In [22]:
```
import random
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

Dodatkowe zadanie kwalifikacyjne

Opis:

- Celem zadania jest opracowanie strategii optymalizującej zysk, uwzględniającej ryzyko, w warunkach deterministycznych generacji cen.

Opis mechanizmu generacji ścieżek cenowych:

- Ścieżka cenowa zaczyna się od poziomu 100 punktów i składa się z 101 okresów.
- W każdym okresie cena ulega 1 punktowej zmianie na podstawie losowanej wartości z listy zmian.
- Lista zmian zawiera 50 wartości "+1 pkt." oraz 50 wartości "-1 pkt.", tak aby suma zmian ceny była równa 0.

Zasady tworzenia strategii:

- Strategia opiera się na iteracji poprzez każdy okres wygenerowanej ścieżki.
- Podczas każdej iteracji strategia korzysta tylko z informacji o ścieżce cenowej z poprzednich i bieżących iteracji.
- Strategia zwraca sygnał podczas każdej iteracji, informujący o zmianie pozycji w następnej iteracji.
- Sygnał strategii mieści się w przedziale od -10 do +10 i oznacza zmianę (nie deklarację!) pozycji o wartość sygnału.\

np:
okres 1) pozycja = 10, sygnał -5
okres 2) pozycja = 5 ponieważ: pozycja(t) = pozycja(t-1) + sygnał(t-1)

- Kierunek pozycji oraz jej wielkość bezpośrednio wpływają na wynik strategii w każdej iteracji, zgodnie z równaniem:\

wynik(t) = (cena(t)-cena(t-1)) * pozycja

- Dodatkowo, od każdej zmiany pozycji naliczana jest opłata transakcyjna wynosząca 0,2 punkta za każdy punkt w zmianie pozycji.\

np:
okres 1) pozycja = 10, sygnał -5
okres 2) pozycja = 5, opłata = 2,5 ponieważ: opłata(t) = |pozycja(t-1) - pozycja(t)| * 0,5

Metodyka obliczania wyniku strategii:

- Ostateczny wynik strategii będzie oceniany na podstawie listy wyników strategii z 10 000 wygenerowanych ścieżek cenowych.
- Wynik strategii to suma pojedynczych wyników z każdej ścieżki, podzielona przez ich standardowe odchylenie.

Implementacja strategii:

- Strategię można przygotować w formie kodu lub opisać jako logiczny proces składający się z zdań typu "jeżeli x, to y".
- Strategie będą przedstawiane przez max. 10 min. podczas rozmowy z testującym.
- Podczas rozmowy indywidualnej sprawdzane będzie, czy dostarczony kod spełnia zasady tworzenia strategii. W przypadku przedstawienia tylko procesu logicznego, testujący przeprowadzi implementację logiki do postaci kodu.
- Testowanie strategii odbędzie się w obecności wszytskich uczestników po etapie rozmów indywidualnych.

funkcja generacji ścieżki (nie zmieniać)

In [47]:
```python
def generate_path():

    # tworzenie listy 50 ruchów +1 i 50 ruchów -1
    moves_lst = ([1] * 50) + ([-1] * 50)

    # losowa zmiana kolejności
    random.shuffle(moves_lst)

    # zamiana ruchów na ścieżkę cenową
    moves_lst = [100] + moves_lst
    path = {i: price for i, price in enumerate(np.cumsum(moves_lst))}

    return path
```

In [49]:
```python
fig, ax = plt.subplots()

ax.plot(generate_path().values())
ax.axhline(y=100, color='gray')

ax.set_title('Przykładowa losowa ścieżka cenowa')

plt.show()
```

funkcja strategii

In [201…

```python
def strategy(posted_path, position):
    '''
    Parametry:

    -posted_path(list): Lista zawierająca ceny z poprzednich i aktualnych iteracji ścieżki cenowe

    -position(float): Aktualna pozycja przedstawiona liczbowo.

    Zwraca:

    - signal(float): Sygnał strategii na daną iteracje.

    '''

    # TUTAJ TEKST LOGICZNY STRATEGII:

    # Przykładowa strategia:

    # cena z ostatniej iteracji
    last_price = posted_path[-1]

    # zmiana ceny z ostaniej iteracji
    if len(posted_path)>=2:
        price_change = posted_path[-1] - posted_path[-2]
    else:
        price_change = 0

    # sygnał, że pozycja będzie równa zmianie ceny
    signal = price_change-position

    return signal
```

funkcja testowania strategii (nie zmieniać)

In [202…]
```
def test(no_of_iterations):
    iterations_results = []

    for i in range(no_of_iterations):
        price_path = generate_path()

        results_dict = {t: {'price':price, 'position':0, 'fee': 0, 'result':0} for t, price in pr

        position = 0
        fee = 0
        result = 0

        ticks = sorted(results_dict.keys())
        for t in range(len(ticks)):
            results_dict[t]['position'] = position
            results_dict[t]['fee'] = fee
            results_dict[t]['result'] = result

            posted_path = [results_dict[tick]['price'] for tick in ticks if tick<=t]
            positions = [results_dict[tick]['position'] for tick in ticks if tick<=t]

            if t>0:
                result = ((posted_path[t] - posted_path[t-1])*position)-fee
            else:
                result = 0


            signal = strategy(posted_path, positions[-1])

            position = positions[-1] + signal
            fee = abs(positions[-1] - position) * 0.2



        results = sum([results_dict[t]['result'] for t in results_dict.keys()])
        iterations_results.append(results)

    return iterations_results
```

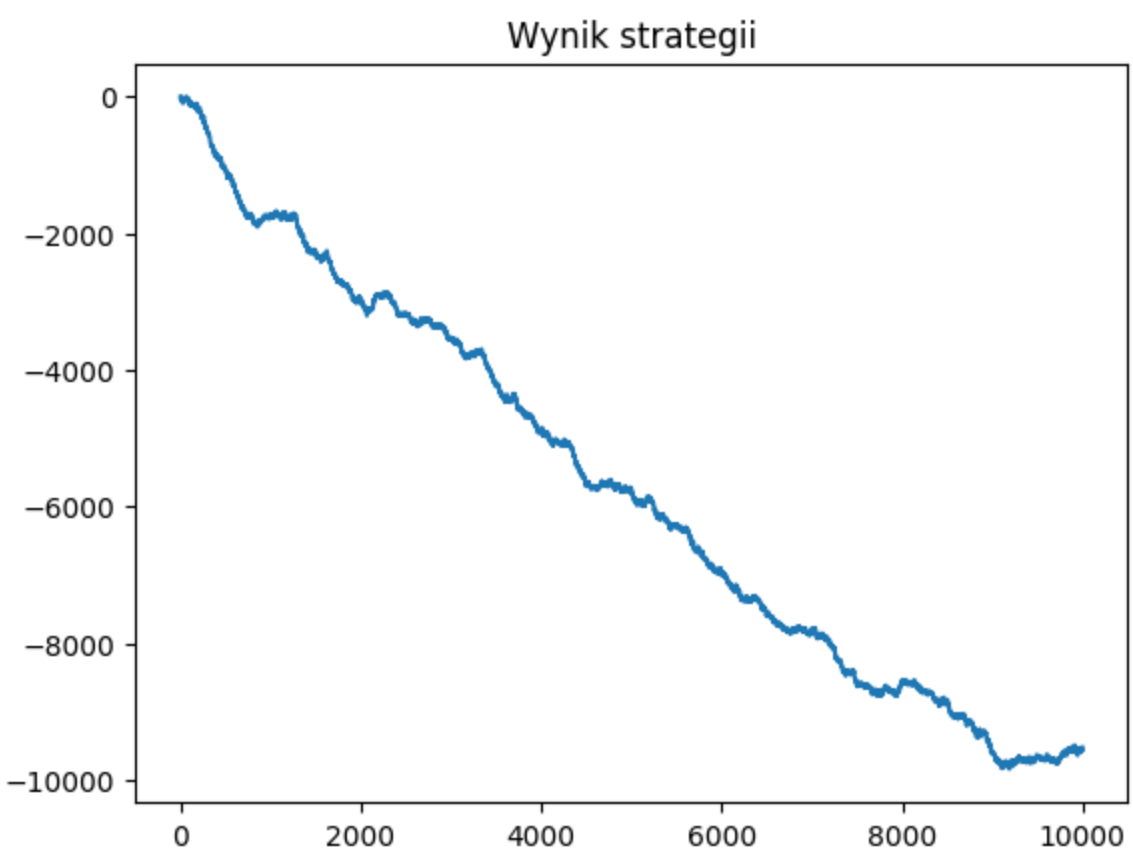
In [208…]
```
results = test(10000)
```

In [210…]
```
fig, ax = plt.subplots()

ax.plot(np.cumsum(results))

ax.set_title('Wynik strategii')

plt.show()
```


Wynik strategii
0
−2000
−4000
−6000
−8000
−10000
0
2000
4000
6000
8000
10000

In [ ]: