Laboratorio de Datos

Primer Cuatrimestre 2024

Práctica N° 1: Nociones básicas de Python.

1. Realizar las siguientes operaciones básicas en la consola

   (a) 2+2

   (b) a = 2

   (c) b = 5

   (d) a**3

   (e) 16

   (f) 2*a**2+0.5*b+(a+b)/2

   (g) asignar el resultado anterior a la variable c, imprimir el contenido de c en la consola (corriendo c o print(c)). ¿Ven un [1] delante del valor de c? Esto indica que es un vector.

2. Interpretar las siguientes operaciones lógicas y predecir el resultado antes de probar en la consola.

```
a = TRUE
b = FALSE
a == b
a || b
a == !b
```

3. Antes de probar en la consola, piense que van a dar estas operaciones:

```
a = 3
b = 4
print(a > b)
print(a <= b)
print(a != b)
print(a == b)
print(not(a > b))
x = 2
print(((x > a) or (10*x>b)) and not(b/a>x))
```

4. **Listas.** Las listas permiten guardar valores de distintos tipos en forma ordenada y acceder a los distintos elementos por su índice, comenzando desde 0.

```
s = [1, 2, 3.0, "hola", 7 + 3]
print(s)
print(s[0])
print(s[1])
print(s[-1])
```

5. **Vectores.** Para trabajar con vectores en Python (y en general para todo tipo de operaciones matemáticas) vamos a usar el paquete numpy. Para eso importamos primero la biblioteca numpy y definimos vectores con el comando np.array. Ejecutar el siguiente código y observar los resultados.

```
import numpy as np
v = np.array([1,2,3])
w = np.array([1.2, 7, np.pi])
print(v)
print(w)
print(v + w)
```

6. **La magia de Numpy** La biblioteca Numpy reproduce muchas funcionalidades de Matlab. La mayoría de las operaciones con vectores de Numpy se hacen coordenada a coordenada. Esto permite en muchos casos evitar usar ciclos o ciclos anidados y realizarlos con un solo comando.

   Ejecturar los siguientes comandos e interpretar los resultados.

```
v = np.array([1,2,np.e,7])
w = np.array([1.2, np.pi, 4, 5])
print("v = ", v)
print("w = ", w)
print("v + w = ", v + w)
print ("v**2 = ", v**2)
print ("v%2 = ", v%2)
print ("np.sum(v) = ", np.sum(v))
print ("np.sqrt(w) = ", np.sqrt(w))
print ("v > 3?", v > 3)
print ("w < 3.5?", w < 3.5)
```

7. Las operaciones lógicas or y and no se pueden aplicar a vectores. Debemos usar los símbolos | (or) y & (and).

```
print((v > 3) | (w < 3.5))
print((v > 3) & (w < 3.5))
```

8. ¿Cómo se puede aplicar not a un vector de variables booleanas (TRUE o FALSE)? Pueden probar algunas ideas o buscar la respuesta en Internet.

9. Algunos comandos pueden dar resultados inesperados. Intenten adivinar cuál va a ser el resultado de cada comando.

```
v = np.array([1,2,np.e,7])
w = np.array([1.2, np.pi, 4, 5])
z = np.array([0,1])
print(v * w)
print(v+2)
print(v+z)
```

10. Explorar estas distintas formas de extraer información de un vector.

```
v = np.array([1,2,np.e,7,5])
print(v[0])
print(v[1])
print(v[-1])
print(v[[0,3]])
print(v[0:3])
print(v[0:1])
```

11. También podemos seleccionar los elementos que cumplan alguna propiedad.

```
v = np.array([1,2,np.e,7,5])
w = np.array([1,0,2,5,0])
print(v[v>2])
print(v[w != 0])
```

12. **Matrices** Las matrices se definen en numpy como arrays de filas. Las operaciones usuales se realizan coordenada a coordenada al igual que con vectores.

```
A = np.array([[3, 2, 2], [-1, 0, 1], [-2, 2, 4]])
B = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
C = np.array([[0, 1, -1], [5, -2, 1]])
print(A + B)
print(A * B)
print(C**2)
```

13. El producto usual de matrices se realiza con el comando @

```
print(A @ B)
print(B @ C)
print(C @ A)
```

Funciones

Las funciones son bloques de código organizado que se usan para realizar tares específicas. Reciben un input (un número o una variable, por ejemplo) y devuelven un output. Los inputs van entre paréntesis y separados por una coma, si hay más de uno. Muchas funciones están disponibles en la biblioteca estándar de Python, otras estás agrupadas en distintas bibliotecas, como numpy que agrupa una gran cantidad de funciones matemáticas. El objetivo de estos ejercicios es familiarizarse con varias funciones básicas de Python.

14. Ejecutar estas operaciones en la consola para entender qué hacen las funciones de numpy:

```
a = np.sqrt(2)
print(a)
print(np.round(a))
print(np.round(a,2))
np.info(np.round)
```

```
np.info(np.ceil)
```

15. Muchas funciones de numpy se pueden aplicar también en arrays:

```
v = np.array([a, a**2, a**3, a**(.5)])
print("v = ", v)
print("np.floor(v) = ", np.floor(v))
```

16. Explorar las funciones np.max(), np.min(), np.sum(), np.mean() y np.sort() aplicadas al vector v del ejercicio anterior. ¿Qué hace cada una?

17. Utilizando solo las operaciones y funciones vistas en los ejercicios anteriores, escribir códigos de una sola linea para las siguientes funciones matemáticas.

    (a) $\|v\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = \sqrt{\sum_{i=1}^n v_i^2}$

    (b) $\|v - w\|_2$

    (c) $\langle v, w \rangle = \sum_{i=1}^n v_i \cdot w_i$

18. En Python podemos definir nuestras propias funciones utilizando def. ¿Qué hace la siguiente función? ¿Qué resultados esperan al aplicar la función a los vectores v_1 y v_2?

```
def todosPositivos(v):
  r = np.all(v>0)
  return(r)

v1 = np.array([3, 4])
v2 = np.array([3, 5, -1, 1])
print(todosPositivos(v1))
print(todosPositivos(v2))
```

19. Definir una función que calcule la norma-2 de un vector y verificarla en los vectores v_1 y v_2 del ejercicio anterior.

20. Usar las funciones np.argmin() y np.argmax() con el mismo vector de alturas. Interpretar qué hace cada una.

21. Reproducir estos usos de la función random.choice() e interpretar qué hace esta función.

```
x = np.array(["cara", "ceca"])
s = random.choices(x, k = 10)
print(s)
y = np.array([1,2,3,4,5,6])
t = random.choices(x, y = 10)
print(t)
u = random.sample(x, y = 4)
```

Realicen otras pruebas para descubrir la diferencia entre random.choices() y np.sample()

22. **El teorema central del límite.** Este teorema asegura que si tiramos n veces una moneda, el promedio de veces que sale cara tiende a $1/2$ cuando n tiende a infinitio. Utilizando un código de una línea, simular 10 lanzamientos de una moneda y calcular el promedio de veces que sale cara. Repetir para $n = 1000$ y $n = 100.000$.

    Sugerencia: hay varias formas distintas de hacerlo, una posibilidad es usar el comando count que puede aplicarse a listas.

Archivos de datos

23. La biblioteca Pandas nos permite trabajar fácilmente con archivos de datos.

    (a) Leer el archivo casos_coronavirus.csv.

    (b) Graficar la curva de casos por día.

    (c) Graficar la curva de casos acumulados.

    (d) Definir y como el logaritmo de la cantidad de casos acumulados y graficar y en función de la cantidad de días transcurridos.

    (e) Estimar tomando dos valores la pendiente de la recta para los datos a partir del dia 30.

    Utilicen o modifiquen el siguiente código.

```
import pandas as pd
datos = pd.read_csv("casos_coronavirus.csv")    # dataFrame
print(datos)

# Convertimos los datos a np.array
datosNP = datos.to_numpy()
print(datosNP)

x = np.linspace(1,96,96)
print(x)

plt.plot(x, datosNP[:,2])

# Tomamos logaritmos para linealizar
y = np.log(np.float64(datosNP[:,2]))
print(y)

plt.plot(x,y)
```