Laboratorio de Datos

Primer Cuatrimestre 2024

Práctica N° 1: Nociones básicas de Python.

Como le dijo el Sr. Miyagi a Daniel, "*encerar... pulir*". No se puede hacer ciencia de punta sin antes volver intuitivos los conceptos fundacionales de un lenguaje. Así que *copien los comandos de esta guía a mano en una consola*, y traten de estimar qué van a devolver, antes de ejecutarlos.

Si ya instalaron el entorno de trabajo que sugiere el `README.md`, pueden lanzar una consola adecuada ejecutando `source venv/bin/activate && python`.

1. Realizar las siguientes operaciones básicas en la consola

   (a)

```
2 + 2
a = 2
b = 5
a**3
16 %% 5  # probar con otros numeros para entender que significa
2 * a**2 + 0.5 * b + (a + b) / 2
```

   (b) 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 == (not b)
```

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

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

4. Bonus: Abra la documentación de Python y lea sobre tuplas.

5. **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]
s, s[0], s[1], s[-1]
```

6. **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 cxdigo y observar los resultados.

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

7. **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.

   Ejecutrar los siguientes comandos e interpretar los resultados.

```
np.set_printoptions(precision=2, suppress=True)
v = np.array([1,2,np.e,7])
w = np.array([1.2, np.pi, 4, 5])
for expr in [
    'v', 'w', 'v + w', 'v ** 2', 'v % 2', 'np.sum(v)', 'np.sqrt(w)', 'v > 3', 'w < 3.5'
]:
    print(f"{expr:11s} == {eval(expr)}")
```

8. Bonus: descrifrar qué magia hace la f-string que recibe como argumento `print`. ¿Y eso de eval?

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

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

10. ¿Cómo se puede aplicar `not` a un vector de variables booleanas ($x \in \{\text{True, False}\}$? Pueden probar algunas ideas o buscar la respuesta en Internet.

11. 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])
v * w
v+2
v+z
```

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

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

13. 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])
v[v > 2]
v[w != 0]
```

14. **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]])
A + B
A * B
C**2  # bonus: pruebe con np.
```

15. El producto usual de matrices se realiza con el comando `@`. `A.T` es la transpuesta de A.

```
A @ B
B @ C
B.shape, C.shape
B @ C.T
```

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.

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

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

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

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

18. 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?

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

20. 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):
    return np.all(v > 0)


v0 = np.array([3, 4])
v1 = np.array([3, 5, -1, 1])
for i, v in enumerate([v0, v1]):
    assert todosPositivos(v), f"No todos los elementos son positivos en v{i}"
```

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

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

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

```
import random


x = ["cara", "ceca"]
random.choices(x, k=10)
y = range(1, 7)
random.choices(y, k=10)
random.sample(y, k=4)
random.sample(x, 4)  # Entienden el error? Usen ‘help(random.sample)‘ para indagar.
```

    Realicen otras pruebas para descubrir la diferencia entre `random.choices` y `random.sample`

24. **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 $n = 100.000$.

25. Bonus: Plantee al menos tres formas distintas de hacerlo; al menos una de ellas usando sólo tipos de datos nativos (lista, tupla, diccionario, et cetera).

Archivos de datos

26. 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 log_cum_casos como el logaritmo de la cantidad de casos acumulados y graficar 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
import matplotlib.pyplot as plt

df = pd.read_csv("Datos/casos_coronavirus.csv")
df.info()
df["fecha"] = pd.to_datetime(df.fecha, format="%d-%m-%Y")
casos = df.set_index("fecha").confirmados_Nuevos

plt.plot(casos)
plt.show()
# Bonus: pueden rotar las etiquetas del eje X para que no se superpongan?

cum_casos = casos.cumsum()
plt.plot(cum_casos)
plt.show()

# Tomamos logaritmos para linealizar
plt.plot(np.log(cum_casos))
plt.show()

# Alternativamente, podemos mantener los datos "al natural" y cambiar la _escala_ del
    eje y.
plt.plot(cum_casos), plt.yscale("log")
plt.show()
```