COIT20277 Introduction to Artificial Intelligence

Week 1 - Tutorial

- Anaconda Installation
- Jupyter Notebook Basics
- Python Review

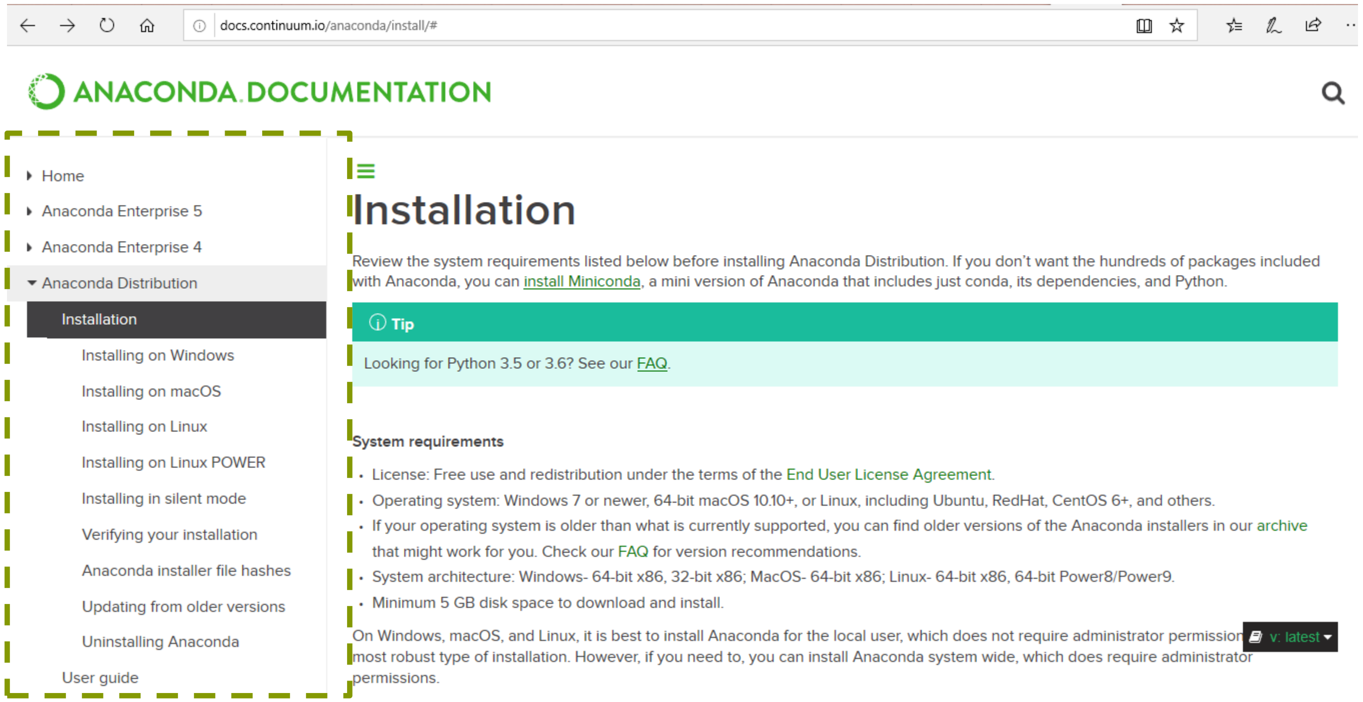
Acknowledgement of Country

I respectfully acknowledge the Traditional Custodians of the land on which we live, work and learn. I pay my respects to the First Nations people and their Elders, past, present and future

CQUniversity AUSTRALIA

BE WHAT YOU WANT TO BE
cqu.edu.au

Depending on the machine (Windows/Mac/Linux), follow the installation guides as per the link here: http://docs.continuum.io/anaconda/install/#



Installation

Review the system requirements listed below before installing Anaconda Distribution. If you don't want the hundreds of packages included with Anaconda, you can install Miniconda, a mini version of Anaconda that includes just conda, its dependencies, and Python.

Tip

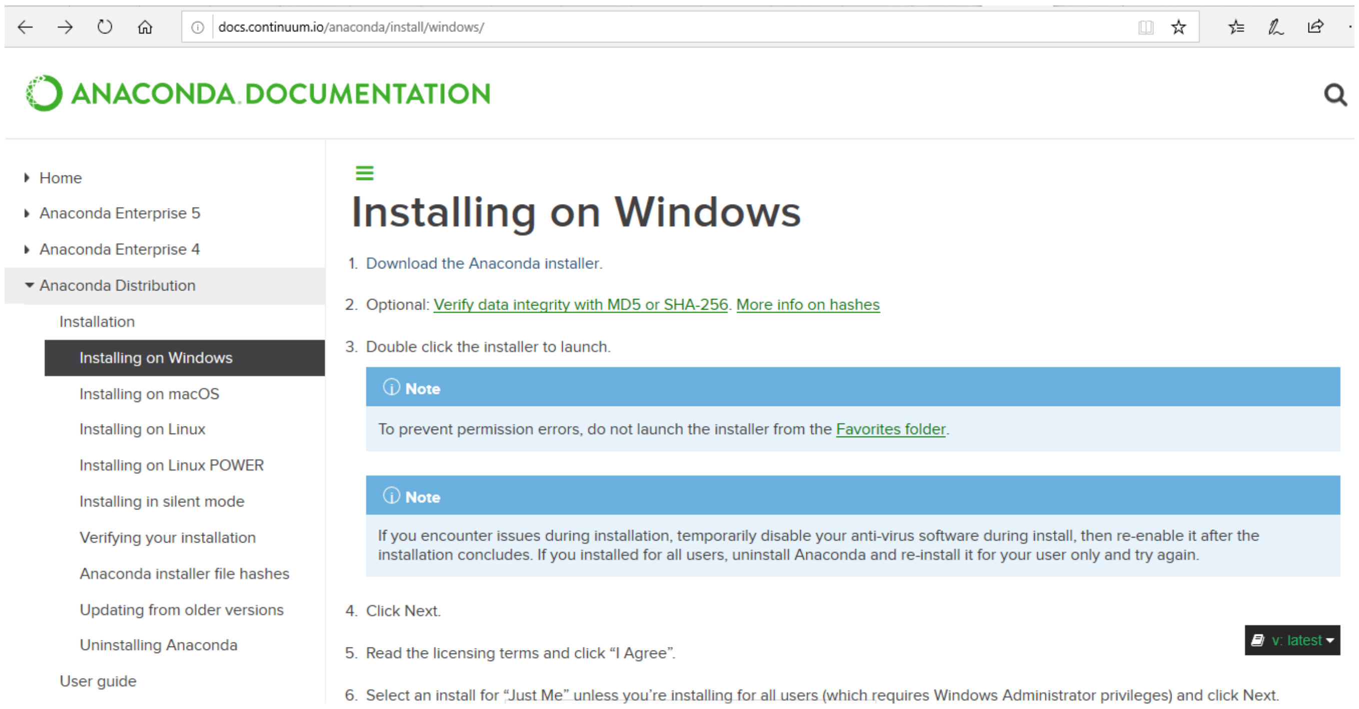
Looking for Python 3.5 or 3.6? See our FAQ.

System requirements

- License: Free use and redistribution under the terms of the End User License Agreement.
- Operating system: Windows 7 or newer, 64-bit macOS 10.10+, or Linux, including Ubuntu, RedHat, CentOS 6+, and others.
- If your operating system is older than what is currently supported, you can find older versions of the Anaconda installers in our archive that might work for you. Check our FAQ for version recommendations.
- System architecture: Windows- 64-bit x86, 32-bit x86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bit Power8/Power9.
- Minimum 5 GB disk space to download and install.

On Windows, macOS, and Linux, it is best to install Anaconda for the local user, which does not require administrator permission most robust type of installation. However, if you need to, you can install Anaconda system wide, which does require administrator permissions.

For examples, you should be able to see similar screen as below. Click on 'Download the Anaconda installer' to save the installer on your computer.



Installing on Windows

1. Download the Anaconda installer.
2. Optional: Verify data integrity with MD5 or SHA-256. More info on hashes
3. Double click the installer to launch.

   Note

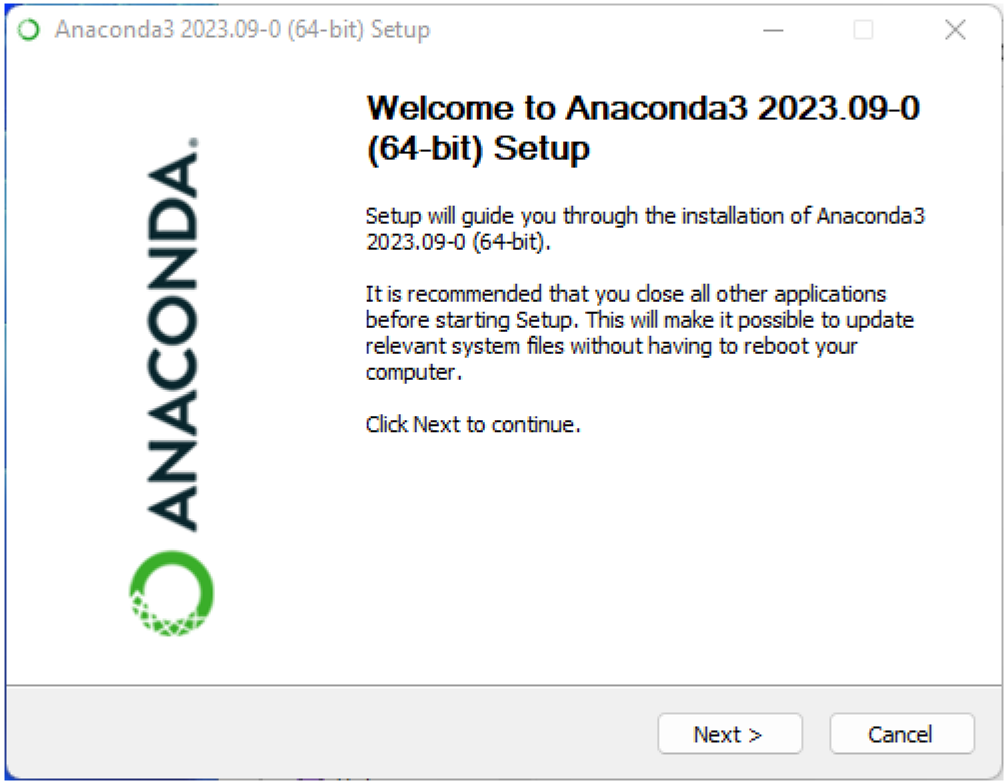
   To prevent permission errors, do not launch the installer from the Favorites folder.

   Note

   If you encounter issues during installation, temporarily disable your anti-virus software during install, then re-enable it after the installation concludes. If you installed for all users, uninstall Anaconda and re-install it for your user only and try again.
4. Click Next.
5. Read the licensing terms and click "I Agree".
6. Select an install for "Just Me" unless you're installing for all users (which requires Windows Administrator privileges) and click Next.

Once downloaded, double click the installer and follow the installation guide to install Anaconda.

You should accept all recommended (or default) settings during the installation.


Anaconda3 2023.09-0 (64-bit) Setup

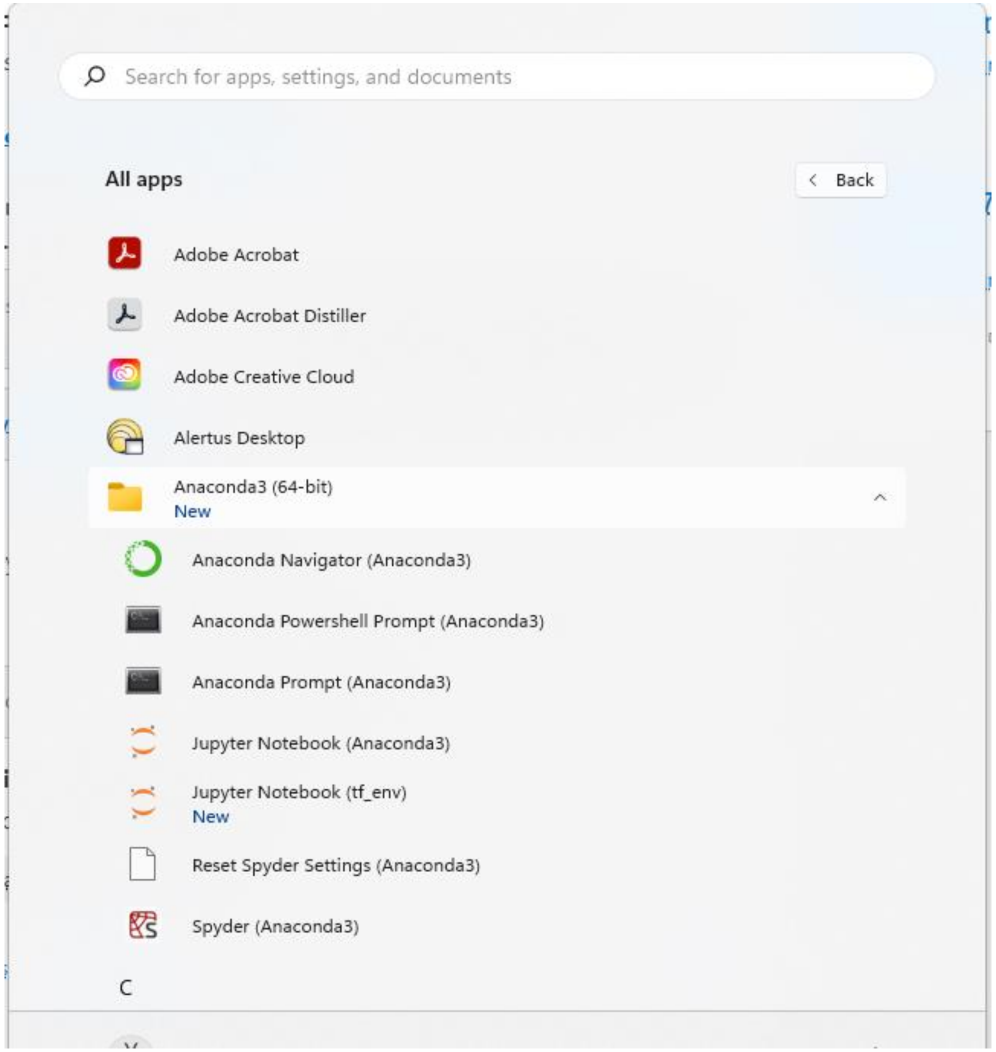
Welcome to Anaconda3 2023.09-0 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 2023.09-0 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

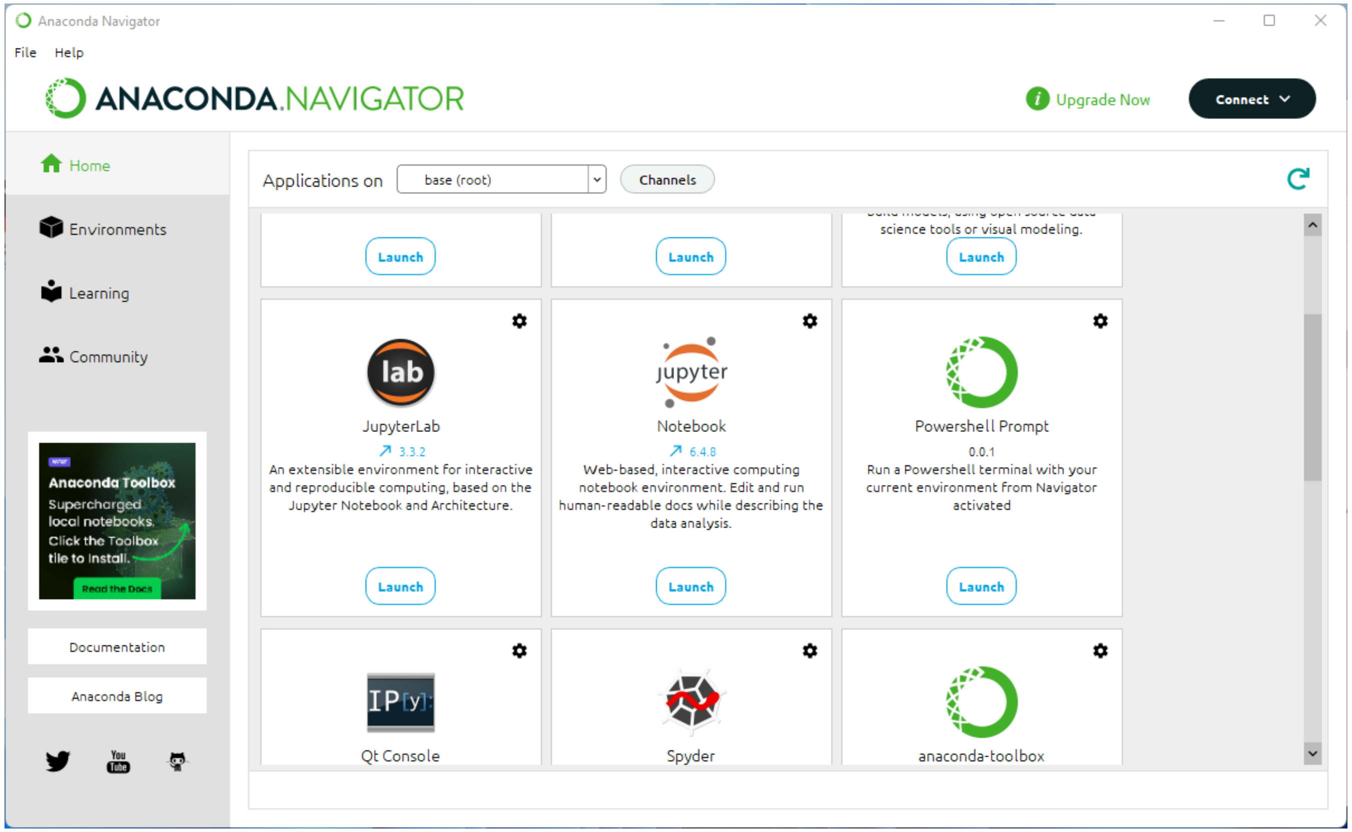
Once the installation is done, you can find on Windows “Anaconda3 (64-bit)” in the list of installed programs.



Adobe Acrobat
Adobe Acrobat Distiller
Adobe Creative Cloud
Alertus Desktop
Anaconda3 (64-bit)
New
Anaconda Navigator (Anaconda3)
Anaconda Powershell Prompt (Anaconda3)
Anaconda Prompt (Anaconda3)
Jupyter Notebook (Anaconda3)
Jupyter Notebook (tf_env)
New
Reset Spyder Settings (Anaconda3)
Spyder (Anaconda3)
C

Clicking on ‘Anaconda Navigator’ should take you to a screen like below.

Clicking ‘Launch’ in Jupyter Notebook.



science tools or visual modeling.

JupyterLab
3.3.2
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.4.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated

Qt Console

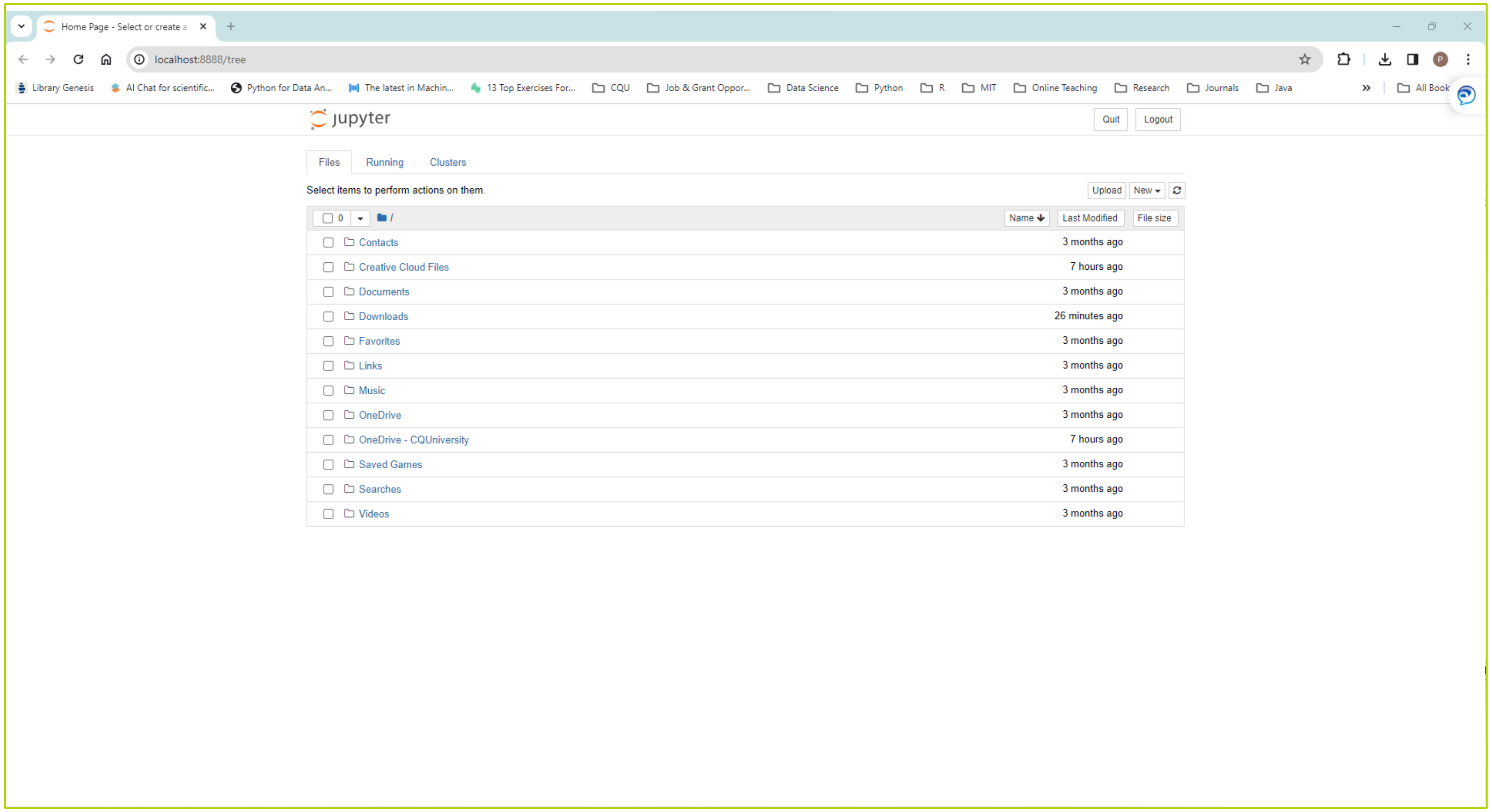
Spyder

anaconda-toolbox

Anaconda Toolbox
Supercharged local notebooks. Click the Toolbox tile to install.
Read the Docs

BE WHAT YOU WANT TO BE
cqu.edu.au

A new tab in your web browser will be created to run the Jupyter Notebook.



Select items to perform actions on them.

Name	Last Modified
Contacts	3 months ago
Creative Cloud Files	7 hours ago
Documents	3 months ago
Downloads	26 minutes ago
Favorites	3 months ago
Links	3 months ago
Music	3 months ago
OneDrive	3 months ago
OneDrive - CQUniversity	7 hours ago
Saved Games	3 months ago
Searches	3 months ago
Videos	3 months ago

Acknowledgment

The next section of slides on Jupyter Notebook and Python Basics/Review is used with courtesy and acknowledgment of:

- Angelica Lo Duca, angelica.loduca@iit.cnr.it
- National Research Council (Cnr), Italy

Data Journalism

Jupyter and Python Basics

Angelica Lo Duca
angelica.loduca@iit.cnr.it

Jupyter Basics

extracted from: https://www.dataquest.io/blog/jupyter-notebook-tutorial/

Jupyter Notebook

Notebook documents (or “notebooks”, all lower case) are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc…).

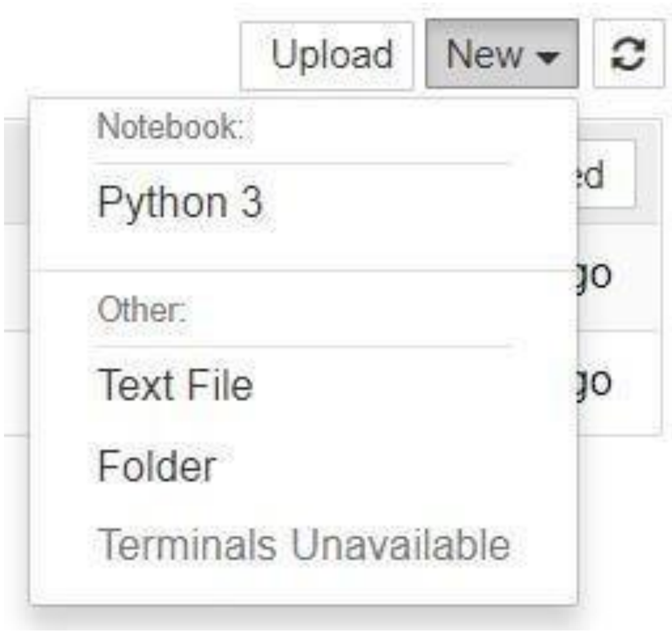
From Jupyter Notebook Beginner Guide

Build the first Notebook

Create the first notebook, click the “New” button in the top-right and select “Python 3”

The notebook opens

Its name is `Untitled.ipynb`



The Notebook Interface

- A **kernel** is a “computational engine” that executes the code contained in a notebook document.
- A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook’s kernel.
    - A code cell contains code to be executed in the kernel. When the code is run, the notebook displays the output below the code cell that generated it.
    - A Markdown cell contains text formatted using Markdown and displays its output in-place when the Markdown cell is run.

To run a cell: Ctrl + Enter / Cmd + Enter, or click Run

Commands

- Toggle between edit and command mode with Esc and Enter, respectively.
- Once in command mode:
  - Scroll up and down your cells with your Up and Down keys.
  - Press A or B to insert a new cell above or below the active cell.
  - M will transform the active cell to a Markdown cell.
  - Y will set the active cell to a code cell.
  - D + D (D twice) will delete the active cell.
  - Z will undo cell deletion.
  - Hold Shift and press Up or Down to select multiple cells at once. With multiple cells selected, Shift + M will merge your selection.

Python Basics

extracted from: https://www.csee.umbc.edu/courses/671/fall09/notes/python1.ppt
and https://www.csee.umbc.edu/courses/671/fall09/notes/python2.ppt

Basic Datatypes

- **Integers (default for numbers)**

```
z = 5 // 2 # Answer 2, integer division
2
```

- **Floats**

```
x = 3.456
```

- **Strings**

```
name = “Angelica”
```

- **Booleans**

```
option = True
option = False
```

Whitespace

- Whitespace is meaningful in Python: especially indentation and placement of newlines
- Use a newline to end a line of code
- Use \ when must go to next line prematurely
- No braces {} to mark blocks of code, use *consistent* indentation instead
    - First line with *less* indentation is outside of the block
    - First line with *more* indentation starts a nested block

Comments

- Start comments with #, rest of line is ignored
- Can include a “documentation string” as the first line of a new function or class you define

```
# this is a comment
```

Assignment

- Basic assignment

```
x = 2
```

- You can assign to multiple names at the same time

```
x, y = 2, 3
```

Accessing Non-Existent Name

Accessing a name before it’s been properly created (by placing it on the left side of an assignment), raises an error

```
y

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    y
NameError: name ‘y' is not defined
y = 3
y
```

Sequence Types

Sequence Types

1. **Tuple: (‘john’, 32, [CMSC])**
   - A simple *immutable* ordered sequence of items
   - Items can be of mixed types, including collection types
2. **Strings: “John Smith”**
   - *Immutable*
   - Conceptually very much like a tuple
4. **List: [1, 2, ‘john’, (‘up’, ‘down’)]**
   - *Mutable* ordered sequence of items of mixed types

Sequence Types 1

- Define tuples using parentheses and commas

```
tu = (23, ‘abc’, 4.56, (2,3), ‘def’)
```

- Define lists are using square brackets and commas

```
li = [“abc”, 34, 4.34, 23]
```

- Define strings using quotes (“, ‘, or “““).

```
st = “Hello World”
st = ‘Hello World’
st = “““This is a multi-line
string that uses triple quotes.”””
```

Sequence Types 2

- Access individual members of a tuple, list, or string using square bracket “array” notation

```
tu = (23, ‘abc’, 4.56, (2,3), ‘def’)
tu[1]     # Second item in the tuple.
    ‘abc’

li = [“abc”, 34, 4.34, 23]
li[1]      # Second item in the list.
    34

st = “Hello World”
st[1]   # Second character in string.
    ‘e’
```

Positive and negative indices

```
t = (23, ‘abc’, 4.56, (2,3), ‘def’)
```

Positive index: count from the left, starting with 0

```
t[1]
    ‘abc’
```

Negative index: count from right, starting with –1

```
t[-3]
    4.56
```

Slicing: return copy of a subset

```
t = (23, ‘abc’, 4.56, (2,3), ‘def’)
```

Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying *before* second.

```
t[1:4]
    (‘abc’, 4.56, (2,3))
```

Negative indices count from end

```
t[1:-1]
    (‘abc’, 4.56, (2,3))
```

Slicing: return copy of a =subset

```
t = (23, 'abc', 4.56, (2,3), 'def')
```

Omit first index to make copy starting from beginning of the container

```
t[:2]
    (23, 'abc')
```

Omit second index to make copy starting at first index and going to end

```
t[2:]
    (4.56, (2,3), 'def')
```

Copying the Whole Sequence

- [:] makes a *copy* of an entire sequence

```
t[:]
        (23, ‘abc’, 4.56, (2,3), ‘def’)
```

The ‘in’ Operator

- Boolean test whether a value is inside a container:

```
t = [1, 2, 4, 5]
3 in t
      False
4 in t
      True
4 not in t
      False
```

- For strings, tests for substrings

```
a = 'abcde'
'c' in a
      True
'cd' in a
      True
'ac' in a
      False
```

The + Operator

The + operator produces a *new* tuple, list, or string whose value is the concatenation of its arguments.

```
(1, 2, 3) + (4, 5, 6)
    (1, 2, 3, 4, 5, 6)

[1, 2, 3] + [4, 5, 6]
    [1, 2, 3, 4, 5, 6]

"Hello" + " " + "World"
 'Hello World'
```

The * Operator

- The * operator produces a *new* tuple, list, or string that "repeats" the original content.

```
(1, 2, 3) * 3
    (1, 2, 3, 1, 2, 3, 1, 2, 3)

[1, 2, 3] * 3
    [1, 2, 3, 1, 2, 3, 1, 2, 3]

“Hello” * 3
    ‘HelloHelloHello’
```

Operations on Lists Only

```
li = [1, 11, 3, 4, 5]

li.append(‘a’)   # Note the method syntax
li
    [1, 11, 3, 4, 5, ‘a’]

li.insert(2, ‘i’)
li
    [1, 11, ‘i’, 3, 4, 5, ‘a’]
```

Operations on Lists Only

Lists have many methods, including index, count, remove, reverse, sort

```
li = [‘a’, ‘b’, ‘c’, ‘b’]
li.index(‘b’)  # index of 1st occurrence
   1
li.count(‘b’)  # number of occurrences
   2
li.remove(‘b’) # remove 1st occurrence
li
  [‘a’, ‘c’, ‘b’]
```

Operations on Lists Only

```
li = [5, 2, 6, 8]

li.reverse()    # reverse the list *in place*
li
  [8, 6, 2, 5]

li.sort()       # sort the list *in place*
li
  [2, 5, 6, 8]
```

Dictionaries

Dictionaries: A Mapping type

- Dictionaries store a *mapping* between a set of keys and a set of values
  - Keys can be any *immutable* type.
  - Values can be any type
  - A single dictionary can store values of different types
- You can define, modify, view, lookup or delete the key-value pairs in the dictionary
- Python’s dictionaries are also known as *hash tables* and *associative arrays*

Creating & accessing dictionaries

```
d = {‘user’:‘bozo’, ‘pswd’:1234}
d[‘user’]
    ‘bozo’
d[‘pswd’]
   123
d[‘bozo’]
Traceback (innermost last):
  File ‘<interactive input>’ line 1, in ?
KeyError: bozo
```

Updating Dictionaries

- Assigning to an existing key replaces its value

```
d = {'user':'bozo', 'pswd':1234}
d['user'] = 'clown'
d
    {'user':'clown', 'pswd':1234}
```

- A new key-value pair can be added to the dictionary

```
d['id'] = 45
d
{'user':'clown', 'id':45, 'pswd':1234}
```

- Dictionaries are unordered
  - New entries can appear anywhere in output

Removing dictionary entries

```
d = {'user':'bozo', 'p':1234, 'i':34}
del d['user']    # Remove one.
d
    {'p':1234, 'i':34}
d.clear()        # Remove all.
d
    {}
a=[1,2]
del a[1]         # del works on lists, too
a
    [1]
```

Useful Accessor Methods

```
d = {‘user’:‘bozo’, ‘p’:1234, ‘i’:34}

d.keys()   # List of keys, VERY useful
    [‘user’, ‘p’, ‘i’]

d.values() # List of values
    [‘bozo’, 1234, 34]

d.items()   # List of item tuples
    [(‘user’,‘bozo’), (‘p’,1234), (‘i’,34)]
```

Functions

Defining Functions

Function definition begins with “def.”

Function name and its arguments.

```
def get_final_answer(filename):
    line1
    line2
    return total_counter
```

Colon.

The indentation matters…
First line with less indentation is considered to be outside of the function definition.

The keyword ‘return’ indicates the value to be sent back to the caller.

Calling a Function

- The syntax for a function call is:

```
def myfun(x, y):
    return x * y

myfun(3, 4)
     12
```

Default Values for Arguments

- You can provide default values for a function’s arguments
- These arguments are optional when the function is called

```
def myfun(b, c=3, d=“hello”):
           return b + c
myfun(5,3,”hello”)
myfun(5,3)
myfun(5)
```

All of the above function calls return 8

Keyword Arguments

- You can call a function with some or all of its arguments out of order as long as you specify their names
- You can also just use keywords for a final subset of the arguments.

```
def myfun(a, b, c):
    return a-b
myfun(2, 1, 43)
  1
myfun(c=43, b=1, a=2)
  1
myfun(2, c=43, b=1)
  1
```

Lambda Notation

- Python uses a lambda notation to create anonymous functions

```
applier(lambda z: z * 4, 7)
     28
```

- Python supports functional programming idioms, including closures and continuations

Control of Flow

if Statements

```
if x == 3:
  print “X equals 3.”
elif x == 2:
  print “X equals 2.”
else:
  print “X equals something else.”
print “This is outside the ‘if’.”
```

Be careful! The keyword *if* is also used in the syntax of filtered *list comprehensions*. Note:

- Use of indentation for blocks
- Colon (*:*) after boolean expression

while Loops

```
x = 3
while x < 5:
        print x, "still in the loop"
        x = x + 1
```

```
        3 still in the loop
        4 still in the loop
```

```
x = 6
while x < 5:
        print x, "still in the loop"
```

break and continue

- You can use the keyword *break* inside a loop to leave the *while* loop entirely.

- You can use the keyword *continue* inside a loop to stop processing the current iteration of the loop and to immediately go on to the next one.

assert

- An *assert* statement will check to make sure that something is true during the course of a program.
  - If the condition is false, the program stops
    - (more accurately: the program throws an exception)

```
assert(number_of_players < 5)
```

For Loops 1

- A for-loop steps through each of the items in a collection type, or any other type of object which is “iterable”

```
for <item> in <collection>:
  <statements>
```

- If <collection> is a list or a tuple, then the loop steps through each element of the sequence
- If <collection> is a string, then the loop steps through each character of the string

```
for someChar in “Hello World”:
   print someChar
```

For Loops 2

```
for <item> in <collection>:
  <statements>
```

- <item> can be more than a single variable name
- When the <collection> elements are themselves sequences, then <item> can match the structure of the elements.
- This multiple assignment can make it easier to access the individual parts of each element

```
for (x,y) in [(a,1),(b,2),(c,3),(d,4)]:
  print x
```

For loops & the range() function

- Since a variable often ranges over some sequence of numbers, the *range()* function returns a list of numbers from 0 up to but not including the number we pass to it.
- range(5) returns [0,1,2,3,4]
- So we could say:

```
for x in range(5):
    print x
```

- (There are more complex forms of *range()* that provide richer functionality…)

For Loops and Dictionaries

```
ages = { "Sam" : 4, "Mary" : 3, "Bill" : 2 }
ages

      {'Bill': 2, 'Mary': 3, 'Sam': 4}
for name in ages.keys():
      print name, ages[name]

      Bill 2
      Mary 3
      Sam 4
```

THANK YOU

TIME FOR DISCUSSION & QUESTIONS