Python Training for RMB

RMB-PYTHON-ONL - Session 01 - Prep and Prereading

by Philip Booysen <philip@piguy.co.za>

Note: You'll write a small test on the first day of training on this content

Read: Sections 1.1 up to 2.3 helps you to start up Jupyter Notebook

Code: Section 2.3 onwards are to be read and coded in Jupyter Notebook

Overview of this Python Notebook

1. How to install, open and run Jupyter Notebook

Prerequisites: Make sure Anaconda was installed on your Windows Desktop

Installation steps to install Anaconda.

In essence, it is necessary to install Anaconda, which installs all the required software, most importantly the included Jupyter Notebook.

For any of the Python Coding courses, you can install the required Anaconda software as follows:

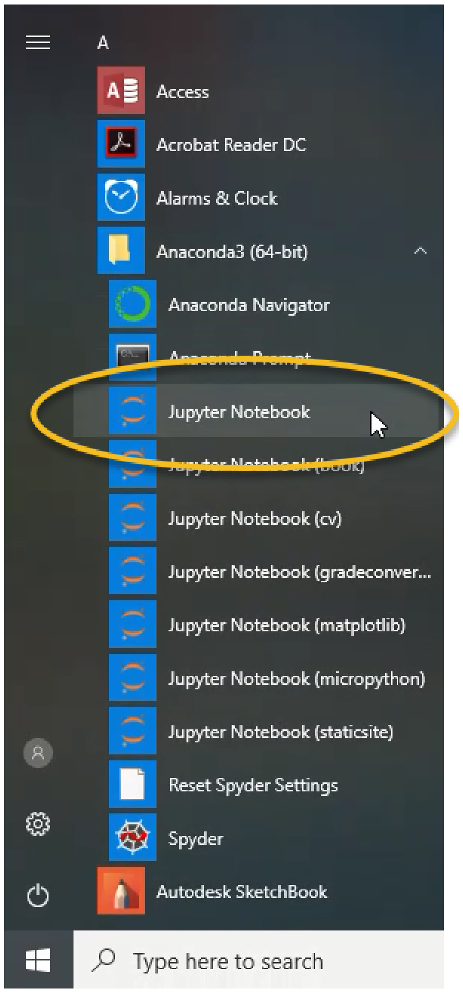
1. Go to link: https://www.anaconda.com/distribution/ (https://www.anaconda.com/distribution/)
2. No need for special admin privileges to install this
3. Click on Individual Edition "Download" icon (https://www.anaconda.com/products/individual#Downloads (https://www.anaconda.com/products/individual#Downloads))
4. Choose: Windows Python 3.8 64-bit Graphical Installer [457MB / requires 30min+ to install]
5. Download & Install by opening up the exe installation file which was downloaded from above link
6. Follow the on screen instructions provided by the installation guide

1.1 Firing up Jupyter Notebook

With the required Anaconda software now installed on your laptop, we can start up Jupyter Notebook.

To start Jupyter Notebook, do the following on your Laptop in Windows 10:

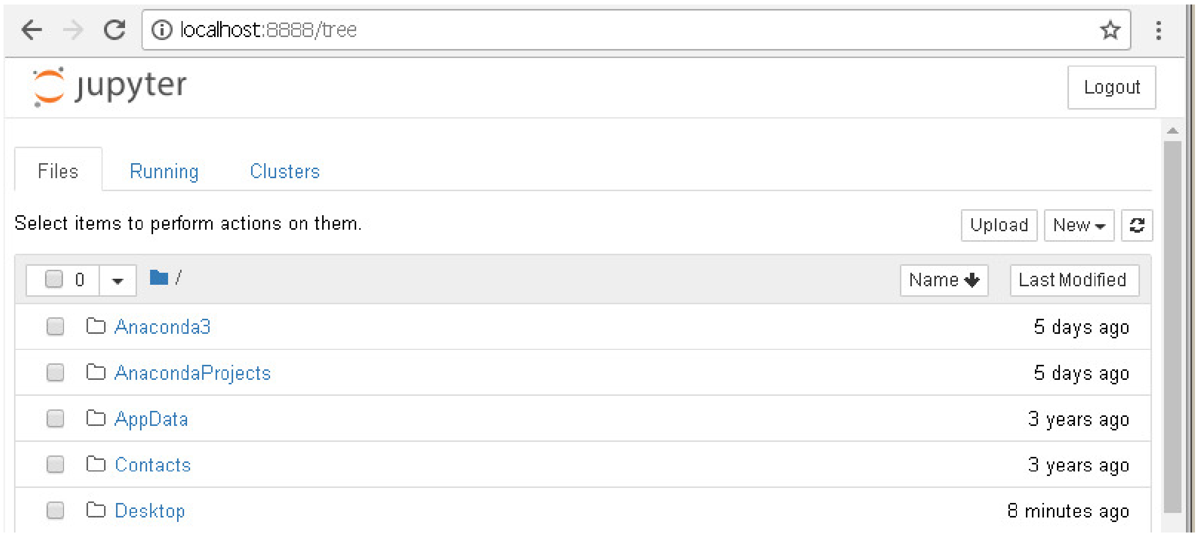
1. Open the Windows start menu and select [Anaconda3(64 bit)] → [Jupyter Notebook]



2. Wait for Jupyter Notebook to start up, a console with the backend will start up

3. Your default browser should open up with a new tab pointing to localhost:8888/tree

Once up and running, **Jupyter Notebook is now running** in your browser:



Select items to perform actions on them.

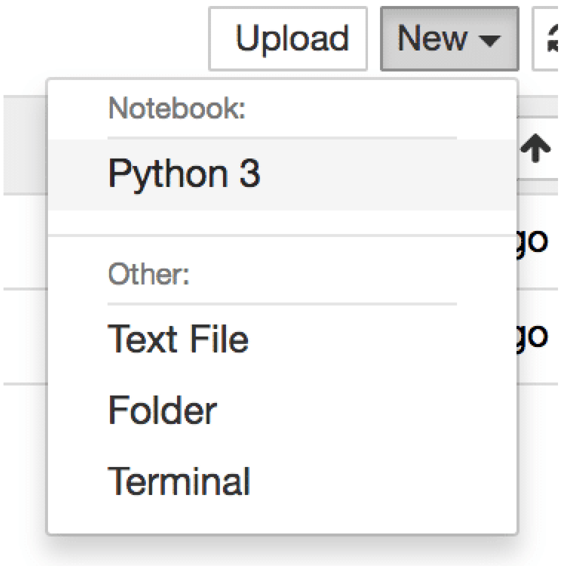
Anaconda3	5 days ago
AnacondaProjects	5 days ago
AppData	3 years ago
Contacts	3 years ago
Desktop	8 minutes ago

2. How to use Jupyter Notebook

2.1 Creating A New Notebook

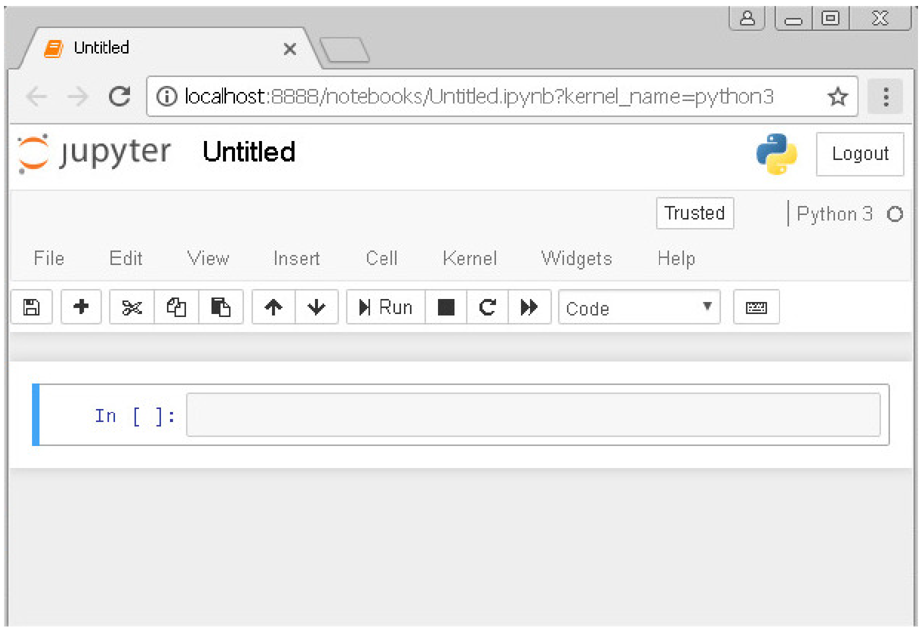
To create a new Notebook:

- Make sure you are in the **"Files"** tab (as per previous section)
- Then use the **"New" dropdown** menu (in top right-hand corner) and you'll see the following options:



Select the "Python 3" option to open a new Jupyter Notebook for Python.

The notebook gets created and you should be able to now see something similar to the following:


Untitled
localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3
jupyter Untitled
Logout
Trusted
Python 3
File Edit View Insert Cell Kernel Widgets Help
Run
Code
In [ ]:

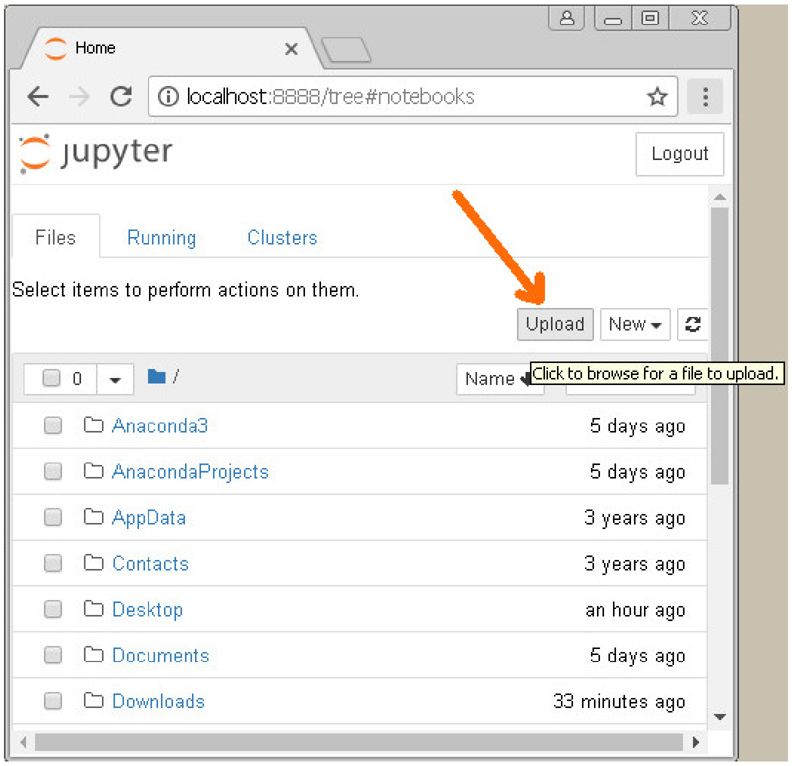
2.2 Uploading an existing Notebook

In the files section, you could browse to a location where there is a downloaded Pyhton Notebook (.ipynb).

You can also **upload** an existing Notebook as follows:

- Make sure you are in the **"Files"** tab
- Then click the **"Upload"** button

- Select the .ipynb file to upload and **click "Open"**:


Home
localhost:8888/tree#notebooks
jupyter
Logout
Files
Running
Clusters
Upload
New
Name
Click to browse for a file to upload.

Select items to perform actions on them.

	Anaconda3	5 days ago
	AnacondaProjects	5 days ago
	AppData	3 years ago
	Contacts	3 years ago
	Desktop	an hour ago
	Documents	5 days ago
	Downloads	33 minutes ago

2.3 Working With This "Pi Guy" Python Training Prep Notebook

Ensure you have **downloaded the RMB-PYTHON-ONL-Prep-For-Day-01.ipynb** file which we emailed as an attachment together with the PDF version of this page and document you are reading now.

Make sure you follow the steps outlined in the previous section on **"Uploading an existing Notebook"** and load the **RMB-PYTHON-ONL-Prep-For-Day-01.ipynb**

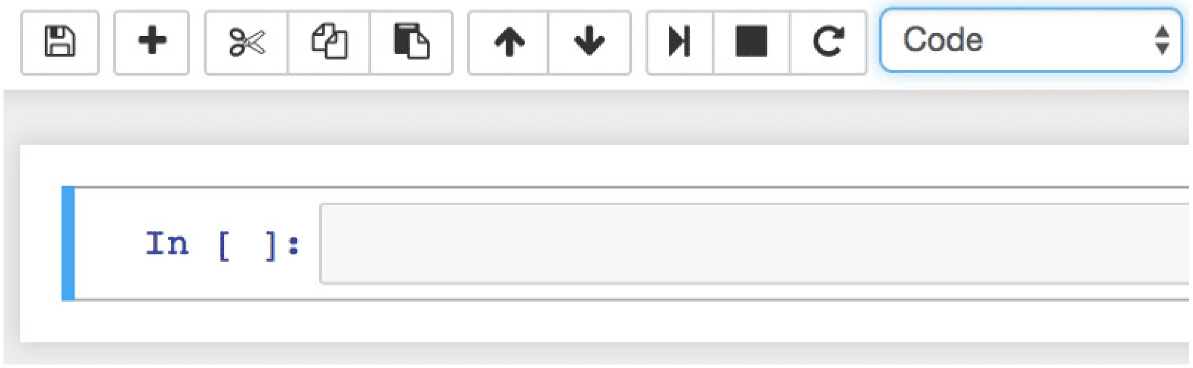
From this point onwards in this Notebook, you can interact live with Python code.

Follow and interact with the rest of this document by using Python Notebook (**not** the PDF).

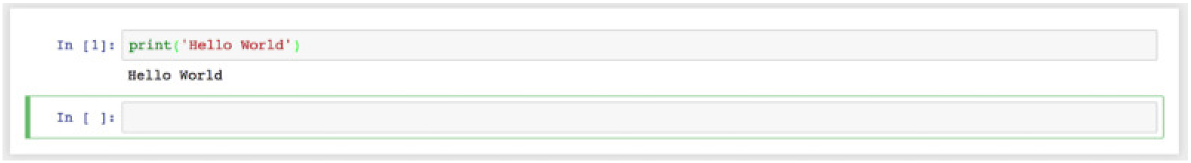
It is imperative to get some practical hands-on experience, and also ensure Jupyter Notebook is running on your laptop, so ensure you carry out the rest of this preparation in Jupyter Notebook.

2.4 Working With The Jupyter Notebook

The notebook itself consists of cells. A first empty cell is already available after having created the new notebook:



This cell is of type “Code” and you can start typing in Python code directly. Executing code in this cell can be done by either clicking on the run cell button or hitting SHIFT + ENTER keys:



```
In [1]: print('Hello World')
        Hello World

In [ ]:
```

The resulting output becomes visible right underneath the cell.

If you would use "ALT + ENTER" it would execute and create a new cell for you to write some more code in.

This should make to output look exactly like the figure above.

Try out yourself here under, click inside the code block, and press ALT + ENTER to execute it and create a new cell to code in:

```
In [6]: print("Hello World")
```

```
Hello World
```

You can also just run a piece of Python code in a block, as mentioned earlier, by hitting SHIFT + ENTER. This will not create a new cell, just execute the code.

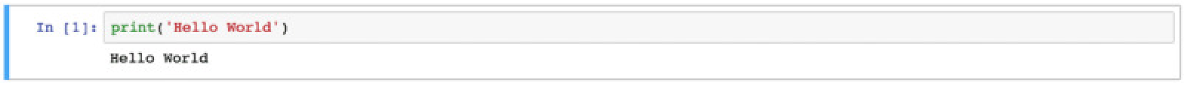
Try and change "Hello World" above to a different string, and then hit SHIFT + ENTER.

2.5 Edit And Command Mode

If a cell is active two modes distinguished:

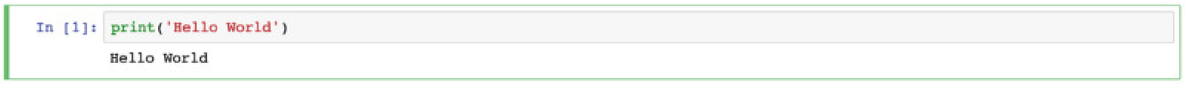
- edit mode
- command mode

If you just click in one cell the cell is opened in **command mode** which is indicated by a **blue border** on the left:



```
In [1]: print('Hello World')
        Hello World
```

The **edit mode** is entered if you click into the code area of that cell. This mode is indicated by a **green border** on the left side of the cell:

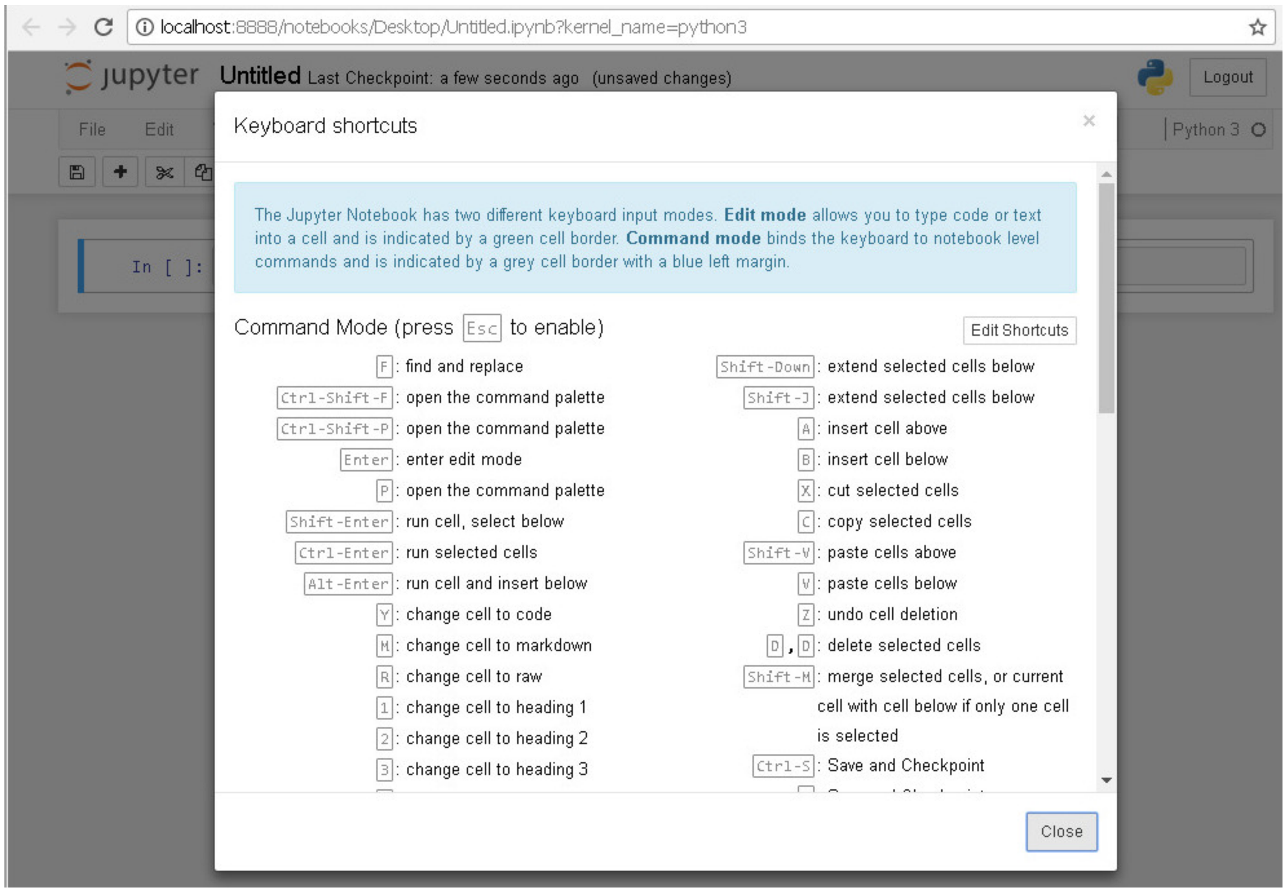


```
In [1]: print('Hello World')
        Hello World
```

If you'd like to leave edit mode and return to command mode again you just need to hit ESC.

2.6 Keyboard Shortcuts

To get an overview of functions which are available in command and in edit mode you can open up the overview of key shortcuts by using menu entry *Help → Keyboard Shortcuts* or by just hitting the key "H" in command mode:


localhost:8888/notebooks/Desktop/Untitled.ipynb?kernel_name=python3
jupyter
Untitled Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File
Edit
Python 3
In [ ]:

Keyboard shortcuts

The Jupyter Notebook has two different keyboard input modes. **Edit mode** allows you to type code or text into a cell and is indicated by a green cell border. **Command mode** binds the keyboard to notebook level commands and is indicated by a grey cell border with a blue left margin.

Command Mode (press Esc to enable)

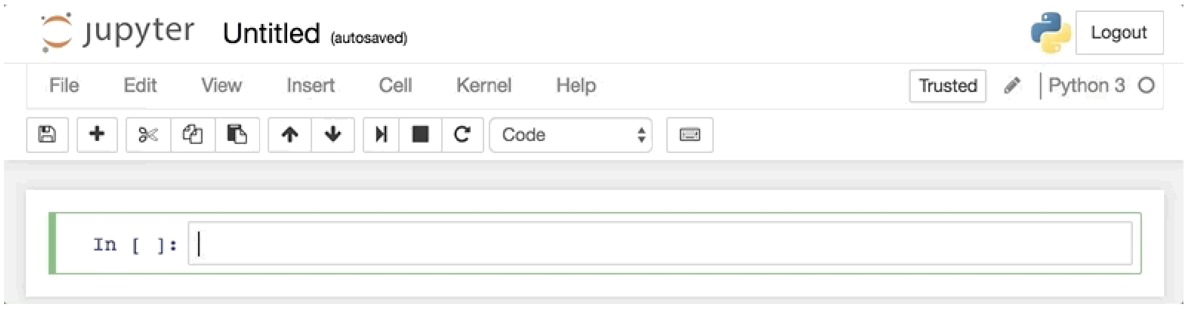
Edit Shortcuts

F	find and replace	Shift-Down	extend selected cells below
Ctrl-Shift-F	open the command palette	Shift-J	extend selected cells below
Ctrl-Shift-P	open the command palette	A	insert cell above
Enter	enter edit mode	B	insert cell below
P	open the command palette	X	cut selected cells
Shift-Enter	run cell, select below	C	copy selected cells
Ctrl-Enter	run selected cells	Shift-V	paste cells above
Alt-Enter	run cell and insert below	V	paste cells below
Y	change cell to code	Z	undo cell deletion
M	change cell to markdown	D, D	delete selected cells
R	change cell to raw	Shift-M	merge selected cells, or current cell with cell below if only one cell is selected
1	change cell to heading 1		
2	change cell to heading 2		
3	change cell to heading 3	Ctrl-S	Save and Checkpoint

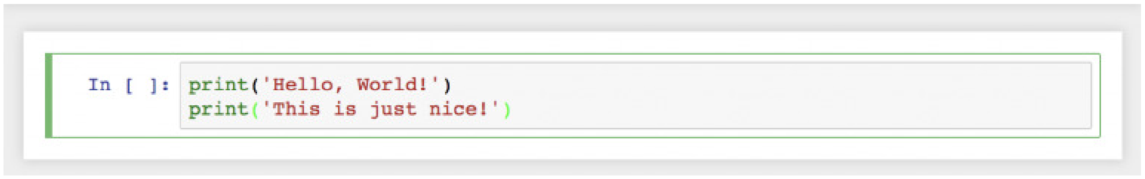
Close

2.7 How to use Jupyter Notebook (Re-cap)

Let's recap how to use Jupyter Notebook

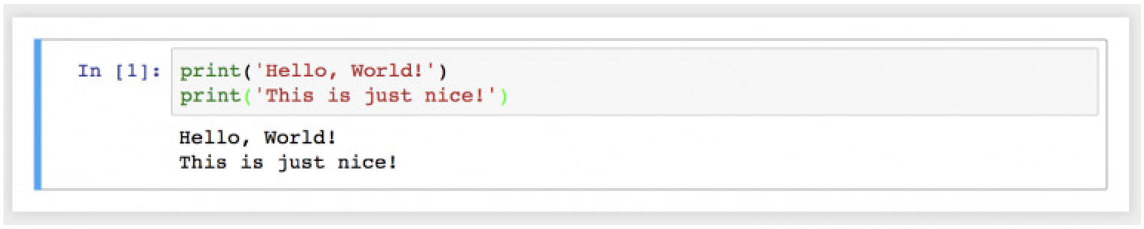


Type your Python command! It can be a multi-line command too – if you hit return/enter, it won't run, it will just start a new line in the same cell!



```
In [ ]: print('Hello, World!')
        print('This is just nice!')
```

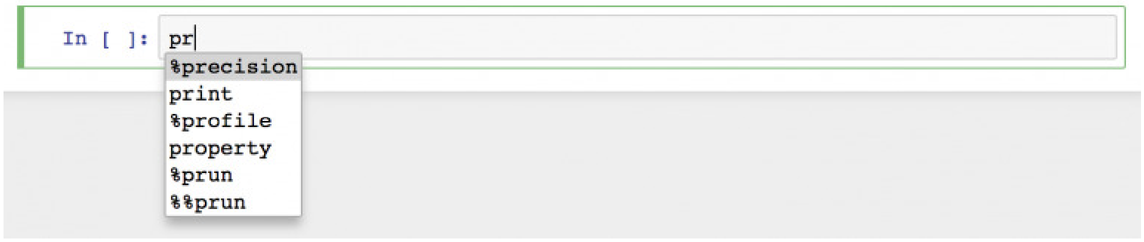
Hit SHIFT + ENTER to run your Python command!



```
In [1]: print('Hello, World!')
        print('This is just nice!')

        Hello, World!
        This is just nice!
```

Start typing and hit TAB! If it's possible, Jupyter will auto-complete your expression (eg. for Python commands or for variables that you have already defined). If there is more than one possibility, you can choose from a drop-down menu.



```
In [ ]: pr
```

3. Python Basics

3.1 Variables

In Python we like to assign values to variables.

Some of the best reasons it because it makes our code better - more flexible, reusable and understandable.

At the same time one of the trickiest things in coding is exactly this "assignment concept".

A name that is used to denote something or a value is called a variable. In python, variables can be declared and values can be assigned to it as follows:

Remember to try and EXECUTE the below code block

- Click anywhere inside the code block
- Change some values if preferred
- To execute and run the code: Press Shift- OR Click the ">| Run" icon in Jupyter Notebook

Maybe change the "Hello" to "Hello World" here under, then EXECUTE to see the change take effect:

```
In [7]: x = 2
        y = 5
        xy = 'Hey'
        print("Hello")
```

```
Hello
```

```
In [8]: print(x+y, xy)
```

```
7 Hey
```

Multiple variables can be assigned with the same value.

```
In [9]: x = y = 1
```

```
In [10]: print (x,y)
```

```
1 1
```

3.2 Operators

3.2.1 Arithmetic Operators

Symbol	Task Performed
+	Addition
-	Subtraction
/	division
%	mod
*	multiplication
//	floor division
**	to the power of

```
In [11]: 1+2
Out[11]: 3
```

```
In [12]: 2-1
Out[12]: 1
```

```
In [13]: 1*2
Out[13]: 2
```

```
In [14]: 1/2
Out[14]: 0.5
```

```
In [15]: 15%10
Out[15]: 5
```

Floor division is nothing but converting the result so obtained to the nearest integer.

```
In [16]: 2.8//2.0
Out[16]: 1.0
```

3.2.2 Relational Operators

Symbol	Task Performed
==	True, if it is equal
!=	True, if not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

```
In [17]: z = 1
```

```
In [18]: z == 1
Out[18]: True
```

```
In [19]: z > 1
Out[19]: False
```

3.3 Print Statement

The **print** statement can be used in the following different ways :

```
- print "Hello World"
- print "Hello", <Variable Containing the String>
- print "Hello" + <Variable Containing the String>
- print "Hello %s" % <variable containing the string>
```

In [20]:
```
print ("Hello World")
```
```
Hello World
```

In Python, single, double and triple quotes are used to denote a string. Most use single quotes when declaring a single character. Double quotes when declaring a line and triple quotes when declaring a paragraph/multiple lines.

In [21]:
```
print ('Hey')
```
```
Hey
```

In [22]:
```
print ("""My name is the Pi Guy

I love Python.""")
```
```
My name is the Pi Guy

I love Python.
```

Strings can be assigned to variable say *string1* and *string2* which can called when using the print statement.

In [23]:
```
string1 = 'World'
print ('Hello', string1)
```
```
Hello World
```

In [24]:
```
string2 = '!'
print ('Hello', string1, string2)
```
```
Hello World !
```

String concatenation is the "addition" of two strings. Observe that while concatenating there will be no space between the strings.

In [25]:
```
print ('Hello' + string1 + string2)
```
```
HelloWorld!
```

%s is used to refer to a variable which contains a string.

In [26]:
```
print ("Hello %s" % string1)
```
```
Hello World
```

Similarly, when using other data types

```
- %s -> string
- %d -> Integer
- %f -> Float
- %o -> Octal
- %x -> Hexadecimal
- %e -> exponential
```

This can be used for conversions inside the print statement itself.

In [27]:
```
print ("Actual Number = %d" %18)
print ("Float of the number = %.5f" %18)
print ("Octal equivalent of the number = %o" %18)
print ("Hexadecimal equivalent of the number = %x" %18)
print ("Exponential equivalent of the number = %e" %18)
```

```
Actual Number = 18
Float of the number = 18.00000
Octal equivalent of the number = 22
Hexadecimal equivalent of the number = 12
Exponential equivalent of the number = 1.800000e+01
```

When referring to multiple variables parenthesis is used.

In [28]:
```
string1 = 'World'
string2 = '!'
print ("Hello %s %s" %(string1,string2))
```

```
Hello World !
```

3.4 Control Flow Statements

3.4.1 If

if some_condition:

```
algorithm
```

In [29]:
```
x = 12
if x > 10:
    print ("Hello")
```

```
Hello
```

3.4.2 If-else

if some_condition:

```
algorithm
```

else:

```
algorithm
```

In [30]:
```
x = 12
if x > 10:
    print ("x was greater than 10")
else:
    print ("x was NOT greater than 10")
```

```
x was greater than 10
```

3.4.3 if-elif-else

if some_condition:

```
algorithm
```

elif some_condition:

```
algorithm
```

else:

```
algorithm
```

```
In [31]: x = 10
         y = 12
         if x > y:
             print ("x>y")
         elif x < y:
             print ("x<y")
         else:
             print ("x=y")
```

```
x<y
```

if statement inside a if statement or if-elif or if-else are called as nested if statements.

```
In [32]: x = 10
         y = 12
         if x > y:
             print ("x>y")
         elif x < y:
             print ("x<y")
             if x==10:
                 print ("x=10")
             else:
                 print ("invalid")
         else:
             print ("x=y")
```

```
x<y
x=10
```

3.4.4 Loop

3.4.4.1 For

for variable in something:

```
    algorithm
```

```
In [33]: for item in [0,1,2,3,4,"Pi Guy","Hello","Python"]:
             print (item)
```

```
0
1
2
3
4
Pi Guy
Hello
Python
```

In the above example, i iterates over the 0,1,2,3,4. Every time it takes each value and executes the algorithm inside the loop. It is also possible to iterate over a nested list illustrated below.

```
In [34]: list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
         for list1 in list_of_lists:
                 print (list1)
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

A use case of a nested for loop in this case would be,

```
In [35]: list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
         for list1 in list_of_lists: # For each value in the list_of_lists saved into list1 for each iteration
             for x in list1:         # For each value in the list1 saved into x for each iteration
                 print (x)
```

```
1
2
3
4
5
6
7
8
9
```

3.4.4.2 While

while some_condition:

```
algorithm
```

```
In [36]: i = 1
         while i < 3:
             print(i ** 2) # i to the power of 2
             i = i+1
         print('Bye')
```

```
1
4
Bye
```

```
- END -
```