☑ Python 3.6

☐ R

Adding a library to the environment is done by searching it here, make sure to mark Not installed or All for new libraries:



[redacted]

[redacted]

[redacted]

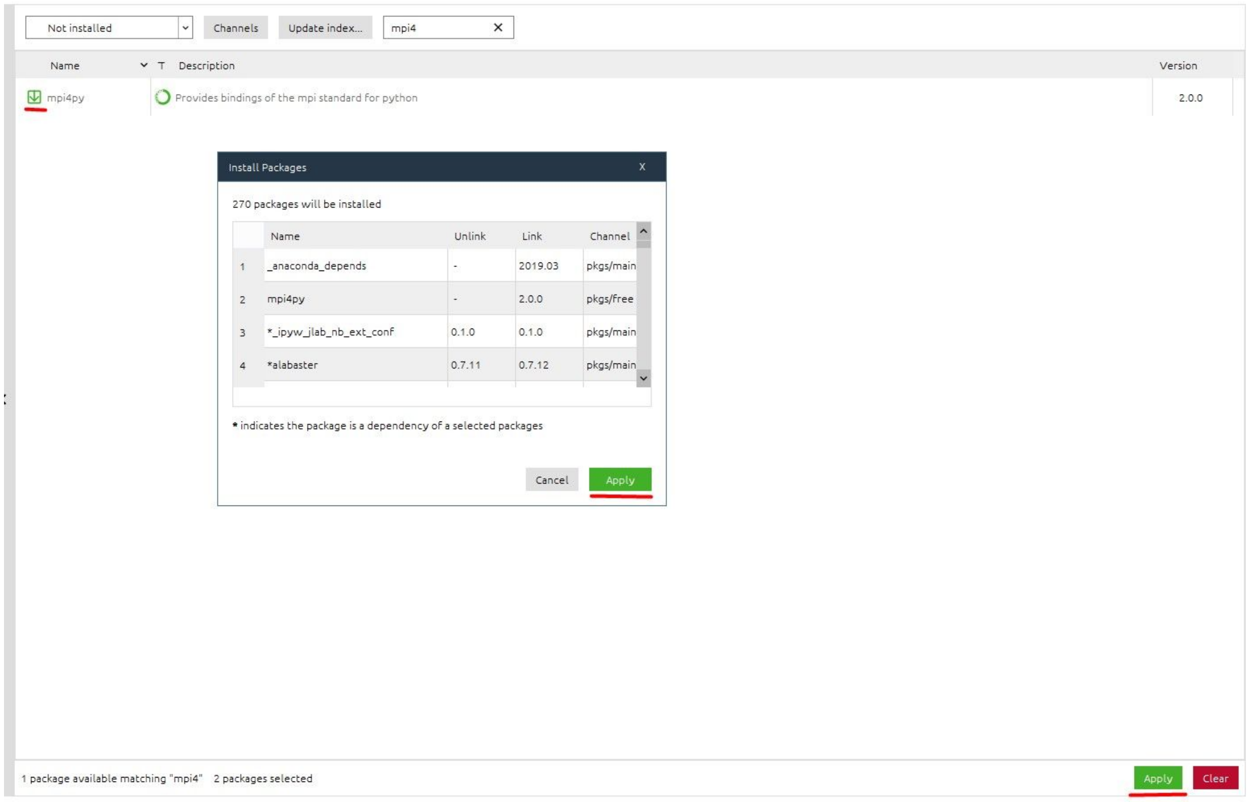
scription

nfiguration metapackage for enabling anaconda-bundled jupyter extensions

figurable, python 2+3 compatible sphinx theme.

plifies package management and deployment of anaconda

To install a library, select its empty mark location, and hit Apply at the bottom right, this will cause anaconda to calculate the dependencies of the new library (could take up to a few minutes in worst cases), and after that to show you the result for approval:



Name	Description	Version
mpi4py	Provides bindings of the mpi standard for python	2.0.0

Install Packages

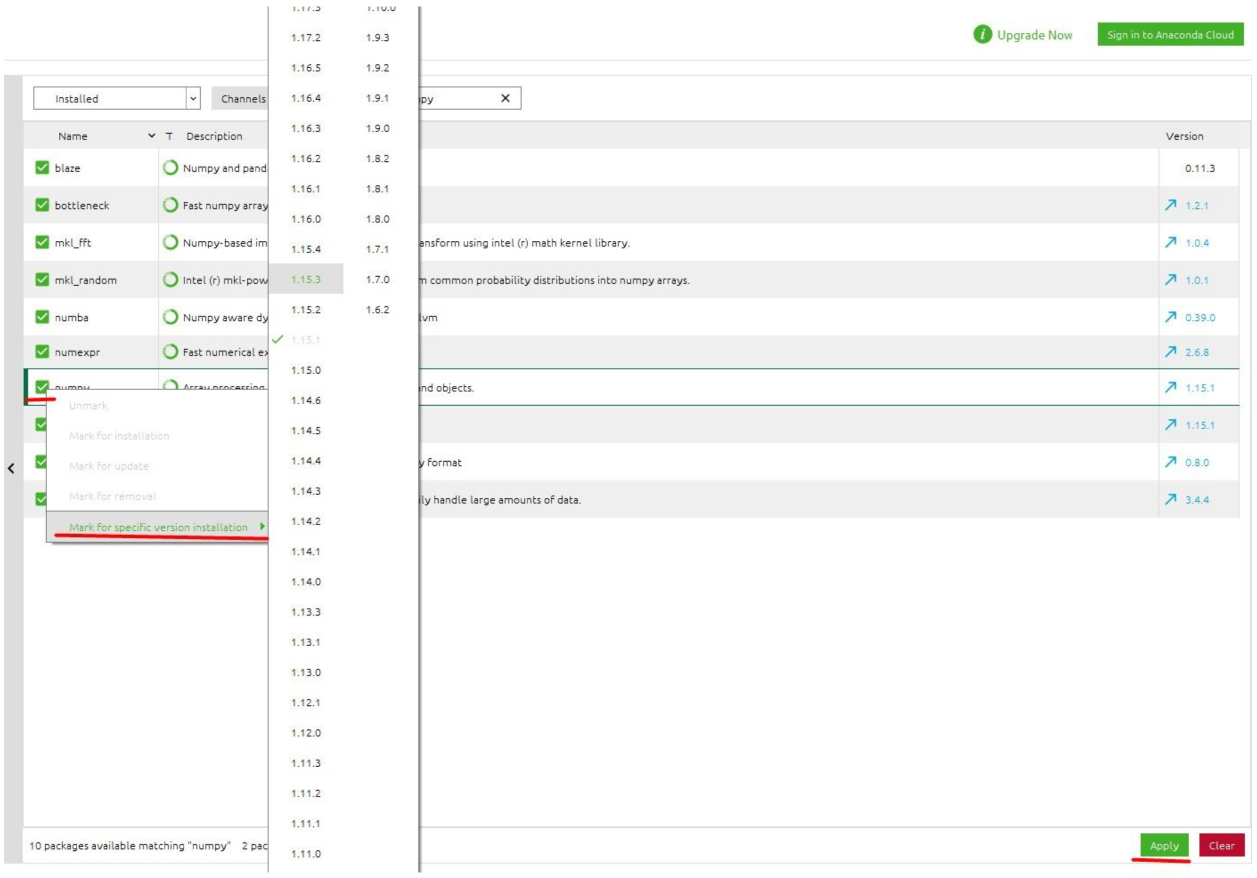
270 packages will be installed

	Name	Unlink	Link	Channel
1	_anaconda_depends	-	2019.03	pkgs/main
2	mpi4py	-	2.0.0	pkgs/free
3	*_ipyw_jlab_nb_ext_conf	0.1.0	0.1.0	pkgs/main
4	*alabaster	0.7.11	0.7.12	pkgs/main

* indicates the package is a dependency of a selected packages

1 package available matching "mpi4"   2 packages selected

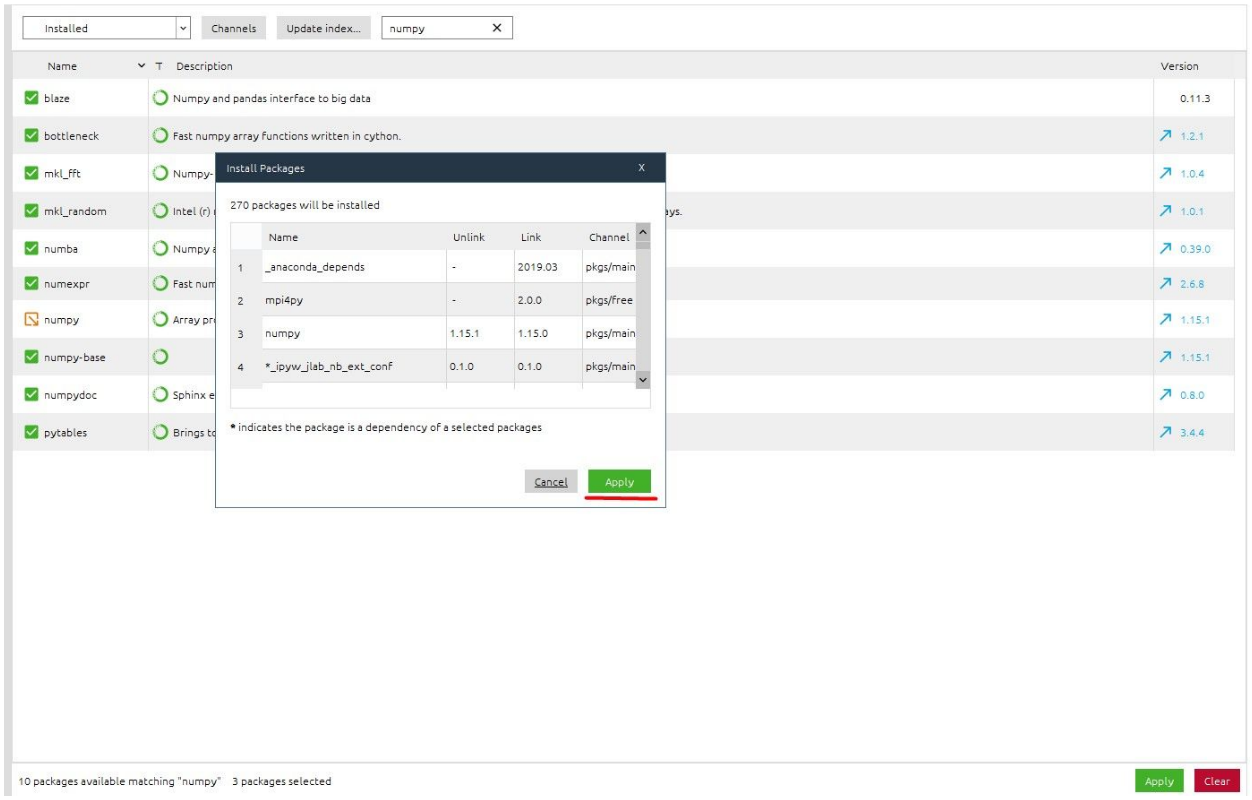
To change a version of a library, hit the (full) mark button of the library, choose a version, and hit the Apply button.
Anaconda will calculate the dependencies of the new version, and make sure that all of the libraries that depends on the current library are still compatible, and if not calculate their new versions:



Name	T	Description	Version
blaze		Numpy and pand	0.11.3
bottleneck		Fast numpy array	1.2.1
mkl_fft		Numpy-based im … ansform using intel (r) math kernel library.	1.0.4
mkl_random		Intel (r) mkl-pow … m common probability distributions into numpy arrays.	1.0.1
numba		Numpy aware dy … vm	0.39.0
numexpr		Fast numerical ex	2.6.8
numpy		Array processing … nd objects.	1.15.1
			1.15.1
		… y format	0.8.0
		… ily handle large amounts of data.	3.4.4

```
1.17.3   1.10.0
1.17.2   1.9.3
1.16.5   1.9.2
1.16.4   1.9.1
1.16.3   1.9.0
1.16.2   1.8.2
1.16.1   1.8.1
1.16.0   1.8.0
1.15.4   1.7.1
1.15.3   1.7.0
1.15.2   1.6.2
✓ 1.15.1
1.15.0
1.14.6
1.14.5
1.14.4
1.14.3
1.14.2
1.14.1
1.14.0
1.13.3
1.13.1
1.13.0
1.12.1
1.12.0
1.11.3
1.11.2
1.11.1
1.11.0
```

10 packages available matching "numpy"   2 pac



Name	T	Description	Version
blaze		Numpy and pandas interface to big data	0.11.3
bottleneck		Fast numpy array functions written in cython.	1.2.1
mkl_fft		Numpy-	1.0.4
mkl_random		Intel (r) ... ays.	1.0.1
numba		Numpy	0.39.0
numexpr		Fast num	2.6.8
numpy		Array pr	1.15.1
numpy-base			1.15.1
numpydoc		Sphinx e	0.8.0
pytables		Brings to	3.4.4

Install Packages

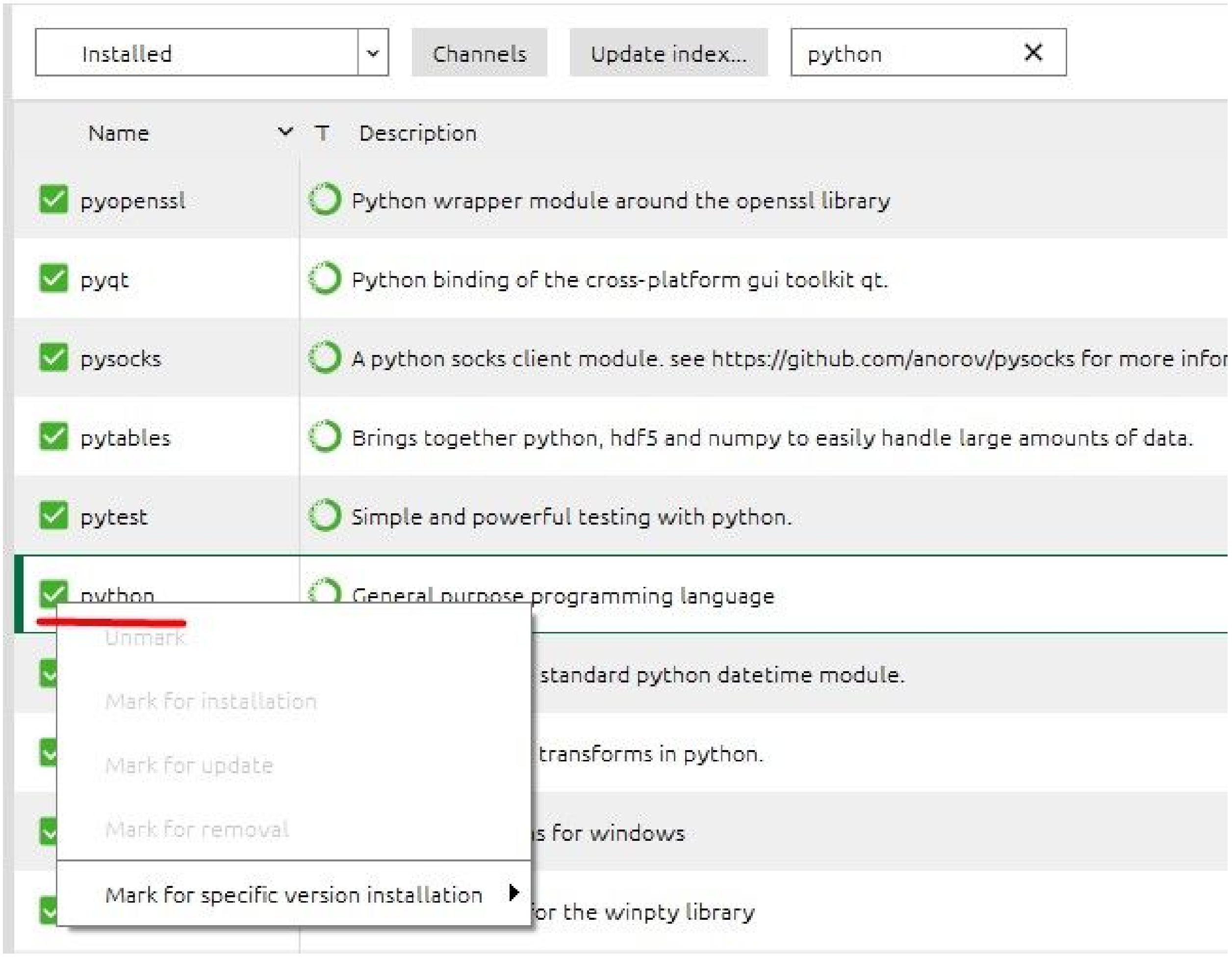
270 packages will be installed

	Name	Unlink	Link	Channel
1	_anaconda_depends	-	2019.03	pkgs/main
2	mpi4py	-	2.0.0	pkgs/free
3	numpy	1.15.1	1.15.0	pkgs/main
4	*_ipyw_jlab_nb_ext_conf	0.1.0	0.1.0	pkgs/main

* indicates the package is a dependency of a selected packages

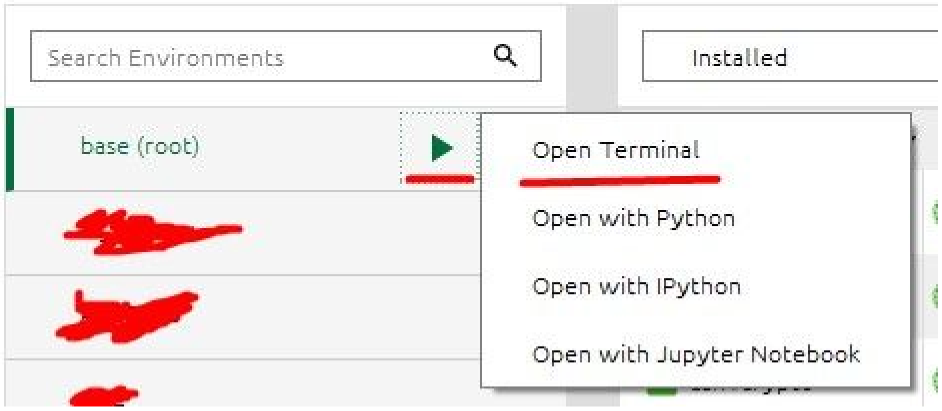
10 packages available matching "numpy" 3 packages selected

Changing python version in done in the same way as any other library:


Installed | Channels | Update index... | python ✕

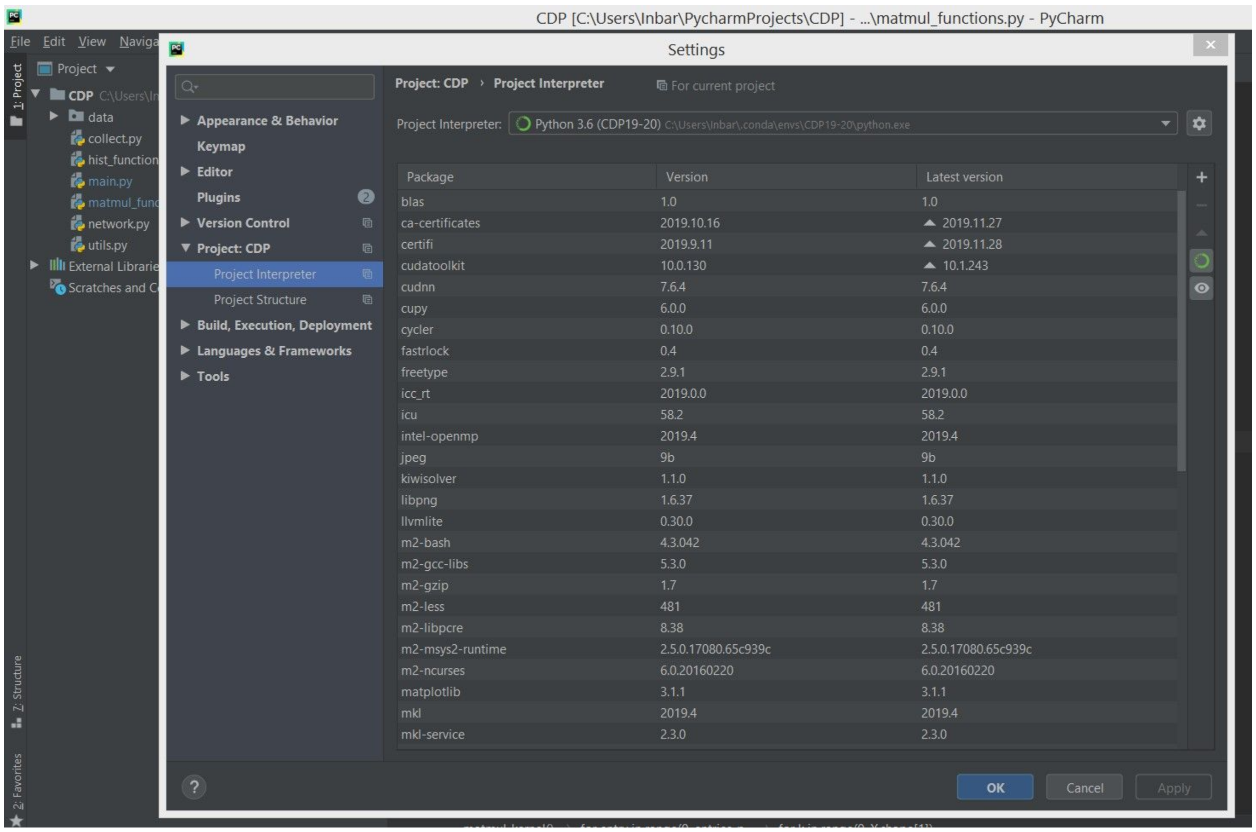
Name	T	Description
✓ pyopenssl		Python wrapper module around the openssl library
✓ pyqt		Python binding of the cross-platform gui toolkit qt.
✓ pysocks		A python socks client module. see https://github.com/anorov/pysocks for more infor
✓ pytables		Brings together python, hdf5 and numpy to easily handle large amounts of data.
✓ pytest		Simple and powerful testing with python.
✓ python		General purpose programming language
✓		standard python datetime module.
✓		transforms in python.
✓		s for windows
✓		for the winpty library

In order to launch a terminal that can executes python code with the defined environment, click on the play sign -> open terminal:



[redacted]
[redacted]
[redacted]

If you are using pycharm (recommended) you can bind an anaconda environment to a project.
In order to bind it to an existing project, go to:
file -> setting -> Project <project name> -> Project interpreter


CDP [C:\Users\Inbar\PycharmProjects\CDP] - ...\matmul_functions.py - PyCharm

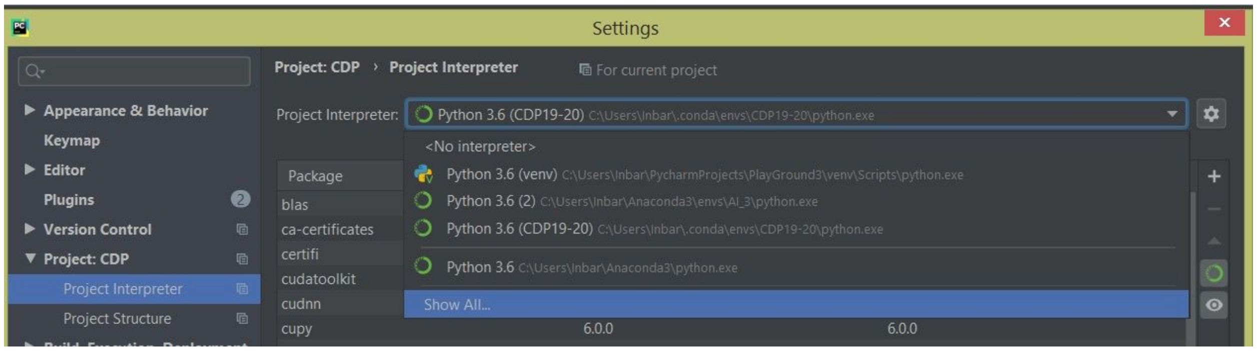
Settings

Project: CDP › Project Interpreter — For current project

Project Interpreter: Python 3.6 (CDP19-20) C:\Users\Inbar\.conda\envs\CDP19-20\python.exe

Package	Version	Latest version
blas	1.0	1.0
ca-certificates	2019.10.16	▲ 2019.11.27
certifi	2019.9.11	▲ 2019.11.28
cudatoolkit	10.0.130	▲ 10.1.243
cudnn	7.6.4	7.6.4
cupy	6.0.0	6.0.0
cycler	0.10.0	0.10.0
fastrlock	0.4	0.4
freetype	2.9.1	2.9.1
icc_rt	2019.0.0	2019.0.0
icu	58.2	58.2
intel-openmp	2019.4	2019.4
jpeg	9b	9b
kiwisolver	1.1.0	1.1.0
libpng	1.6.37	1.6.37
llvmlite	0.30.0	0.30.0
m2-bash	4.3.042	4.3.042
m2-gcc-libs	5.3.0	5.3.0
m2-gzip	1.7	1.7
m2-less	481	481
m2-libpcre	8.38	8.38
m2-msys2-runtime	2.5.0.17080.65c939c	2.5.0.17080.65c939c
m2-ncurses	6.0.20160220	6.0.20160220
matplotlib	3.1.1	3.1.1
mkl	2019.4	2019.4
mkl-service	2.3.0	2.3.0

And at the Project interpreter tab choose the required environment (go to show all if it is not there):

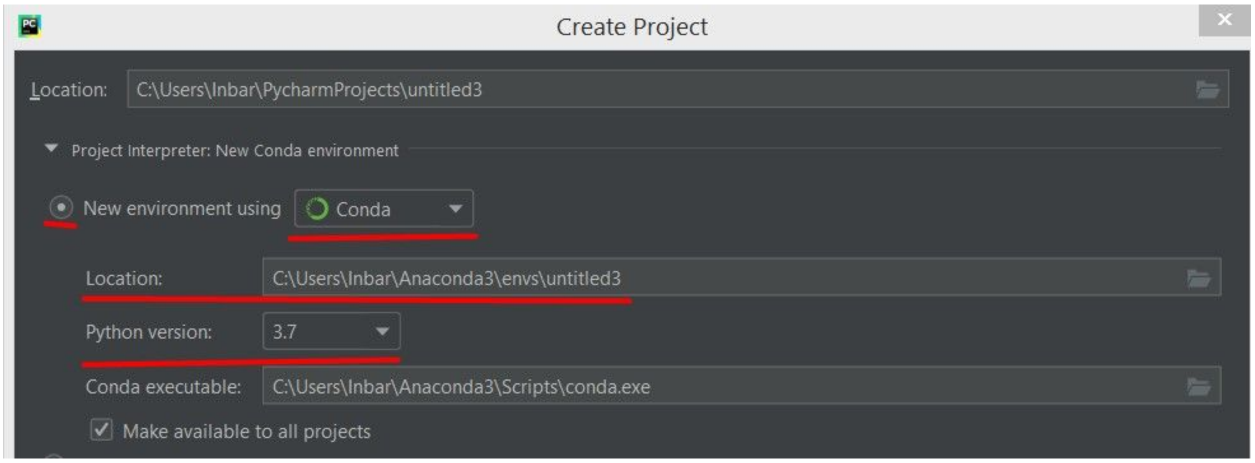

Settings

Project: CDP › Project Interpreter — For current project

Project Interpreter: Python 3.6 (CDP19-20) C:\Users\Inbar\.conda\envs\CDP19-20\python.exe

<No interpreter>
Python 3.6 (venv) C:\Users\Inbar\PycharmProjects\PlayGround3\venv\Scripts\python.exe
Python 3.6 (2) C:\Users\Inbar\Anaconda3\envs\AI_3\python.exe
Python 3.6 (CDP19-20) C:\Users\Inbar\.conda\envs\CDP19-20\python.exe
Python 3.6 C:\Users\Inbar\Anaconda3\python.exe
Show All...

For a new project, there are two ways:
First is to create a new anaconda environment, and configure it in anaconda (configuration can be done also in pycharm, easier in anaconda):


Create Project

Location: C:\Users\Inbar\PycharmProjects\untitled3

Project Interpreter: New Conda environment

New environment using: Conda

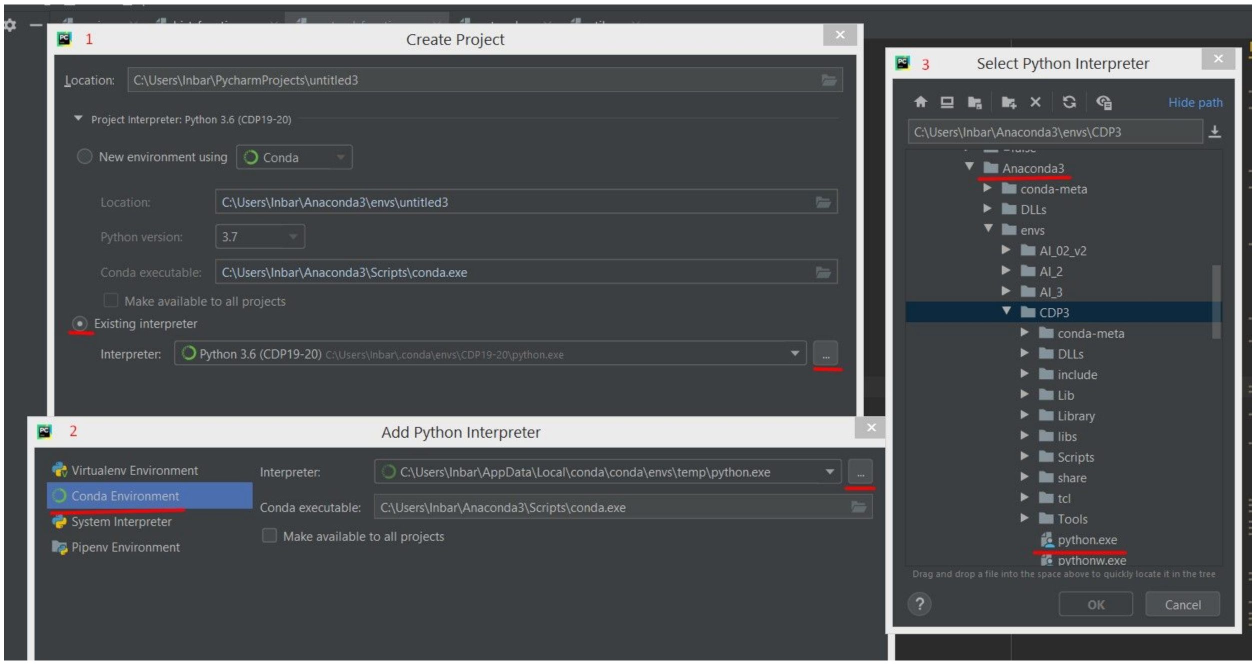
Location: C:\Users\Inbar\Anaconda3\envs\untitled3

Python version: 3.7

Conda executable: C:\Users\Inbar\Anaconda3\Scripts\conda.exe

☑ Make available to all projects

Second way is to import an existing environment, to do it follow the steps in the next picture (steps 1 and 2):


1 Create Project

Location: C:\Users\Inbar\PycharmProjects\untitled3

Project Interpreter: Python 3.6 (CDP19-20)

New environment using: Conda

Location: C:\Users\Inbar\Anaconda3\envs\untitled3

Python version: 3.7

Conda executable: C:\Users\Inbar\Anaconda3\Scripts\conda.exe

Make available to all projects

● Existing interpreter

Interpreter: Python 3.6 (CDP19-20) C:\Users\Inbar\.conda\envs\CDP19-20\python.exe ...

2 Add Python Interpreter

Virtualenv Environment
Conda Environment
System Interpreter
Pipenv Environment

Interpreter: C:\Users\Inbar\AppData\Local\conda\conda\envs\temp\python.exe ...

Conda executable: C:\Users\Inbar\Anaconda3\Scripts\conda.exe

Make available to all projects

3 Select Python Interpreter

Hide path

C:\Users\Inbar\Anaconda3\envs\CDP3

- Anaconda3
  - conda-meta
  - DLLs
  - envs
    - AI_02_v2
    - AI_2
    - AI_3
    - CDP3
      - conda-meta
      - DLLs
      - include
      - Lib
      - Library
      - libs
      - Scripts
      - share
      - tcl
      - Tools
      - python.exe
      - pythonw.exe

Drag and drop a file into the space above to quickly locate it in the tree

OK Cancel

Then (step 3), manually find your installation location of anaconda on your pc, and choose the required environment:
<anaconda installation location>/env/<env name>/python.exe in windows.

Jupyter notebook:

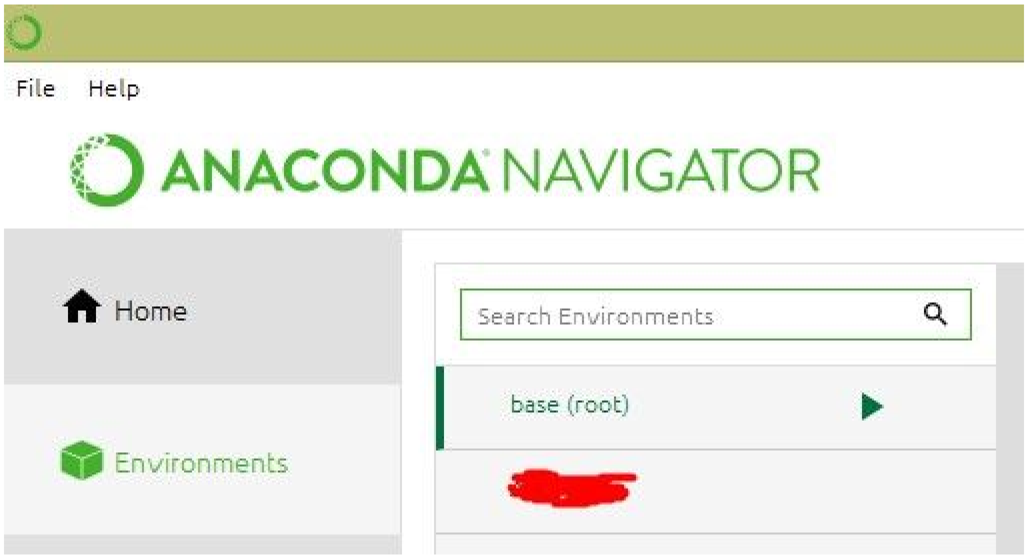
This is from the description of Jupyter at the official site:

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text

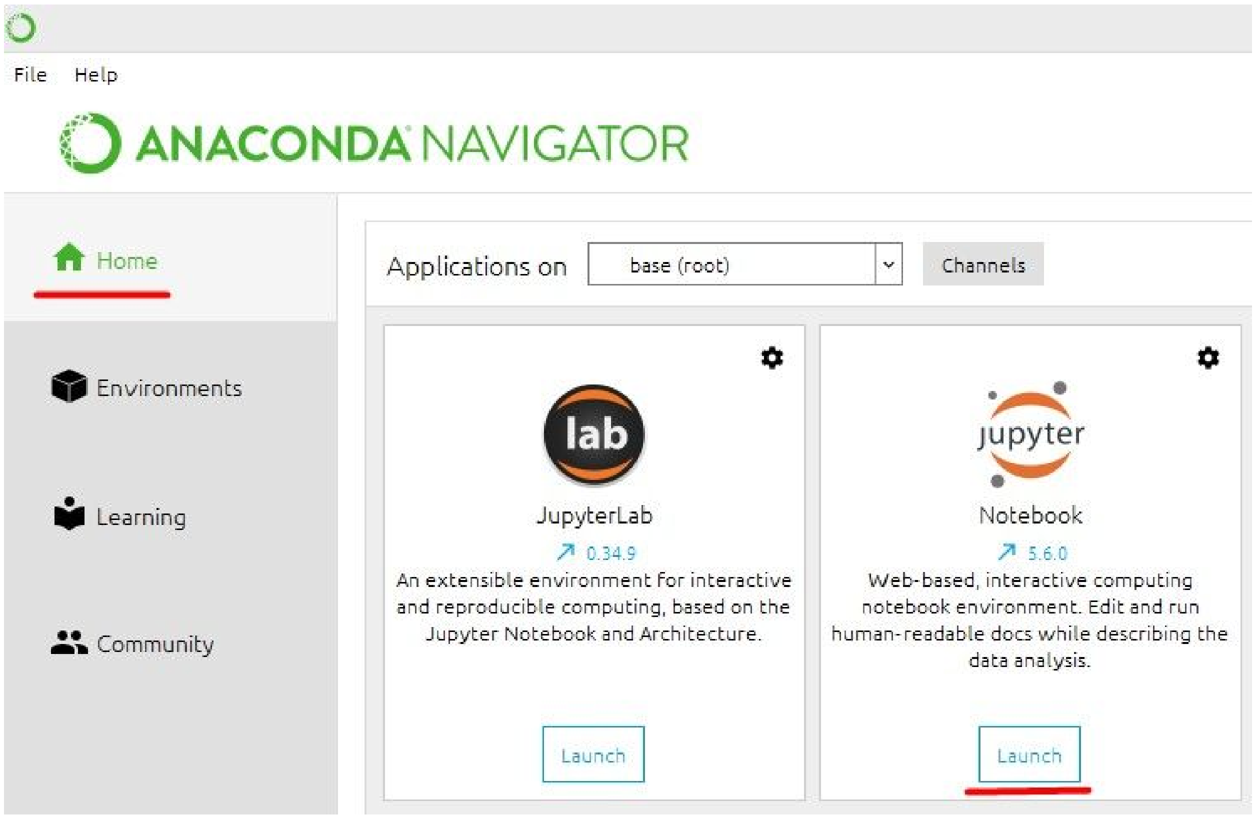
If 30 years ago, a notebook was a bunch of papers that were physically being written on.
10 years ago, it could be a word document containing pictures and links.
Jupyter is the next step in evolution, that enables running actual code inside a notebook.

In the case of python code, the same version mess still needs to be handled.
The common solution is to launch a Jupyter notebook directly from anaconda, with a preconfigured environment that anaconda manages.

To choose an environment simply click on its name, make sure the play sign is located on the required environment:



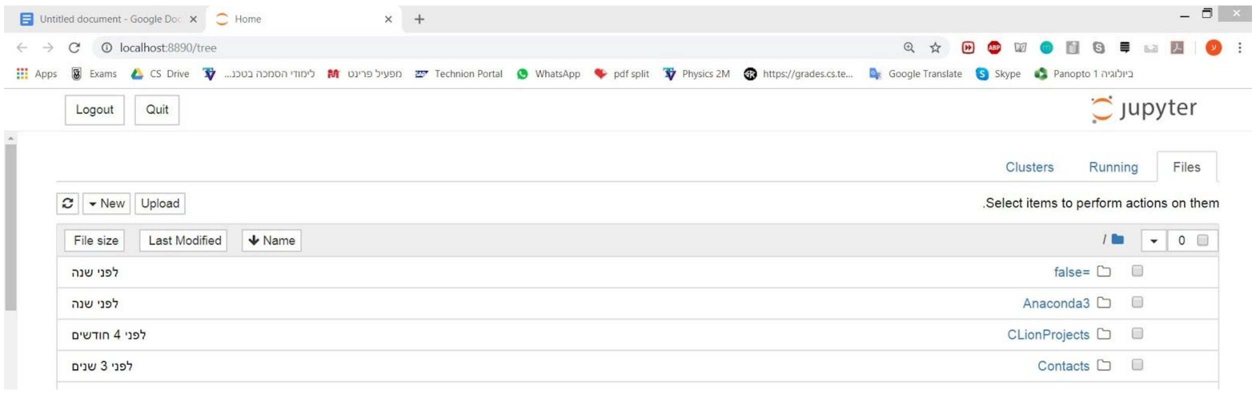
Then go to home and launch Jupyter (first time anaconda will install Jupyter, afterwards it will just launch it), of click the play => Open with Jupyter notebook after it is installed:



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

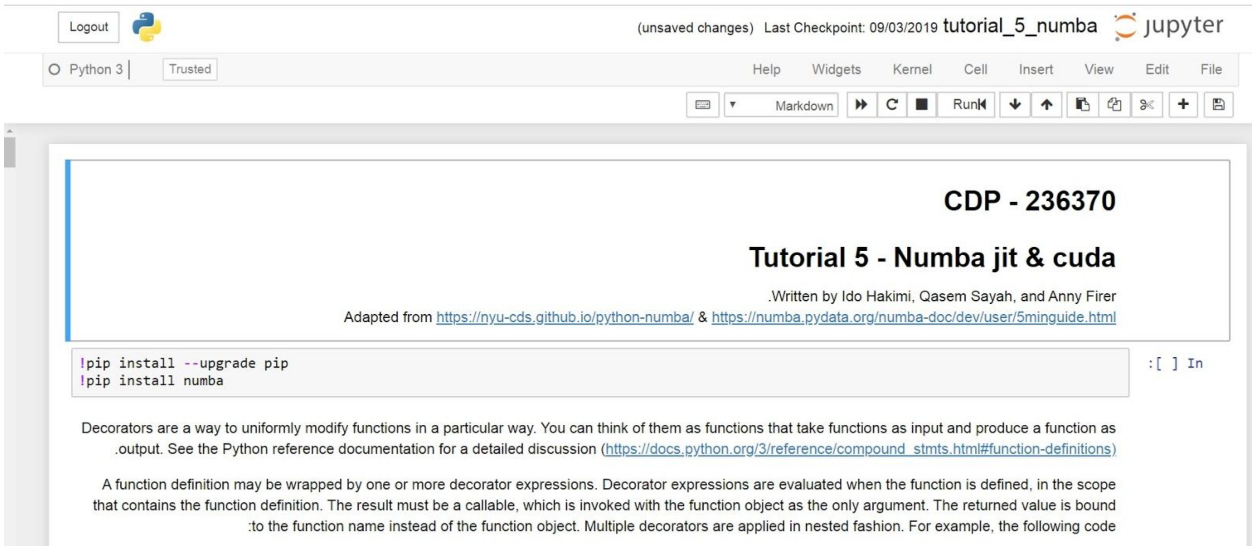
This will launch Jupyter in your browser, inside a page that looks like this:



.Select items to perform actions on them

לפני שנה	false=
לפני שנה	Anaconda3
לפני 4 חודשים	CLionProjects
לפני 3 שנים	Contacts

In order to open a notebook from your pc, find it and click on it, this will open a page that looks like this:


Logout
(unsaved changes) Last Checkpoint: 09/03/2019 tutorial_5_numba
jupyter
Python 3
Trusted
Help Widgets Kernel Cell Insert View Edit File
Markdown
Run

CDP - 236370

Tutorial 5 - Numba jit & cuda

.Written by Ido Hakimi, Qasem Sayah, and Anny Firer
Adapted from https://nyu-cds.github.io/python-numba/ & https://numba.pydata.org/numba-doc/dev/user/5minguide.html

```
!pip install --upgrade pip
!pip install numba
```

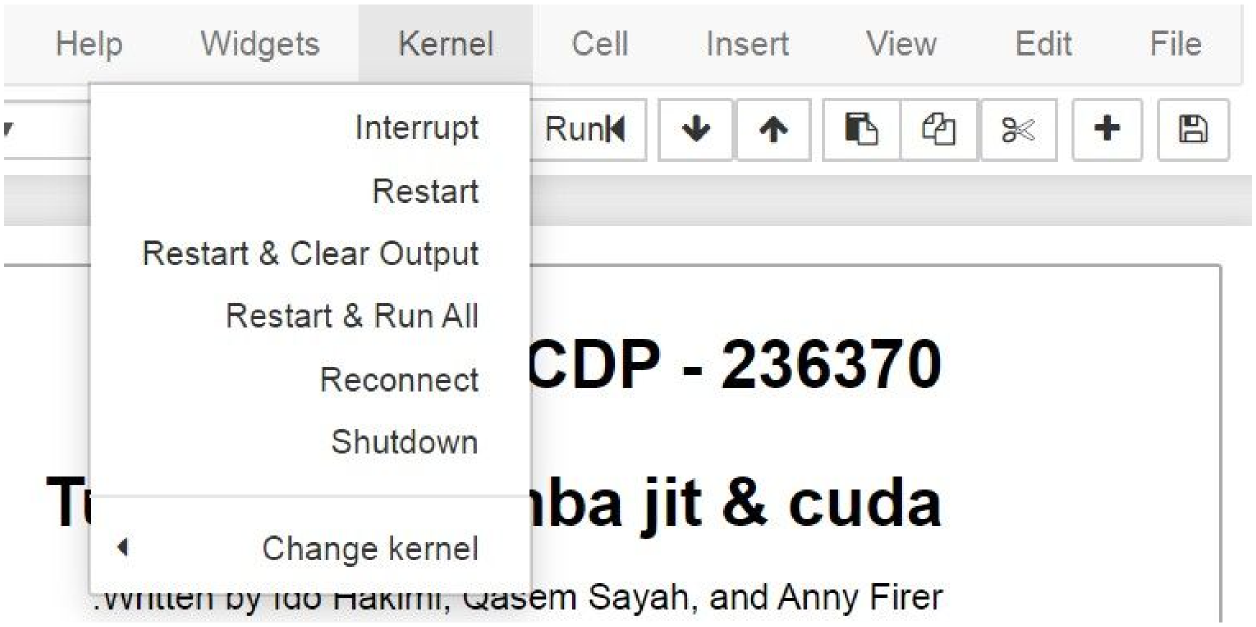
:[ ] In

Decorators are a way to uniformly modify functions in a particular way. You can think of them as functions that take functions as input and produce a function as .output. See the Python reference documentation for a detailed discussion (https://docs.python.org/3/reference/compound_stmts.html#function-definitions)

A function definition may be wrapped by one or more decorator expressions. Decorator expressions are evaluated when the function is defined, in the scope that contains the function definition. The result must be a callable, which is invoked with the function object as the only argument. The returned value is bound :to the function name instead of the function object. Multiple decorators are applied in nested fashion. For example, the following code

The cell that contains the :[ ] sign right of it, is a block of code for the python interpreter to execute, in order to do it, click the sign itself.
After each first run of a block of code, the :[ ] will be filled with 1, and every other execution on any other block will fill it with the appropriate execution number.

At the toolbar you can find the Kernel tab:



The kernel in this context is the component in jupyter that executes code.
Restart will effectively reset the environment.

Jupytes has many tricks under its sleeves, this is the basic that needs in order to use the tutorials.