Sprint 2 - C BiCa

September 29, 2021

1 SPRINT 2

Learning how to use *Jupyter*

aaaand Markdown

Exercices to do:

1.1 Nivell 1

```
- Exercici 1
Instal·la el programa Anaconda amb Python 3, i Jupyter Notebook.

- Exercici 2
Utilitzant Jupyter Notebook executa alguns càlculs senzills, a la vegada que et familiaritzes a

- Exercici 3
Prova de crear títols, llistes, canviar l'estil de la lletra o afegir imatges dins del Notebook
```

1.2 Nivell 2

```
- Exercici 1
Exporta el Notebook com a pdf i com a html.
```

1.3 Nivell 3

```
- Exercici 1
Instal·la Nbextensions al Notebook de Jupyter.
```

1.4 1. Level 1

- **Exercice 1: Install Anaconda, Python and Jupyter Notebook**
  Obvously, I installed Anaconda, Python, the lates version (3.8.8), and now I'm using the Jupyter Notebook :) if not I won't be able to use Jupyter. I'm using the Jupyter lab since I find it easier cause you can see the file tree
- **Exercice 2:**
  Execute Jupyter notebook and do some easy python calculations, at the same time use some Markdown langutge to get use to it!

Markdown: - End a line with two or more spaces to create a line break.

- **bold**
  add two ** or two ___ (underscores) at the beggining and end of the text you want to bold.
- *italics*
  add one * or one _ (underscores) at the beggining and end of the text you want to italics.
- ***Bold and italics***
  add one _ (underscores) and two * at the beggining and end of the text you want to bold & italics.
  ooor two _ and one *.
- to underline:
  since markdown is a markup language and In fact you can use HTML/CSS inside it, the easiest way I've found so far is:
  text here underlined text other text.
- Backslash escape
  Using a backslash \ you can use litera * asterisc, _ underscore and several others

Python:

[2]:
```
print("Hello, world!")
```

```
Hello, world!
```

[3]:
```
2+2
```

[3]: 4

- **Exercice 3:**
  Try to add title, lists and different letter styles or add imatges in the note book

To insert an image:
- cell should be markdown
- inserta el codigo entre <> <img src="IMG_2745.jpg"> for local images (in the same folder as the jupyter file
- or this for web images <img src=http://animals.sandiegozoo.org/sites/default/files/2016-09/animals_hero_armadillo_0.jpg>

iuuujjjuuuuu!! Im using Jupyter and markdown!!

1.5 2. Level 2

Well, I finally manage to export in pdf and html after a lot of installing random packages from Random Packages Repository. I will add them in my new GitHub :smile:

[1]:
```
import emoji
```

[10]:
```
print(emoji.emojize('I\'m importing emojis!!!!! :thumbs_up:'))
```

```
I'm importing emojis!!!!!
```

[11]:
```
print(emoji.emojize('but maybe I want them in Markdown   '))
```

```
but maybe I want them in Markdown
```

and here there is an interesting link on why to use emojis in coding:
https://towardsdatascience.com/why-im-coding-with-emojis-in-jupyter-notebooks-23f08585ed3d

soooo go to the link and copy paste the emoji you want, thaaat easy!
https://getemoji.com/

1.6 3. Level 3

Install nbextensions in Jupyter notebook

but first…I want to know what are they for, no?

this link is very interesting with tips and trick for jupyter Notebook. It gives shorcuts, magic commands and some nbextensions:
https://towardsdatascience.com/optimizing-jupyter-notebook-tips-tricks-and-nbextensions-26d75d502663

and:
https://medium.com/@maxtingle/10-jupyter-notebook-extensions-making-my-lyfe-easier-f40139a334ce

1st of all it is **Not** possible to use nbextensions in Jupyter Lab, to use them I need to be in Jupyter Notebook!

Some useful extensions:

1. **Hinterland**
   It enables code autocompletion menu for every keypress in a code cell, instead of only enabling it with tab

2. **Split Cells Notebook**
   Enable split cells in Jupyter notebooks
   Enter command mode (Esc), use Shift + s to toggle the current cell to either a split cell or full width
3. **Table of Contents**
   The toc extension enables to collect all running headers and display them in a floating window, as a sidebar or with a navigation menu. The extension is also draggable, resizable, collapsable, dockable and features automatic numerotation with unique links ids, and an optional toc cell.
4. **Autopep8**
   Use kernel-specific code to reformat/prettify the contents of code cells
5. **Snippets**
   Adds a drop-down menu to insert snippet cells into the current notebook.
6. **Autopep8:**
   Reformats code to fix basic spacing errors that do not follow Pep8 guidelines. I recommend referencing Pep8 documentation for detailed style guidelines. However, this extension does a good job with spacing errors.

7. **Spellchecker:**
   Highlights incorrectly spelled words in Markdown and Raw cells. It is a pretty straightforward extension that is tremendously helpful in identifying misspelled words, especially when all your text, code, and LaTeX seem to be running together.

8. **Execute Time:**
   If you are like me and work in multiple Jupyter Notebooks at a time, the Execute Time extension is a helpful reminder of what has and has not been executed as you switch between notebooks. This extension displays a line at the bottom of your code block that tells you 1) the date and time the last execution of the code cell occurred and 2) how long it took to execute. The timing information not only allows you to quickly gage how long it will take to re-run the cell, but it also allows you to skip the step of importing Time or Timeit libraries to time the code execution.
9. **Initialization Cells:**
   The Initialization Cells extension allows you to mark cells as 'initialization cells' that will run automatically when the notebook is opened or when clicking the initialization button in the main toolbar.

1.7 4. Some real Python exercices!!

[1]:
```
a = "Hello, world"
print(a)
```

```
Hello, world
```

STRINGs practice!
I will use the len() method to calculate length of the string

[6]:
```
lengthStr = len(a)
print(lengthStr)
```

```
12
```

Strings as arrays

[7]:
```
firstLetter = a[0]
print(firstLetter)
```

```
H
```

[9]:
```
lowerString = a.lower()
print(lowerString)
```

```
hello, world
```

[10]:
```
upperString = a.upper()
print(upperString)
```

```
HELLO, WORLD
```

looping through a String

[14]:
```
for i in a:
    print(i)
```

```
H
e
l
l
o
,

w
o
r
l
d
```

I just realized how the **for** works in Python...I think

[15]:
```
print("world" in a)
```

```
True
```

[16]:
```
print("!" not in a)
```

```
True
```

[17]:
```
print(a + "!")
```

```
Hello, world!
```

[18]:
```
print(a[2:5])
```

```
llo
```

[19]:
```
print(a[5:])
```

```
, world
```

[20]:
```
print(a[:3])
```

```
Hel
```

[21]:
```
print(a[-5:-2])
```

```
wor
```

[22]:
```
print(a[-5])
```

```
w
```

[23]:
```
print(a[-5:])
```

```
world
```

[24]:
```
print(a.replace("H", "J"))
```

```
Jello, world
```

[25]:
```
a = a + "!"
print(a)
```

```
Hello, world!
```

[27]:
```
a = a.replace("H", "J")
print(a)
```

```
Jello, world!
```

Integers and casting practice

[31]:
```
num1 = 2
num2 = 3.3
num3 = "4"
print(type(num1))
print(type(num2))
print(type(num3))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

[33]:
```
num3 = float(num3)
print(type(num3))
print(num3)
```

```
<class 'float'>
4.0
```

[34]:
```
num4 = float(num1)
print(num4)
```

```
2.0
```

[35]:
```
num4 = int(num4)
print(num4)
```

```
2
```