Task 1:

Implement Hill climbing solutions for the given two function. The Hill Climbing solution should provide values for the x and y parameters where the value of the function maximizes. The figures also provide the range of values the solution exist within. Restrict your search within the domain of the variables for a quicker response.

f(x,y)=e-(x2+y2)

In [20]:
```
import random
from math import e

def evaluate(x, y):
    return e**(-(x**2 + y**2))

step = [-0.025, +0.025]

def neighbor(x, y, prev_eval):
    current_x, current_y = x, y
    for i in range(2):
        for value in step:
            x += value if i == 0 else 0
            y += value if i == 1 else 0
            neigh_eval = evaluate(x, y)
            if neigh_eval > prev_eval:
                return x, y, neigh_eval
            x, y = current_x, current_y
    return current_x, current_y, prev_eval

x, y = random.uniform(-2, 2), random.uniform(-2, 2)
current_eval = evaluate(x, y)

iterations = 200
while iterations > 0:
    x, y, current_eval = neighbor(x, y, current_eval)
    iterations -= 1

print(x, y, current_eval)
```

```
0.010865397423668887 -0.004430470337771823 0.9998623235496843
```

$\boldsymbol{f(x, y) = (1 - x)2 + 100(y - x2)2}$

In [21]:
```python
import random
from math import e

def evaluate(x, y):
    return (1 - x**2) + 100 * (y - x**2)**2

step = [-0.025, +0.025]

def neighbor(x, y, prev_eval):
    current_x, current_y = x, y
    for i in range(2):
        for value in step:
            x += value if i == 0 else 0
            y += value if i == 1 else 0
            neigh_eval = evaluate(x, y)
            if neigh_eval > prev_eval:
                return x, y, neigh_eval
            x, y = current_x, current_y
    return current_x, current_y, prev_eval

x, y = random.uniform(-2, 2), random.uniform(-2, 2)
current_eval = evaluate(x, y)

iterations = 200
while iterations > 0:
    x, y, current_eval = neighbor(x, y, current_eval)
    iterations -= 1

print(x, y, current_eval)
```

```
-0.011207789083784996 5.253915931451589 2761.23114379691
```

Task 2:

Implement an Hill climbing search for the 8 queen problem represented below. Using the solution representation shown in the figure below might reduce your solution space size.

• Penalty of one queen: the number of queens she can check.

• Penalty of a configuration: the sum of the penalties of all queens.

• Note: penalty is to be minimized

• Fitness of a configuration: inverse penalty to be maximized

Your current execution might get stuck at a local optimum. Implement a random restart technique hill climbing variant to improve your results.

In [22]:

```python
import random

def evaluation(sol):
    penalty = 0
    for i in range(8):
        for j in range(i+1, 8):
            if abs(i - j) == abs(sol[i] - sol[j]):
                penalty += 1
    return penalty

def neighbor(sol):
    index = random.randint(0, 6)
    index2 = index + 1
    sol[index], sol[index2] = sol[index2], sol[index]
    return sol

def hill_climbing_with_restart():
    max_iterations = 100
    restarts = 10

    best_sol = None
    best_penalty = float('inf')

    for _ in range(restarts):
        sol = list(range(1, 9))
        random.shuffle(sol)
        current_penalty = evaluation(sol)

        for _ in range(max_iterations):
            neighbor_sol = neighbor(sol)
            neighbor_penalty = evaluation(neighbor_sol)

            if neighbor_penalty < current_penalty:
                sol = neighbor_sol
                current_penalty = neighbor_penalty

        if current_penalty < best_penalty:
            best_sol = sol
            best_penalty = current_penalty

    print("Configuration:", best_sol)
    print("Penalty:", best_penalty)

if __name__ == "__main__":
    hill_climbing_with_restart()
```

```
Configuration: [7, 3, 2, 4, 6, 8, 5, 1]
Penalty: 0
```