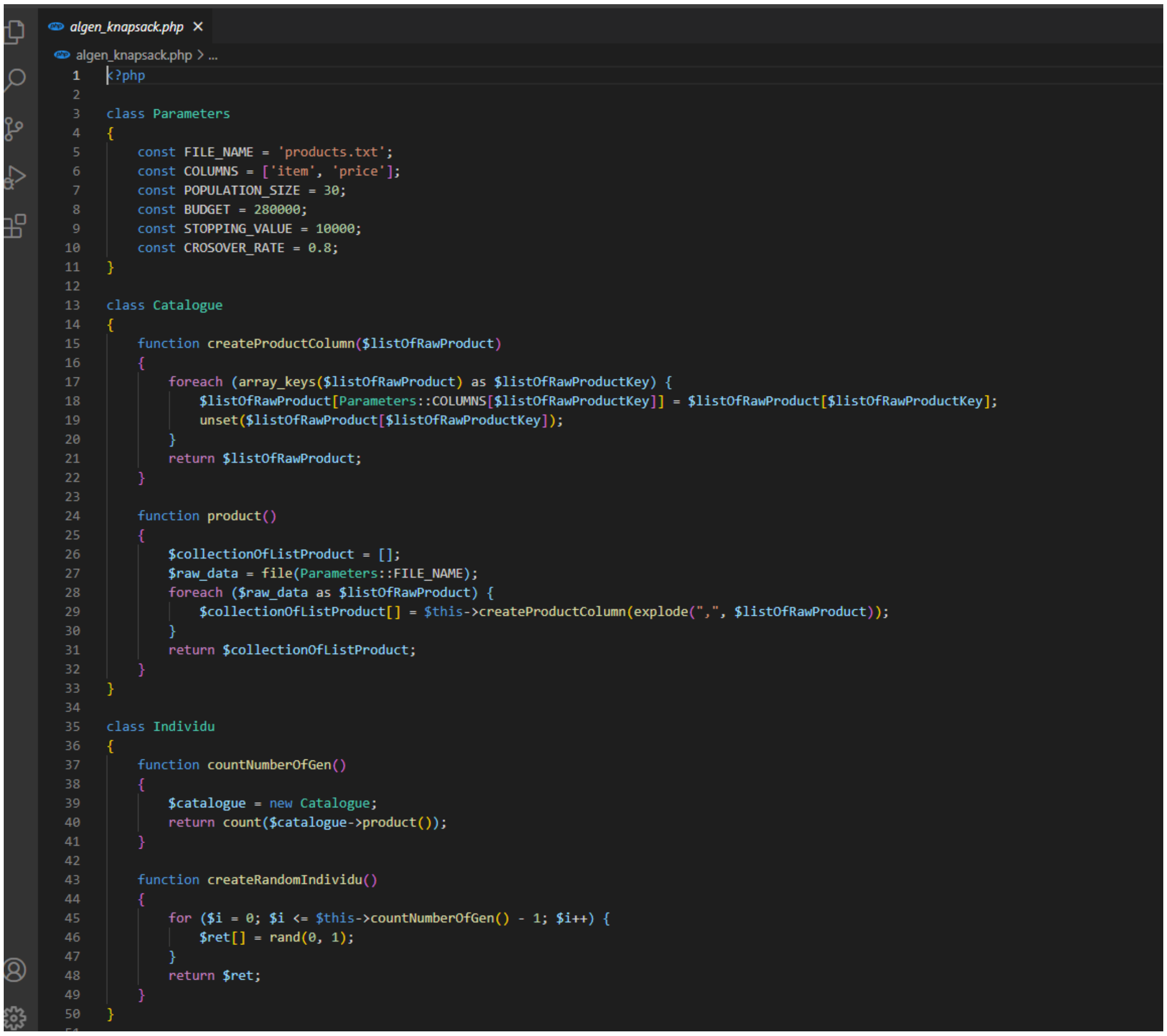
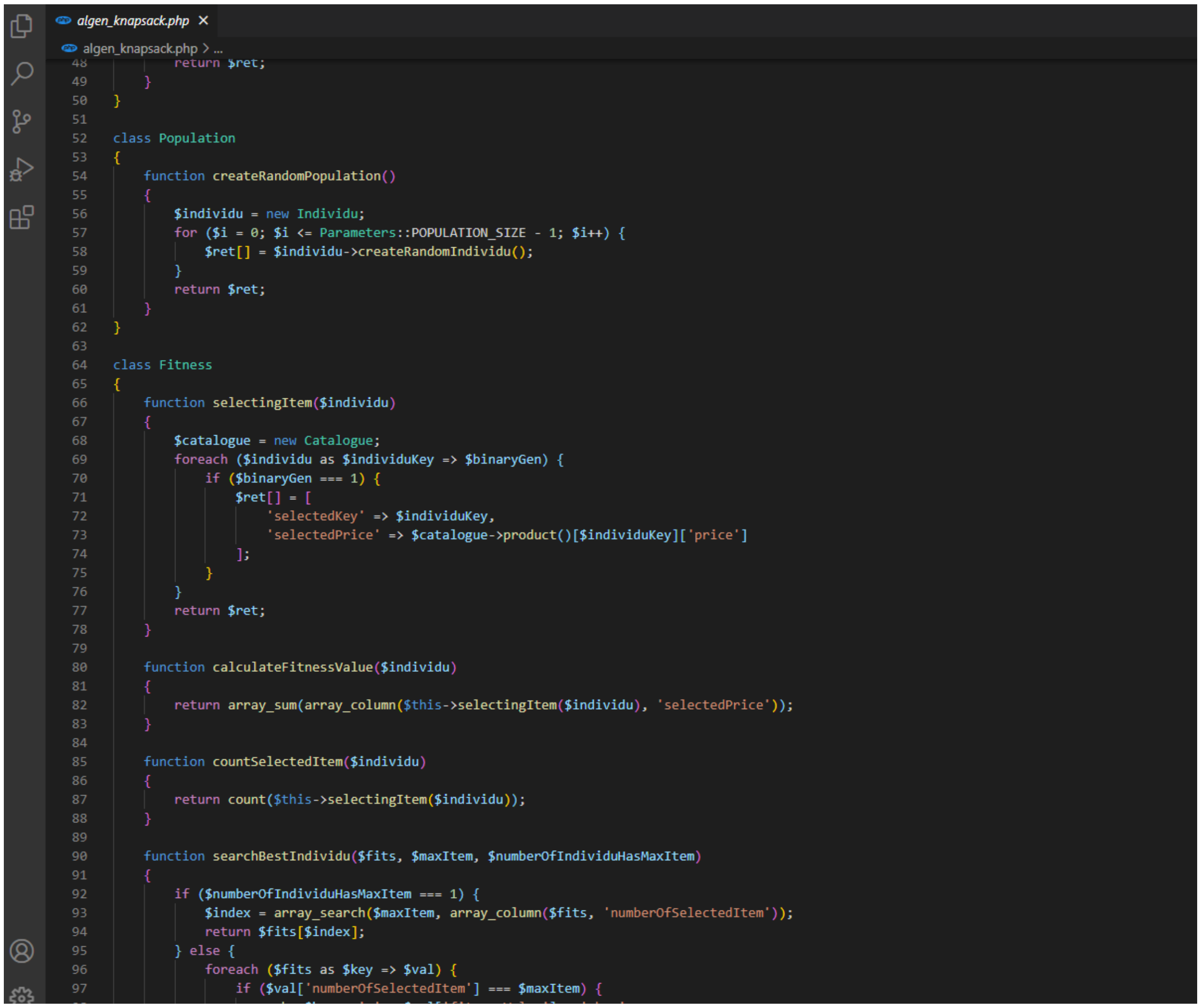
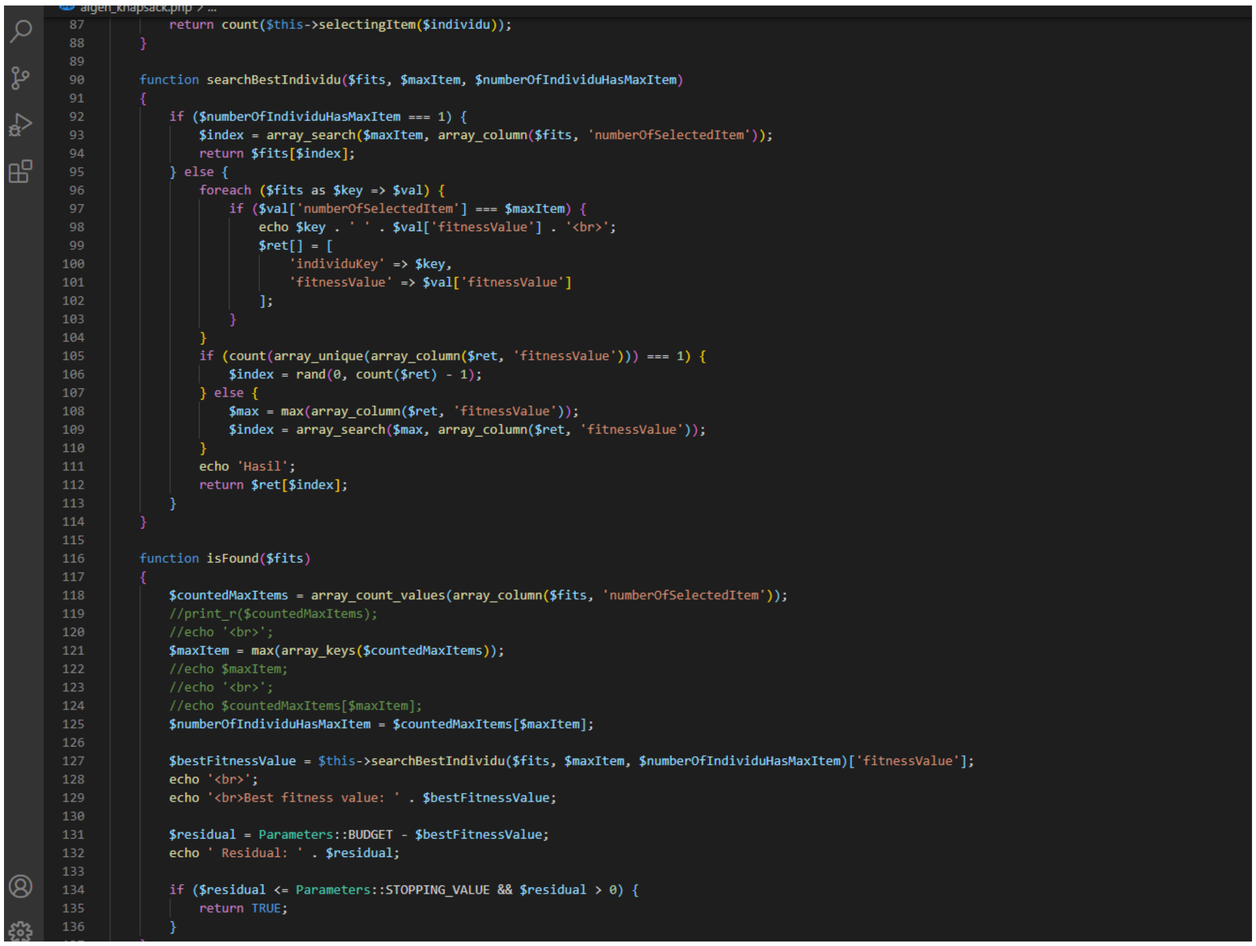

algen_knapsack.php ✕

algen_knapsack.php > ...

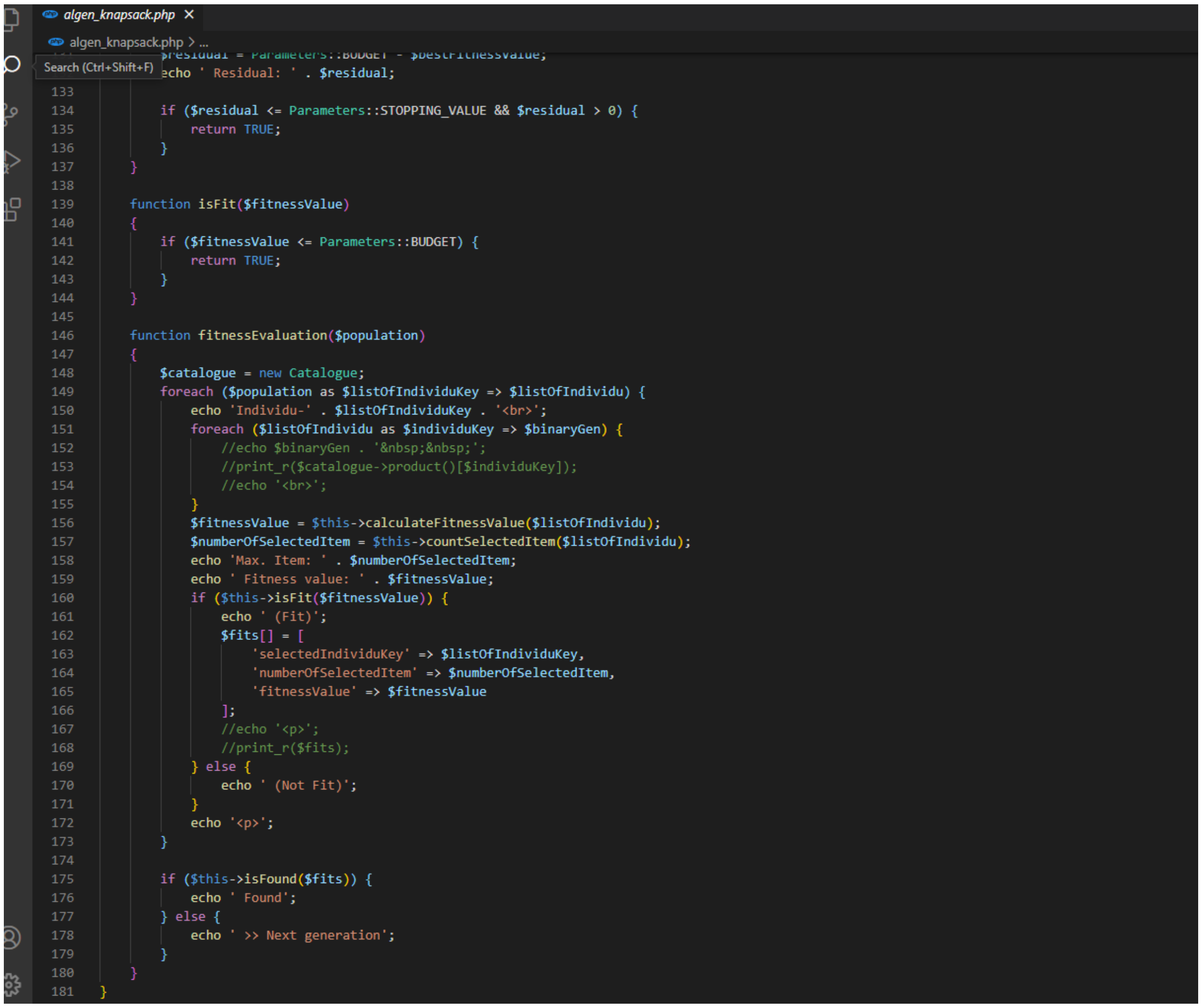
```php
<?php

class Parameters
{
    const FILE_NAME = 'products.txt';
    const COLUMNS = ['item', 'price'];
    const POPULATION_SIZE = 30;
    const BUDGET = 280000;
    const STOPPING_VALUE = 10000;
    const CROSOVER_RATE = 0.8;
}

class Catalogue
{
    function createProductColumn($listOfRawProduct)
    {
        foreach (array_keys($listOfRawProduct) as $listOfRawProductKey) {
            $listOfRawProduct[Parameters::COLUMNS[$listOfRawProductKey]] = $listOfRawProduct[$listOfRawProductKey];
            unset($listOfRawProduct[$listOfRawProductKey]);
        }
        return $listOfRawProduct;
    }

    function product()
    {
        $collectionOfListProduct = [];
        $raw_data = file(Parameters::FILE_NAME);
        foreach ($raw_data as $listOfRawProduct) {
            $collectionOfListProduct[] = $this->createProductColumn(explode(",", $listOfRawProduct));
        }
        return $collectionOfListProduct;
    }
}

class Individu
{
    function countNumberOfGen()
    {
        $catalogue = new Catalogue;
        return count($catalogue->product());
    }

    function createRandomIndividu()
    {
        for ($i = 0; $i <= $this->countNumberOfGen() - 1; $i++) {
            $ret[] = rand(0, 1);
        }
        return $ret;
    }
}
```



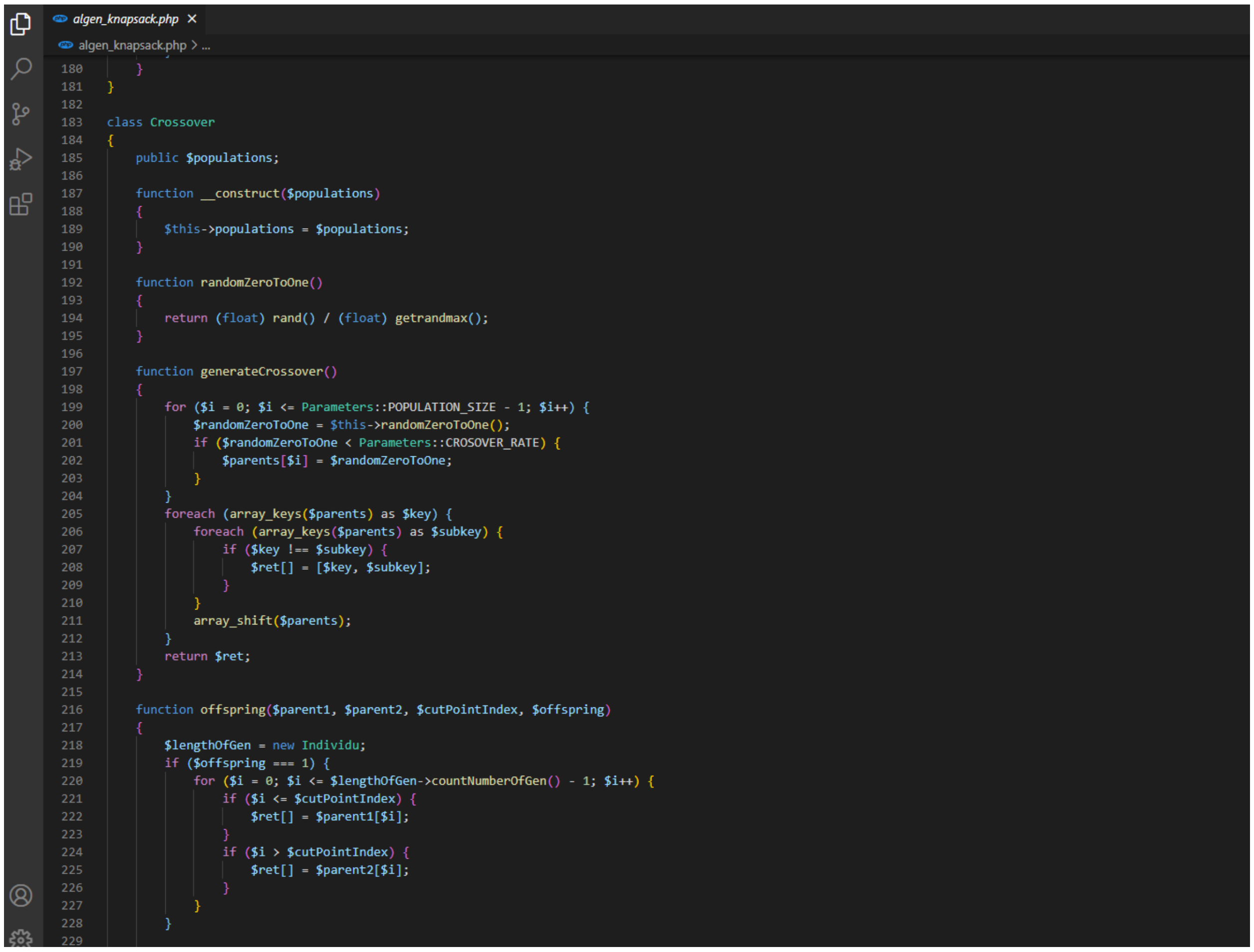
```php
        return $ret;
    }
}

class Population
{
    function createRandomPopulation()
    {
        $individu = new Individu;
        for ($i = 0; $i <= Parameters::POPULATION_SIZE - 1; $i++) {
            $ret[] = $individu->createRandomIndividu();
        }
        return $ret;
    }
}

class Fitness
{
    function selectingItem($individu)
    {
        $catalogue = new Catalogue;
        foreach ($individu as $individuKey => $binaryGen) {
            if ($binaryGen === 1) {
                $ret[] = [
                    'selectedKey' => $individuKey,
                    'selectedPrice' => $catalogue->product()[$individuKey]['price']
                ];
            }
        }
        return $ret;
    }

    function calculateFitnessValue($individu)
    {
        return array_sum(array_column($this->selectingItem($individu), 'selectedPrice'));
    }

    function countSelectedItem($individu)
    {
        return count($this->selectingItem($individu));
    }

    function searchBestIndividu($fits, $maxItem, $numberOfIndividuHasMaxItem)
    {
        if ($numberOfIndividuHasMaxItem === 1) {
            $index = array_search($maxItem, array_column($fits, 'numberOfSelectedItem'));
            return $fits[$index];
        } else {
            foreach ($fits as $key => $val) {
                if ($val['numberOfSelectedItem'] === $maxItem) {
```



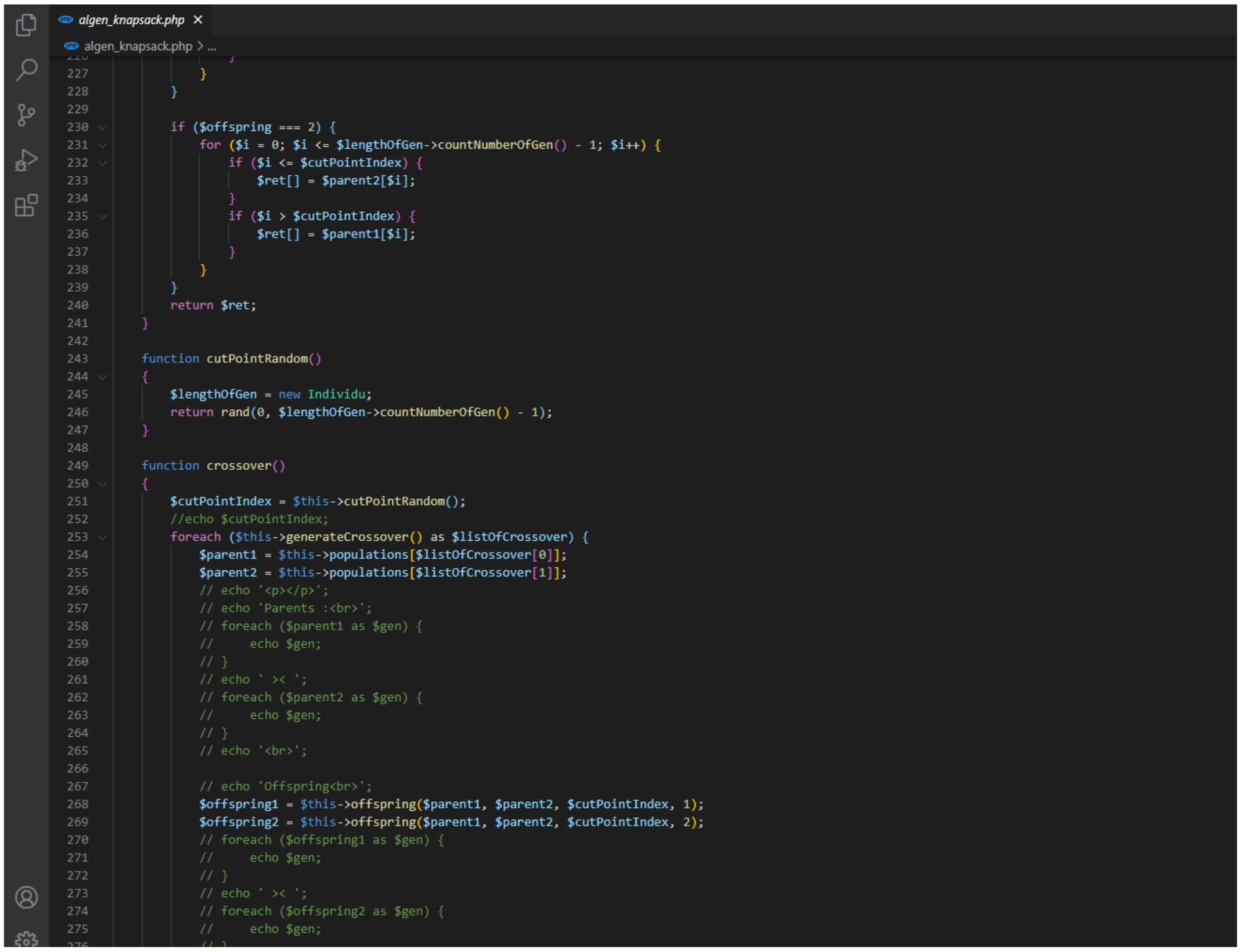
```php
        return count($this->selectingItem($individu));
    }

    function searchBestIndividu($fits, $maxItem, $numberOfIndividuHasMaxItem)
    {
        if ($numberOfIndividuHasMaxItem === 1) {
            $index = array_search($maxItem, array_column($fits, 'numberOfSelectedItem'));
            return $fits[$index];
        } else {
            foreach ($fits as $key => $val) {
                if ($val['numberOfSelectedItem'] === $maxItem) {
                    echo $key . ' ' . $val['fitnessValue'] . '<br>';
                    $ret[] = [
                        'individuKey' => $key,
                        'fitnessValue' => $val['fitnessValue']
                    ];
                }
            }
            if (count(array_unique(array_column($ret, 'fitnessValue'))) === 1) {
                $index = rand(0, count($ret) - 1);
            } else {
                $max = max(array_column($ret, 'fitnessValue'));
                $index = array_search($max, array_column($ret, 'fitnessValue'));
            }
            echo 'Hasil';
            return $ret[$index];
        }
    }

    function isFound($fits)
    {
        $countedMaxItems = array_count_values(array_column($fits, 'numberOfSelectedItem'));
        //print_r($countedMaxItems);
        //echo '<br>';
        $maxItem = max(array_keys($countedMaxItems));
        //echo $maxItem;
        //echo '<br>';
        //echo $countedMaxItems[$maxItem];
        $numberOfIndividuHasMaxItem = $countedMaxItems[$maxItem];

        $bestFitnessValue = $this->searchBestIndividu($fits, $maxItem, $numberOfIndividuHasMaxItem)['fitnessValue'];
        echo '<br>';
        echo '<br>Best fitness value: ' . $bestFitnessValue;

        $residual = Parameters::BUDGET - $bestFitnessValue;
        echo ' Residual: ' . $residual;

        if ($residual <= Parameters::STOPPING_VALUE && $residual > 0) {
            return TRUE;
        }
```



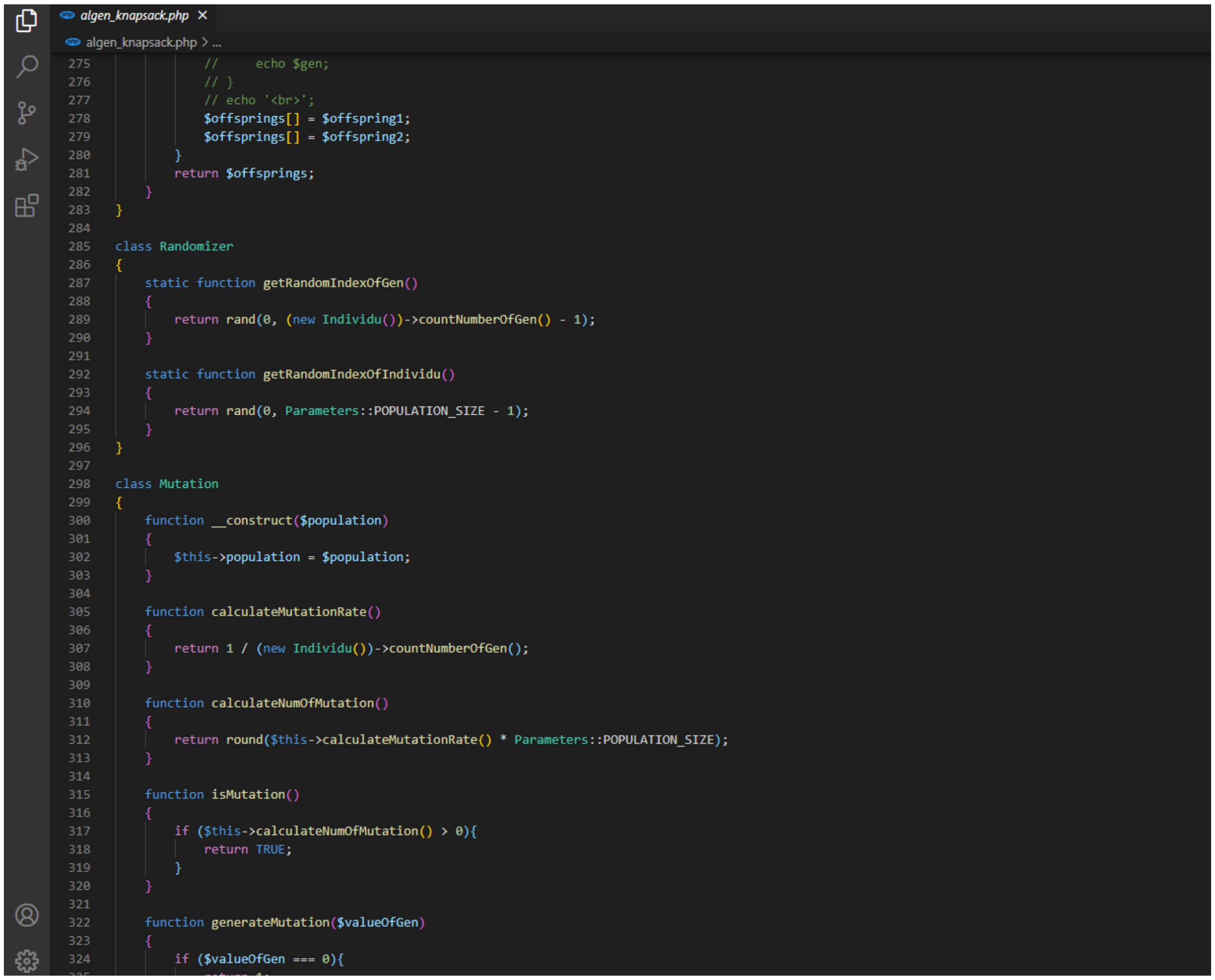
```php
        $residual = Parameters::BUDGET - $bestFitnessValue;
        echo ' Residual: ' . $residual;

        if ($residual <= Parameters::STOPPING_VALUE && $residual > 0) {
            return TRUE;
        }
    }

    function isFit($fitnessValue)
    {
        if ($fitnessValue <= Parameters::BUDGET) {
            return TRUE;
        }
    }

    function fitnessEvaluation($population)
    {
        $catalogue = new Catalogue;
        foreach ($population as $listOfIndividuKey => $listOfIndividu) {
            echo 'Individu-' . $listOfIndividuKey . '<br>';
            foreach ($listOfIndividu as $individuKey => $binaryGen) {
                //echo $binaryGen . '&nbsp;&nbsp;';
                //print_r($catalogue->product()[$individuKey]);
                //echo '<br>';
            }
            $fitnessValue = $this->calculateFitnessValue($listOfIndividu);
            $numberOfSelectedItem = $this->countSelectedItem($listOfIndividu);
            echo 'Max. Item: ' . $numberOfSelectedItem;
            echo ' Fitness value: ' . $fitnessValue;
            if ($this->isFit($fitnessValue)) {
                echo ' (Fit)';
                $fits[] = [
                    'selectedIndividuKey' => $listOfIndividuKey,
                    'numberOfSelectedItem' => $numberOfSelectedItem,
                    'fitnessValue' => $fitnessValue
                ];
                //echo '<p>';
                //print_r($fits);
            } else {
                echo ' (Not Fit)';
            }
            echo '<p>';
        }

        if ($this->isFound($fits)) {
            echo ' Found';
        } else {
            echo ' >> Next generation';
        }
    }
}
```



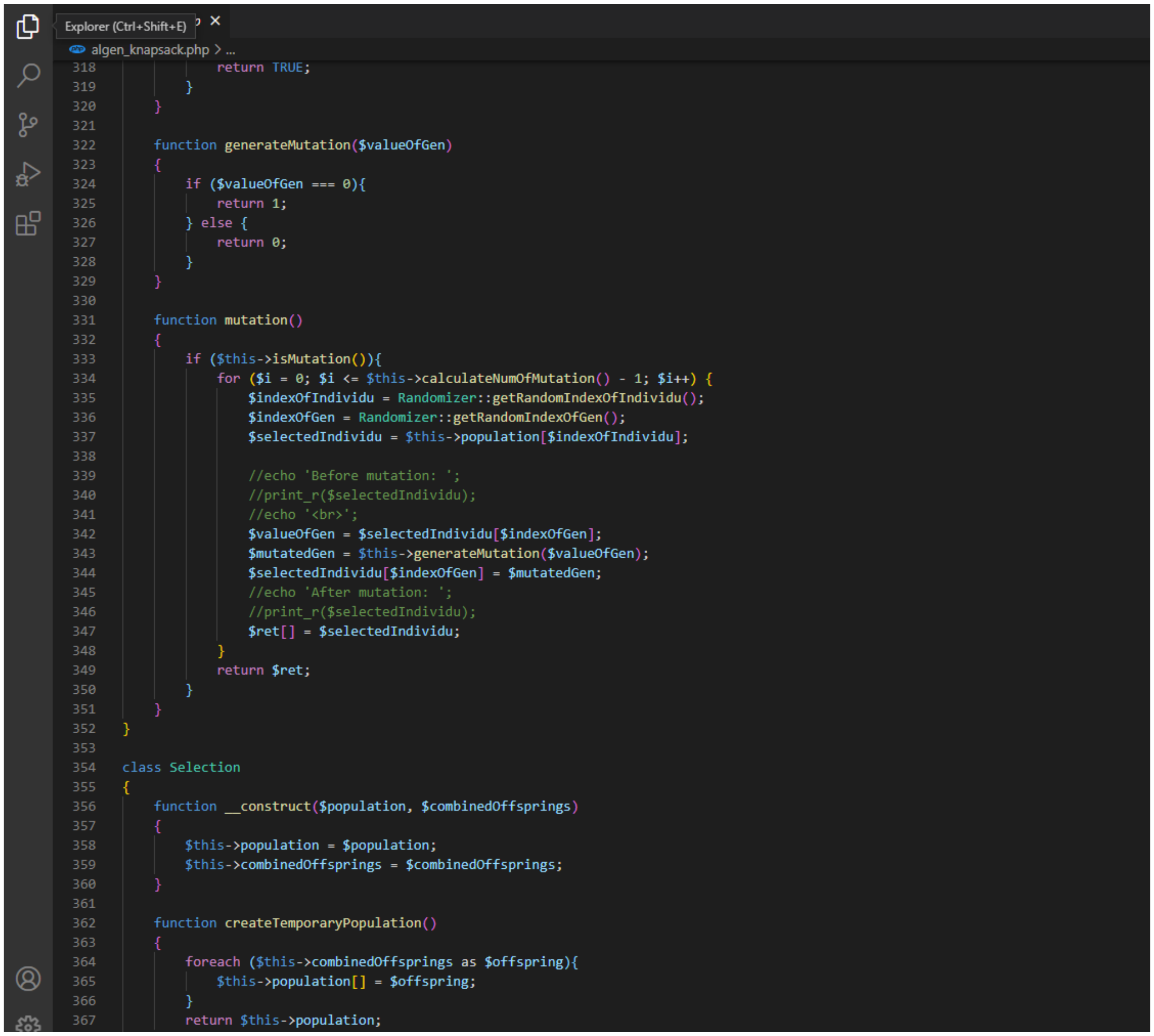
```php
    }
}

class Crossover
{
    public $populations;

    function __construct($populations)
    {
        $this->populations = $populations;
    }

    function randomZeroToOne()
    {
        return (float) rand() / (float) getrandmax();
    }

    function generateCrossover()
    {
        for ($i = 0; $i <= Parameters::POPULATION_SIZE - 1; $i++) {
            $randomZeroToOne = $this->randomZeroToOne();
            if ($randomZeroToOne < Parameters::CROSOVER_RATE) {
                $parents[$i] = $randomZeroToOne;
            }
        }
        foreach (array_keys($parents) as $key) {
            foreach (array_keys($parents) as $subkey) {
                if ($key !== $subkey) {
                    $ret[] = [$key, $subkey];
                }
            }
            array_shift($parents);
        }
        return $ret;
    }

    function offspring($parent1, $parent2, $cutPointIndex, $offspring)
    {
        $lengthOfGen = new Individu;
        if ($offspring === 1) {
            for ($i = 0; $i <= $lengthOfGen->countNumberOfGen() - 1; $i++) {
                if ($i <= $cutPointIndex) {
                    $ret[] = $parent1[$i];
                }
                if ($i > $cutPointIndex) {
                    $ret[] = $parent2[$i];
                }
            }
        }

```



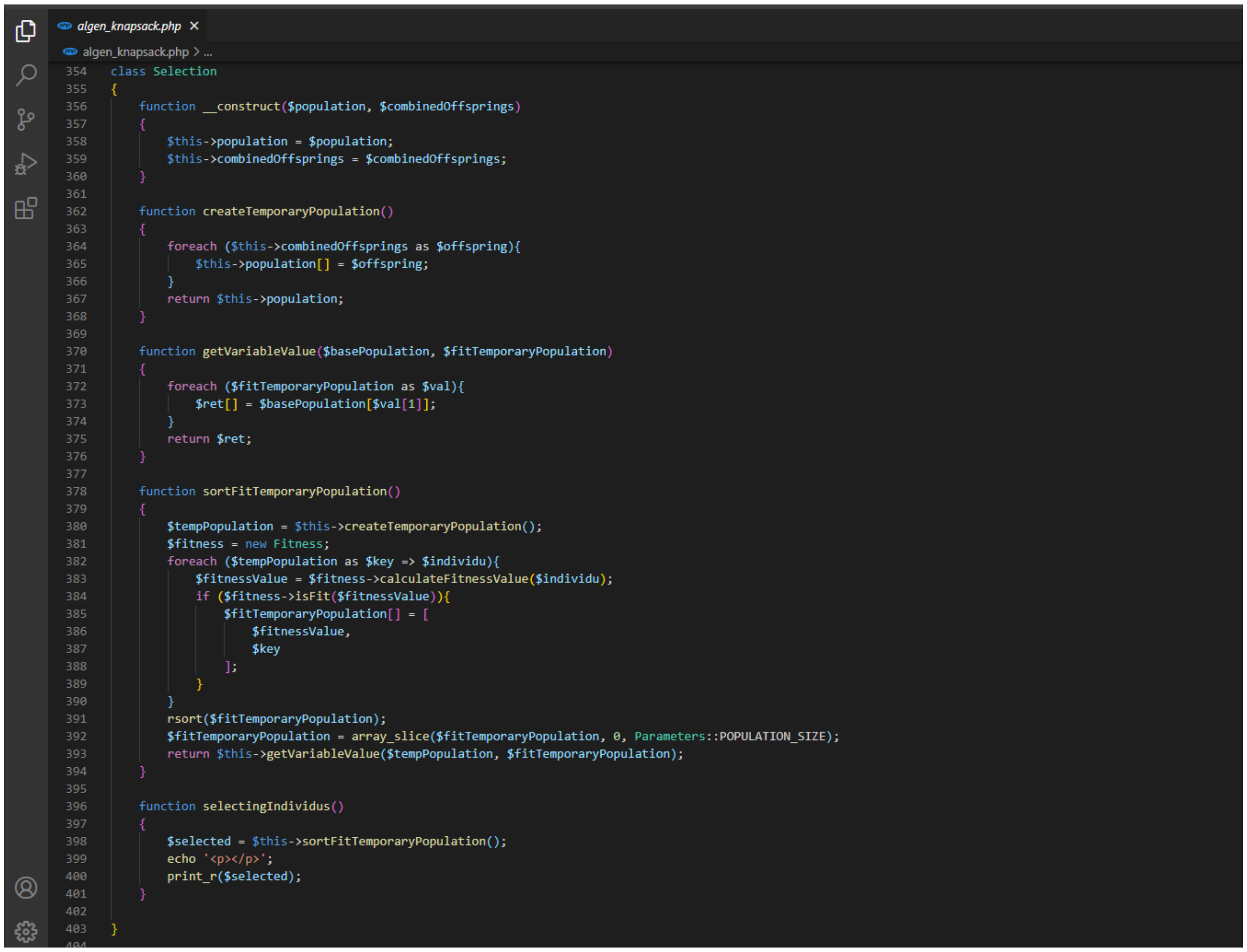
```php
            }
        }

        if ($offspring === 2) {
            for ($i = 0; $i <= $lengthOfGen->countNumberOfGen() - 1; $i++) {
                if ($i <= $cutPointIndex) {
                    $ret[] = $parent2[$i];
                }
                if ($i > $cutPointIndex) {
                    $ret[] = $parent1[$i];
                }
            }
        }
        return $ret;
    }

    function cutPointRandom()
    {
        $lengthOfGen = new Individu;
        return rand(0, $lengthOfGen->countNumberOfGen() - 1);
    }

    function crossover()
    {
        $cutPointIndex = $this->cutPointRandom();
        //echo $cutPointIndex;
        foreach ($this->generateCrossover() as $listOfCrossover) {
            $parent1 = $this->populations[$listOfCrossover[0]];
            $parent2 = $this->populations[$listOfCrossover[1]];
            // echo '<p></p>';
            // echo 'Parents :<br>';
            // foreach ($parent1 as $gen) {
            //     echo $gen;
            // }
            // echo ' >< ';
            // foreach ($parent2 as $gen) {
            //     echo $gen;
            // }
            // echo '<br>';

            // echo 'Offspring<br>';
            $offspring1 = $this->offspring($parent1, $parent2, $cutPointIndex, 1);
            $offspring2 = $this->offspring($parent1, $parent2, $cutPointIndex, 2);
            // foreach ($offspring1 as $gen) {
            //     echo $gen;
            // }
            // echo ' >< ';
            // foreach ($offspring2 as $gen) {
            //     echo $gen;
```



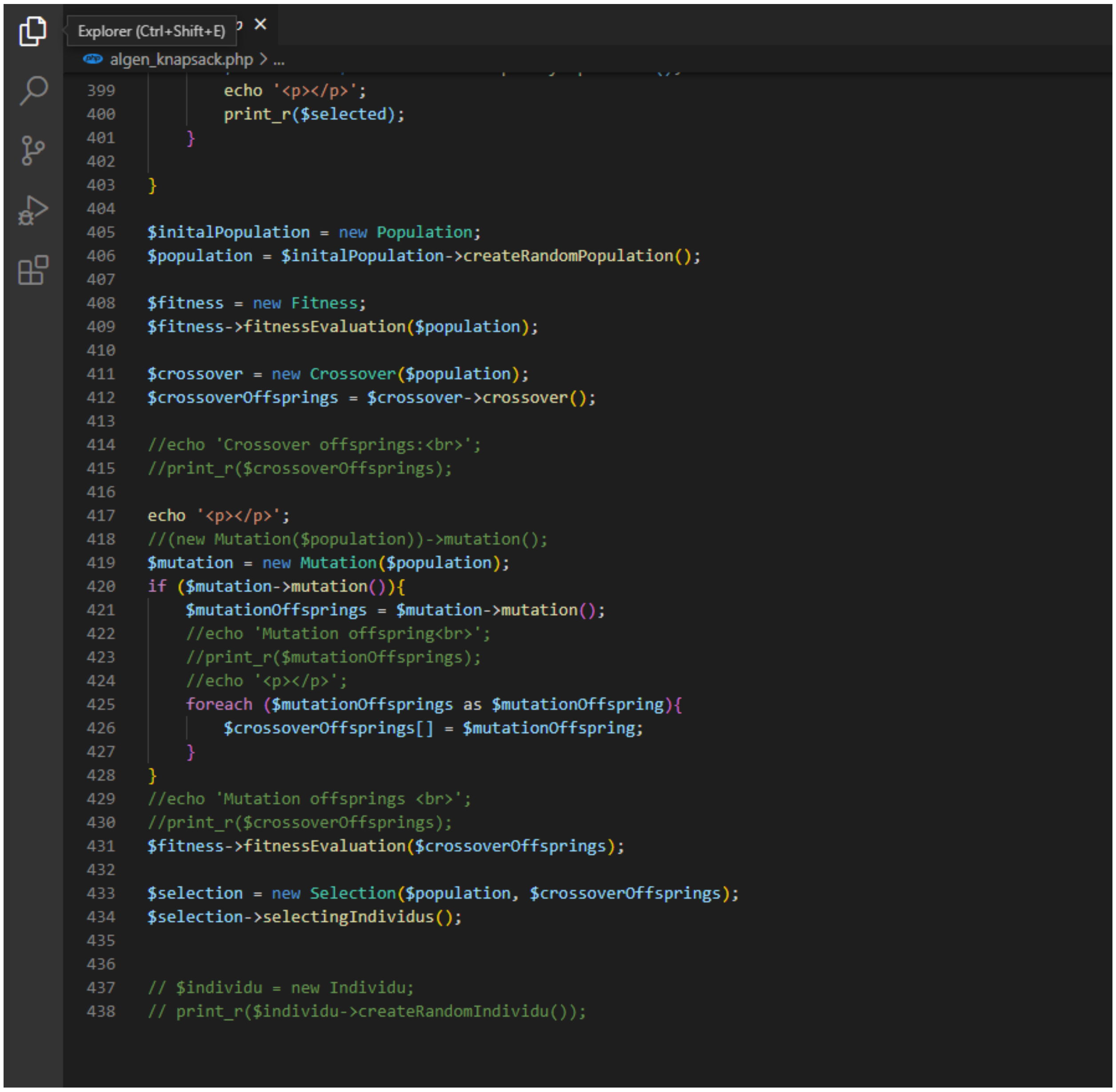
```php
            //     echo $gen;
            // }
            // echo '<br>';
            $offsprings[] = $offspring1;
            $offsprings[] = $offspring2;
        }
        return $offsprings;
    }
}

class Randomizer
{
    static function getRandomIndexOfGen()
    {
        return rand(0, (new Individu())->countNumberOfGen() - 1);
    }

    static function getRandomIndexOfIndividu()
    {
        return rand(0, Parameters::POPULATION_SIZE - 1);
    }
}

class Mutation
{
    function __construct($population)
    {
        $this->population = $population;
    }

    function calculateMutationRate()
    {
        return 1 / (new Individu())->countNumberOfGen();
    }

    function calculateNumOfMutation()
    {
        return round($this->calculateMutationRate() * Parameters::POPULATION_SIZE);
    }

    function isMutation()
    {
        if ($this->calculateNumOfMutation() > 0){
            return TRUE;
        }
    }

    function generateMutation($valueOfGen)
    {
        if ($valueOfGen === 0){
```



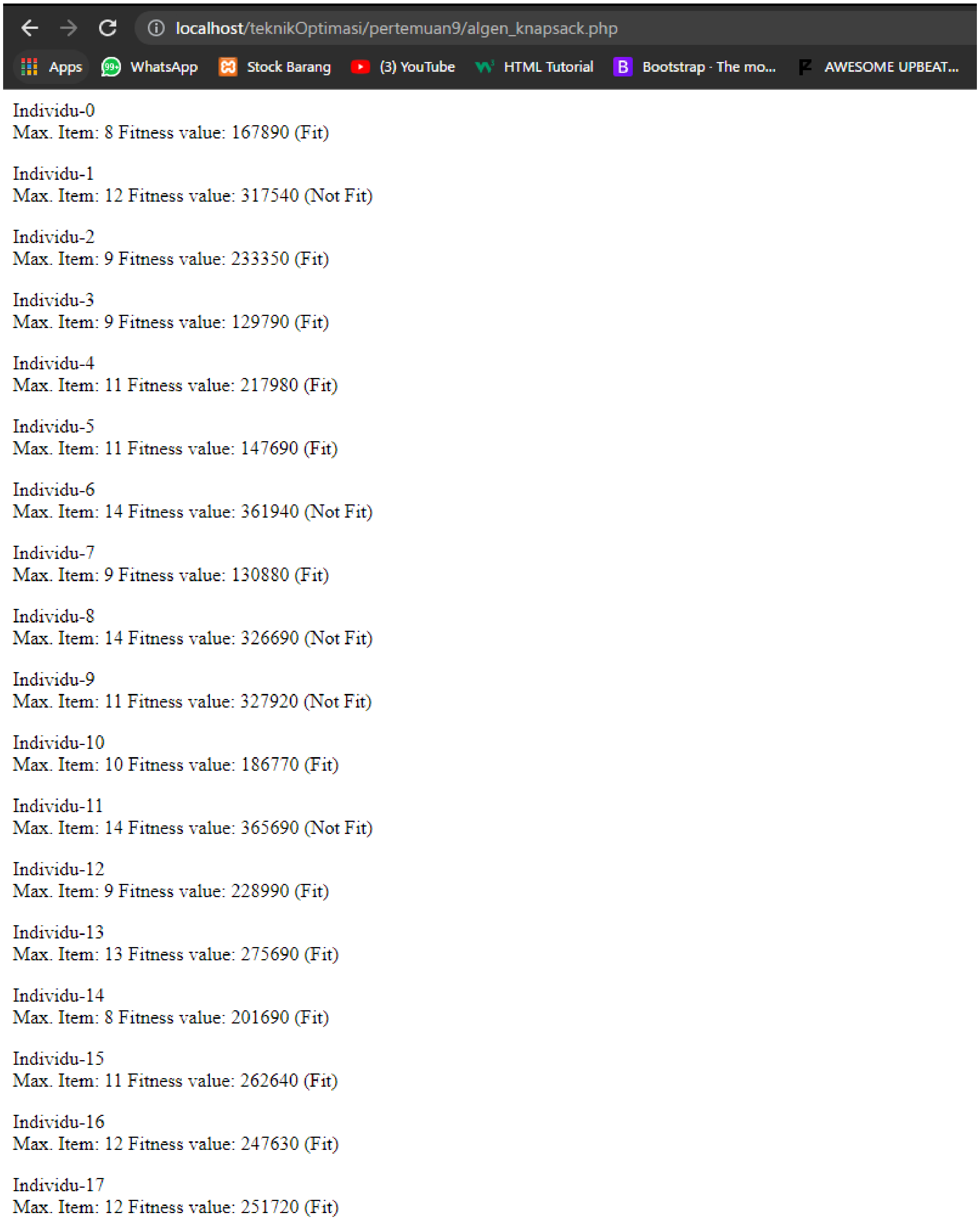
```php
            return TRUE;
        }
    }

    function generateMutation($valueOfGen)
    {
        if ($valueOfGen === 0){
            return 1;
        } else {
            return 0;
        }
    }

    function mutation()
    {
        if ($this->isMutation()){
            for ($i = 0; $i <= $this->calculateNumOfMutation() - 1; $i++) {
                $indexOfIndividu = Randomizer::getRandomIndexOfIndividu();
                $indexOfGen = Randomizer::getRandomIndexOfGen();
                $selectedIndividu = $this->population[$indexOfIndividu];

                //echo 'Before mutation: ';
                //print_r($selectedIndividu);
                //echo '<br>';
                $valueOfGen = $selectedIndividu[$indexOfGen];
                $mutatedGen = $this->generateMutation($valueOfGen);
                $selectedIndividu[$indexOfGen] = $mutatedGen;
                //echo 'After mutation: ';
                //print_r($selectedIndividu);
                $ret[] = $selectedIndividu;
            }
            return $ret;
        }
    }
}

class Selection
{
    function __construct($population, $combinedOffsprings)
    {
        $this->population = $population;
        $this->combinedOffsprings = $combinedOffsprings;
    }

    function createTemporaryPopulation()
    {
        foreach ($this->combinedOffsprings as $offspring){
            $this->population[] = $offspring;
        }
        return $this->population;
```



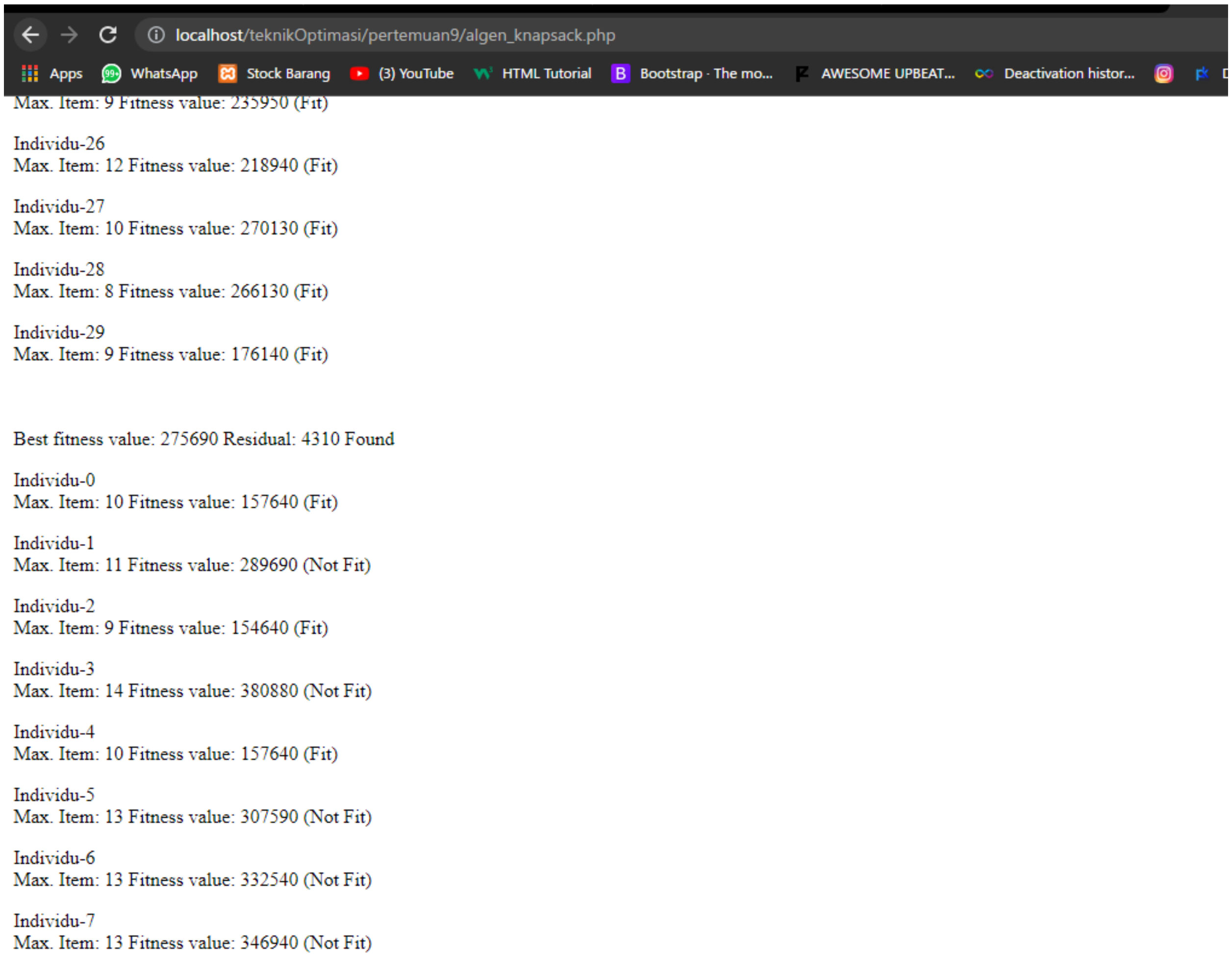
```php
class Selection
{
    function __construct($population, $combinedOffsprings)
    {
        $this->population = $population;
        $this->combinedOffsprings = $combinedOffsprings;
    }

    function createTemporaryPopulation()
    {
        foreach ($this->combinedOffsprings as $offspring){
            $this->population[] = $offspring;
        }
        return $this->population;
    }

    function getVariableValue($basePopulation, $fitTemporaryPopulation)
    {
        foreach ($fitTemporaryPopulation as $val){
            $ret[] = $basePopulation[$val[1]];
        }
        return $ret;
    }

    function sortFitTemporaryPopulation()
    {
        $tempPopulation = $this->createTemporaryPopulation();
        $fitness = new Fitness;
        foreach ($tempPopulation as $key => $individu){
            $fitnessValue = $fitness->calculateFitnessValue($individu);
            if ($fitness->isFit($fitnessValue)){
                $fitTemporaryPopulation[] = [
                    $fitnessValue,
                    $key
                ];
            }
        }
        rsort($fitTemporaryPopulation);
        $fitTemporaryPopulation = array_slice($fitTemporaryPopulation, 0, Parameters::POPULATION_SIZE);
        return $this->getVariableValue($tempPopulation, $fitTemporaryPopulation);
    }

    function selectingIndividus()
    {
        $selected = $this->sortFitTemporaryPopulation();
        echo '<p></p>';
        print_r($selected);
    }

}
```



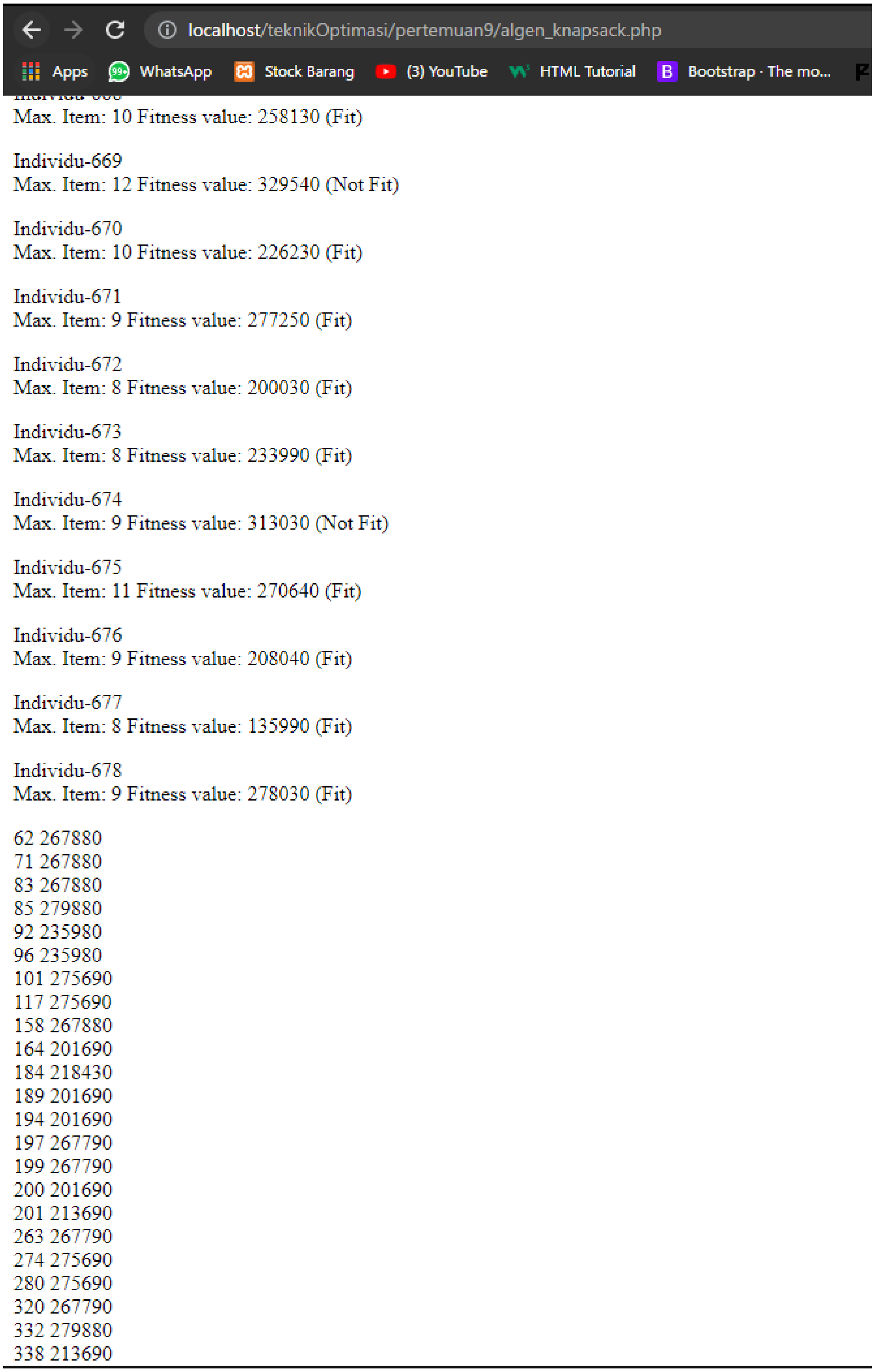
```php
        echo '<p></p>';
        print_r($selected);
    }

}

$initalPopulation = new Population;
$population = $initalPopulation->createRandomPopulation();

$fitness = new Fitness;
$fitness->fitnessEvaluation($population);

$crossover = new Crossover($population);
$crossoverOffsprings = $crossover->crossover();

//echo 'Crossover offsprings:<br>';
//print_r($crossoverOffsprings);

echo '<p></p>';
//(new Mutation($population))->mutation();
$mutation = new Mutation($population);
if ($mutation->mutation()){
    $mutationOffsprings = $mutation->mutation();
    //echo 'Mutation offspring<br>';
    //print_r($mutationOffsprings);
    //echo '<p></p>';
    foreach ($mutationOffsprings as $mutationOffspring){
        $crossoverOffsprings[] = $mutationOffspring;
    }
}
//echo 'Mutation offsprings <br>';
//print_r($crossoverOffsprings);
$fitness->fitnessEvaluation($crossoverOffsprings);

$selection = new Selection($population, $crossoverOffsprings);
$selection->selectingIndividus();


// $individu = new Individu;
// print_r($individu->createRandomIndividu());
```



```
Individu-0
Max. Item: 8 Fitness value: 167890 (Fit)

Individu-1
Max. Item: 12 Fitness value: 317540 (Not Fit)

Individu-2
Max. Item: 9 Fitness value: 233350 (Fit)

Individu-3
Max. Item: 9 Fitness value: 129790 (Fit)

Individu-4
Max. Item: 11 Fitness value: 217980 (Fit)

Individu-5
Max. Item: 11 Fitness value: 147690 (Fit)

Individu-6
Max. Item: 14 Fitness value: 361940 (Not Fit)

Individu-7
Max. Item: 9 Fitness value: 130880 (Fit)

Individu-8
Max. Item: 14 Fitness value: 326690 (Not Fit)

Individu-9
Max. Item: 11 Fitness value: 327920 (Not Fit)

Individu-10
Max. Item: 10 Fitness value: 186770 (Fit)

Individu-11
Max. Item: 14 Fitness value: 365690 (Not Fit)

Individu-12
Max. Item: 9 Fitness value: 228990 (Fit)

Individu-13
Max. Item: 13 Fitness value: 275690 (Fit)

Individu-14
Max. Item: 8 Fitness value: 201690 (Fit)

Individu-15
Max. Item: 11 Fitness value: 262640 (Fit)

Individu-16
Max. Item: 12 Fitness value: 247630 (Fit)

Individu-17
Max. Item: 12 Fitness value: 251720 (Fit)
```


localhost/teknikOptimasi/pertemuan9/algen_knapsack.php

```
Max. Item: 9 Fitness value: 235950 (Fit)

Individu-26
Max. Item: 12 Fitness value: 218940 (Fit)

Individu-27
Max. Item: 10 Fitness value: 270130 (Fit)

Individu-28
Max. Item: 8 Fitness value: 266130 (Fit)

Individu-29
Max. Item: 9 Fitness value: 176140 (Fit)


Best fitness value: 275690 Residual: 4310 Found

Individu-0
Max. Item: 10 Fitness value: 157640 (Fit)

Individu-1
Max. Item: 11 Fitness value: 289690 (Not Fit)

Individu-2
Max. Item: 9 Fitness value: 154640 (Fit)

Individu-3
Max. Item: 14 Fitness value: 380880 (Not Fit)

Individu-4
Max. Item: 10 Fitness value: 157640 (Fit)

Individu-5
Max. Item: 13 Fitness value: 307590 (Not Fit)

Individu-6
Max. Item: 13 Fitness value: 332540 (Not Fit)

Individu-7
Max. Item: 13 Fitness value: 346940 (Not Fit)
```



```
Max. Item: 10 Fitness value: 258130 (Fit)

Individu-669
Max. Item: 12 Fitness value: 329540 (Not Fit)

Individu-670
Max. Item: 10 Fitness value: 226230 (Fit)

Individu-671
Max. Item: 9 Fitness value: 277250 (Fit)

Individu-672
Max. Item: 8 Fitness value: 200030 (Fit)

Individu-673
Max. Item: 8 Fitness value: 233990 (Fit)

Individu-674
Max. Item: 9 Fitness value: 313030 (Not Fit)

Individu-675
Max. Item: 11 Fitness value: 270640 (Fit)

Individu-676
Max. Item: 9 Fitness value: 208040 (Fit)

Individu-677
Max. Item: 8 Fitness value: 135990 (Fit)

Individu-678
Max. Item: 9 Fitness value: 278030 (Fit)

62 267880
71 267880
83 267880
85 279880
92 235980
96 235980
101 275690
117 275690
158 267880
164 201690
184 218430
189 201690
194 201690
197 267790
199 267790
200 201690
201 213690
263 267790
274 275690
280 275690
320 267790
332 279880
338 213690
```