In [12]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Cargar datos
df = pd.read_csv('Valhalla23.csv')
celsius = df['Celsius']
valks = df['Valks']

# Dividir los datos en entrenamiento y prueba (80% entrenamiento, 20%
prueba)
train_df, test_df = train_test_split(df, test_size=0.2)

# Separar características y etiquetas
train_Celsius = train_df['Celsius']
train_Valks = train_df['Valks']
test_Celsius = test_df['Celsius']
test_Valks = test_df['Valks']
```

In [13]:

```
# Sacar las tablas de celsius y valks
Celsius = df.iloc[:, 0]
print(Celsius)
Valks = df.iloc[:, 1]
print(Valks)

# Asignar los valores de las thetas
theta0= 1
theta1 = 1
```

```
print(theta0, theta1)


# Cargar el valor del learning rate (alpha)
alpha = 0.0005
print(alpha)
```

```
0     61.4720
1     70.5790
2     -7.3013
3     71.3380
4     43.2360
       ...
95    -7.0094
96    36.8820
97    26.9390
98   -18.8100
99    13.7120
Name: Celsius, Length: 100, dtype: float64
0    -139.7400
1    -156.6000
2      73.2690
3    -165.4200
4     -75.8350
        ...
95     69.6320
96    -71.2400
97    -34.2550
98    106.4300
99      9.1011
Name: Valks, Length: 100, dtype: float64
1 1
0.0005
```

In [14]:

```
# Crear función lambda para la función de hipótesis

h0 = lambda theta0, theta1, x : theta0 + theta1 * x

# Número de muestras en el conjunto de entrenamiento
n = len(train_Celsius)
```

In [16]:

```
# Calcular delta para theta0
delta = [h0(theta0, theta1, train_Celsius.iloc[i]) - train_Valks.iloc[i]
for i in range(len(train_Celsius))]
print(delta)

# Calcular delta para theta1
deltax = [delta[i] * train_Celsius.iloc[i] for i in range(n)]
print(deltax)

# Calcular sumatorias y promedio
sumdelta = sum(delta)
sumdeltax = sum(deltax)
print(sumdelta)
print(sumdeltax)
```

```
[72.566, 26.486200000000004, 166.263, 187.422, -25.233200000000004,
-71.18140000000001, 40.56, 109.12199999999999, 260.949, 196.498, 250.049,
131.628, -14.098099999999999, 126.58, 166.287, -92.68299999999999, 86.457,
243.869, 186.076, 36.058, 171.50900000000001, -88.43, 90.79599999999999,
-75.6414, 23.4059, 251.86899999999997, 212.573, 190.154, 80.97, 60.536,
154.124, 237.75799999999998, 223.154, 247.67200000000003, 259.571, 122.379,
-15.8302, 210.73, 142.895, 99.822, 263.064, 5.610899999999999, -81.3472,
```

```
172.27, -16.4775, -74.21260000000001, 49.067, -68.1766, -25.6885, 62.194,
55.361000000000004, 21.036099999999998, 102.043, 146.716, -124.24000000000001,
6.2745999999999995, 14.626199999999999, -17.7557, 244.161, -38.7778, 159.498,
255.289, -3.433, -60.3673, 228.17899999999997, 133.617, -118.889, -79.5703,
230.719, 184.433, -105.141, 252.11599999999999, 177.81, 105.106, 128.131,
156.217, -26.786800000000003, 203.518, 90.991, 144.228]
[2165.079176, 477.94347900000014, 9030.242319, 10599.088944,
-145.11108656000002, 413.66358796000003, 968.32944, 4024.637603999999,
19959.728061, 12070.479143999999, 19268.525891, 5874.689267999999,
-121.01668058999999, 5765.46584, 9630.844179, 949.630018, 3001.959954,
18516.729301, 11601.094296, 799.6222079999999, 9513.432721000001,
909.6794100000002, 3149.531648, 530.20082916, 449.93161569999995,
19129.198680999998, 13409.742559, 10605.649196, 2477.35812, 1727.576368,
7378.532376, 16961.180203999997, 15972.024395999999, 18580.848784,
19662.762821000004, 5091.455916, -124.26390396, 14559.3357, 7070.873285,
3671.652804, 19986.024336000002, 76.93666079999998, 658.92858944,
10253.510400000001, -125.27019374999999, 455.48725376000004,
1205.0364530000002, 345.69626795999994, -141.53079075, 1675.444166,
1512.9607690000003, 381.93143159999994, 3931.4106610000003, 7473.126176000001,
2336.9544, 88.0891094, 219.36374759999998, -136.58217111000002, 17921.661561,
-92.33769736, 7960.226184, 18309.071791, -40.200430000000004, 173.93630149,
16104.645640999997, 6080.909669999999, 1953.2273809999997, 580.9666313900001,
16934.543881, 10212.608509, 1535.584305, 18385.811416, 9907.5732, 4049.313756,
5839.442194, 8611.774559000001, -145.40410776000002, 12216.778504,
3009.9822799999997, 6918.617160000001]
7435.103300000001
522204.57842842
```

In [17]:

```
# Actualizar theta0
theta0 = theta0 - alpha * sumdelta/n
print(theta0)

# Actualizar theta1
theta1 = theta1 - alpha * sumdeltax/n
print(theta1)
```

```
0.953530604375
-2.2637786151776256
```

In [20]:

```
# Hacer 100 Iteraciones

for i in range(100):
  # No need to redefine h0 in each iteration
  # h0 = lambda theta0, theta1, x : theta0 + theta1 * x

  # Calculate the number of samples from the training data
  n = len(train_Celsius)  # Use len(train_Celsius) instead of len(df)

  # Calculate delta for theta0 for each sample
  delta = [h0(theta0, theta1, train_Celsius.iloc[i]) - train_Valks.iloc[i]
for i in range(n)]

  # Calculate delta for theta1 for each sample
  deltax = [delta[i] * train_Celsius.iloc[i] for i in range(n)]

  # Calculate sums and averages
  sumdelta = sum(delta)
  sumdeltax = sum(deltax)

  # Update theta0
  theta0 = theta0 - alpha * sumdelta/n

  # Update theta1
  theta1 = theta1 - alpha * sumdeltax/n
```

In [21]:

```
# Usar el modelo entrenado para predecir los valores en el conjunto de
prueba
test_Valks2Estimado = [h0(theta0, theta1, x) for x in test_Celsius]

# Calcular la función de costo en el conjunto de prueba
test_n = len(test_Celsius)
test_Costo = sum((test_Valks2Estimado[i] - test_Valks.iloc[i])**2 for i in
range(test_n)) / (2 * test_n)
print(test_Costo)
```

```
785.1200603205673
```

In [22]:

```
# Graficar datos originales del conjunto de prueba
plt.scatter(test_Celsius, test_Valks, label='Datos originales')

# Graficar datos predichos
plt.scatter(test_Celsius, test_Valks2Estimado, color='red',
label='Predicciones')

# Etiquetas y leyenda
plt.xlabel('Celsius')
plt.ylabel('Valks')
plt.legend()

# Mostrar gráfica
```

```
plt.show()
```

In [11]:

```
# Codigo para pasar el notebook a html
import os
from google.colab import drive
drive.mount('/content/drive')
# Listar archivos en el directorio MyDrive/Tarea
os.listdir('/content/drive/MyDrive/Tarea')
```

```
---------------------------------------------------------------------------
MessageError                                Traceback (most recent call last)
<ipython-input-11-06640da1abf9> in <cell line: 4>()
      2 import os
      3 from google.colab import drive
----> 4 drive.mount('/content/drive')
      5 # Listar archivos en el directorio MyDrive/Tarea
      6 os.listdir('/content/drive/MyDrive/Tarea')

/usr/local/lib/python3.10/dist-packages/google/colab/drive.py in
mount(mountpoint, force_remount, timeout_ms, readonly)
     98 def mount(mountpoint, force_remount=False, timeout_ms=120000,
readonly=False):
     99   """Mount your Google Drive at the specified mountpoint path."""
--> 100   return _mount(
    101       mountpoint,
    102       force_remount=force_remount,

/usr/local/lib/python3.10/dist-packages/google/colab/drive.py in
_mount(mountpoint, force_remount, timeout_ms, ephemeral, readonly)
    131   )
    132   if ephemeral:
```

```
--> 133     _message.blocking_request(
    134         'request_auth',
    135         request={'authType': 'dfs_ephemeral'},

/usr/local/lib/python3.10/dist-packages/google/colab/_message.py in
blocking_request(request_type, request, timeout_sec, parent)
    174       request_type, request, parent=parent, expect_reply=True
    175   )
--> 176   return read_reply_from_input(request_id, timeout_sec)

/usr/local/lib/python3.10/dist-packages/google/colab/_message.py in
read_reply_from_input(message_id, timeout_sec)
    101     ):
    102       if 'error' in reply:
--> 103         raise MessageError(reply['error'])
    104       return reply.get('data', None)
    105

MessageError: Error: credential propagation was unsuccessful
```

In [ ]:

```
!jupyter nbconvert --to html "/content/drive/MyDrive/Tarea/Challenge.ipynb"
```

Cometarios del profe¶

- Entregar el readme correctamente
- Entrenar sobre train y probar en test