Short introduction to Python for Sciences

Tiny tutorial for jupyter notebook, numpy and matplotlib use

I. Gerber 25/01/24

Python

— Python is a high-level, dynamically typed multiparadigm programming language.

— Python code is often said to be almost like pseudocode, since it allows you to express very powerful ideas in very few lines of code while being very readable.

Jupyter Notebook

— A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media.

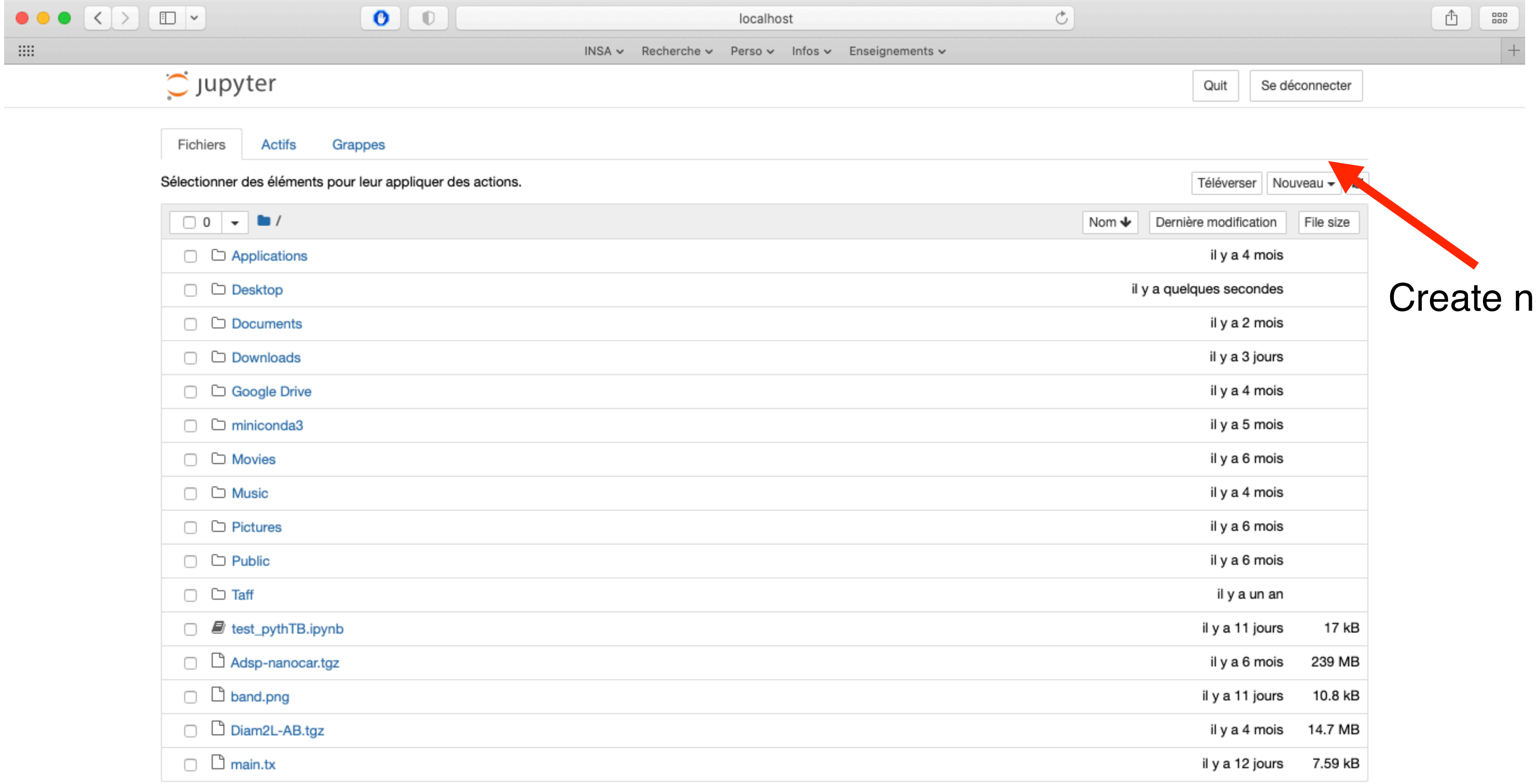
In other words: it’s a single document where you can run code, display the output, and also add explanations, formulas, charts, and make your work more transparent, understandable, repeatable, and shareable.

— Using Notebooks is now a major part of the data science workflow at companies across the globe. If your goal is to work with data, using a Notebook will speed up your workflow and make it easier to communicate and share your results.

Let’s start:

Just type « jupyter notebook » command in a terminal

Jupyter Notebook

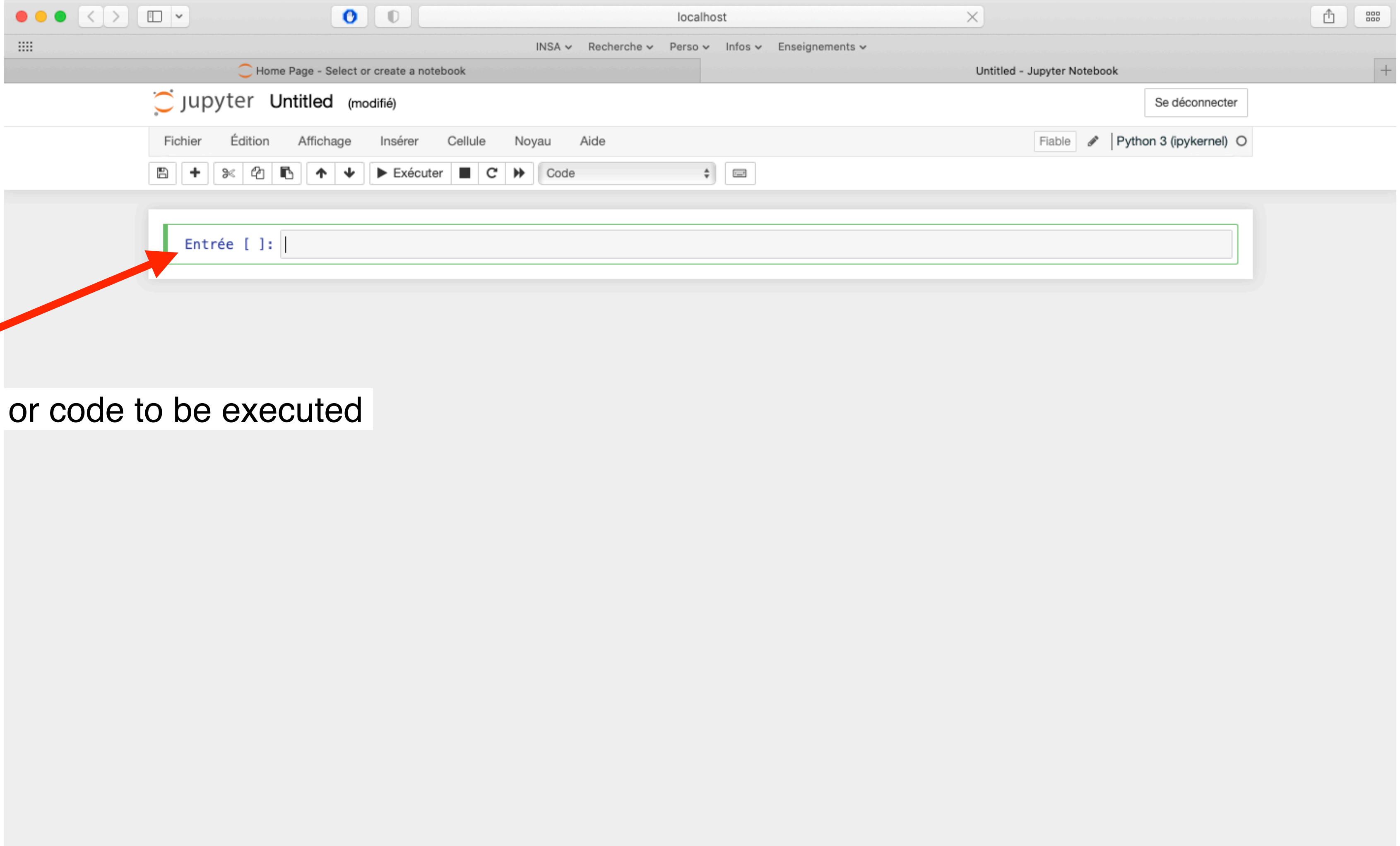


Sélectionner des éléments pour leur appliquer des actions.

Nom	Dernière modification	File size
□ Applications	il y a 4 mois	
□ Desktop	il y a quelques secondes	
□ Documents	il y a 2 mois	
□ Downloads	il y a 3 jours	
□ Google Drive	il y a 4 mois	
□ miniconda3	il y a 5 mois	
□ Movies	il y a 6 mois	
□ Music	il y a 4 mois	
□ Pictures	il y a 6 mois	
□ Public	il y a 6 mois	
□ Taff	il y a un an	
□ test_pythTB.ipynb	il y a 11 jours	17 kB
□ Adsp-nanocar.tgz	il y a 6 mois	239 MB
□ band.png	il y a 11 jours	10.8 kB
□ Diam2L-AB.tgz	il y a 4 mois	14.7 MB
□ main.tx	il y a 12 jours	7.59 kB

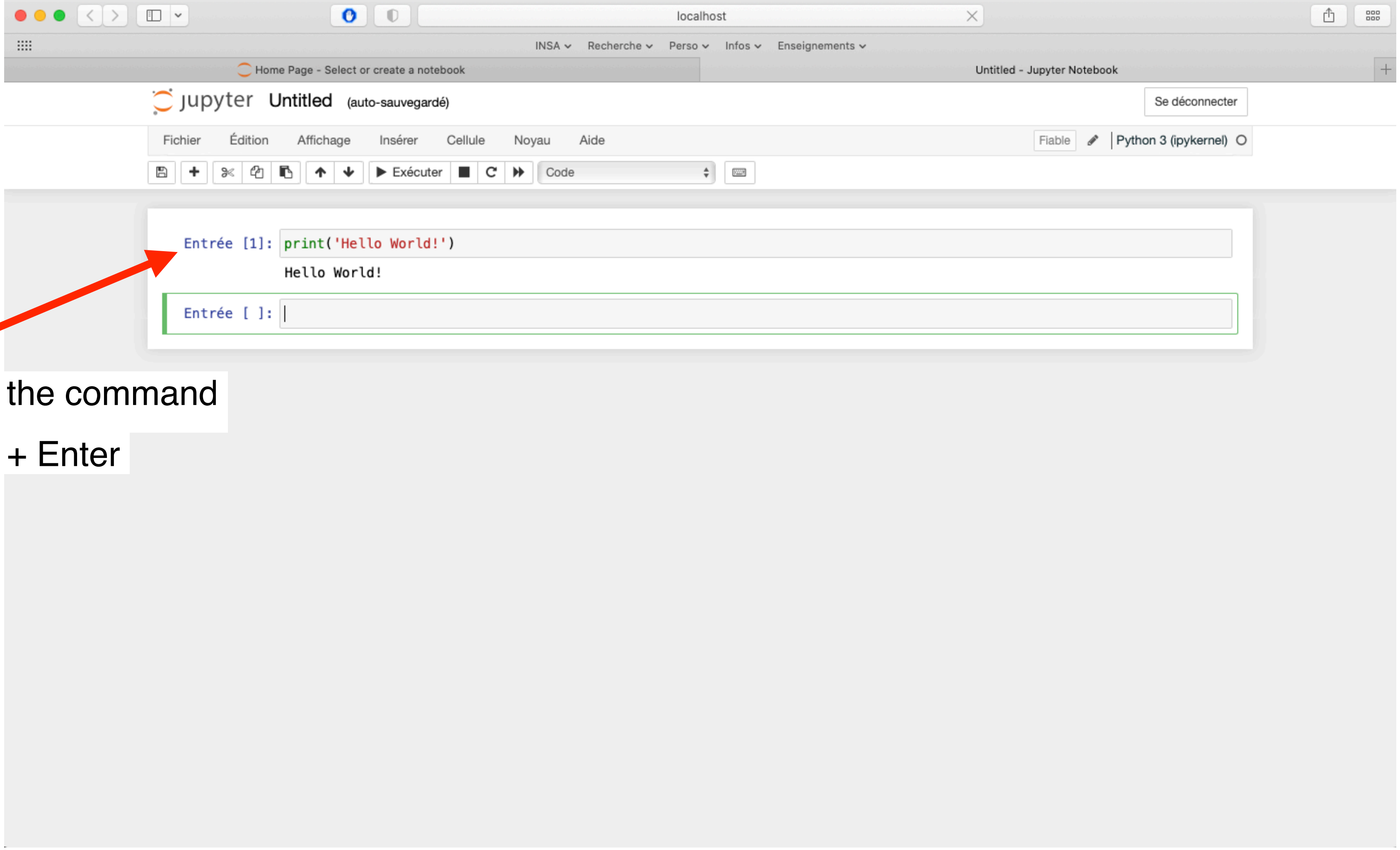
Create new notebook

A cell



A cell : text or code to be executed

A cell

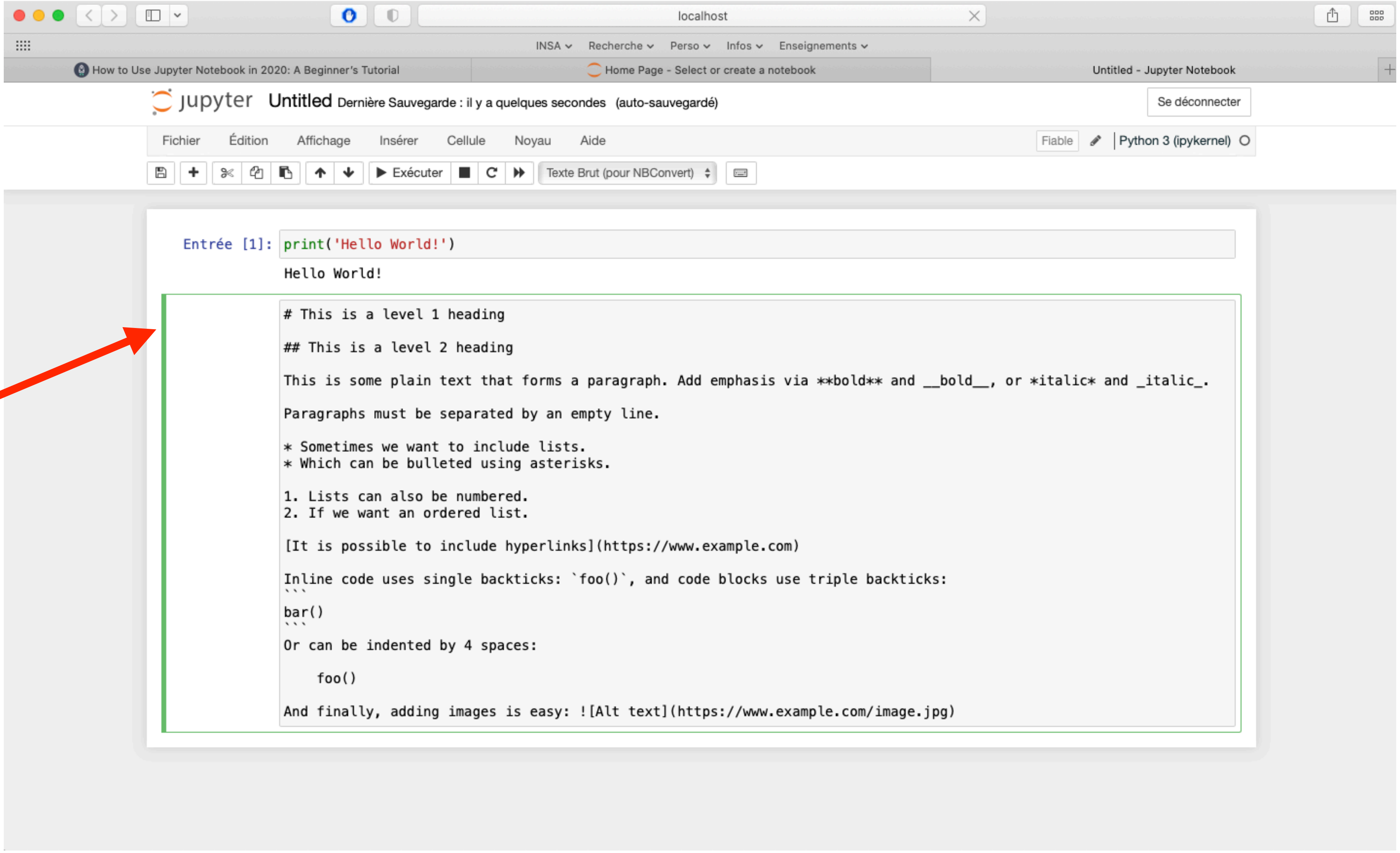


```
Entrée [1]: print('Hello World!')
            Hello World!
Entrée [ ]:
```

To execute the command

Press Shift + Enter

A cell



localhost
INSA
Recherche
Perso
Infos
Enseignements
How to Use Jupyter Notebook in 2020: A Beginner's Tutorial
Home Page - Select or create a notebook
Untitled - Jupyter Notebook
jupyter
Untitled Dernière Sauvegarde : il y a quelques secondes (auto-sauvegardé)
Se déconnecter
Fichier
Édition
Affichage
Insérer
Cellule
Noyau
Aide
Fiable
Python 3 (ipykernel)
Exécuter
Texte Brut (pour NBConvert)

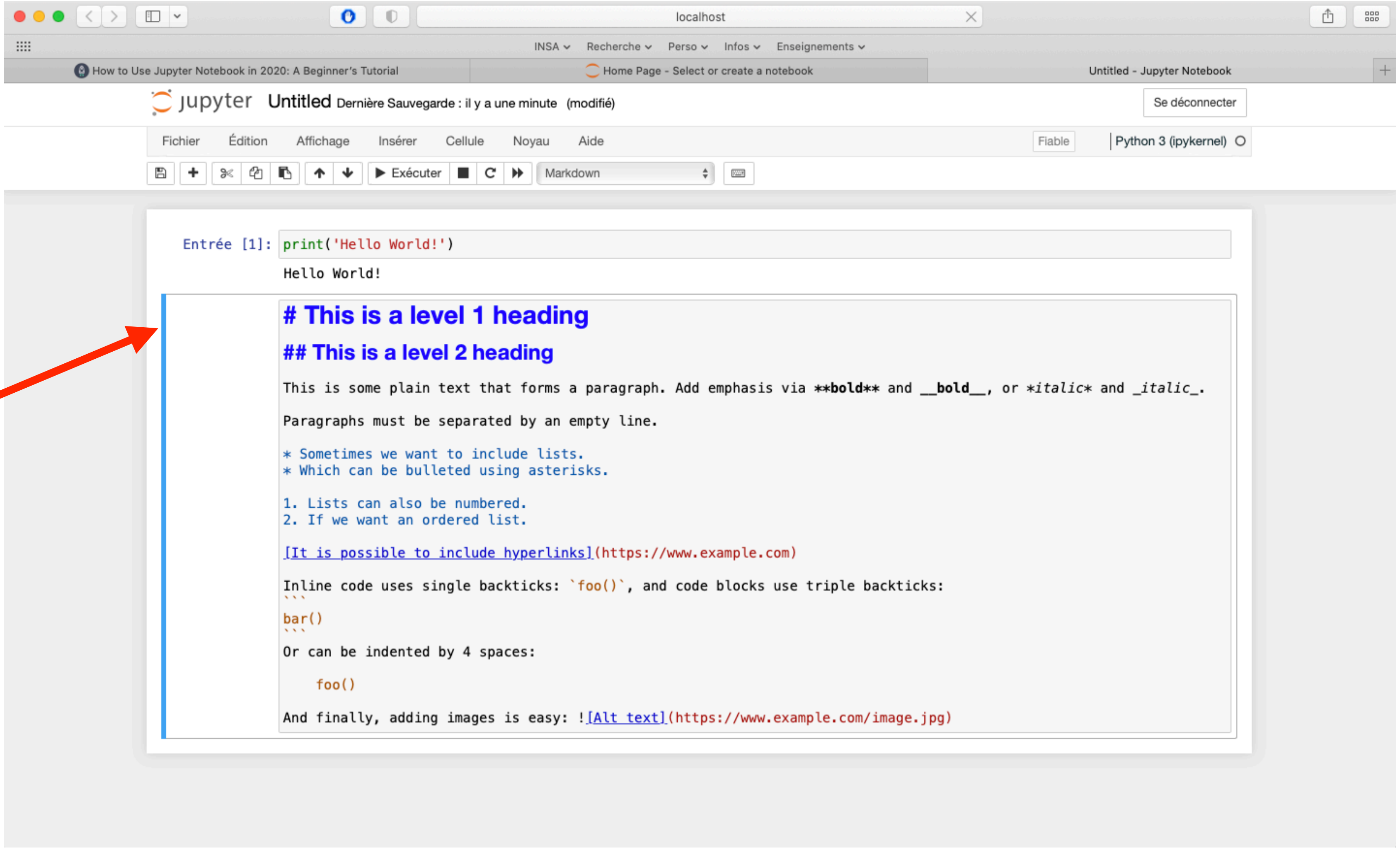
```
Entrée [1]: print('Hello World!')

            Hello World!
```

```
# This is a level 1 heading

## This is a level 2 heading

This is some plain text that forms a paragraph. Add emphasis via **bold** and __bold__, or *italic* and _italic_.

Paragraphs must be separated by an empty line.

* Sometimes we want to include lists.
* Which can be bulleted using asterisks.

1. Lists can also be numbered.
2. If we want an ordered list.

[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: `foo()`, and code blocks use triple backticks:
```
bar()
```
Or can be indented by 4 spaces:

    foo()

And finally, adding images is easy: ![Alt text](https://www.example.com/image.jpg)
```

Brut text

A cell

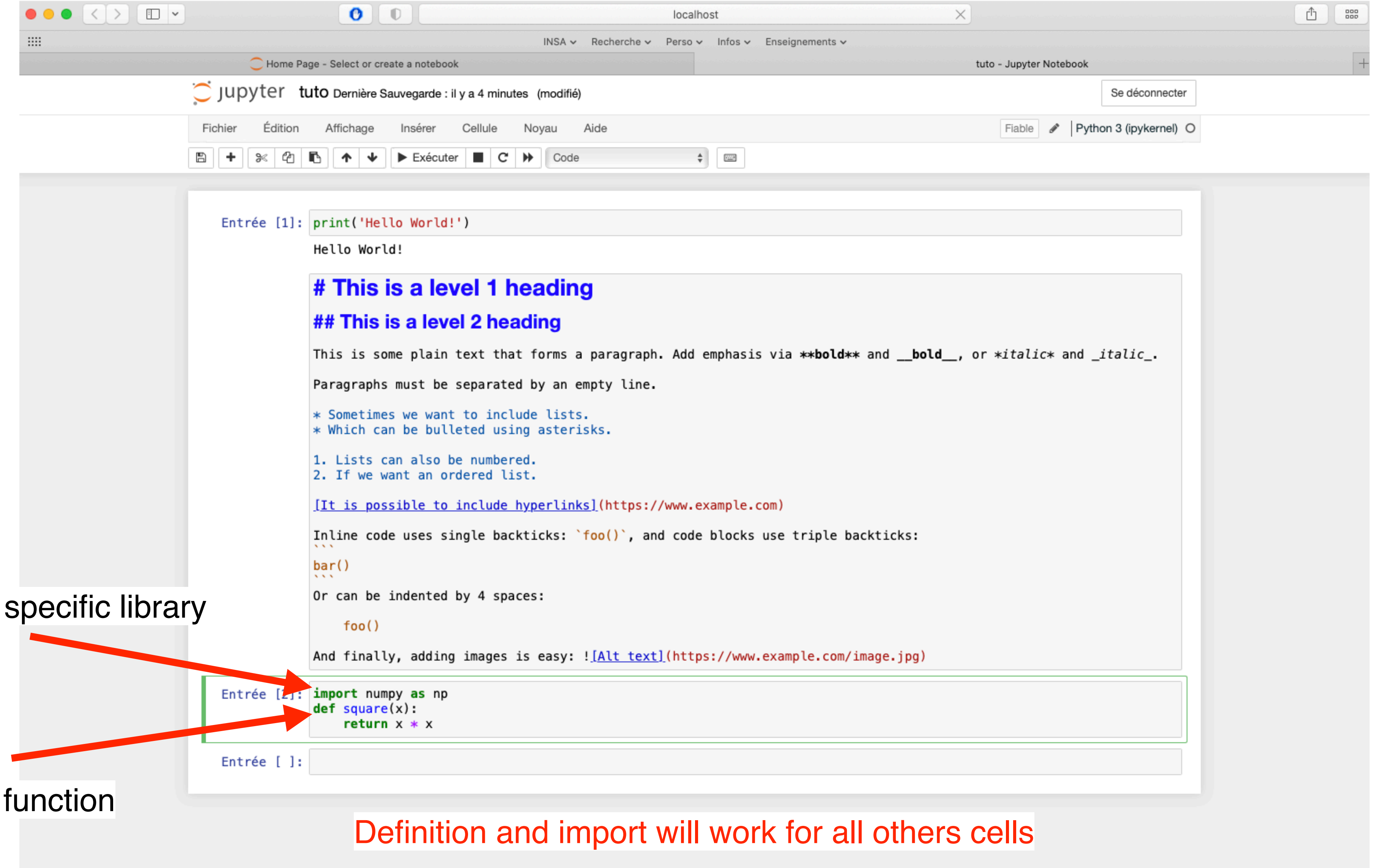
Markdown


localhost
INSA
Recherche
Perso
Infos
Enseignements
How to Use Jupyter Notebook in 2020: A Beginner's Tutorial
Home Page - Select or create a notebook
Untitled - Jupyter Notebook
jupyter
Untitled Dernière Sauvegarde : il y a une minute (modifié)
Se déconnecter
Fichier
Édition
Affichage
Insérer
Cellule
Noyau
Aide
Fiable
Python 3 (ipykernel) ○
Exécuter
Markdown

```
Entrée [1]: print('Hello World!')
            Hello World!
```

```
# This is a level 1 heading
## This is a level 2 heading

This is some plain text that forms a paragraph. Add emphasis via **bold** and __bold__, or *italic* and _italic_.

Paragraphs must be separated by an empty line.

* Sometimes we want to include lists.
* Which can be bulleted using asterisks.

1. Lists can also be numbered.
2. If we want an ordered list.

[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: `foo()`, and code blocks use triple backticks:
```
bar()
```
Or can be indented by 4 spaces:

    foo()

And finally, adding images is easy: ![Alt text](https://www.example.com/image.jpg)
```

Kernel



```
Entrée [1]: print('Hello World!')
```

```
Hello World!
```

```
# This is a level 1 heading
## This is a level 2 heading
This is some plain text that forms a paragraph. Add emphasis via **bold** and __bold__, or *italic* and _italic_.

Paragraphs must be separated by an empty line.

* Sometimes we want to include lists.
* Which can be bulleted using asterisks.

1. Lists can also be numbered.
2. If we want an ordered list.

[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: `foo()`, and code blocks use triple backticks:
```
bar()
```
Or can be indented by 4 spaces:

    foo()

And finally, adding images is easy: ![Alt text](https://www.example.com/image.jpg)
```

```
Entrée [2]: import numpy as np
            def square(x):
                return x * x
```

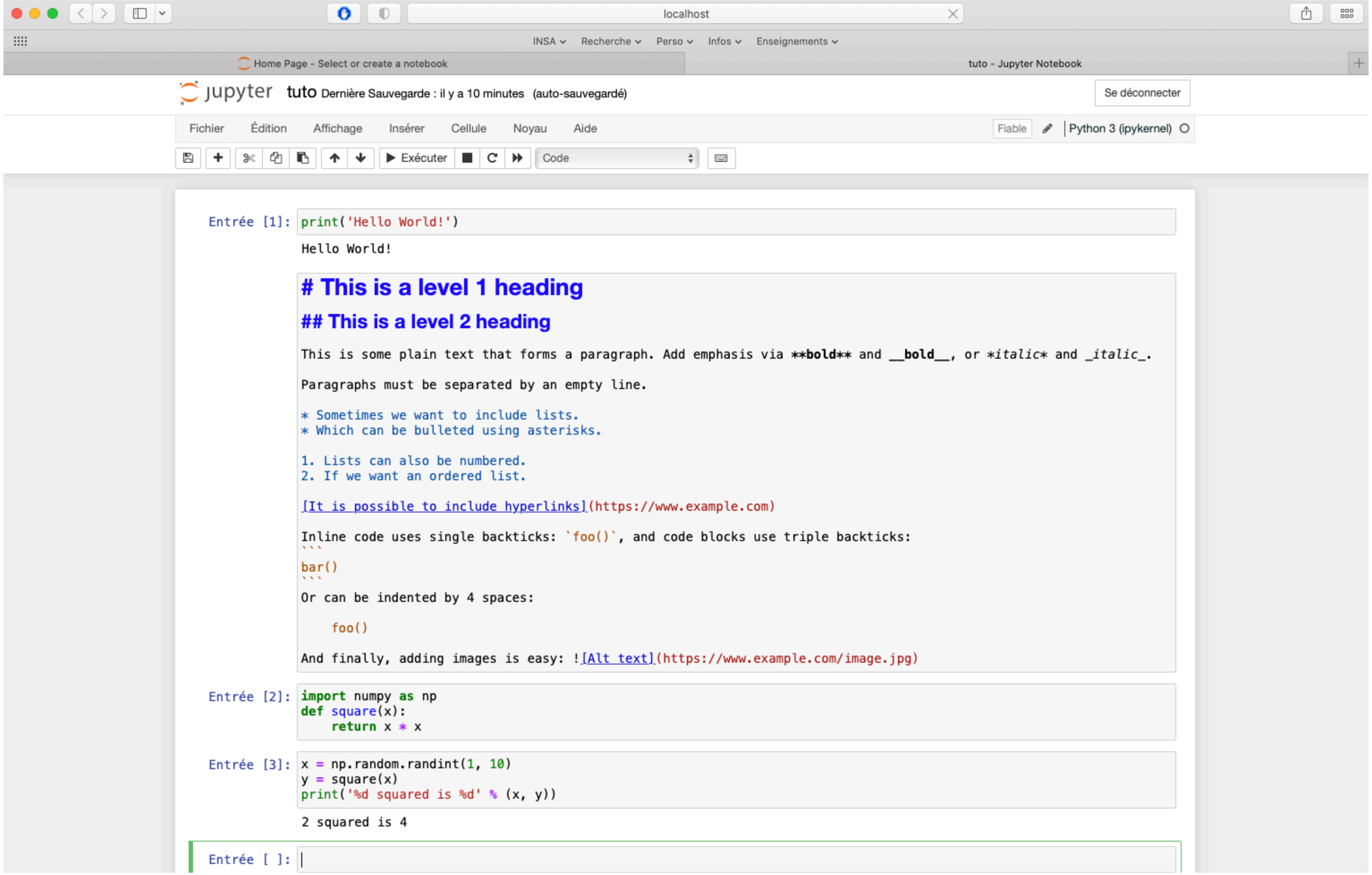
```
Entrée [ ]:
```

Import specific library

Define function

Definition and import will work for all others cells

Kernel



```
Entrée [1]: print('Hello World!')
            Hello World!
```

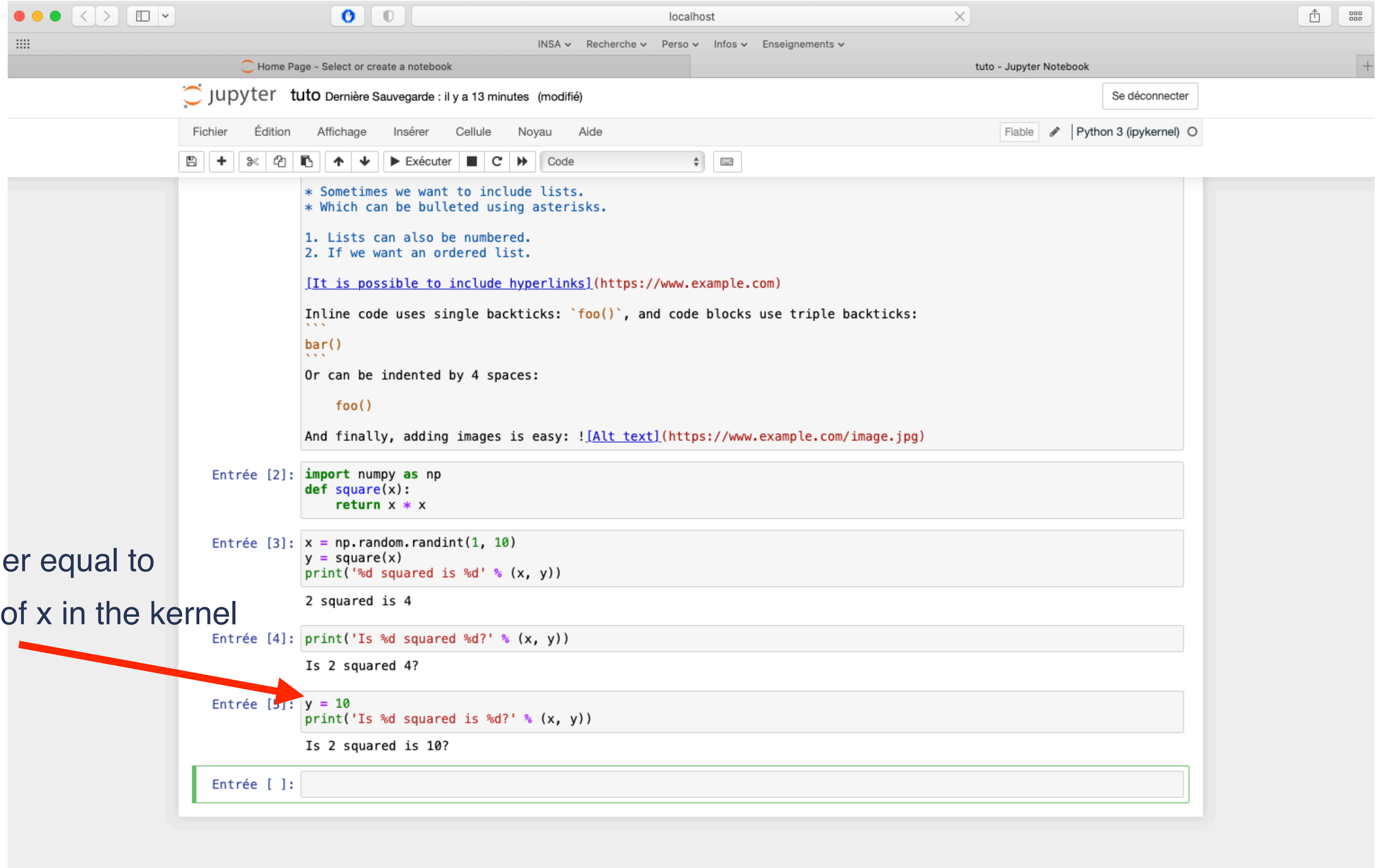
```
# This is a level 1 heading
## This is a level 2 heading
This is some plain text that forms a paragraph. Add emphasis via **bold** and __bold__, or *italic* and _italic_.

Paragraphs must be separated by an empty line.

* Sometimes we want to include lists.
* Which can be bulleted using asterisks.

1. Lists can also be numbered.
2. If we want an ordered list.

[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: `foo()`, and code blocks use triple backticks:
```
bar()
```
Or can be indented by 4 spaces:

    foo()

And finally, adding images is easy: ![Alt text](https://www.example.com/image.jpg)
```

```
Entrée [2]: import numpy as np
            def square(x):
                return x * x
```

```
Entrée [3]: x = np.random.randint(1, 10)
            y = square(x)
            print('%d squared is %d' % (x, y))
            2 squared is 4
```

```
Entrée [ ]:
```

Kernel



```
* Sometimes we want to include lists.
* Which can be bulleted using asterisks.

1. Lists can also be numbered.
2. If we want an ordered list.

[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: `foo()`, and code blocks use triple backticks:
```
bar()
```
Or can be indented by 4 spaces:

    foo()

And finally, adding images is easy: ![Alt text](https://www.example.com/image.jpg)
```

Entrée [2]:

```
import numpy as np
def square(x):
    return x * x
```

Entrée [3]:

```
x = np.random.randint(1, 10)
y = square(x)
print('%d squared is %d' % (x, y))
```

```
2 squared is 4
```

Entrée [4]:

```
print('Is %d squared %d?' % (x, y))
```

```
Is 2 squared 4?
```

Entrée [5]:

```
y = 10
print('Is %d squared is %d?' % (x, y))
```

```
Is 2 squared is 10?
```

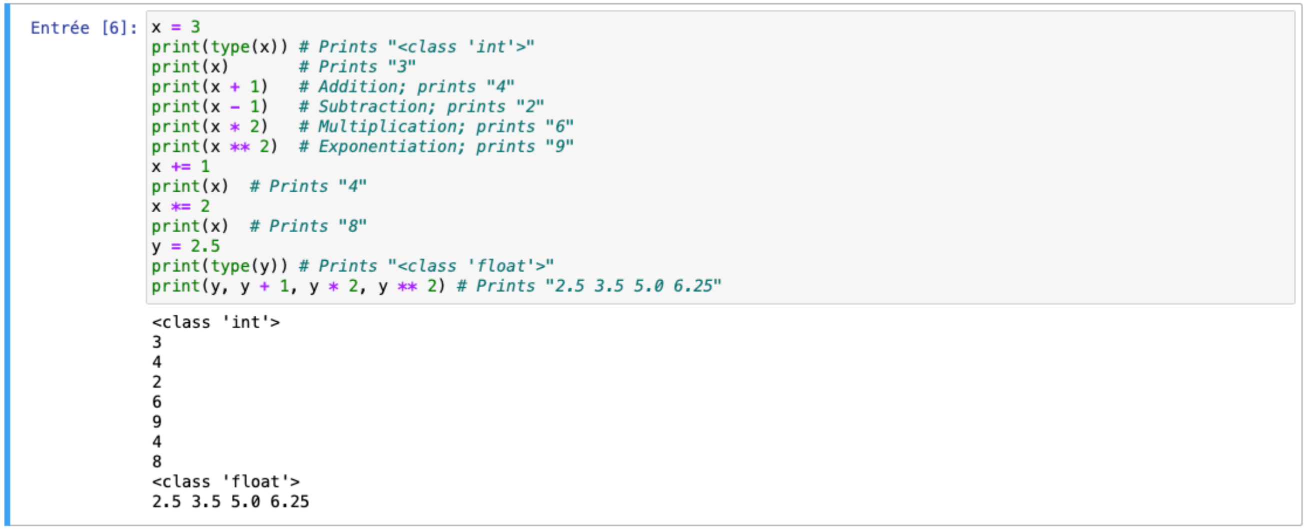
Entrée [ ]:

y is no longer equal to

the square of x in the kernel

Basic data types

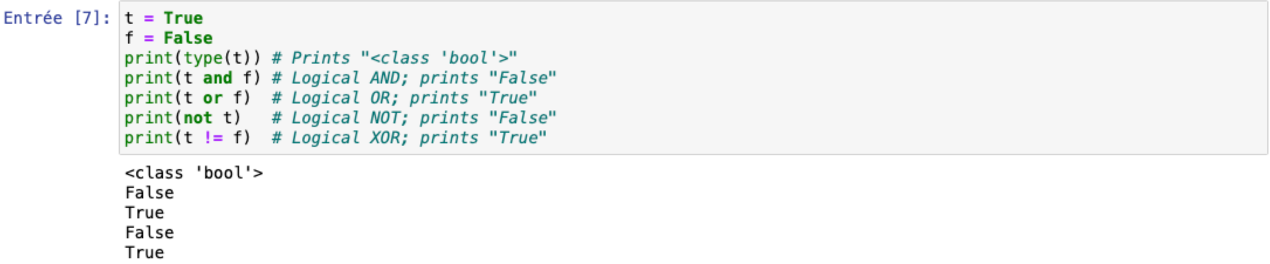
Numbers



```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

```
<class 'int'>
3
4
2
6
9
4
8
<class 'float'>
2.5 3.5 5.0 6.25
```

Boolean

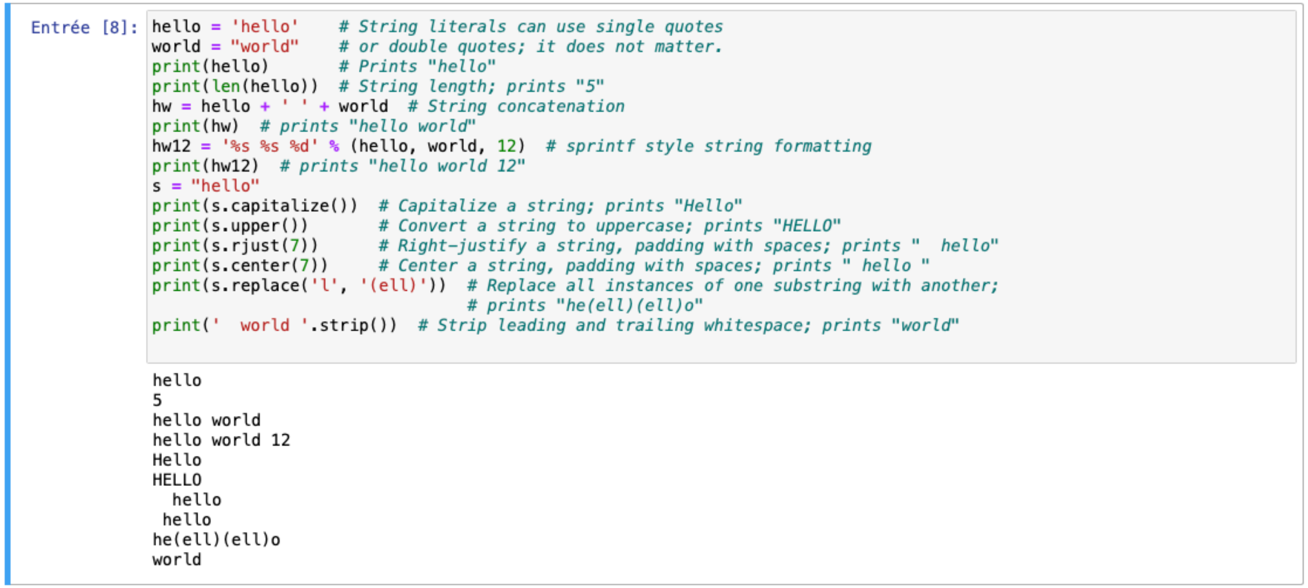


```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

```
<class 'bool'>
False
True
False
True
```

Basic data types

Strings

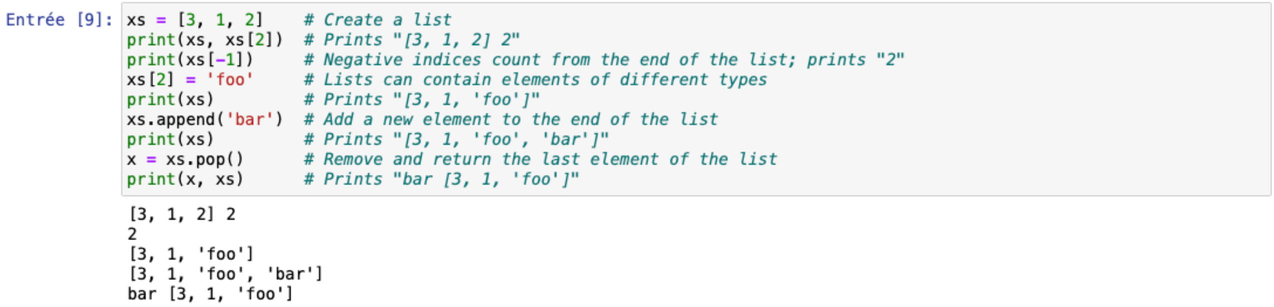


```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
s = "hello"
print(s.capitalize())  # Capitalize a string; prints "Hello"
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces; prints "  hello"
print(s.center(7))     # Center a string, padding with spaces; prints " hello "
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with another;
                                # prints "he(ell)(ell)o"
print('  world '.strip())  # Strip leading and trailing whitespace; prints "world"
```

```
hello
5
hello world
hello world 12
Hello
HELLO
  hello
 hello
he(ell)(ell)o
world
```

Containers

Lists

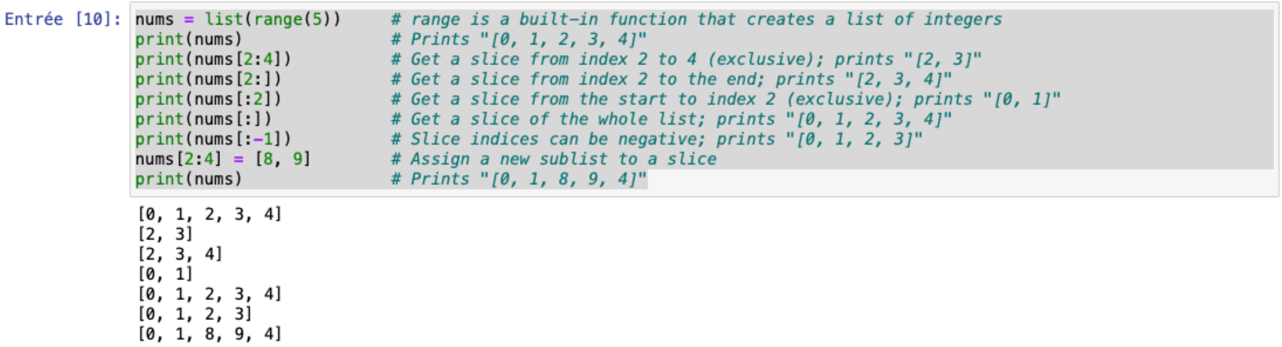


Entrée [9]:

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)         # Prints "[3, 1, 'foo']"
xs.append('bar')  # Add a new element to the end of the list
print(xs)         # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

```
[3, 1, 2] 2
2
[3, 1, 'foo']
[3, 1, 'foo', 'bar']
bar [3, 1, 'foo']
```

Slicing

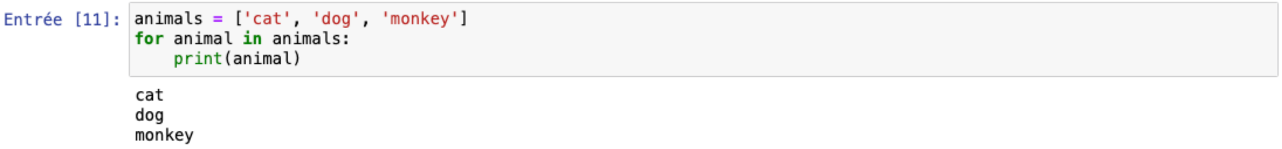


Entrée [10]:

```
nums = list(range(5))     # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]"
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

Loops

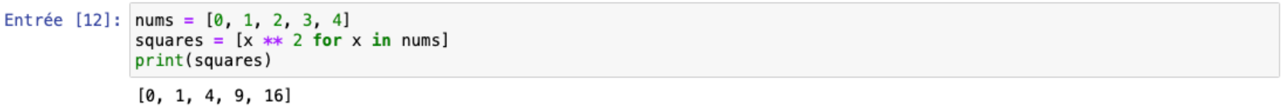


Entrée [11]:

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

List comprehension



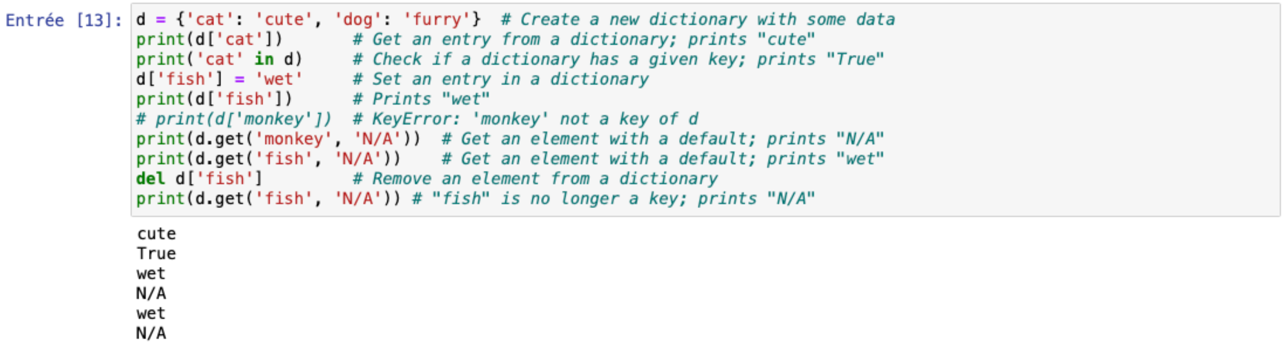
Entrée [12]:

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)
```

```
[0, 1, 4, 9, 16]
```

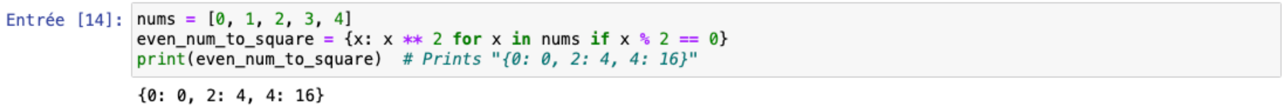
Containers

Dictionary



```
Entrée [13]: d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
             print(d['cat'])       # Get an entry from a dictionary; prints "cute"
             print('cat' in d)     # Check if a dictionary has a given key; prints "True"
             d['fish'] = 'wet'     # Set an entry in a dictionary
             print(d['fish'])      # Prints "wet"
             # print(d['monkey'])  # KeyError: 'monkey' not a key of d
             print(d.get('monkey', 'N/A'))  # Get an element with a default; prints "N/A"
             print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
             del d['fish']         # Remove an element from a dictionary
             print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

```
cute
True
wet
N/A
wet
N/A
```



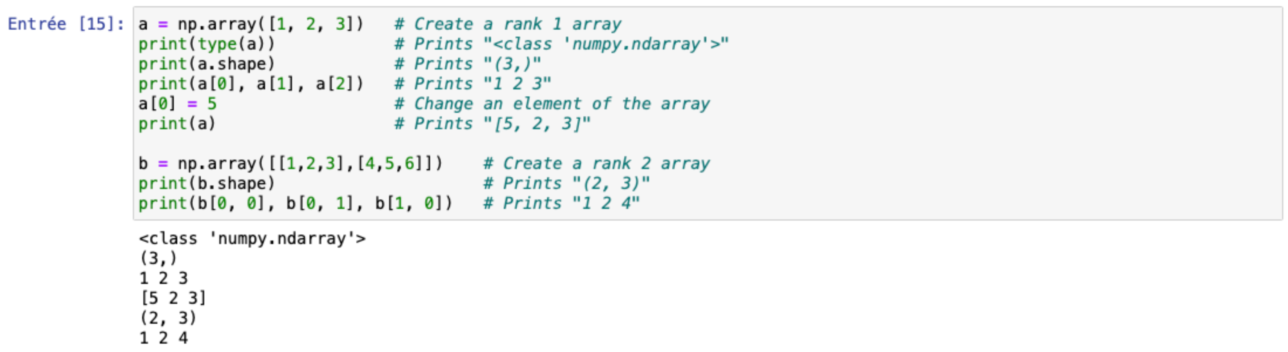
```
Entrée [14]: nums = [0, 1, 2, 3, 4]
             even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
             print(even_num_to_square)  # Prints "{0: 0, 2: 4, 4: 16}"
```

```
{0: 0, 2: 4, 4: 16}
```

Set, tuples are also possible…

NumPy

Arrays

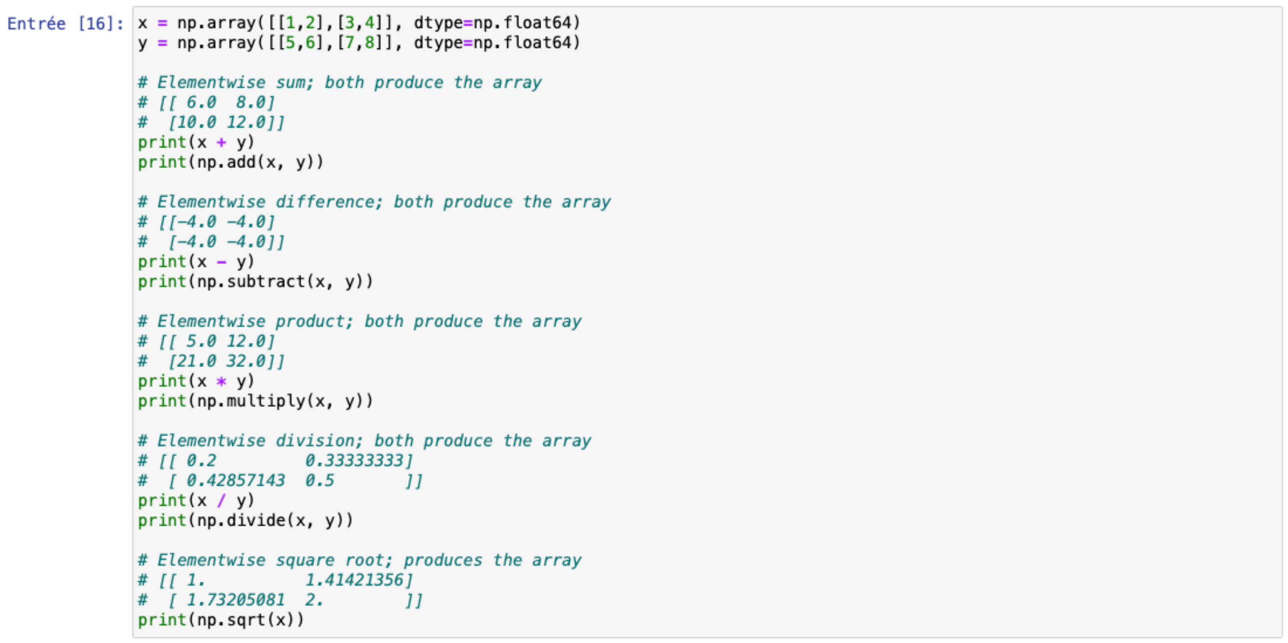


Entrée [15]:

```
a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
(2, 3)
1 2 4
```

Array math

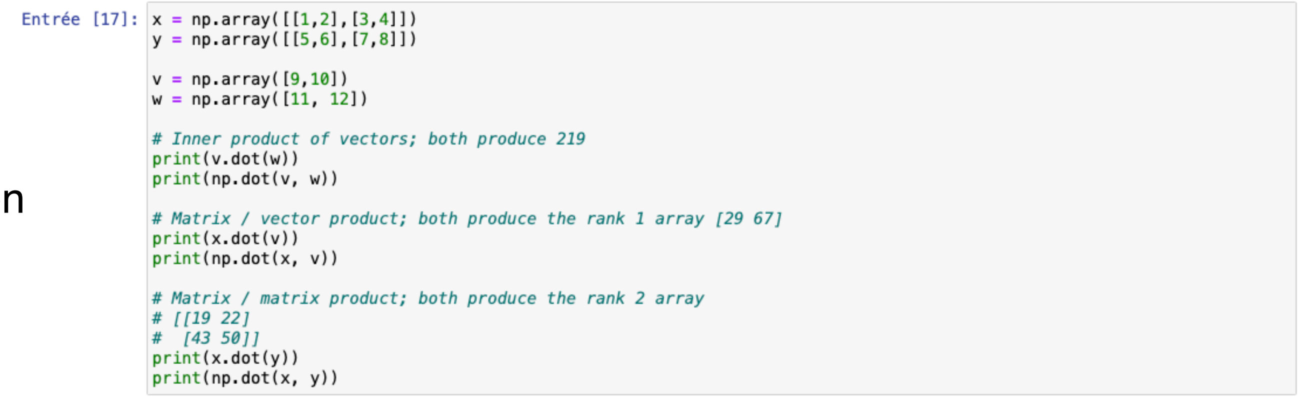


Entrée [16]:

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

NumPy

Matrix multiplication

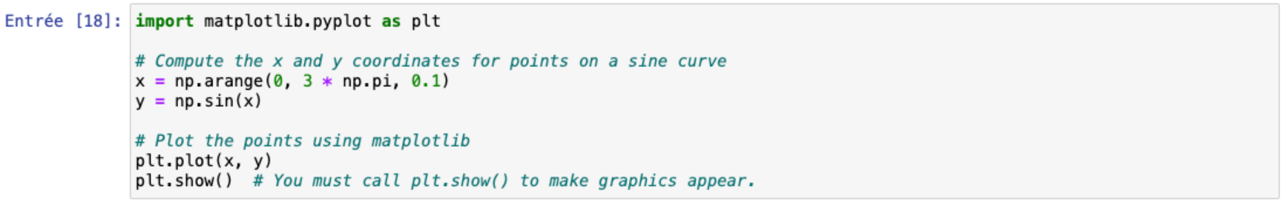


Entrée [17]:

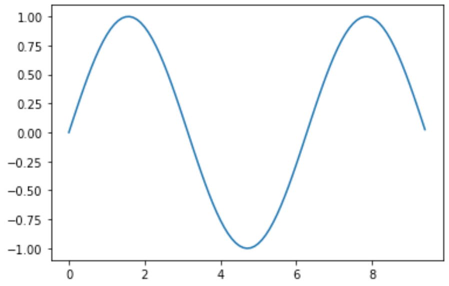
```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

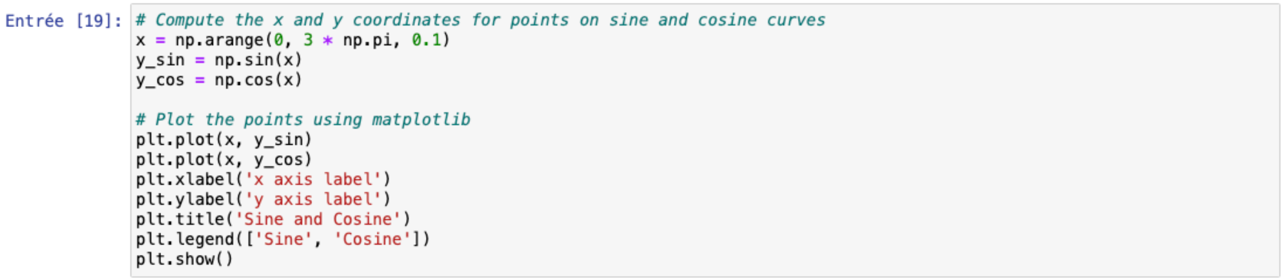
Matplotlib

Plot graphs

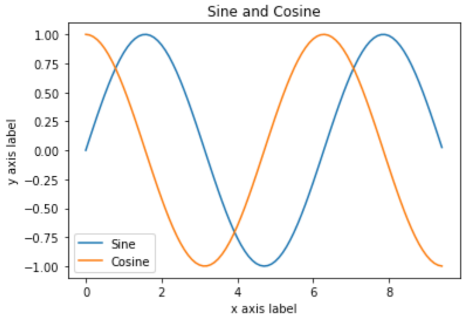


```
Entrée [18]: import matplotlib.pyplot as plt

             # Compute the x and y coordinates for points on a sine curve
             x = np.arange(0, 3 * np.pi, 0.1)
             y = np.sin(x)

             # Plot the points using matplotlib
             plt.plot(x, y)
             plt.show()  # You must call plt.show() to make graphics appear.
```



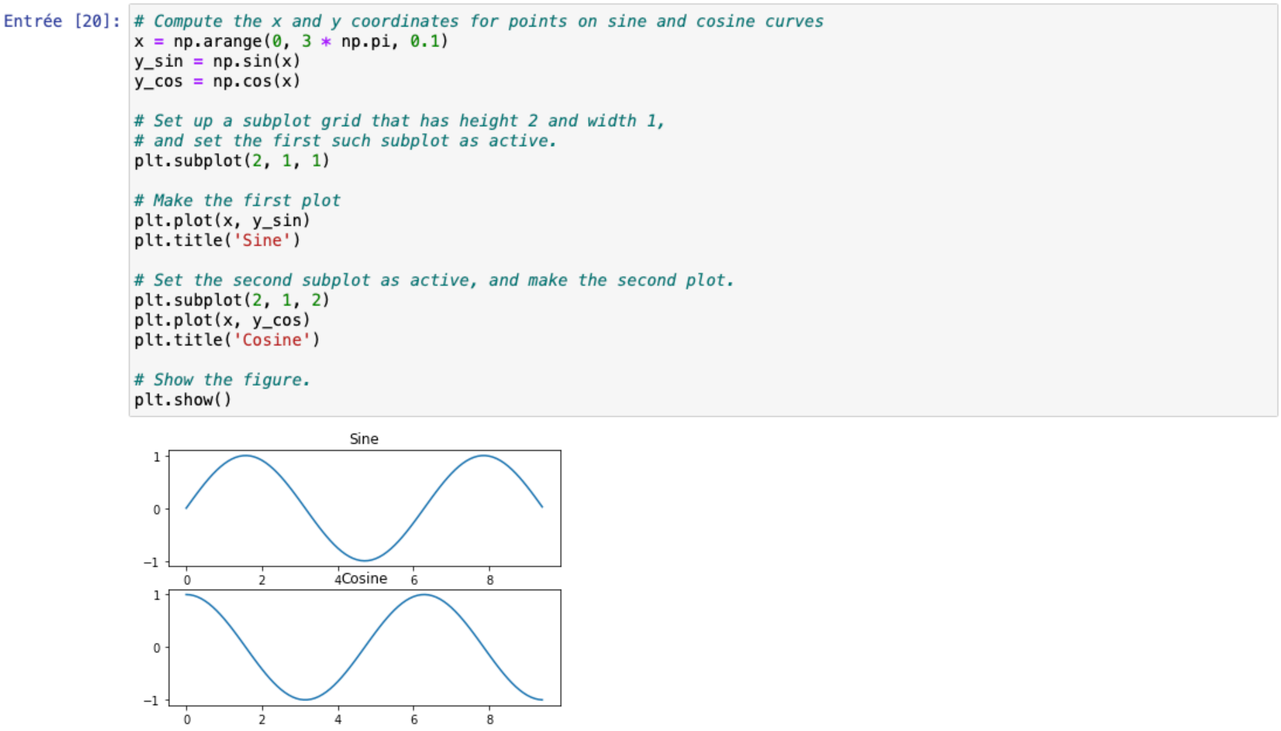


```
Entrée [19]: # Compute the x and y coordinates for points on sine and cosine curves
             x = np.arange(0, 3 * np.pi, 0.1)
             y_sin = np.sin(x)
             y_cos = np.cos(x)

             # Plot the points using matplotlib
             plt.plot(x, y_sin)
             plt.plot(x, y_cos)
             plt.xlabel('x axis label')
             plt.ylabel('y axis label')
             plt.title('Sine and Cosine')
             plt.legend(['Sine', 'Cosine'])
             plt.show()
```



Matplotlib

Subplot



Entrée [20]:

```
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

Last thing before we start : internet is your best ally !

Few references:

https://jupyter.org

https://numpy.org/doc/stable/

https://matplotlib.org