

✓ Introducción a Python en la Ciencia

Python en el Ámbito Científico

Sesión 1

Alan Badillo Salas (alan@nomadacode.com)

✓ Bienvenida del curso

La naturaleza y la realidad están compuestas de elementos complejos de entender para el humano, y el descubrimiento del cómo funcionan las cosas ha llevado siglos de aprendizaje por parte de la humanidad.

Un artesano es capaz de tomar materiales y crear un artefacto capaz de reflejar un diseño que mezcla física, química, matemáticas, biología y demás ciencias del entendimiento. Parecería imposible construir una guitarra capaz de tocar música, un mole con el sabor de mil especias o una hamaca con diseños extraordinarios, pero todo esto abunda en nuestro día a día.

¿Por qué nos interesa saber cómo funcionan las cosas?

Piensa brevemente en el porqué y aporta una opinión al respecto.

¿En qué se diferencia la ciencia del arte y en qué convergen?

Piensa acerca del método científico y las formas no científicas de explicar y expresar la realidad

¿Qué es un fenómeno natural y cómo se describe?

Intenta pensar en un fenómeno real que esté relacionado al área más cercana de tu interés, explica brevemente en qué consiste de un modo informal y luego intenta volver a explicarlo del modo más formal posible

¿Cómo se plantea un problema y porqué es un problema?

Intenta definir qué es un problema y da un ejemplo de un problema cercano a tu realidad, explica para quién eso que ejemplificas representa un problema

✓ Introducción a los problemas del ámbito científico

Quizás no todos los problemas parezcan problemas, algunos podrían pensar que eso no es un problema, más bien un conflicto particular o divagar al respecto. Pero, lo cierto es que la naturaleza está llena de problemas. Piensa en como una hormiga tiene buscar comida y planificar una ruta para llevar la comida a la colmena, estimar cuántas hormigas adicionales se necesitarán para llevar el trozo de comida al hormiguero y calcular el costo-beneficio del esfuerzo involucrado. Además, piensa en que la hormiga tiene que hacer esto cada día para sobrevivir, quizás un día no sea necesario, quizás varias semanas de comida estén garantizadas, pero eventualmente volverá a tener que resolver los mismos problemas.

Sin importar si somos humanos u hormigas, el problema este problema está planteado y su planteamiento es el mismo, desde la perspectiva de la hormiga es *lo que se tiene que hacer*, desde la perspectiva del humano es *lo que la hormiga tiene que hacer*, pero ambas perspectivas sobre el mismo problema no modifican el planteamiento del problema: es *lo que se tiene que resolver*.

Cuándo observamos un fenómeno, este involucra casi infinitos factores, meteorológicos, espaciales, químicos, físicos, interacciones de seres vivos. Todo esto hace que los fenómenos sean complejos de entender, observar fenómenos reales es similar o superior a observar sistemas complejos, llenos de variables desconocidas, factores ocultos o magia chamánica.

El entendimiento de los fenómenos es lo que ilumina a la humanidad, le quita el velo ocultista, revela la caja negra, explica la magia de las cosas y se vuelve tecnología. Entre la magia y la tecnología sólo hay un paso: el control de la realidad.

Si pudiéramos entender todos los fenómenos que nos rodean, nada estaría oculto a nuestra conciencia, podríamos controlarlo todo o al menos entenderlo y aceptarlo. Pero, siempre hay una búsqueda incansable por la verdad, y esta búsqueda nos motiva a ser más inteligentes, astutos y superiores.

Imagina al primer ladrón de la naturaleza, aquél organismo vivo que con un esfuerzo mínimo logró controlar a otro para que obedecer sus órdenes. ¿Crees que fue un virus, una bacteria o un primate?

La astucia para resolver problemas nos vuelve inteligentes, no basta saber cómo funciona algo, sino tener la capacidad de aprovechar ese conocimiento. Para lograrlo, hace falta más que plantear un problema, se debe construir un modelo del problema con el que seamos capaces de manipularlo y proyectar los datos, poder viajar al futuro o al pasado a través del modelo.

Piensa en una bola de cañon que es disparada para atacar a un enemigo, si se dispara demasiado fuerte, la bola del cañón caerá bastante lejos del enemigo, y si se dispara demasiado suave, la bola apenas caera a unos metros del caño. ¿Cuál debería ser una fuerza suficiente para disparar un cañón?, ¿Cuánta polvora ponemos dentro del cañón?

Crear un modelo que explique la realidad no es sencillo, Newton tardó décadas para poder describir cómo una manzana caía de un árbol, el fenómeno es generalmente más sencillo que el problema, el fenómeno es que una manzana cae de un árbol, el problema es que todos los cuerpos son atraídos hacia el suelo, pero plantear un modelo que describa cómo todos los objetos dentro de un espacio son atraídos a un cuerpo de mayor masa por influencia de una fuerza gravitatoria, está en otro nivel.

¿Qué es un fenómeno real y natural?

Intenta explicar qué es un fenómeno real, explica si es lo mismo hablar de fenómenos reales y naturales, da un ejemplo de lo que se te ocurra por fenómeno real y por fenómeno natural, resume brevemente qué entiendes por realidad y qué entiendes por naturaleza

¿Cómo se plantea un problema?

Explica cuáles son los factores más importantes para plantear un problema, explica si existe un sujeto ligado al problema, explica si el problema es general o particular, da un ejemplo de un planteamiento de algún problema, explica si el problema puede ser entendido por una máquina o qué necesitaría la máquina para entenderlo

¿Qué es un modelo matemático?

Explica qué es un modelo en general, luego explica que sería un modelo matemático y finalmente da un ejemplo de un modelo matemático, explica si el modelo matemático puede ser entendido por una máquina o qué necesitaría la máquina para poder entenderlo

✓ Problema 1 - Levantar un objeto del suelo

Fenómeno: Hay un objetos pequeños regados en el suelo y se desea levantarlos para tirarlos a la basura si no valen nada o venderlos si se detecta algún valor.

Plantamiento del problema: En un suelo plano, hay objetos de dimensión limitada a la sujeción de una pinza, los cuales deben ser sujetados por la pinza, analizados y dirigidos a un contenedor de objetos valiosos o un contenedor de objetos basura.

Modelo matemático: En un plano infinito sobre los ejes X y Y , hay objetos que pueden ser contenidos en un poliedro convexo y acotado a un radio máximo R . Definimos como una **pinza** de longitud R a un segmento de tamaño variable en $[0, R]$ con un punto de referencia externo que establece la posición del centro del segmento y el vector normal al segmento, así como el tamaño de apertura del segmento, donde 0 será una apertura nula y 1 la apertura máxima que lleve al segmento a tener un tamaño R . Se define la operación de **sujeción** de la *pinza* hacia un objeto, si la posición de la pinza y su apertura coincide con dos puntos en la superficie del objeto, que no atraviesen al mismo. Se plantea un problema de optimización, de llevar la pinza de la posición $U_0 = (x_0, y_0, z_0)$ a la posición $U_1 = (x_1, y_1, z_1)$ de tal forma que la pinza sea capaz de sujetar el objeto, este problema requiere encontrar la normal $n_1 = (a_1, b_1, c_1)$ tal que la pinza sea capaz de sujetar el objeto sin atravesarlo

✓ Ejemplo: Camino aleatorio óptimo

Reolver el **Problema 1** sobre levantar un objeto del suelo parece bastante complejo, sin embargo, podemos plantear un subproblema más pequeño para resolver una de las partes del problema.

Por ejemplo, pasar de una posición $U_0 = (x_0, y_0)$ a una posición $U_1 = (x_1, y_1)$ mediante un camino de pasos, y luego medir el error entre el camino y la posición deseada para intentar optimizarlo.

Paso 1: Calcular una secuencia de pasos aleatorios entre el punto U_0 y el punto U_1 .

Imaginemos que no sabemos cuál es el camino más corto entre U_0 y U_1 como para establecer una dirección hacia donde movernos en cada paso, entonces, tendríamos que movernos aleatoriamente en cada paso sin saber si nos estamos acercando o alejando.

Vamos a suponer que en 10 pasos tenemos que llegar lo más cerca que sea posible al punto U_1 partiendo del punto U_0 . Cada paso lo daremos en una dirección aleatoria.

```

# 1. Definimos el punto (x0, y0) de donde partiremos
# por ejemplo, de punto (5, 3)
x0, y0 = (5, 3)

# 2. Generamos una lista vacía donde guardar los pasos aleatorios
pasos = []

# 3. Agregamos el punto de partida a los pasos, siendo el primer paso
pasos.append((x0, y0))

# 4. Suponemos que nos ubicamos actualmente en la posición (x, y)
# la cual será actualizada en cada paso que demos
x, y = (x0, y0)

# 5. Para cada paso que demos, buscamos una dirección aleatoria
# actualizamos la ubicación (x, y) y guardamos la posición
# a la que llegamos en la lista de pasos
import random
for i in range(10):
    # Calculamos una dirección aleatoria en x y y
    # podemos desplazarnos aleatoriamente entre (-1, 1) en x y y
    dx = random.uniform(-1, 1)
    dy = random.uniform(-1, 1)

    # Actualizamos la posición usando la posición actual
    # más la diferencia aleatoria en x o y
    x = x + dx
    y = y + dy

    # Guardamos el paso
    pasos.append((x, y))

# Así quedarían los 10 pasos más el inicial
pasos

```

```

➡ [(5, 3),
   (4.73309149372453, 2.7709192343224007),
   (4.161457741412595, 3.616247060884234),
   (3.203719816164581, 3.020049361570387),
   (3.1008143412474194, 3.9001804865068688),
   (3.9617372484608833, 3.582033757063316),
   (4.778524588866312, 3.5886447312032774),
   (4.505354520082653, 3.078234213159549),
   (4.830695836176243, 3.7914818064714857),
   (5.795913731959092, 2.989391523793527),
   (6.270827267836054, 3.6518350933929167)]

```

Paso 2: Calcular que tan lejos está cada paso de U_1

Ahora tenemos una lista de 10 pasos más el punto inicial al principio, pero no sabemos si esos pasos están cerca o lejos del punto U_1 al que queremos llegar, por lo que tenemos que calcular el error entre cada posición al dar los pasos y el punto $U_1 = (x_1, y_1)$.

Entre menor sea el error, significará que el camino es más corto, es decir, en cada paso estamos lo más cerca posible del punto U_1 .

Para medir el error entre un punto (x, y) y el punto (x_1, y_1) debemos usar una función que mida la distancia entre ambos puntos, por ejemplo, en un plano, podemos usar una distancia euclidiana, pero en una esfera deberíamos usar una distancia esférica.

```
# 1. Definimos el punto (x1, y1) a donde queremos llegar
x1, y1 = (10, 10)

# 2. Definimos una función que mida la distancia entre el punto (x, y)
# y el punto (x1, y1)
def distancia(x, y, x1, y1):
    return ((x - x1) ** 2 + (y - y1) ** 2) ** 0.5

# 3. Acumulamos el error entre las distancias de cada paso y el punto (x1, y1)
# partiendo de 0
e = 0

# 4. Recorremos cada paso y calculamos el error entre el punto alcanzado
# en cada paso y el punto (x1, y1) usando la función de distancia
for x, y in pasos:
    # Calculamos la distancia entre los puntos
    d = distancia(x, y, x1, y1)

    # Acumulamos el error
    e = e + d

e
```

↔ 94.68273833408716

Paso 3: Minimizamos el error

El error total entre cada paso que dimos y el punto $(x1, y1)$ puede ser alto, al menos sabemos que si no es cero este tendrá mayor distancia entre los pasos y el punto al que queremos llegar.

Para poder optimizar el error, podríamos sustituir un paso por otro aleatorio, si el error es menor, entonces podemos conservar ese paso, sino podemos descartarlo ya que no mejora nada.

Entonces, el procedimiento que podríamos diseñar para optimizar los pasos, será reemplazar un paso seleccionado aleatoriamente a la vez, hasta que después de muchas épocas de reemplazo, tengamos el mejor camino posible.

```
# 1. Primero creamos una copia de los pasos para no perder los originales
pasos_optimizados = pasos[:]
```

```
# 2. Definimos la función que mide el error
def error_pasos(pasos):
    return sum([distancia(x, y, x1, y1) for x, y in pasos])
```

```
# 3. Durante 5 épocas buscamos minimizar el error
for epoca in range(5):
```

```
    # Creamos una copia con los pasos de prueba
    pasos_prueba = pasos_optimizados[:]
```

```
    # Primero seleccionamos el índice de un paso a reemplazar
    # excepto para el primer punto que es el inicial
    i = random.choice(range(1, 11))
```

```
    # Luego obtenemos el punto anterior para tomar una dirección diferente
    j = i - 1
    xj, yj = pasos[j]
```

```
    # Suponemos que la ubicación actual es (xj, yj)
    x, y = xj, yj
```

```
    # Ahora calculamos nuevos pasos partiendo de xj, yj
    # hasta completar los pasos faltantes reemplazados
    for h in range(i, 11):
        dx = random.uniform(-1, 1)
        dy = random.uniform(-1, 1)
```

```
        x = x + dx
        y = y + dy
```

```
    # Reemplazamos el h-ésimo punto desde i hacia enfrente
```



```

# en la lista con los pasos de prueba
pasos_prueba[h] = (x, y)

# Medimos el error entre los pasos y los nuevos pasos de prueba
e_pasos = error_pasos(pasos_optimizados)
e_prueba = error_pasos(pasos_prueba)

# Si el error es menor en los pasos de prueba
# reemplazamos los pasos
if e_prueba < e_pasos:
    pasos_optimizados = pasos_prueba

# Se repetirá en otra época

# Ahora se han alcanzado los mejores pasos en 5 épocas
pasos_optimizados

```

```

→ [(5, 3),
    (4.73309149372453, 2.7709192343224007),
    (4.161457741412595, 3.616247060884234),
    (3.203719816164581, 3.020049361570387),
    (4.1156379106906025, 3.7337844300148424),
    (3.328271791493484, 3.8386156324110683),
    (4.1353408488115875, 4.731239205250039),
    (5.046126994415438, 4.105125349184581),
    (5.07554555981286, 4.873696065431507),
    (5.868603913616463, 4.897802469926322),
    (5.73651859127162, 5.19277946383623)]

```

Inspeccionamos el error entre los pasos originales y los pasos optimizados

```

print("Error pasos originales:", error_pasos(pasos))
print("Error pasos optimizados:", error_pasos(pasos_optimizados))

```

```

→ Error pasos originales: 94.68273833408716
   Error pasos optimizados: 89.30013496433983

```

Se observa que el error es ligeramente mejor para los pasos optimizados.

Paso 4: Graficamos la diferencia entre los pasos originales y los pasos optimizados

```
import matplotlib.pyplot as pyplot
```

```
X, Y = zip(*pasos)
```

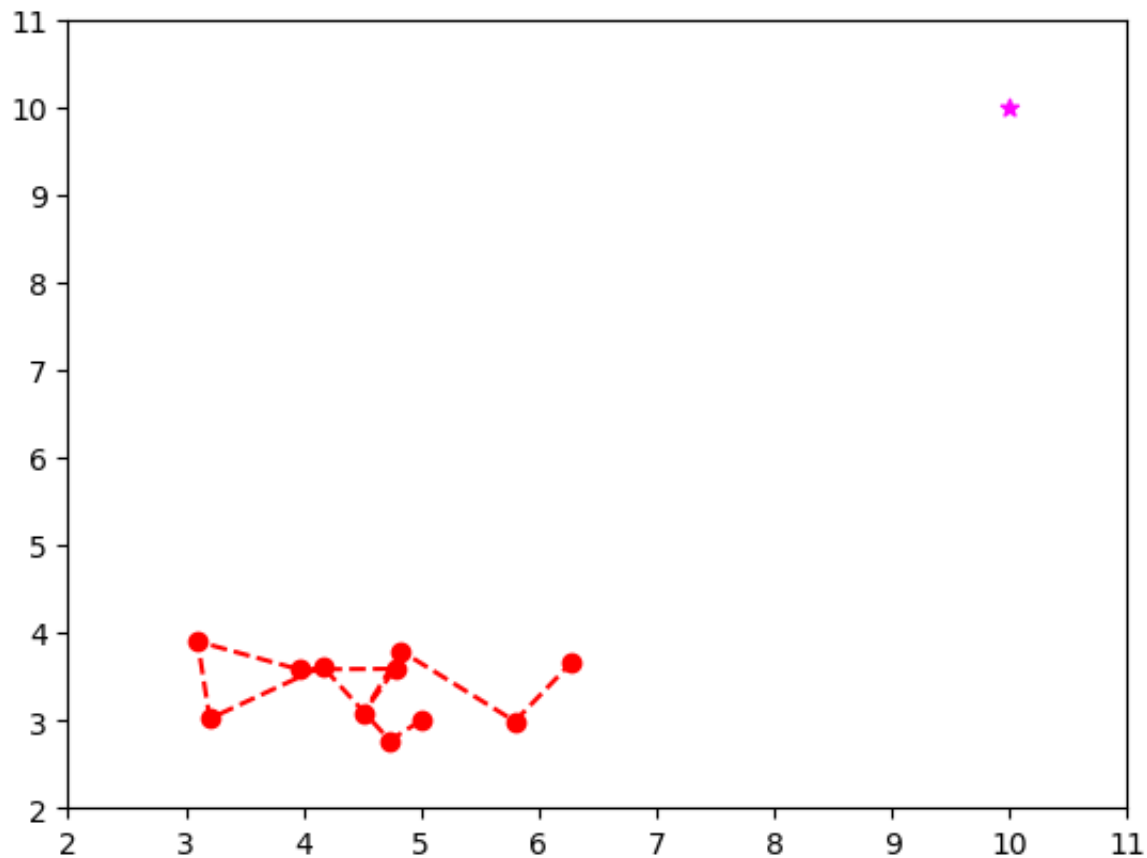
```
pyplot.plot(X, Y, "ro--")
```

```
pyplot.scatter([10], [10], c="magenta", marker="*")
```

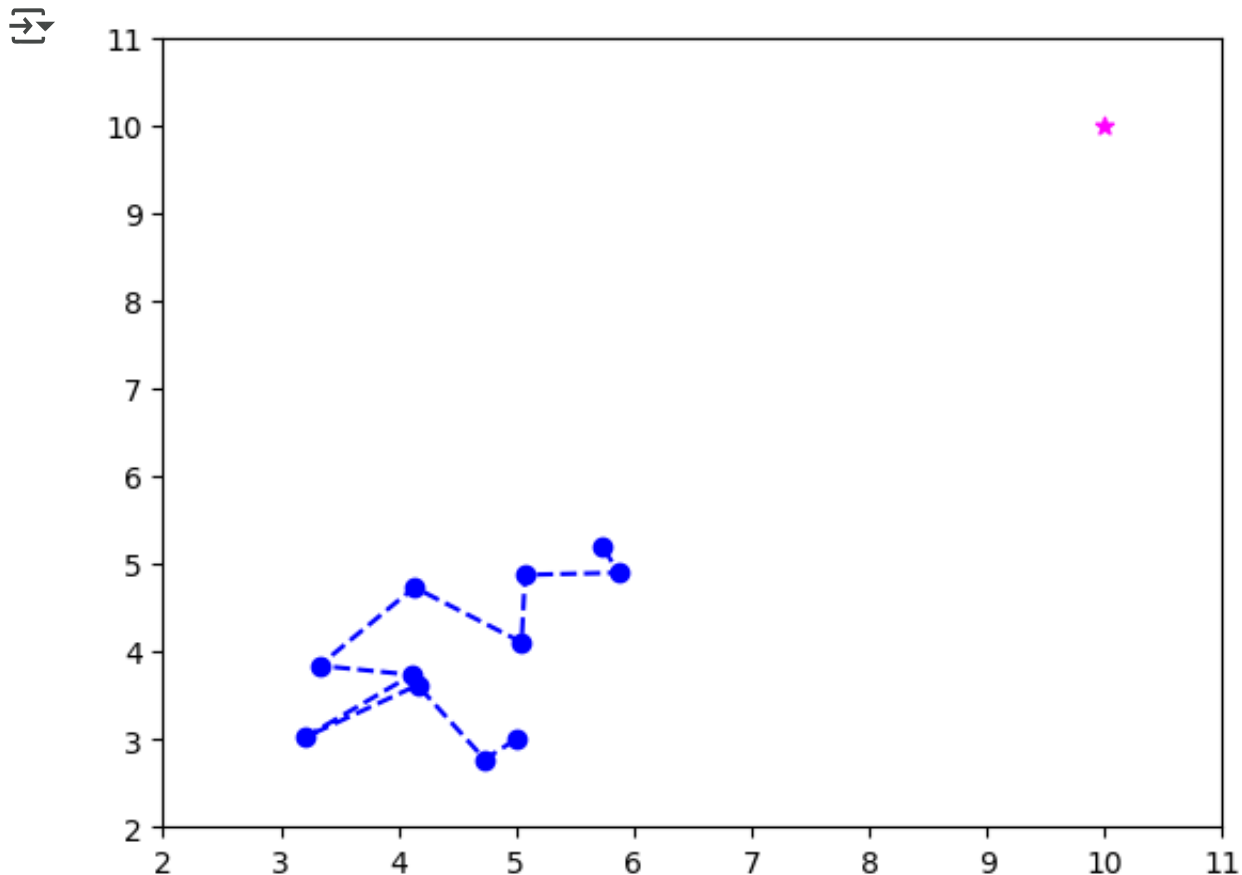
```
pyplot.xlim(2, 11)
```

```
pyplot.ylim(2, 11)
```

```
pyplot.show()
```



```
Xopt, Yopt = zip(*pasos_optimizados)
pyplot.plot(Xopt, Yopt, "bo--")
pyplot.scatter([10], [10], c="magenta", marker="*")
pyplot.xlim(2, 11)
pyplot.ylim(2, 11)
pyplot.show()
```



Paso 5: Ejecutamos durante 5,000 épocas de optimización

Parece que el camino se optimiza ligeramente en 5 épocas, pero no es visible ni natural ver como el camino mejora, pero si probamos optimizar durante otras 5,000 épocas tenemos

```
for epoca in range(5000):
    pasos_prueba = pasos_optimizados[:]

    i = random.choice(range(1, 11))

    j = i - 1
    xj, yj = pasos[j]

    x, y = xj, yj

    for h in range(i - 1):
```

```

for h in range(1, 11):
    dx = random.uniform(-1, 1)
    dy = random.uniform(-1, 1)

    x = x + dx
    y = y + dy

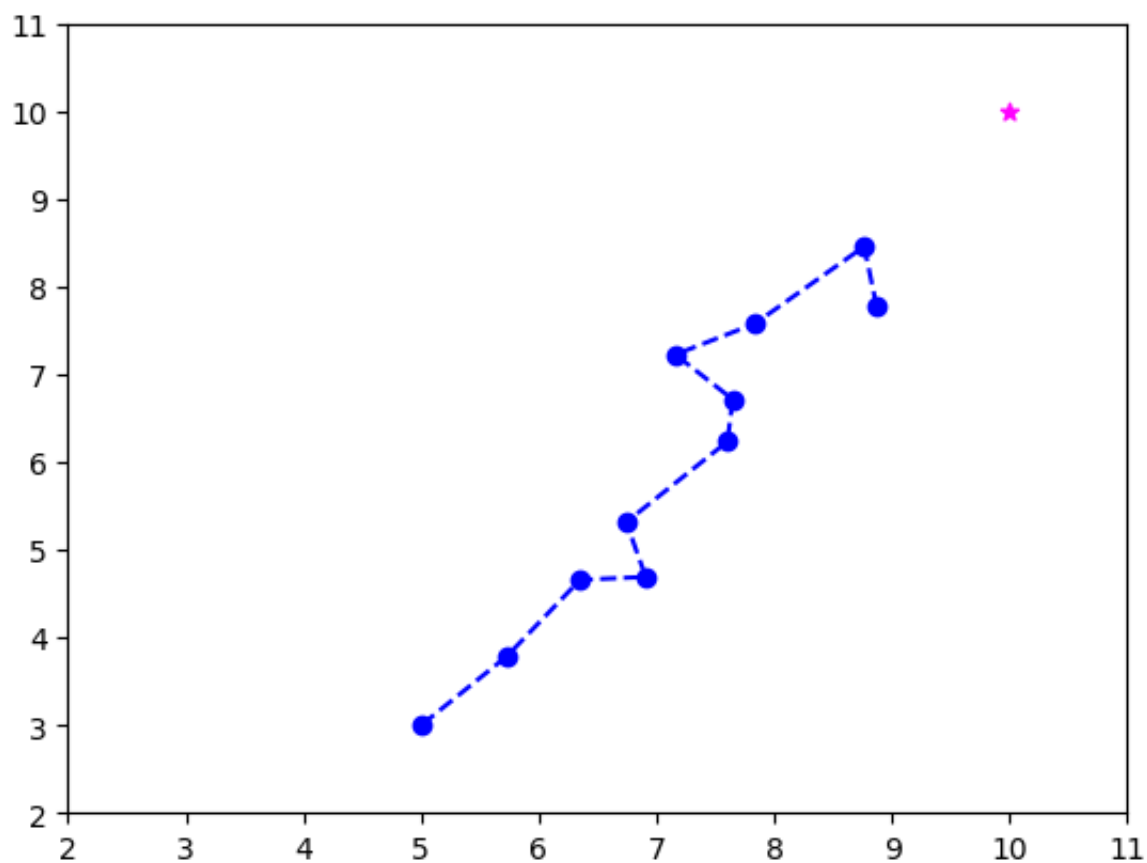
    pasos_prueba[h] = (x, y)

e_pasos = error_pasos(pasos_optimizados)
e_prueba = error_pasos(pasos_prueba)

if e_prueba < e_pasos:
    pasos_optimizados = pasos_prueba

Xopt, Yopt = zip(*pasos_optimizados)
pyplot.plot(Xopt, Yopt, "bo--")
pyplot.scatter([10], [10], c="magenta", marker="*")
pyplot.xlim(2, 11)
pyplot.ylim(2, 11)
pyplot.show()

```



Si ejecutamos varias veces, el camino será cada vez mejor hasta encontrar una ruta óptima que nos lleve del punto U_0 al punto U_1 .

✓ Problema 2 - Número de monedas que carga una persona

Fenómeno: El banco ha observado que muchas personas prefieren cargar monedas que billetes, esto debido a que las usan en su día a día para el transporte o resultado de las transacciones que realiza. En una hora aleatoria del día, se pueden encuestar a las personas en la calle o un salón de clases sobre la cantidad de monedas que carga la persona, considerando solo las monedas de \$1, \$2, \$5 y \$10 pesos mexicanos.

Planteamiento del problema: ¿Cómo podemos comparar el número de monedas que carga una persona para las diferentes denominaciones?.

Modelo matemático: Primero podemos comparar cómo se distribuye el número de monedas para cada denominación, luego si asumimos que se distribuyen de forma normal, podemos crear una caja estadística para cada denominación, para analizar sus cuantiles.

✓ Ejemplo: Cajas estadísticas

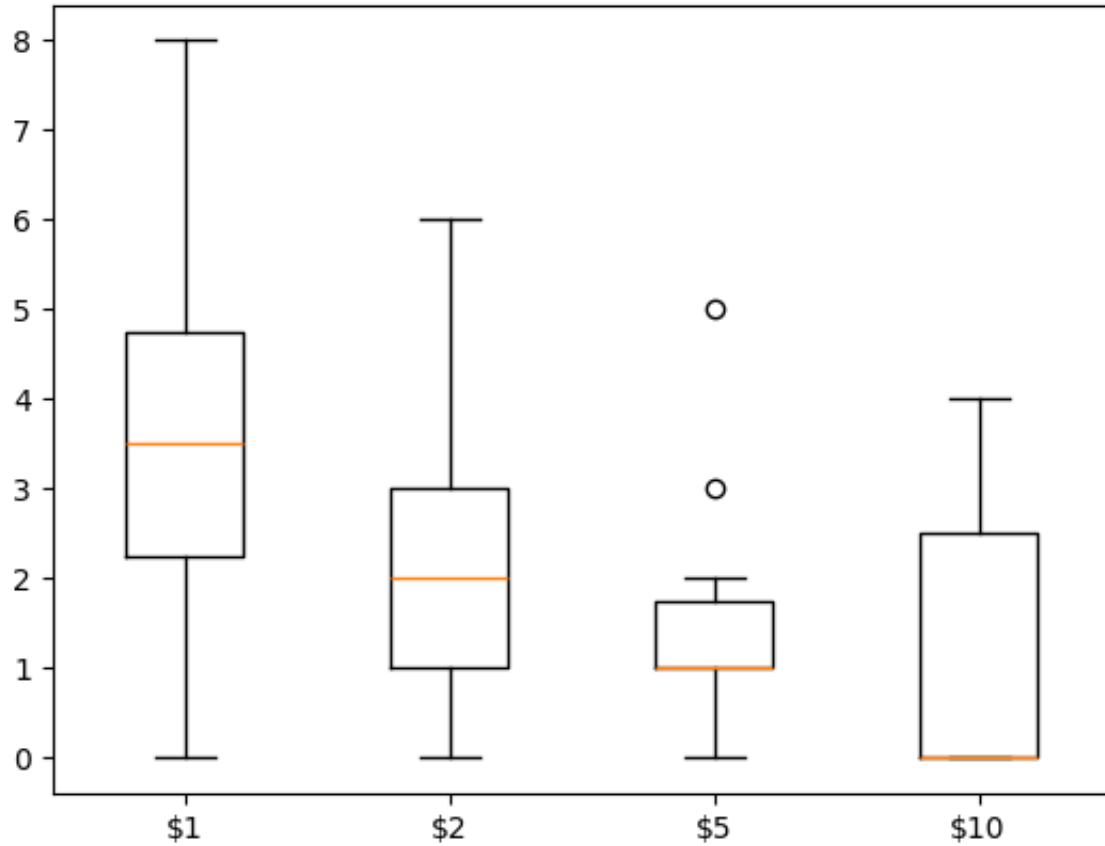
Supongamos que consultamos a los alumnos del curso para determinar el número de monedas en las 4 denominaciones fijadas de \$1 a \$10 pesos, excluyendo otras denominaciones.

```
# 1. Construimos una matriz de conteos para el número de monedas
# para cada denominación (columnas), de cada alumno (filas)
monedas = [
    [3, 2, 1, 0],
    [2, 3, 5, 4],
    [4, 1, 3, 4],
    [8, 0, 1, 0],
    [4, 0, 2, 0],
    [3, 1, 1, 3],
    [5, 2, 1, 0],
    [1, 3, 0, 1],
    [0, 4, 1, 0],
    [6, 6, 0, 0],
]

# 2. Separamos cada columna para tener el número de monedas
# para cada denominación
monedas1, monedas2, monedas5, monedas10 = zip(*monedas)
```

```
# 3. Graficamos las cajas estadísticas de cada denominación
import matplotlib.pyplot as pyplot

pyplot.boxplot([monedas1, monedas2, monedas5, monedas10],
                tick_labels=["$1", "$2", "$5", "$10"])
pyplot.show()
```



Las cajas estadísticas nos dan una idea del comportamiento entre los cuantiles de cada serie, pero, si queremos los datos podemos usar

```
import numpy

print("Moneda $1 peso")
print("-" * 15)
print("Min: ", numpy.min(monedas1))
print("Q1 : ", numpy.quantile(monedas1, 0.25))
print("Q2 : ", numpy.quantile(monedas1, 0.5))
print("Q3 : ", numpy.quantile(monedas1, 0.75))
print("Max: ", numpy.max(monedas1))
```

```
➡ Moneda $1 peso
-----
Min:  0
Q1 :  2.25
Q2 :  3.5
Q3 :  4.75
Max:  8
```

✓ Problema 3 - Ganancias de un producto

Fenómeno: Una tienda de abarrotes registró el monto total sobre las ventas de un producto y número de unidades vendidas, de las cuales tiene que calcular la ganancia sobre el precio establecido a principios del mes que considera el promedio del costo de adquisición del producto de las últimas 2 veces.

Planteamiento del problema: ¿Cómo podemos calcular la ganancia del producto para el mes actual respecto al promedio del costo de adquisición de las últimas dos veces?

Modelo matemático: Primero tenemos que multiplicar el número de unidades vendidas por el costo promedio asignado, luego calcularemos la diferencia entre el monto total vendido, menos el costo promedio del producto.

✓ Ejemplo: Ganancia relativa

Dado un monto total y un número de productos vendidos, así como los últimos dos precios de adquisición, calcularemos la ganancia relativa al promedio del costo de adquisición, esto tiene sentido si pensamos que el producto que se vende no modifica mucho su precio entre cada adquisición. Tomar el último precio podría reflejar una ganancia menor considerando que el producto pudo subir mucho de precio en la última adquisición, pero aún teníamos productos resagados de la penúltima adquisición.

```

# 1. Definimos el monto total y cantidad de productos vendidos en todo el mes
monto_total = 2300
unidades_vendidas = 19

# 2. Definimos los costos de adquisición del producto
# para la última y penúltima adquisición (cuánto nos costó cada unidad)
costo_1 = 75
costo_2 = 84

# 3. Determinamos el costo total fijado
# (costo de adquisición de cada producto)
costo_unidad = (costo_1 + costo_2) / 2

# 4. Calculamos el costo de adquisición por el número de productos vendidos
# esto significa cuánto se descontará en promedio de todas las unidades
# vendidas (lo que nos costaron todas las unidades pero de forma relativa)
costo_total = costo_unidad * unidades_vendidas

# 5. Calculamos la ganancia respecto al monto total
ganancia = monto_total - costo_total

# 6. Imprimimos un reporte
print("Reporte de Ganancia relativa del producto")
print("-" * 45)
print(f"Unidades vendidas : {unidades_vendidas:9}")
print(f"Total vendido      : ${monto_total:8.2f}")
print("-" * 45)
print(f"Costo promedio      : ${costo_unidad:8.2f}")
print(f"Costo total         : ${costo_total:8.2f}")
print("-" * 45)
print(f"Ganancia relativa : ${ganancia:8.2f}")
print(f"Ganancia (%)      : {100 * ganancia/monto_total:8.2f}%")

```

Reporte de Ganancia relativa del producto

```

-----
Unidades vendidas :          19
Total vendido      : $ 2300.00
-----
Costo promedio     : $   79.50
Costo total        : $ 1510.50
-----
Ganancia relativa : $   789.50
Ganancia (%)       :    34.33%

```


✓ Introducción a la programación con Python

Hasta ahora hemos visto 3 problemas del ámbito científico que podrían ser cercanos a nuestra realidad. Sin embargo, se ha mostrado mucho código que podría resultar extraño como leer chino.

Aunque intentemos leer el código y entender algunas palabras, es fundamental aprender las estructuras lógicas que guían el código para poder entender su funcionamiento y poder hacer adaptaciones o reutilizarlo en otros problemas.

Primero entenderemos cómo los lenguajes de programación se pueden aplicar en la vida cotidiana.

✓ Los lenguajes de programación y la vida cotidiana

Aunque no es muy evidente cómo un lenguaje de programación se presenta en nuestras actividades del día a día, están presentes por todos lados.

Los lenguajes de programación son abstracciones de instrucciones, comandos y sentencias que dictan cómo una máquina debería hacer operaciones para resolver un problema.

Pensemos en una máquina capaz de recibir cuatro tipos de instrucciones:

- A - Avanzar un paso la máquina
- B - girar la máquina 90° a la izquierda
- C - Retroceder un paso la máquina
- D - girar la máquina 90° a la derecha

La máquina puede recibir una cinta finita de instrucciones y relizar una a una de forma secuencial, por ejemplo, si escribimos una cinta con *ABABABA*, la máquina avanzará un paso, dará un giro de 90° a la izquierda, luego volverá a avanzar un paso, después volverá a girar a la izquierda y así hasta completar todas las instrucciones.

Entonces, si quisieramos guiar a la máquina hacia algún lugar, bastaría calcular si hay que avanzar, girar o retroceder. Cada que la máquina reciba una cinta de instrucciones ejecutará las operaciones asociadas.

✓ Ejemplo: Máquina que avanza y gira

Veamos una implementación de la máquina que avanza y gira capaz de ejecutar una cinta de instrucciones

```
# 1. Definimos las coordenadas iniciales de la máquina,
# para reportar su posición tras cada actualización
x, y = 0, 0

# 2. Definimos la dirección hacia donde apunta la máquina
# NORTE, SUR, ESTE u OESTE, por ejemplo, al avanzar
# si está viendo al NORTE, la coordenada y incrementará
d = "NORTE"

# 3. Creamos una memoria para recordar las coordenadas de la máquina
# en cada instrucción, inicializando la memoria en la primer coordenada
# y guardamos también la dirección en la memoria
memoria = [ (x, y, d) ]

# 4. Definimos una función que representa a la máquina
# y ejecuta una cinta de instrucciones, modificando
# las coordenadas y dirección donde se encuentra y apunta
def maquina(cinta, silencio = False):
    global x, y, d
    for i in cinta:
        if not silencio:
            print(f"INICIO: ({x}, {y}) [{d:5}] -> EJECUTA {i}")
        if i == "A":
            # Avanzar
            if d == "NORTE":
                y += 1
            elif d == "SUR":
                y -= 1
            elif d == "ESTE":
                x += 1
            elif d == "OESTE":
                x -= 1
        elif i == "C":
            # Retroceder
            if d == "NORTE":
                y -= 1
            elif d == "SUR":
                y += 1
            elif d == "ESTE":
                x -= 1
            elif d == "OESTE":
```

```

    x += 1
elif i == "B":
    # Girar izquierda
    if d == "NORTE":
        d = "OESTE"
    elif d == "SUR":
        d = "ESTE"
    elif d == "ESTE":
        d = "NORTE"
    elif d == "OESTE":
        d = "SUR"
elif i == "D":
    # Girar derecha
    if d == "NORTE":
        d = "ESTE"
    elif d == "SUR":
        d = "OESTE"
    elif d == "ESTE":
        d = "SUR"
    elif d == "OESTE":
        d = "NORTE"
else:
    if not silencio:
        print(">>> INSTRUCCIÓN NO VÁLIDA")
if not silencio:
    print(f"FIN    : ({x}, {y}) [{d:5}]")
    print("-" * 40)
memoria.append((x, y, d))

```

5. Ejecutamos la máquina con una cinta de instrucciones
maquina("ABCDEAABBCDDACDXDA")

```

⇌ INICIO: (0, 0) [NORTE] -> EJECUTA A
FIN    : (0, 1) [NORTE]
-----
INICIO: (0, 1) [NORTE] -> EJECUTA B
FIN    : (0, 1) [OESTE]
-----
INICIO: (0, 1) [OESTE] -> EJECUTA C
FIN    : (1, 1) [OESTE]
-----
INICIO: (1, 1) [OESTE] -> EJECUTA D
FIN    : (1, 1) [NORTE]
-----
INICIO: (1, 1) [NORTE] -> EJECUTA E
>>> INSTRUCCIÓN NO VÁLIDA
FIN    : (1, 1) [NORTE]
-----
INICIO: (1, 1) [NORTE] -> EJECUTA A
FIN    : (1, 2) [NORTE]

```

INICIO: (1, 2) [NORTE] -> EJECUTA A
FIN : (1, 3) [NORTE]

INICIO: (1, 3) [NORTE] -> EJECUTA B
FIN : (1, 3) [OESTE]

INICIO: (1, 3) [OESTE] -> EJECUTA B
FIN : (1, 3) [SUR]

INICIO: (1, 3) [SUR] -> EJECUTA C
FIN : (1, 4) [SUR]

INICIO: (1, 4) [SUR] -> EJECUTA D
FIN : (1, 4) [OESTE]

INICIO: (1, 4) [OESTE] -> EJECUTA D
FIN : (1, 4) [NORTE]

INICIO: (1, 4) [NORTE] -> EJECUTA A
FIN : (1, 5) [NORTE]

INICIO: (1, 5) [NORTE] -> EJECUTA C
FIN : (1, 4) [NORTE]

INICIO: (1, 4) [NORTE] -> EJECUTA D
FIN : (1, 4) [ESTE]

INICIO: (1, 4) [ESTE] -> EJECUTA X
>>> INSTRUCCIÓN NO VÁLIDA
FIN : (1, 4) [ESTE]

INICIO: (1, 4) [ESTE] -> EJECUTA D
FIN : (1, 4) [SUR]

INICIO: (1, 4) [SUR] -> EJECUTA A
FIN : (1, 3) [SUR]

Para visualizar mejor la máquina, podemos crear una matriz que reconstruya en las coordenadas cómo se movió la máquina según su memoria

```

import numpy

mat = numpy.zeros((5, 5))

x, y = 0, 0
d = "NORTE"
memoria = [ (x, y, d) ]

maquina("AAAADAAAADABCCCBCCDABADA", silencio = True)

i = 1
for x, y, d in memoria:
    mat[y, x] = i
    i += 1

print(mat)

```

```

→ [[ 1.  0.  0.  0.  0.]
   [ 2. 20. 22.  0.  0.]
   [ 3. 18. 24. 25.  0.]
   [ 4. 17. 15. 14. 13.]
   [ 6.  7.  8.  9. 11.]]

```

✓ Python en el ámbito científico

Quizás los lenguajes de programación no aparezcan en nuestro día a día, pero seguramente aparecerá el lenguaje natural que hablamos o escribimos.

Si recibimos un correo que diga:

Querido empleado, se le solicita que entregue un reporte de sus actividades desarrolladas durante el mes, para que concurse al bono del empleado más productivo. Se le adjunta en el correo una lista de las actividades que le fueron asignadas en el mes, por favor llene las casillas de las actividades que ya entregó y deje vacías las actividades que aún no completa o no se las ha enviado a su jefe inmediato (las que no ha reportado). Sin más, le deseo un buen día. Saludos cordiales.

Seguramente descargaremos el archivo adjunto, revisaremos las actividades que ya le hemos enviado a nuestro jefe inmediato y rellenaremos las casillas, luego guardaremos el archivo y responderemos el correo adjuntando el archivo modificado.

¿Dónde hay un lenguaje de programación oculto? Quizás no lo parezca, pero en el fondo de nuestro ser somos máquinas. Quizás máquinas biológicas, espirituales, racionales o alquímicas, el punto es que nuestros racionamientos y la forma en la que percibimos la realidad

y respondemos al contexto nos vuelven máquinas.

Para una máquina sintética como un robot o una computadora, es más difícil recibir y procesar las instrucciones, se necesita un lenguaje de programación que codifique las complejas instrucciones en estructuras con una sintaxis específica y compile o interprete cada instrucción para generar un código máquina capaz de ejecutar en su procesador.

Para un humano es más sencillo, simplemente recibe una instrucción en un lenguaje natural, aproxima una idea de lo que significa esa instrucción y planifica una solución o simplemente dice: "yo no entiendo nada".

Entender la sintaxis del lenguaje de programación, nos permitirá codificar instrucciones simples o complejas que resuelva la máquina. A una cinta de instrucciones la conoceremos como el programa o *script* y la máquina la entenderemos por el interprete.

Nosotros aprenderemos a usar Python, el cual es un lenguaje de programación sencillo, flexible y potente, con el cual podemos manipular datos, crear algoritmos inteligentes y construir reportes gráficos y de texto para resolver problemas y automatizar nuestro día a día.

Pero la pregunta es ¿Qué hace Python en el ámbito científico?

La ciencia moderna requiere datos y los datos requieren memoria. De aquí parte el problema de la Ciencia de Datos. Para resolver un problema en la actualidad, se necesita transformar el problema a datos.

Por ejemplo, ¿Cuál es la temperatura ideal que debe tener un reactor bioquímico para generar la máxima glucosa posible? "*Yo no entiendo nada*"

Un biorreactor puede ser entendido como un bote de metal en el que se vierten sustancias como agua, azúcar, levadura o lo que se le ocurra al experimentador y a partir de esto genere alguna sustancia nueva, por ejemplo, glucosa, almidones o sustratos obtenidos de las reacciones químicas.

Las reacciones químicas dentro del biorreactor requieren procedimientos específicos para que los compuestos logren formarse, como hornear un pastel, si aplicamos demasiada levadura, seguramente ocurrirá un caos, si la temperatura es muy baja seguramente el pastel nunca se horneará y si la temperatura es muy alta seguramente terminaremos pidiendo una pizza a domicilio.

Entonces, ¿Cuál es la temperatura ideal que debe tener un reactor bioquímico para generar la máxima glucosa posible? Esta pregunta nos hace pensar que podemos modificar la temperatura del biorreactor o mantenerla constante. Para poder responder esta pregunta hace falta que tengamos experimentos y datos.

¿Qué deberíamos medir en el biorreactor? Entrada y salida es lo básico en una medición, pero a veces no es suficiente, muestras continuas cada cierto tiempo es lo ideal, pero muchas veces no podremos hacerlo porque ¿Cómo medimos lo que pasa en el biorreactor sin abrirlo, ni detener el proceso?

Muchas veces, extraer los datos de experimentación será costo o casi imposible, pero, por eso nos pagan.

Debemos ser sensibles que el problema podrá ser resuelto a través de datos, y que los datos serán costos y difíciles de extraer. Por esto mismo, debemos pensar siempre en cómo extraer correctamente los datos, si estos nos servirán y si son suficientes para dar una conclusión o resolver un problema (responder una pregunta).

Generalmente, el fenómeno real explica una situación de cómo suceden las cosas que se observan (son los datos), el planteamiento del problema es una pregunta que se hace sobre el fenómeno observado (una respuesta que buscaremos resolver a través de los datos) y el modelo matemático consistirá en construir un proceso, algoritmo o máquina que nos dé la solución y podemos determinar que tan precisa es la solución y qué tan bien se ajusta a los datos nuestro modelo.

El enfoque que daremos en este curso, será el de resolver problemas del ámbito científico, a través de la ciencia de datos, usando Python.

✓ Ejemplo: Antes de entender Python

Aunque a estas alturas ya debes estar ansioso por aprender Python por alguna razón "porque resuelve cosas", es importante motivar aun más las aplicaciones de Python en el ámbito científico.

En este ejemplo, vamos a tomar un problema clásico y aplicar lo visto en los 3 problemas científicos de esta sesión:

1. Optimizar de forma aleatoria una solución
2. Analizar el comportamiento en grupos de variables aleatorias
3. Generar un reporte sobre el problema planteado

Primero vamos a tomar un conjunto de datos (*dataset*) de la supervivencia en el hundimiento del Titanic.

Luego, vamos a plantear el problema de comparar la supervivencia del titanic entre hombres y mujeres.

Finalmente, vamos a crear un modelo matemático, para calcular la probabilidad de supervivencia general, la probabilidad de supervivencia dado que se es hombre o mujer y la probabilidad de ser mujer u hombre dado que se ha sobrevivido o no.

Paso 1: Cargar los datos de supervivencia del titanic

Aunque no hemos aprendido Python y hemos estado usando librerías de Ciencia de datos con Python, continuaremos para ver hasta donde llegamos y qué tanto entendemos


```
# Cargamos los datos del CSV con Pandas desde Github
import pandas

# URL donde se ubica el dataset
url = "https://github.com/dragonnomada" + \
      "/ipn-cic-pycien-abril-2025" + \
      "/raw/main" + \
      "/datasets/titanic.csv"




print(url)

# Carga del DataFrame de Pandas
titanic = pandas.read_csv(url)

# Datos de interés
data = titanic[["Sex", "Survived"]]

data
```

 <https://github.com/dragonnomada/ipn-cic-pycien-abril-2025/raw/main/datasets/ti>

	Sex	Survived	
0	male	0	
1	female	1	
2	female	1	
3	female	1	
4	male	0	
...	
886	male	0	
887	female	1	
888	female	0	
889	male	1	
890	male	0	

891 rows x 2 columns

Próximos pasos:

 [Ver gráficos recomendados](#)

[New interactive sheet](#)

Paso 2 - Analizar los datos

Para analizar cómo se comporta la supervivencia por sexo, generaremos diferentes visualizaciones y reportes.

Primero segmentamos los datos para mujeres y hombres, y visualizamos su supervivencia mediante una gráfica de violín, la cual muestra dónde están las concentraciones o densidad de los datos de forma simétrica

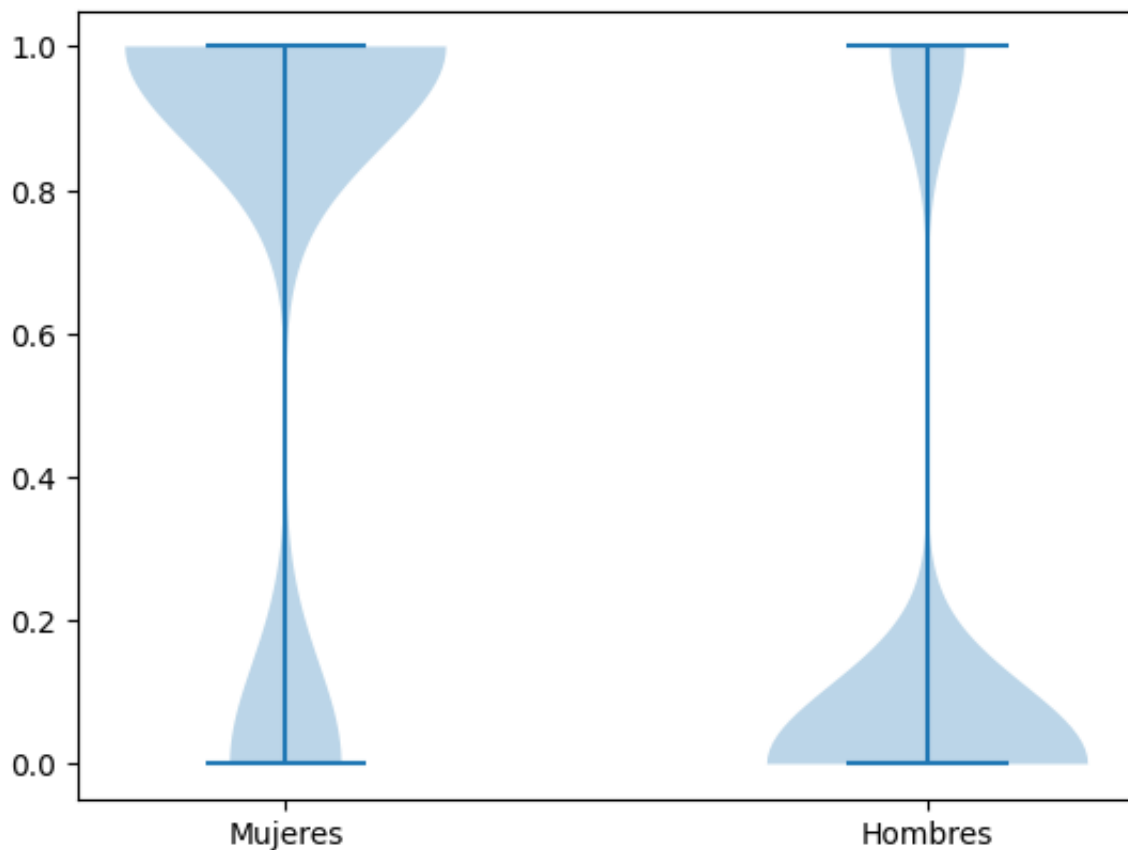
```
# 1. Proporción de la supervivencia por sexo
```

```
data_mujeres = data[ data["Sex"] == "female" ]  
data_hombres = data[ data["Sex"] == "male"    ]
```

```
import matplotlib.pyplot as pyplot
```

```
figure, axis = pyplot.subplots()
```

```
axis.violinplot([data_mujeres["Survived"], data_hombres["Survived"]])  
axis.set_xticks([1, 2])  
axis.set_xticklabels(["Mujeres", "Hombres"])  
pyplot.show()
```



En la gráfica se ve cómo la supervivencia de mujeres es mayor que la no-supervivencia, contrario a los hombres.

Obtenemos las probabilidades generales de sobrevivir y no-sobrevivir para hombres y mujeres y las reportamos.

```

total_mujeres = data_mujeres["Survived"].count()
total_mujeres_sobreviven = data_mujeres["Survived"].sum()
total_mujeres_no_sobreviven = total_mujeres - data_mujeres["Survived"].sum()

total_hombres = data_hombres["Survived"].count()
total_hombres_sobreviven = data_hombres["Survived"].sum()
total_hombres_no_sobreviven = total_hombres - data_hombres["Survived"].sum()

total_sobreviven = total_mujeres_sobreviven + total_hombres_sobreviven
total_no_sobreviven = total_mujeres_no_sobreviven + total_hombres_no_sobreviven

total = total_sobreviven + total_no_sobreviven

print("Conteos de supervivencia de hombres y mujeres")
print("=" * 50)
print("      Sobrevive      No-Sobrevive      Total")
print("-" * 50)
print(f"Mujeres      {total_mujeres_sobreviven:6} {total_mujeres_no_sobreviven:15} {")
print(f"Hombres      {total_hombres_sobreviven:6} {total_hombres_no_sobreviven:15} {")
print("-" * 50)
print(f"Total      {total_sobreviven:6} {total_no_sobreviven:15} {total:14}")

```



Conteos de supervivencia de hombres y mujeres

```

=====
      Sobrevive      No-Sobrevive      Total
-----
Mujeres      233      81      314
Hombres      109      468      577
-----
Total      342      549      891

```

Ahora sabemos que de 891 personas, 314 eran mujeres y 577 eran hombres, además 233 mujeres sobrevivieron y 81 no, mientras que 109 hombres sobrevivieron y 468 no lo hicieron, es decir, que 342 personas lograron sobrevivir, mientras que 549 no lo lograron.

Pero, ¿Cómo es la probabilidad de supervivencia?

Ahora calcularemos las probabilidades de supervivencia y las reportaremos

```

# Probabilidad de ser mujer - P(M)
pM = total_mujeres / total
# Probabilidad de ser mujer y sobrevivir - P(M,S)
pMS = total_mujeres_sobreviven / total
# Probabilidad de ser mujer y no-sobrevivir - P(M,X)
pMX = total_mujeres_no_sobreviven / total

# Probabilidad de ser hombre - P(H)
pH = total_hombres / total
# Probabilidad de ser hombre y sobrevivir - P(H,S)
pHS = total_hombres_sobreviven / total
# Probabilidad de ser hombre y no-sobrevivir - P(H,X)
pHX = total_hombres_no_sobreviven / total

# Probabilidad de sobrevivir - P(S)
pS = total_sobreviven / total
# Probabilidad de no-sobrevivir - P(X)
pX = total_no_sobreviven / total

print("Probabilidad conjunta de supervivencia")
print("=" * 50)
print("          Sobrevive          No-Sobrevive          Total")
print("-" * 50)
print(f"Mujeres      {100 * pMS:6.2f}% {100 * pMX:15.2f}% {100 * pM:13.2f}%")
print(f"Hombres      {100 * pHS:6.2f}% {100 * pHX:15.2f}% {100 * pH:13.2f}%")
print("-" * 50)
print(f"Total         {100 * pS:6.2f}% {100 * pX:15.2f}% {100:13.2f}%")

```



Probabilidad conjunta de supervivencia

	Sobrevive	No-Sobrevive	Total
Mujeres	26.15%	9.09%	35.24%
Hombres	12.23%	52.53%	64.76%
Total	38.38%	61.62%	100.00%

Ahora podemos apreciar mejor los datos en probabilidades, por ejemplo, la probabilidad de que una persona haya sobrevivido si era mujer es de 26.15%.

Pero, ¿Cuáles son las probabilidades reales de supervivencia y no-supervivencia?

Ahora calcularemos las probabilidades condicionales usando la probabilidad conjunta y el Teorema de Bayes.

```
print("Probabilidad de Sobrevivir dado que la persona fue Mujer")
```

```
print(f"P(S | M) = {100 * pMS / pM:.2f}%")

print("=" * 60)

print("Probabilidad de Sobrevivir dado que la persona fue Hombre")
print(f"P(S | H) = {100 * pHS / pH:.2f}%")

print("=" * 60)

print("Probabilidad de No-Sobrevivir dado que la persona fue Mujer")
print(f"P(X | M) = {100 * pMX / pM:.2f}%")

print("=" * 60)

print("Probabilidad de No-Sobrevivir dado que la persona fue Hombre")
print(f"P(X | H) = {100 * pHX / pH:.2f}%")

print("=" * 60)

print("Probabilidad de ser Mujer dado que la persona Sobrevivió")
print(f"P(M | S) = {100 * pMS / pS:.2f}%")

print("=" * 60)

print("Probabilidad de ser Hombre dado que la persona Sobrevivió")
print(f"P(H | S) = {100 * pHS / pS:.2f}%")

print("=" * 60)

print("Probabilidad de ser Mujer dado que la persona No-Sobrevivió")
print(f"P(M | X) = {100 * pMX / pX:.2f}%")

print("=" * 60)

print("Probabilidad de ser Hombre dado que la persona No-Sobrevivió")
print(f"P(H | X) = {100 * pHX / pX:.2f}%")
```



Probabilidad de Sobrevivir dado que la persona fue Mujer

$$P(S | M) = 74.20\%$$

Probabilidad de Sobrevivir dado que la persona fue Hombre

$$P(S | H) = 18.89\%$$

Probabilidad de No-Sobrevivir dado que la persona fue Mujer

$$P(X | M) = 25.80\%$$

Probabilidad de No-Sobrevivir dado que la persona fue Hombre

$$P(X | H) = 81.11\%$$

Probabilidad de ser Mujer dado que la persona Sobrevivió

$$P(M | S) = 68.13\%$$

Probabilidad de ser Hombre dado que la persona Sobrevivió

$$P(H | S) = 31.87\%$$

Probabilidad de ser Mujer dado que la persona No-Sobrevivió

$$P(M | X) = 14.75\%$$

Probabilidad de ser Hombre dado que la persona No-Sobrevivió

$$P(H | X) = 85.25\%$$

Estas probabilidades reflejan la probabilidad real de supervivencia entre hombres y mujeres, por ejemplo, la probabilidad de que una persona que sobrevivió fuera mujer es del 68.13%, lo que indica que casi 7 de cada 10 sobrevivientes fueron mujeres, quienes tuvieron prioridad, mientras que la probabilidad de que una persona que no sobrevivió fuera hombre es del 85.25%, lo que indica que aproximadamente 17 de cada 20 personas que no sobrevivieron fueron hombres.

Además de los supervivientes, solo el 20% fueron hombres.

Ahora podemos graficar las probabilidades comparativas usando gráficas de pastel.

```

figure, axis = pyplot.subplots(2, 2)

axis[0, 0].pie([pM, pH], labels=["Mujeres", "Hombres"],
               colors=["deeppink", "royalblue"], autopct="%.1f%%")
axis[0, 0].set_xlabel("Proporción del Género")

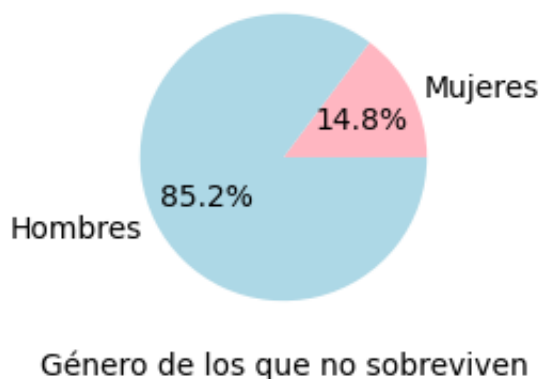
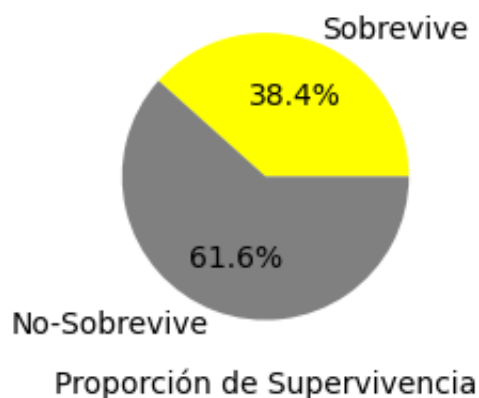
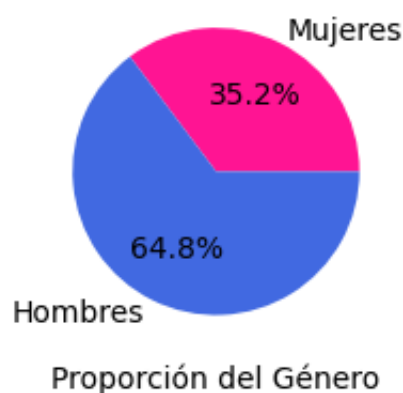
axis[0, 1].pie([pS, pX], labels=["Sobrevive", "No-Sobrevive"],
               colors=["yellow", "gray"], autopct="%.1f%%")
axis[0, 1].set_xlabel("Proporción de Supervivencia")

axis[1, 0].pie([pMS / pS, pHX / pS], labels=["Mujeres", "Hombres"],
               colors=["hotpink", "cornflowerblue"], autopct="%.1f%%")
axis[1, 0].set_xlabel("Género de los sobrevivientes")

axis[1, 1].pie([pMX / pX, pHX / pX], labels=["Mujeres", "Hombres"],
               colors=["lightpink", "lightblue"], autopct="%.1f%%")
axis[1, 1].set_xlabel("Género de los que no sobreviven")

pyplot.show()

```



Ahora podemos entender las probabilidades de ser hombre y mujer, así como de sobrevivir y no hacerlo.

Pero, ¿Podemos predecir si un hombre o mujer sobrevivirá?

Aunque ya sabemos las probabilidades reales, muchas veces calcular estas probabilidades serán más difíciles que entrenar un modelo que prediga las probabilidades.

Como ejemplo final, construiremos un predictor sencillo que intente determinar si una persona es capaz o no de sobrevivir solo sabiendo el género.

El modelo de predicción que usaremos es un regresor logístico que ajustará dos pesos desconocidos a la función logística para intentar aprender si la persona sobrevive o no lo hace.

El modelo de la función logísitica es:

$$\phi(x|\beta_0, \beta_1) = \frac{1}{1+e^{-(\beta_0+\beta_1x)}}$$

donde β_0, β_1 son desconocidos y tenemos que encontrarlos de alguna forma.

Usaremos la optimización aleatoria para intentar encontrar β_0, β_1 que generen el menor error entre la supervivencia y lo que produce $\phi(x|\beta_0, \beta_1)$ donde x será 1 si es hombre y 0 si es mujer.

```

# 1. Establecemos los pesos iniciales
beta_0 = 0
beta_1 = 0

# 2. Definimos la función logística dada x, beta_0 y beta_1
import math
def logistic(x, beta_0, beta_1):
    return 1 / (1 + math.exp(-(beta_0 + beta_1 * x)))

# 3. Definimos la función que mide el error entre la función logística
# y los datos de supervivencia
def error_supervivencia(beta_0, beta_1):
    e = 0
    for i, genero, supervivencia in data.itertuples():
        x = { "male": 1, "female": 0 }[genero]
        y = supervivencia
        yp = logistic(x, beta_0, beta_1)
        e = e + (y - yp) ** 2
    return e

# 4. Medimos el error inicial para la beta_0 y beta_1 actuales
print(f"Error ({beta_0}, {beta_1}): {error_supervivencia(beta_0, beta_1)}")

```

➡ Error (0, 0): 222.75

Ahora realizamos la optimización de las β_0 y β_1 para calcular las mejores durante 5,000 épocas

```
# 1. Durante 5,000 épocas modificamos aleatoriamente los pesos beta_0 y beta_1
# y determinamos si son mejores que los anteriores, para actualizarlos
import numpy
for epoca in range(5000):
    # Proponemos una beta_0, beta_1 modificada
    beta_0_propuesta = numpy.random.normal(0, 100)
    beta_1_propuesta = numpy.random.normal(0, 100)

    # Comparamos si error de la beta_0, beta_1 actuales
    # es mayor al error de las beta_0, beta_1 propuestas
    e_actual = error_supervivencia(beta_0, beta_1)
    e_propuesta = error_supervivencia(beta_0_propuesta, beta_1_propuesta)

    if e_propuesta < e_actual:
        beta_0 = beta_0_propuesta
        beta_1 = beta_1_propuesta

# 2. Medimos el error inicial para la beta_0 y beta_1 optimizadas
print(f"Error ({beta_0}, {beta_1}): {error_supervivencia(beta_0, beta_1)}")
```

➡ Error (1.453992981104975, -4.153916775159738): 159.14097746106174

En este punto hemos encontrado la mejores β_0, β_1 que al aplicar la regresión logística predicen un dato.

Ahora podemos calcular la diferencia entre las predicciones de supervivencia y las reales para cada género

```
print(f"Supervivencia de mujeres: {100 * logistic(0, beta_0, beta_1):.2f}%")
print(f"Supervivencia de hombres: {100 * logistic(1, beta_0, beta_1):.2f}%")
```

➡ Supervivencia de mujeres: 81.06%
Supervivencia de hombres: 6.30%

Los valores reales deberían ser 74.20% y 18.89%.

¿Queda cerca lo lejos el regresor logístico?

Conociendo Python: Sintaxis, variables e impresión

Ha llegado el momento de aprender a utilizar Python, ya vimos demasiado código en chino como para seguir progresando en problemas científicos, sin entender cómo se resolvieron a través de Python.

Primero daremos una introducción a la sintaxis de Python, luego al manejo de variables y finalmente a la impresión y generación de reportes con formato.

✓ Sintaxis

La sintaxis de Python es muy clara, pero abstracta.

Primero partamos de dos componentes básicos del lenguaje:

- **Sentencia** - Es una línea de código que genera una instrucción directa mediante una expresión, por ejemplo, asignar el valor de una variable o ejecutar una función, las cuales son las dos sentencias principales y mayormente utilizadas
- **Bloque de sentencias** - Un bloque de sentencias es una anidación de sentencias guiada por una estructura de control o la declaración de una función. En Python, una anidación siempre debe marcarse por una indentación, la cual es un número fijo de caracteres espacios o tabuladores que indicarán el desplazamiento a la derecha en las sentencias que pertenecen al bloque. Lo común es usar un tabulador, dos espacios o cuatro espacios, pero el código debe ser consistente, es decir, si se usa un tabulador, en todo el código la indentación siempre debe ser de un tabulador por cada nivel anidado.

Esto suena complejo, pero es más sencillo con una esquematización.

Por ejemplo, para un bloque simple tenemos:

```
BLOQUE:
    SENTENCIA_1
    SENTENCIA_2
    ...
    SENTENCIA_N
```

Y para un bloque anidado tenemos:

```
BLOQUE_PRINCIPAL:
    SENTENCIA_P1
```

```
SENTETANCIA_P2
...

BLOQUE_ANIDADO:
    SENTENCIA_A1
    SENTENCIA_A2
    ...
    SENTENCIA_AM

...
SENTETANCIA_PN
```

Intenta definir el esquema para un tercer bloque anidado, dentro del bloque anidado.

Veamos el bloque simple con un ejemplo real:

```
edad = input("Dame tu edad:")

if edad >= 18:
    print("Eres mayor de edad")
    print("Puedes pasar :D")
```

Intenta reproducir el código y ejecutarlo en una celda.

Ahora veamos el bloque anidado con un ejemplo real:

```
pares = []

for n in range(1, 101):
    esPar = n % 2 == 0
    if esPar:
        pares.append(n)

print(pares)
```

Analiza cada línea de código y determina en qué nivel se encuentra:

- Nivel 0 - Es parte del programa principal
- Nivel 1 - Pertenece al bloque inmediato (for)
- Nivel 2 - Pertenece al bloque anidado (if)

Ahora que sabemos la sintaxis general de Python, empecemos a construir programas.

Pero antes, debemos aprender el uso de variables e impresiones.

✓ Sintaxis de python basada en ejemplos

Importar la librería math, random y numpy

```
import math
import random
import numpy
```

Usar las funciones dentro de las librerías

```
math.cos(math.pi / 3)
random.choice(["manzana", "pera", "kiwi"])
numpy.zeros((3, 5))
```

Declarar y asignar una variable con diferentes valores

```
a = 123
b = True
c = "Ana"
d = 123.456
```

Crear una lista con elementos

```
edades = [18, 23, 45, 32, 64]
```

Recorrer una lista de elementos

```
for edad in edades:
    print(f"La edad es: {edad}")
```

Definir una función suma que recibe dos parámetros

```
def suma(a, b):
    return a + b
```

Llamar a la función suma con diferentes parámetros y guardar el resultado en variables

```
s1 = suma(100, 200)
s2 = suma(23, 9)
s3 = suma(198.57, 325.76)
```

Definir una función distancia que calcula la distancia entre los puntos (x_1, y_1) y (x_2, y_2)

```
import math

def distancia(x1, y1, x2, y2):
    dx = x1 - x2
    dy = y1 - y2

    d2 = dx ** 2 + dy ** 2

    return math.sqrt(d2)
```

Llamar a la función distancia para distintos parámetros y almacenar el resultado

```
d1 = distancia(1, 1, 2, 2)
d2 = distancia(0, 0, 5, 3)
d3 = distancia(-2, 3, 5, -11)
```

Transcribe sin copiar y pegar cada ejemplo y ejecútalo.

✓ Variables

En el apartado anterior de sintaxis hemos visto que $a = 123$ asigna una variable llamada a al valor de 123, pero no es suficiente conocer la sintaxis para manejar variables.

Las variables son memorias nombradas que sirven para almacenar valores.

Por ejemplo, al crear una variable llamada *edad*, python reservará una memoria para almacenar un valor, sin importar lo grande o pequeño que sea. Cuando no estamos seguros de qué almacenar en una variable podemos usar `mi_variable = None`.

A través de las variables podremos acceder al valor almacenado en esa variable. No todos los nombres que se nos ocurran serán válidos, el nombre de la variable es muy restrictivo, pero en general podemos usar palabras sin acentos ni espacios, ni caracteres especiales. También podremos usar guines bajos (`_`) y números, pero nunca podremos empezar por un número el nombre de la variable, ya que pensará que es un número y no una variable.

Hay cinco valores básicos que podrá almacenar una variable:

- **Números enteros** - que se construyen de dígitos sin punto decimal
- **Números decimales** - que se construyen de dígitos con un punto decimal
- **Valores lógicos** - que representan verdadero o falso y se construyen por las palabras `True` y `False`.
- **Textos** - que representan cadenas de caracteres libres que contienen texto, y se construyen mediante caracteres o emoticones encerrados entre comillas dobles o simples (como la cadena de texto `"abc 🍕"` o la cadena `'Ho la mundo 🌐'`).
- **None** - que representa un valor vacío y se construye con la palabra `None`.

Las variables también pueden almacenar listas, tuplas, diccionarios, arreglos, series, dataframes y demás objetos complejos que en el fondo sólo tendrán los cinco valores básicos.

✓ Ejemplo: Variables sobre un producto

Veamos las variables que necesitaríamos tener para representar los datos de un producto


```
nombre = "Tenis Pumba"
marca = 'Adidas'

precio_adquisicion = 450.80
precio_venta = 650.51

existencias = 234

modelo_hombre = True
modelo_mujer = False

color_principal = "#ffffff"
color_secundario = "#ff0000"
color_acento = "#00ff00"

valoracion = None
```

Es simple, no necesitamos más tipos de datos para procesar la información, basta tener textos, números y valores lógicos.

✓ Impresión

Algo fundamental para manejar variables y programas de Python en general es la impresión.

Para formar buenos reportes impresos, debemos aprender el formato moderno de impresión, es cual nos permite insertar variables o cálculos dentro de una cadena de texto a la que llamaremos **Texto con Formato** y podremos imprimirla o guardarla en un archivo.

Primero veamos que un texto con formato:

- **Texto** - Le llamaremos así a cualquier cadena de texto que ponemos entre comillas (simples o dobles), por ejemplo, `"Hola tú"`
- **Texto con Formato** - Le llamaremos así a las cadenas de texto que ponemos entre comillas (simples o dobles), pero que tienen una `f` inicial pegada al principio, esta `f` indicará que se trata de un formato o *texto con formato*. A estos textos les podremos inyectar variables, por ejemplo, `f"Hola {nombre}"` o `f'La distancia es: {d} del punto ({x1}, {y1}) al punto ({x2}, {y2})'`.

Dentro de los formatos al inyectar una variable, se puede especificar el número de espacios requeridos, por ejemplo, `"Producto: {nombre:20}"`, inyectará el valor de la variable `nombre`, pero garantizará al menos 20 caracteres, por ejemplo, si `nombre = "Coca Cola"`, se rellenará el texto a `"Producto: Coca Cola"`.

Podemos usar modificadores para que el contenido inyectado se alinee a la izquierda, derecha o enmedio:

- `{x:>20}` - inyecta `x` y rellena a 20 caracteres con espacios, el valor queda justificado a la derecha.
- `{x:<20}` - inyecta `x` y rellena a 20 caracteres con espacios, el valor queda justificado a la izquierda.
- `{x:^20}` - inyecta `x` y rellena a 20 caracteres con espacios, el valor centrado.

También podemos especificar el número de puntos decimales si el valor inyectado es un número, por ejemplo, `{x:20.3f}` inyecta el valor de `x`, garantiza 20 caracteres y limita el valor a 3 decimales. Observa que `.<k>f` especifica `k` decimales y la `f` dice que es un número que puede tener decimales.

Finalmente, un texto puede ser repetido al multiplicarlo, por ejemplo, `"Hola" * 3` será equivalente al texto `"HolaHolaHola"`. Esto es útil para formar separadores en los reportes, por ejemplo, `"-" * 8` que creará la cadena `"-----"`.

✓ Ejemplo: Reporte simple

Veamos como podemos construir un reporte simple usando el valor de las variables del producto y la función `print(<texto>)` que imprime en la pantalla el texto que se le pasa

```

print("Reporte del producto")
print("=" * 40)

print()

print(f"Producto: {nombre} / {marca}")
print("-" * 40)
print(f"Precio adquisición : $ {precio_adquisicion:9.2f}")
print(f"Precio al público : $ {precio_venta:9.2f}")
print("-" * 40)
print(f"Ganancia por unidad : $ {precio_venta - precio_adquisicion:9.2f}")
print("-" * 40)

print()

print("Inventario")
print("•" * 40)
print(f"Existencias : {existencias:12}")
print("•" * 40)
print(f"Valor retenido : $ {precio_adquisicion * existencias:9.2f}")
print(f"Valor esperado : $ {precio_venta * existencias:9.2f}")
print("•" * 40)
print(f"Ganancias esperadas : $ {(precio_venta - precio_adquisicion) * existencias}")
print("•" * 40)

```



Reporte del producto

=====

Producto: Tenis Pumba / Adidas

Precio adquisición	:	\$	450.80
--------------------	---	----	--------

Precio al público	:	\$	650.51
-------------------	---	----	--------

Ganancia por unidad	:	\$	199.71
---------------------	---	----	--------

Inventario

.....

Existencias	:	234
-------------	---	-----

.....

Valor retenido	:	\$	105487.20
----------------	---	----	-----------

Valor esperado	:	\$	152219.34
----------------	---	----	-----------

.....

Ganancias esperadas	:	\$	46732.14
---------------------	---	----	----------

.....

Al generar buenos reportes de texto, podremos explicar la información de nuestros datos y los resultados que derivemos de forma clara y precisa.

Pero, si queremos almacenar el reporte, podemos almacenar todo el reporte en una sola cadena de texto y luego mandarla a guardar a un archivo

```
reporte = f"""\n
Reporte del producto\n
{'=' * 40}\n\n
Producto: {nombre} / {marca}\n
{'-' * 40}\n
Precio adquisición   : $ {precio_adquisicion:9.2f}\n
Precio al público    : $ {precio_venta:9.2f}\n
{'-' * 40}\n
Ganancia por unidad : $ {precio_venta - precio_adquisicion:9.2f}\n
{'-' * 40}\n\n
Inventario\n
{'•' * 40}\n
Existencias          : {existencias:12}\n
{'•' * 40}\n
Valor retenido       : $ {precio_adquisicion * existencias:9.2f}\n
Valor esperado       : $ {precio_venta * existencias:9.2f}\n
{'•' * 40}\n
Ganancias esperadas : $ {(precio_venta - precio_adquisicion) * existencias:9.2f}\n
{'•' * 40}\n
"""\n\n
archivo = open("reporte1.txt", "w")\n\n
archivo.write(reporte.strip())\n\n
archivo.close()\n\n
print(reporte)
```



Reporte del producto

=====

Producto: Tennis Pumba / Adidas

Precio adquisición : \$ 450.80

Precio al público : \$ 650.51

Ganancia por unidad : \$ 199.71

Inventario

.....

Existencias : 234

.....

Valor retenido : \$ 105487.20

Valor esperado : \$ 152219.34

.....

Ganancias esperadas : \$ 46732.14

.....

Observa que hemos definido un texto en múltiples líneas usando `""" ... """`, además hemos abierto un archivo con la función `open(<ruta>, <modo>)`, donde el modo `w` significa escritura y luego hemos escrito el reporte en el archivo, pero usando `reporte.strip()` que quita el salto de línea inicial y final. Finalmente, hemos cerrado el archivo y ahora podemos consultarlo en los archivos de la libreta.