Tools

Genetic algorithms base themselves on natural selection, meaning the reproductive advantage of an individual that fits better in said environment. They make use of tools inspired by biology allowing the specie to evolve through generations.

Selection

This tool resemble the natural selection.

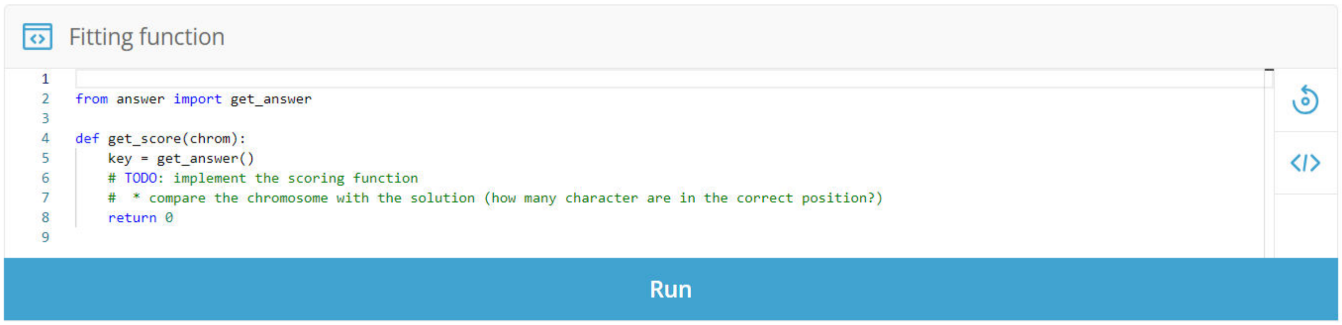
Each individual gets a fitting score, depending on the given problem. In this step, we will select individuals in order to create the next generation. This selection can be done that way:

- A well fitted individual (high fitting score) has good chances at being selected
- The lower the fitting score, the lower the chances at being selected

The first thing to do is to create a fitting function. It will score each individual. The function generally returns a floating number between

0 (bad score) and 1 (good score).

In this exercise, you will implement the fitting function. We will simply compare the chromosome with the solution.



```python

from answer import get_answer

def get_score(chrom):
    key = get_answer()
    # TODO: implement the scoring function
    #  * compare the chromosome with the solution (how many character are in the correct position?)
    return 0

```

In this exercise, you will implement the fitting function. We will simply compare the chromosome with the solution.

Fitting function

```python

from answer import get_answer

def get_score(chrom):
    key = get_answer()
    # TODO: implement the scoring function
    #  * compare the chromosome with the solution (how many character are in the correct position?)
    count = 0
    for i in range(len(key)):
        if chrom[i] == key[i]:
            count += 1

    score = count/len(key)

    return score

```

Success!

Standard Error

```
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
```

Once the fitting function has been defined, we can apply a selection on our population. The goal is to keep the best fitted individuals.

An example of selection function would be:

- Select the best 30% individuals
- Randomly select 20% of the rest

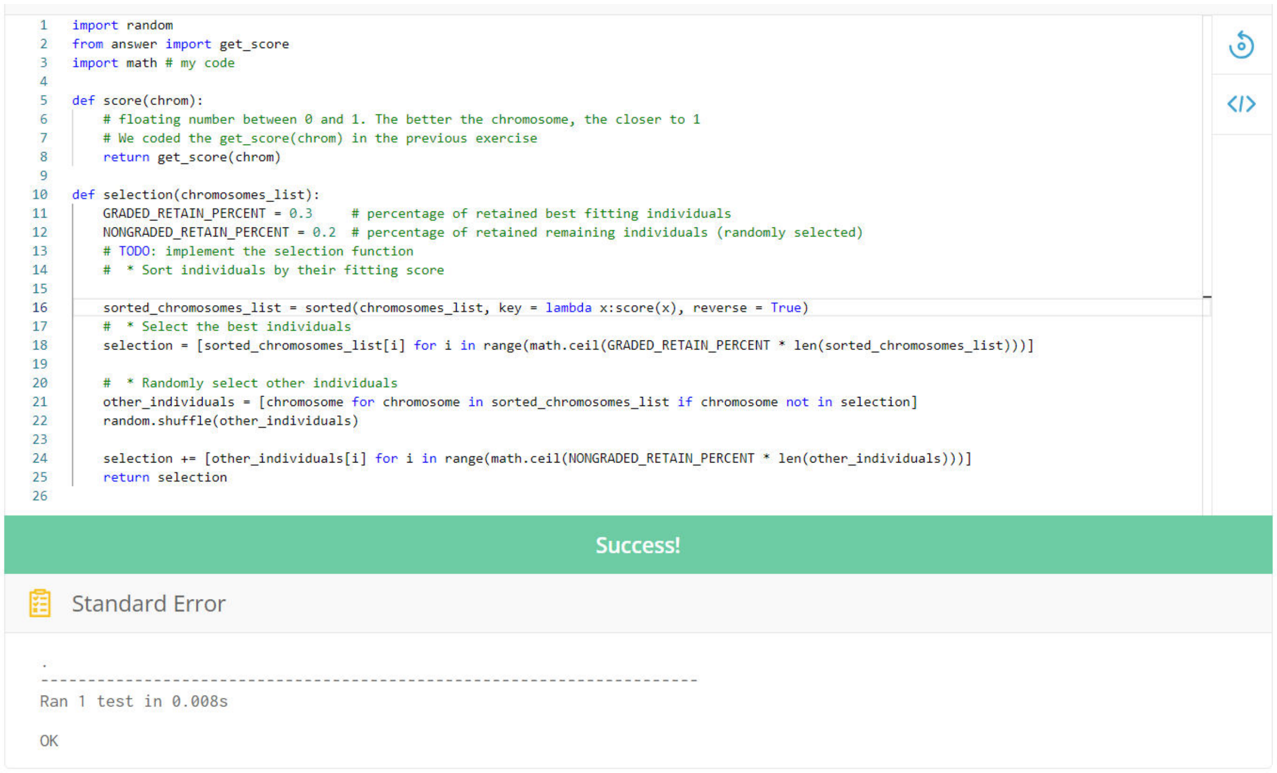
If only the best ones were to be kept (elitism), the risk is to head towards a local extremum without having the possibility to explore other potentially rewarding ways.

In the next exercise, we will implement the selection function.

Chromosomes selection

```python
import random
from answer import get_score

def score(chrom):
    # floating number between 0 and 1. The better the chromosome, the closer to 1
    # We coded the get_score(chrom) in the previous exercise
    return get_score(chrom)

def selection(chromosomes_list):
    GRADED_RETAIN_PERCENT = 0.3     # percentage of retained best fitting individuals
    NONGRADED_RETAIN_PERCENT = 0.2  # percentage of retained remaining individuals (randomly selected)
    # TODO: implement the selection function
    #  * Sort individuals by their fitting score
    #  * Select the best individuals
    #  * Randomly select other individuals
    return []

```

Run



```python
import random
from answer import get_score
import math # my code

def score(chrom):
    # floating number between 0 and 1. The better the chromosome, the closer to 1
    # We coded the get_score(chrom) in the previous exercise
    return get_score(chrom)

def selection(chromosomes_list):
    GRADED_RETAIN_PERCENT = 0.3     # percentage of retained best fitting individuals
    NONGRADED_RETAIN_PERCENT = 0.2  # percentage of retained remaining individuals (randomly selected)
    # TODO: implement the selection function
    #  * Sort individuals by their fitting score

    sorted_chromosomes_list = sorted(chromosomes_list, key = lambda x:score(x), reverse = True)
    #  * Select the best individuals
    selection = [sorted_chromosomes_list[i] for i in range(math.ceil(GRADED_RETAIN_PERCENT * len(sorted_chromosomes_list)))]

    #  * Randomly select other individuals
    other_individuals = [chromosome for chromosome in sorted_chromosomes_list if chromosome not in selection]
    random.shuffle(other_individuals)

    selection += [other_individuals[i] for i in range(math.ceil(NONGRADED_RETAIN_PERCENT * len(other_individuals)))]
    return selection

```

Success!

Standard Error

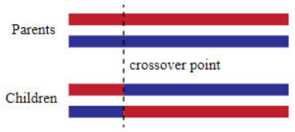
```
.
----------------------------------------------------------------------
Ran 1 test in 0.008s

OK
```

Genetic operators

These tools will directly impact the genetic material of a new individual.

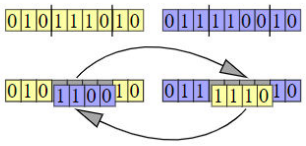
Crossover / reproduction

We now need to fill our population with a new generation. In order to create this generation, we will make individuals reproduce. During this step, the parents will be exchanging their genetic material to produce a child Genetic recombination



A solution is to take each parent 50% of their genetic material, making the crossover in the middle of the chromosome. For instance if the parents are `ABCDEFGH` and `1345678`, the child will be `ABCD5678`. There is no change in the genes' places.

There are other types of crossovers. One can take 70% of the genetic material of a parent and 30% of the other. Or we could do the crossover on multiple locations.



Chromosome crossover

```python
def crossover(parent1, parent2):
    # TODO: implement the crossover function
    #  * Select half of the parent genetic material
    #  * child = half_parent1 + half_parent2
    #  * Return the new chromosome
    #  * Genes should not be moved
    return ""

```

Run

```python
def crossover(parent1, parent2):
    # TODO: implement the crossover function
    #  * Select half of the parent genetic material
    #  * child = half_parent1 + half_parent2

    stop_point = int(len(parent1)/2)
    child = parent1[:stop_point] + parent2[stop_point:]

    #  * Return the new chromosome
    #  * Genes should not be moved
    return child
```

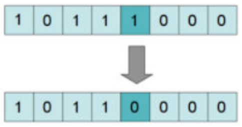
Success!

Standard Error

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Mutation

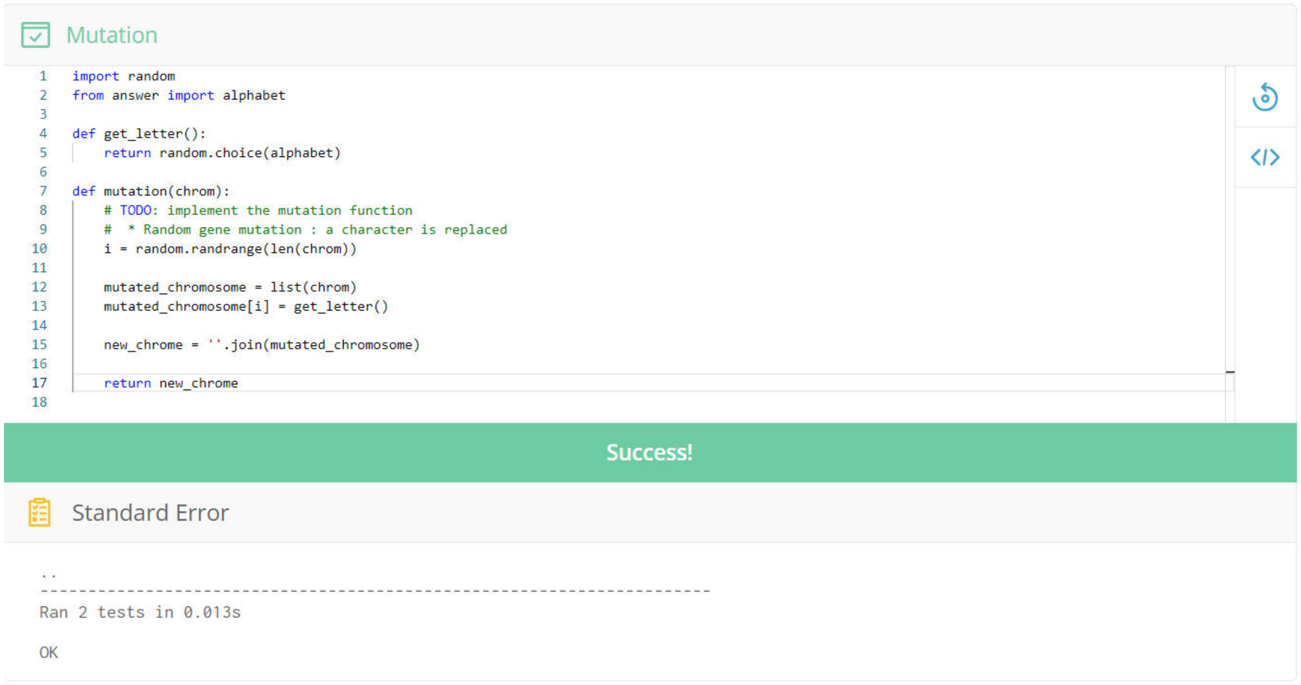
In order to create new genetic material, some individuals will mutate. A gene will randomly be mutated.



Mutation

```python
import random
from answer import alphabet

def get_letter():
    return random.choice(alphabet)

def mutation(chrom):
    # TODO: implement the mutation function
    #  * Random gene mutation : a character is replaced
    return chrom

```

Run


Mutation

```python
import random
from answer import alphabet

def get_letter():
    return random.choice(alphabet)

def mutation(chrom):
    # TODO: implement the mutation function
    #  * Random gene mutation : a character is replaced
    i = random.randrange(len(chrom))

    mutated_chromosome = list(chrom)
    mutated_chromosome[i] = get_letter()

    new_chrome = ''.join(mutated_chromosome)

    return new_chrome

```

Success!

Standard Error

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.013s

OK
```