

Tipo : Guía de Laboratorio
Capítulo : Fundamentos de Python
Duración : 60 minutos

I. OBJETIVO

- Ejecutar código de Python en un workbook Jupyter.
- Conocer los fundamentos de Python.

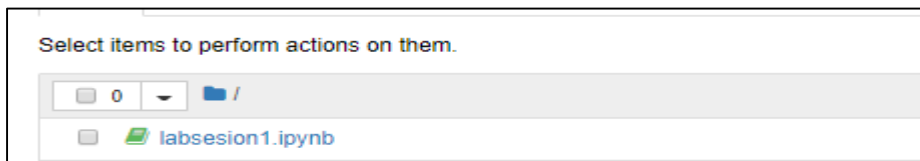
II. REQUISITOS

Los siguientes elementos de software son necesarios para la realización del laboratorio:

- Instalar Anaconda en Windows
- Navegador web

III. EJECUCIÓN DEL LABORATORIO

- Ejercicio: Ejecutar Python en un workbook Jupyter
 - Crear un entorno virtual
 - Conda create --name sesion1 python=3.5
 - Activate sesion1
 - Pip install jupyter
 - Activar jupyter en la línea de comandos con jupyter notebook
 - Abrir labsesion1.ipynb en el browser
 - Ejecutar el código y consultar



1. Sección **números**

```
Números

In [2]: 1 + 1
Out[2]: 2

In [3]: 1 * 3
Out[3]: 3

In [4]: 1 / 2
Out[4]: 0.5

In [5]: 2 ** 4
Out[5]: 16

In [6]: 4 % 2
Out[6]: 0

In [7]: 5 % 2
Out[7]: 1

In [8]: (2 + 3) * (5 + 5)
Out[8]: 50
```

2. Sección **variables**

```
Variables

In [9]: nombre_var = 2

In [10]: x = 2
         y = 3

In [11]: z = x + y

In [12]: z
Out[12]: 5
```

3. Sección **cadena**s

```
Cadenas

In [13]: 'sencillo'
Out[13]: 'sencillo'

In [14]: "doble"
Out[14]: 'doble'

In [15]: "mixto"
Out[15]: 'mixto'
```

4. Sección **impresión**

```
Impresión

In [16]: x = 'hola'

In [17]: x
Out[17]: 'hola'

In [18]: print(x)
         hola

In [19]: num = 12
         name = 'Sam'

In [25]: print('Mi número es: {one}, y mi nombre es: {two}'.format(one=num,two=name))
         Mi número es: 12, y mi nombre es: Sam
```

5. Sección listas

```
In [26]: [1,2,3]
Out[26]: [1, 2, 3]

In [27]: ['hola',1,[1,2]]
Out[27]: ['hola', 1, [1, 2]]

In [28]: mi_lista = ['a','b','c']
In [29]: mi_lista.append('d')
In [30]: mi_lista
Out[30]: ['a', 'b', 'c', 'd']

In [31]: mi_lista[0]
Out[31]: 'a'

In [32]: mi_lista[1]
Out[32]: 'b'

In [33]: mi_lista[1:]
Out[33]: ['b', 'c', 'd']

In [34]: mi_lista[:1]
Out[34]: ['a']
```

```
In [35]: mi_lista[0] = 'NEW'
In [36]: mi_lista
Out[36]: ['NEW', 'b', 'c', 'd']

In [37]: nest = [1,2,3,[4,5,['target']]]
In [38]: nest[3]
Out[38]: [4, 5, ['target']]

In [39]: nest[3][2]
Out[39]: ['target']

In [40]: nest[3][2][0]
Out[40]: 'target'
```

6. Sección condiciones

```
In [41]: if 1 < 2:
          print('Sip!')
          Sip!

In [42]: if 1 < 2:
          print('Sip!')
          Sip!

In [43]: if 1 < 2:
          print('primero')
        else:
          print('ultimo')
          primero

In [44]: if 1 > 2:
          print('primero')
        else:
          print('ultimo')
          ultimo

In [45]: if 1 == 2:
          print('primero')
        elif 3 == 3:
          print('medio')
        else:
          print('ultimo')
          medio
```

7. Sección bucles

```
In [46]: seq = [1,2,3,4,5]

In [47]: for item in seq:
          print(item)

1
2
3
4
5

In [48]: for item in seq:
          print('Yep')

Yep
Yep
Yep
Yep
Yep

In [49]: for jelly in seq:
          print(jelly+jelly)

2
4
6
8
10
```

8. Sección funciones

```
In [50]: def mi_func(param1='default'):
          """
          Docstring goes here.
          """
          print(param1)

In [51]: mi_func()

default

In [52]: mi_func('nuevo param')

nuevo param

In [53]: mi_func(param1='new param')

new param

In [54]: def cuadrado(x):
          return x**2

In [55]: out = cuadrado(2)

In [56]: print(out)

4
```

9. Sección librerías

```
In [57]: import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

IV. EVALUACIÓN

1. ¿Para qué sirve el nesting en las listas?
 - a. **Respuesta:** permite acceder a sub elementos de una lista.
2. ¿Cuáles son los componentes de una función?
 - a. **Respuesta:** nombre, parámetro y retorno.
- 3.
4. ¿Por qué es importante el Zen de Python?
 - a. **Respuesta:** son principios fundamentales de todo programador de python.