First Practical Exercise - Random Search

This is going to be the starting point for your implementation as it is relatively straightforward to do and other components will also make us of the functions we develop.

There are 3 things needed:

- A function that generates a random string of 1's and 0's of a length defined by the parameter provided to the function.
- A fitness function that scores the string based on how good a solution it is. This will also be your fitness function used to evaluated the solutions and its neighbours in the hill climber. There are a few options about how you do this, but a way to approach it is to think about how you would identify the better solution from two random strings.
- A function to use the above two functions to generate and evaluate multiple random candidated solutions. It will return the best one found and it's associated fitness.

Constants and Imports

We will need various constants such as the target string along with imports for the packages we are going to make use of.

```
import string
import random
random.seed(42) # initialse and make repeatable
```

Generate a Random Solution

Create a random string of 1's and 0's of length defined in the parameter.

```
# Generate a random string of 1's and 0's as defined by the parameter
# Input parameters: length - the length of the string to generate
# Returns: The randomly generated string
def gen_random_string(length):
  return ''.join(random.choice('01') for _ in range(length))

# test... run this a few times to check the output is as expected
print(gen_random_string(1))
print(gen_random_string(5))
print(gen_random_string(10))
```

```
0
01000
0010000000
```

Fitness Function

This defines how we score or evaluate a solution. The fitness function has an important role in guiding the search towards the solution (but not for the random search, obviously), so it is essential that we are able to identify when one solution is better than another.

There are various alternatives but the more information you provide the quicker the search is likely to be.

```
# Function to calculate a candidates solution fitness - solutions closer to the goal should score higher
# Input parameters: candidate solution
# Returns: fitness of the candidate solution
def fitness(solution):
    # Your code goes here
    return solution.count('1')

# test the above (precise results will depend on your function and may vary with string length)
print(fitness("1111111111")) # output should be the highest score
print(fitness("1010110010")) # output should be a mid-range score
print(fitness("0000000000")) # output should be the lowest score
print(fitness("11111")) # output should be the highest score
print(fitness("10101")) # output should be a mid-range score
print(fitness("00000")) # output should be the lowest score
print(fitness("1")) # output should be the highest score
print(fitness("0")) # output should be the lowest score
```

```
10
5
0
5
3
0
```

```
1
0
```

Function to generate, evaluate and report on multiple random solutions

The number of possible soutions to explore is input as a parameter. Solutions are generated (by calling get_random_string(...)) and a note is kept of the best scoring one (using the fitness function) and the average fitness of all solutions.

```
# Input parameters: An integer designating the number of solutions to be generated and evaluated
# Returns: A tuple with the the average fitness of all solutions, the best solution generated, and its fitness value
def evaluate_random_solutions(target_number):
    # Your code goes here
    solution_length = 10

    total_fitness = 0
    best_solution = None
    best_fitness = -1

    for _ in range(target_number):
      candidate_solution = gen_random_string(solution_length)

      candidate_fitness = fitness(candidate_solution)

      total_fitness += candidate_fitness

      if candidate_fitness > best_fitness:
            best_fitness = candidate_fitness
            best_solution = candidate_solution

    average_fitness = total_fitness / target_number

    return (average_fitness, best_solution, best_fitness)

# tests
print(evaluate_random_solutions(10))
print(evaluate_random_solutions(100))
print(evaluate_random_solutions(1000))
```

```
(5.1, '1011001110', 6)
(5.15, '1111111111', 10)
(5.027, '1111111111', 10)
```