MOOC – Software testing

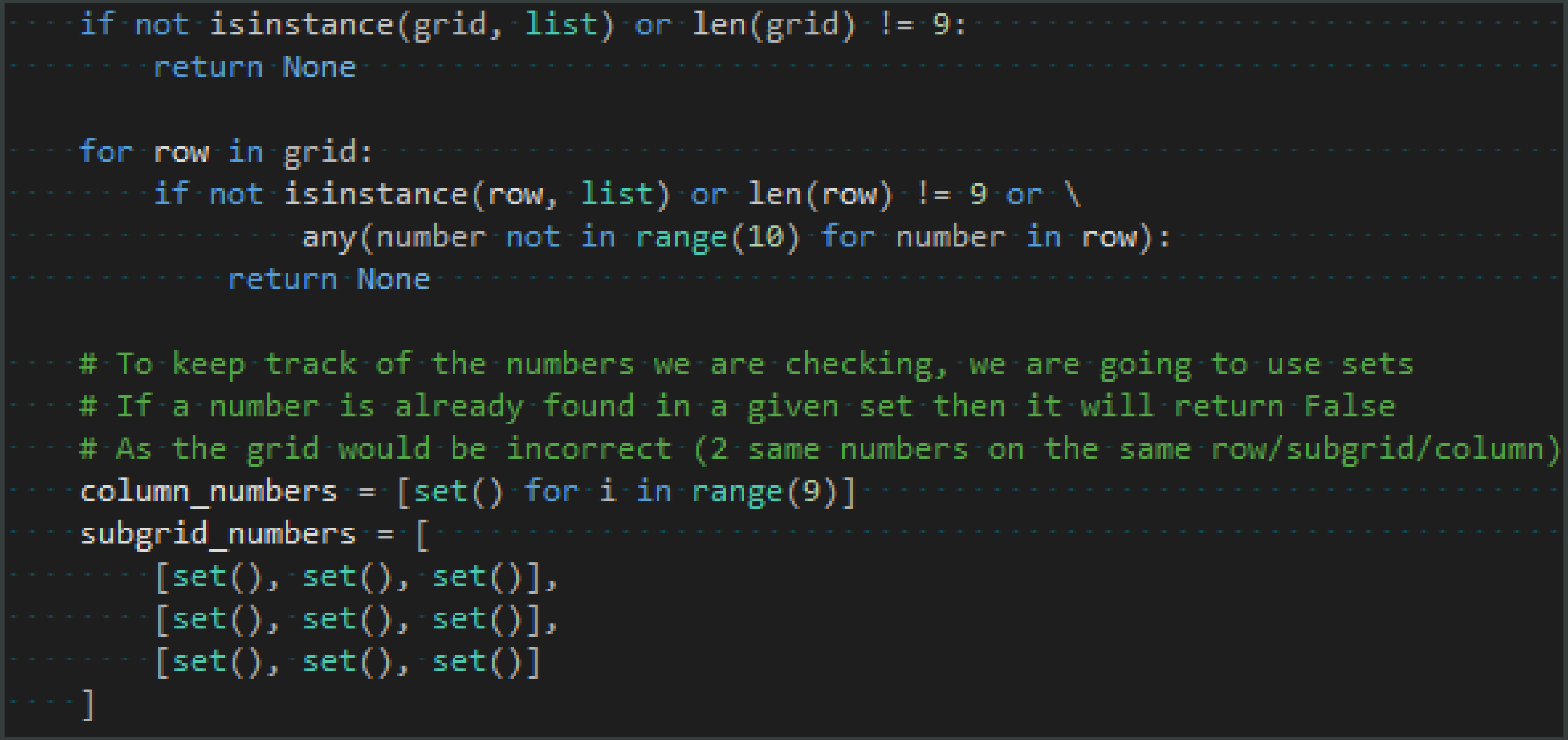
PRÉSENTATION DU SUDOKU CHECKER ET SOLVER

GENTILE DONATO

Sudoku Checker

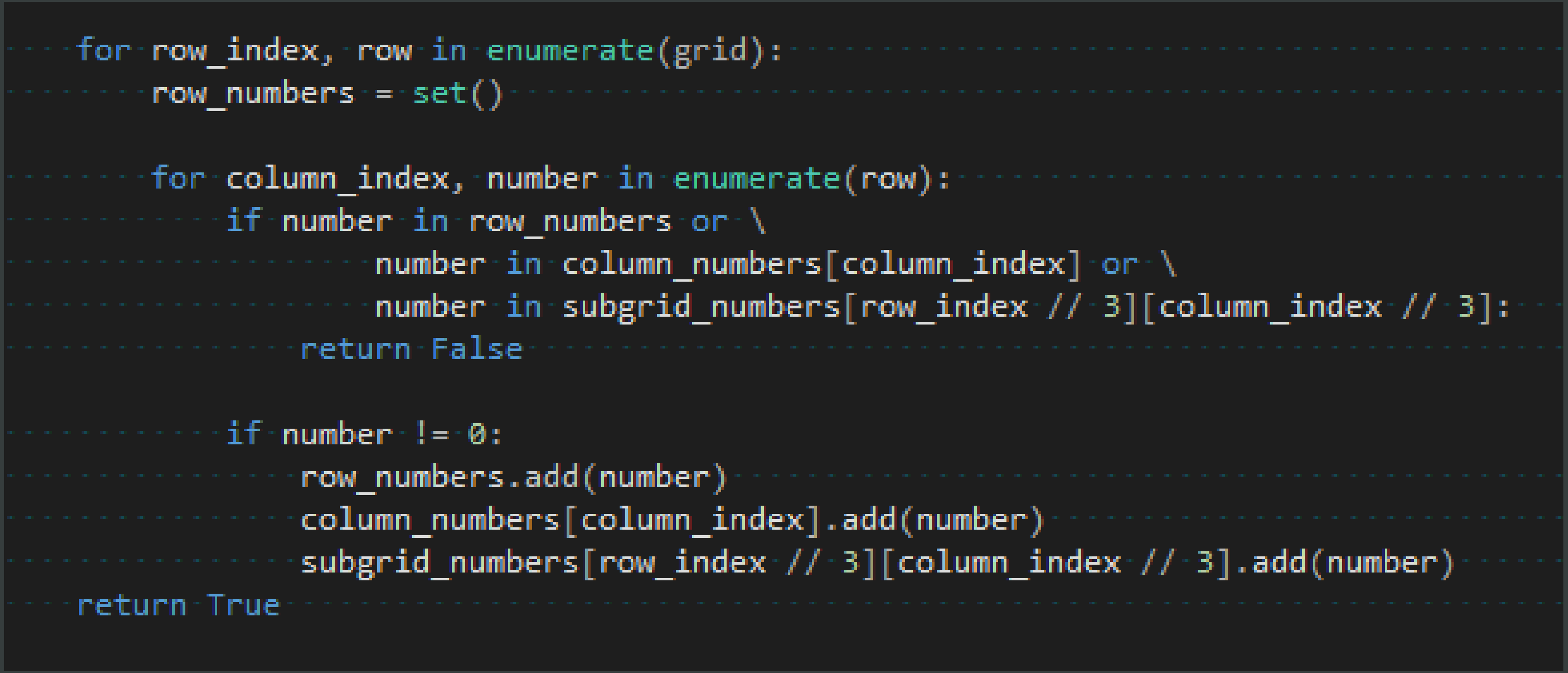
- Spécification du problème
- Algorithme
  - Est-ce que la grille est bien formée ?
    - Taille de la grille de 9 * 9
    - Grille ayant 81 chiffres
  - Est-ce que la grille est valide ?
    - Grilles avec des lignes et colonnes allant de 1 – 9 avec aucune répétition
    - Grilles avec sous-grilles allant de 1 – 9 avec aucune répétition dans chacune des sous-grilles

Un peu de code…



```
    if not isinstance(grid, list) or len(grid) != 9:
        return None

    for row in grid:
        if not isinstance(row, list) or len(row) != 9 or \
                any(number not in range(10) for number in row):
            return None

    # To keep track of the numbers we are checking, we are going to use sets
    # If a number is already found in a given set then it will return False
    # As the grid would be incorrect (2 same numbers on the same row/subgrid/column)
    column_numbers = [set() for i in range(9)]
    subgrid_numbers = [
        [set(), set(), set()],
        [set(), set(), set()],
        [set(), set(), set()]
    ]
```

Un peu de code…

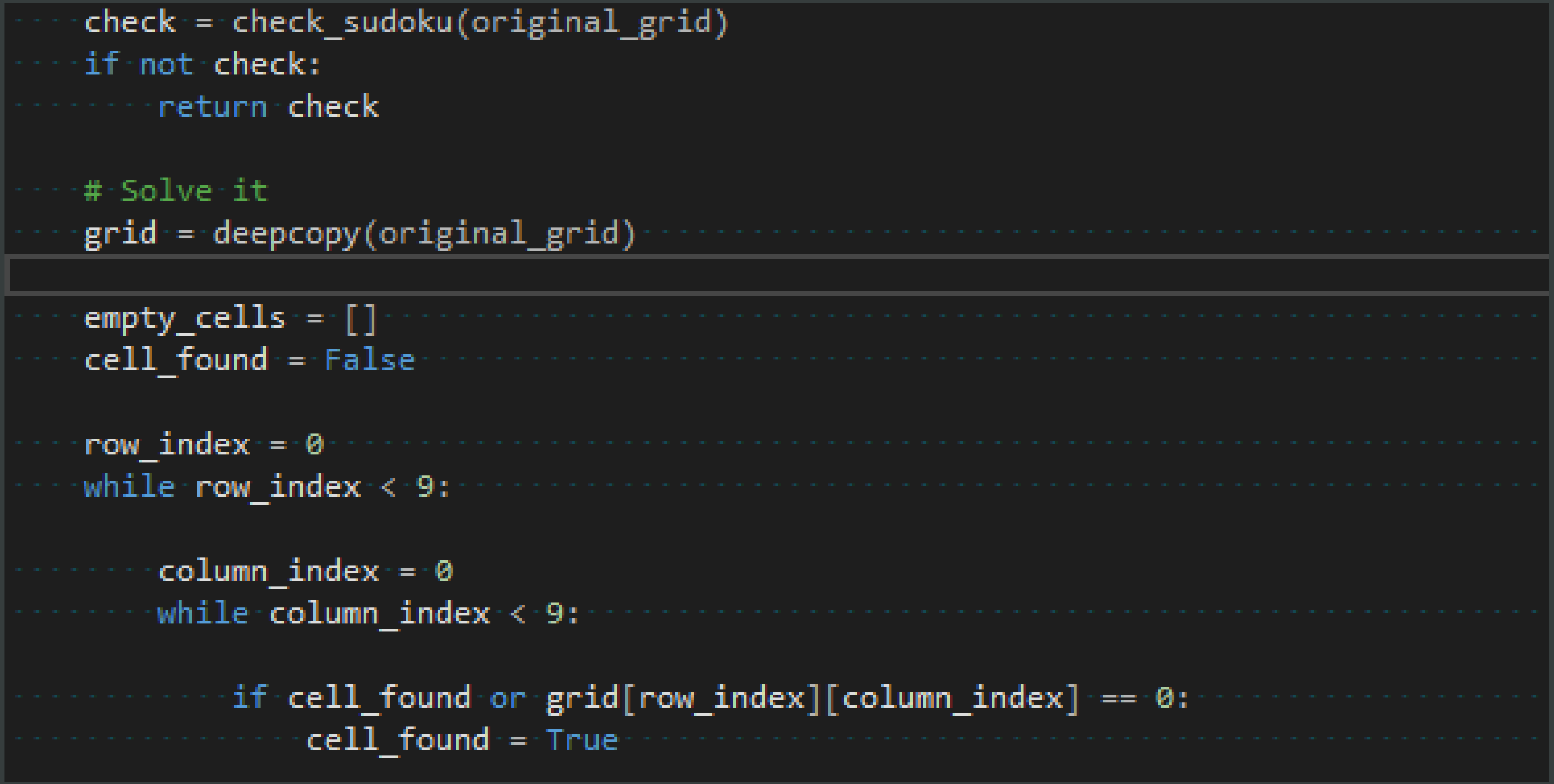


```
    for row_index, row in enumerate(grid):
        row_numbers = set()

        for column_index, number in enumerate(row):
            if number in row_numbers or \
                    number in column_numbers[column_index] or \
                    number in subgrid_numbers[row_index // 3][column_index // 3]:
                return False

            if number != 0:
                row_numbers.add(number)
                column_numbers[column_index].add(number)
                subgrid_numbers[row_index // 3][column_index // 3].add(number)
    return True
```

Sudoku Solver

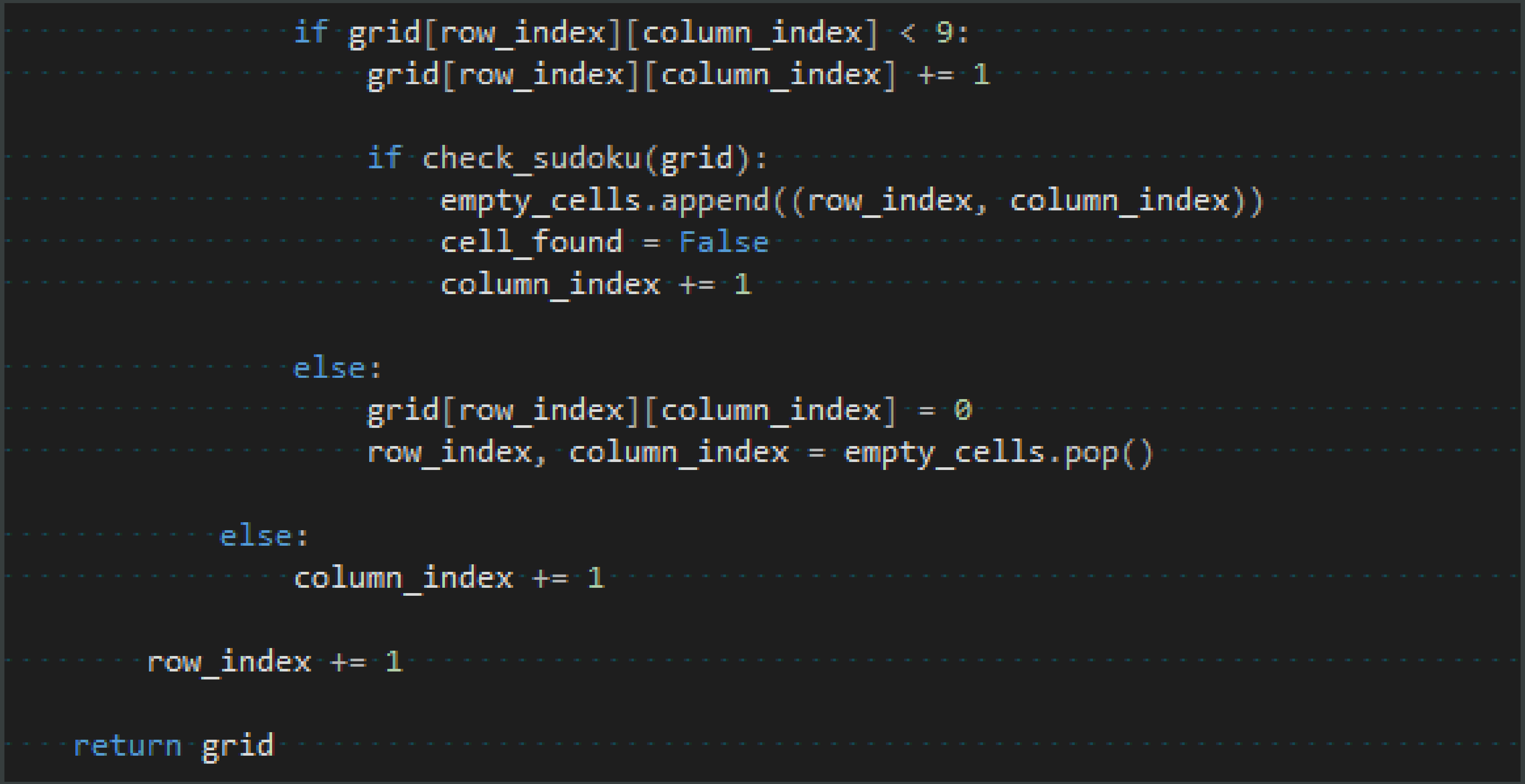
- Spécification du problème
- Algorithme
    - Création d’une copie de la grille d’origine
    - Création d’une stack pour le backtracking
    - Vérification de chaque ligne/colonne/sous-grille
    - Essais erreur jusqu’à trouver la solution
    - Peu être gourmand en ressources

Code du Sudoku Solver



```
    check = check_sudoku(original_grid)
    if not check:
        return check

    # Solve it
    grid = deepcopy(original_grid)

    empty_cells = []
    cell_found = False

    row_index = 0
    while row_index < 9:

        column_index = 0
        while column_index < 9:

            if cell_found or grid[row_index][column_index] == 0:
                cell_found = True
```

Code du Sudoku Solver

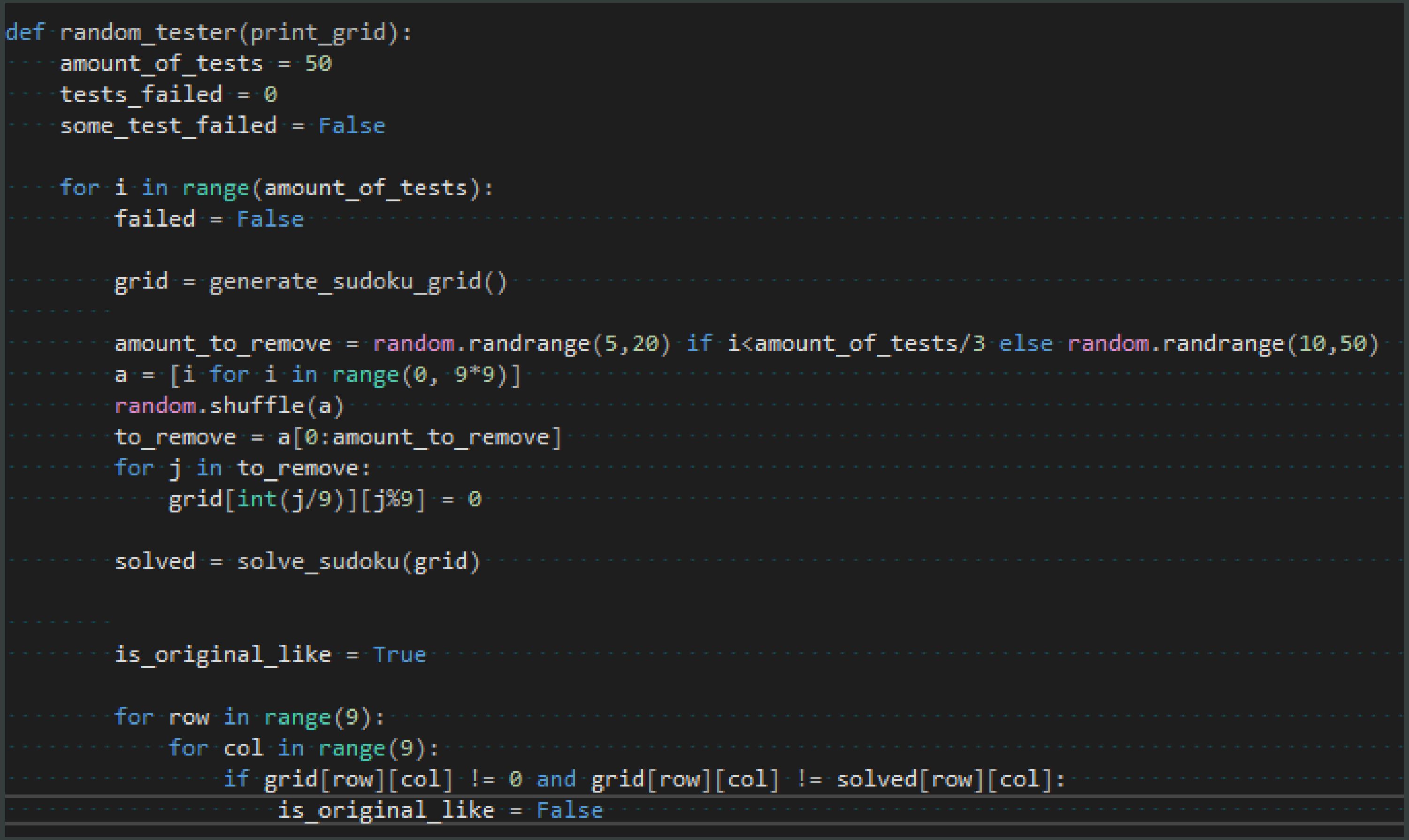


```
                if grid[row_index][column_index] < 9:
                    grid[row_index][column_index] += 1

                    if check_sudoku(grid):
                        empty_cells.append((row_index, column_index))
                        cell_found = False
                        column_index += 1

                else:
                    grid[row_index][column_index] = 0
                    row_index, column_index = empty_cells.pop()

            else:
                column_index += 1

        row_index += 1

    return grid
```

Random Tester

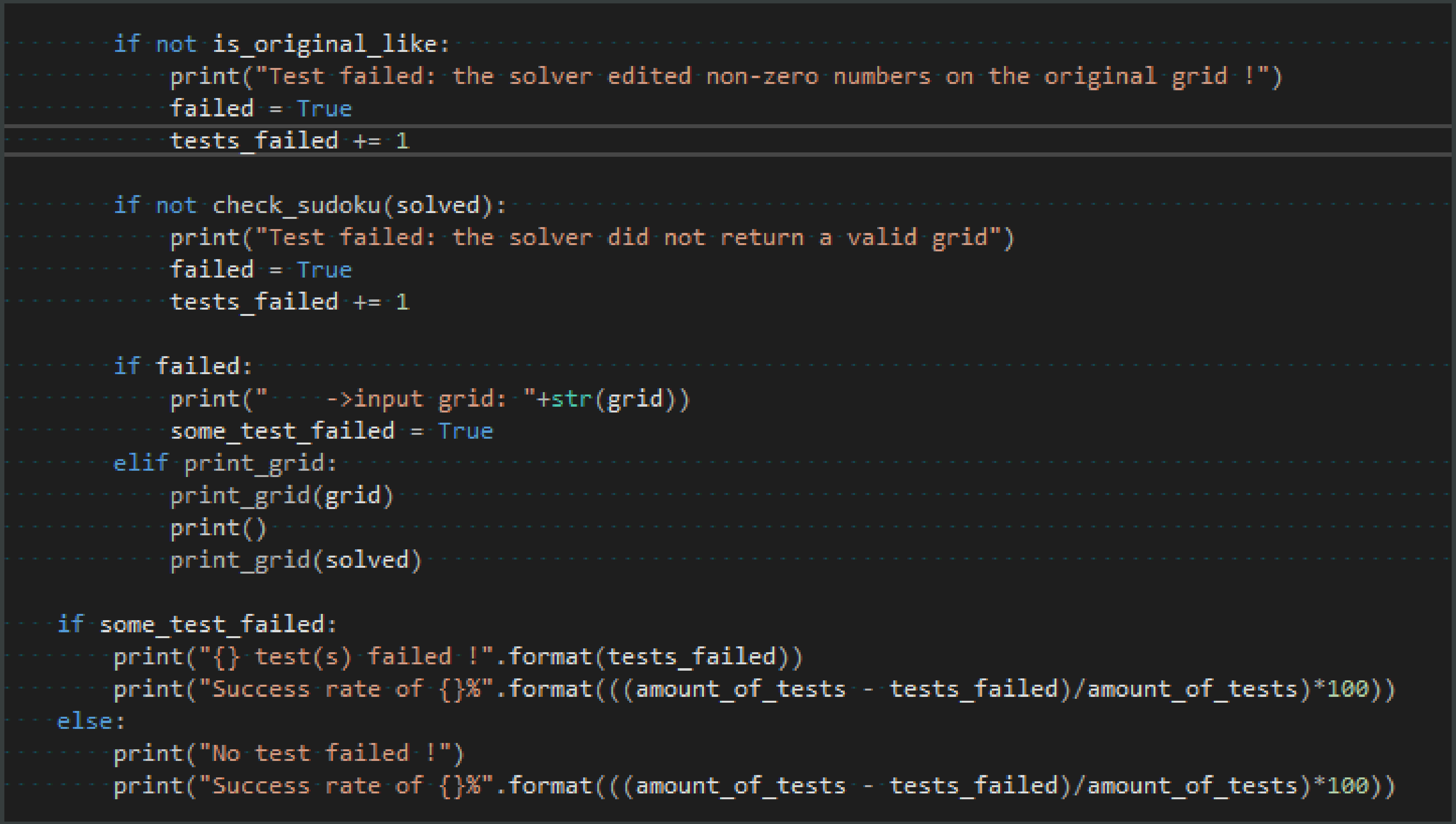
- Test de la fonction solve_sudoku
    - Générer une complète grille valide
    - Insérer des zéros
    - Résoudre la grille avec le solver
    - Vérifier que la grille soit valide et complétée
        - Elle doit être correct (check_sudoku)
        - Elle doit correspondre à la grille de départ

Code du Random Tester



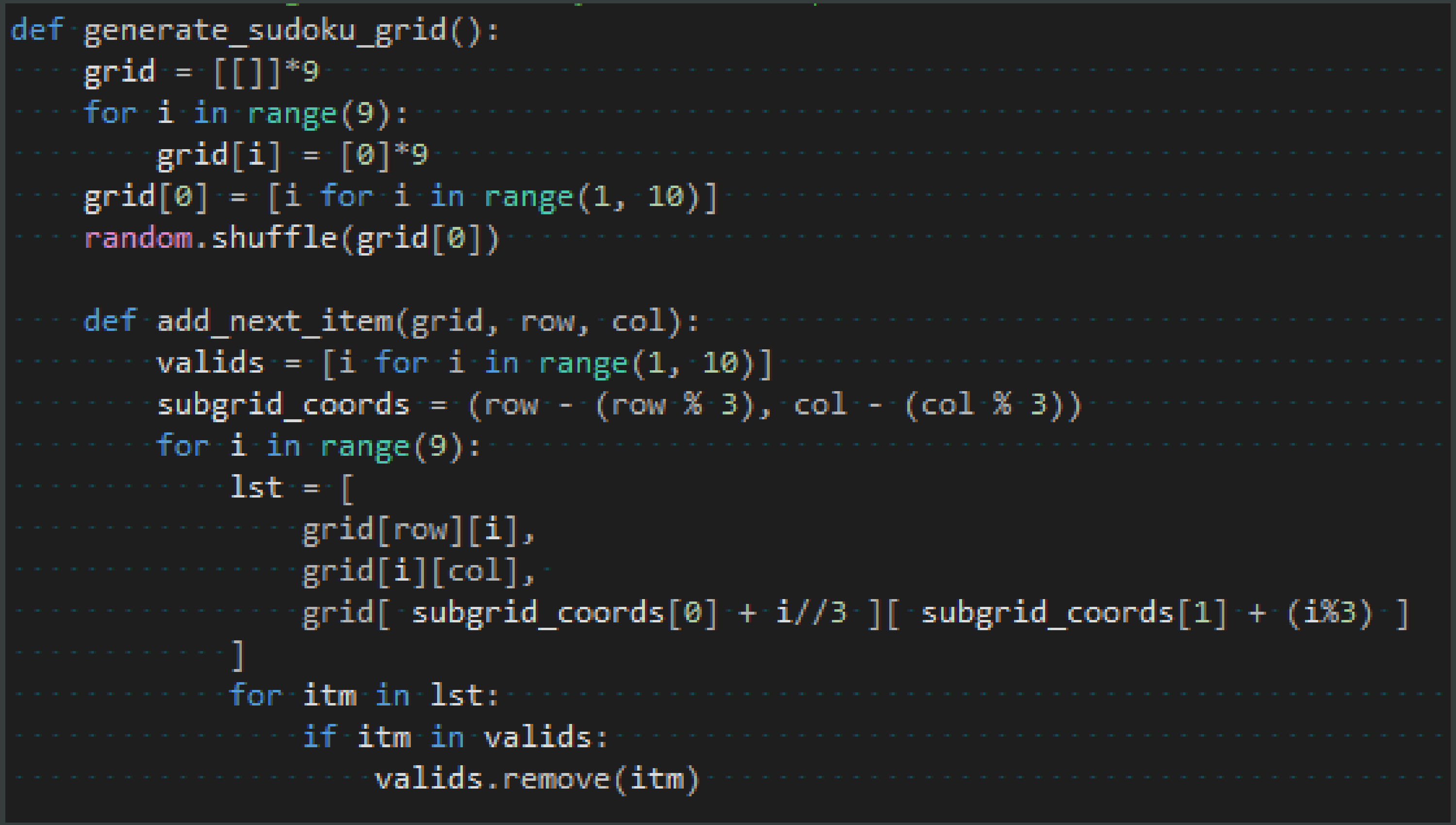
```python
def random_tester(print_grid):
    amount_of_tests = 50
    tests_failed = 0
    some_test_failed = False

    for i in range(amount_of_tests):
        failed = False

        grid = generate_sudoku_grid()

        amount_to_remove = random.randrange(5,20) if i<amount_of_tests/3 else random.randrange(10,50)
        a = [i for i in range(0, 9*9)]
        random.shuffle(a)
        to_remove = a[0:amount_to_remove]
        for j in to_remove:
            grid[int(j/9)][j%9] = 0

        solved = solve_sudoku(grid)


        is_original_like = True

        for row in range(9):
            for col in range(9):
                if grid[row][col] != 0 and grid[row][col] != solved[row][col]:
                    is_original_like = False
```

Code du Random Tester



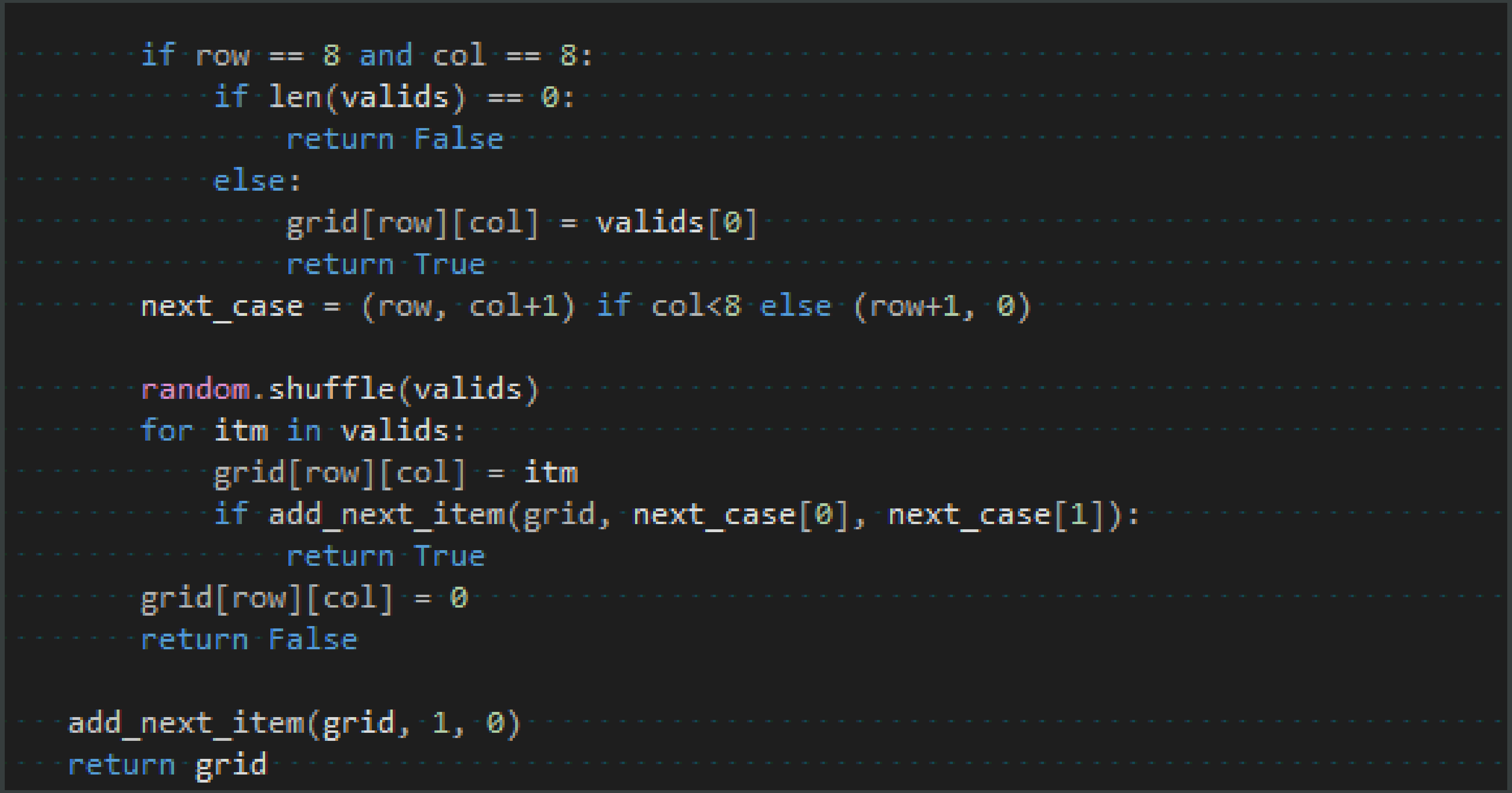
```
        if not is_original_like:
            print("Test failed: the solver edited non-zero numbers on the original grid !")
            failed = True
            tests_failed += 1

        if not check_sudoku(solved):
            print("Test failed: the solver did not return a valid grid")
            failed = True
            tests_failed += 1

        if failed:
            print("    ->input grid: "+str(grid))
            some_test_failed = True
        elif print_grid:
            print_grid(grid)
            print()
            print_grid(solved)

    if some_test_failed:
        print("{} test(s) failed !".format(tests_failed))
        print("Success rate of {}%".format(((amount_of_tests - tests_failed)/amount_of_tests)*100))
    else:
        print("No test failed !")
        print("Success rate of {}%".format(((amount_of_tests - tests_failed)/amount_of_tests)*100))
```

Code du Générateur de grilles



```python
def generate_sudoku_grid():
    grid = [[]]*9
    for i in range(9):
        grid[i] = [0]*9
    grid[0] = [i for i in range(1, 10)]
    random.shuffle(grid[0])

    def add_next_item(grid, row, col):
        valids = [i for i in range(1, 10)]
        subgrid_coords = (row - (row % 3), col - (col % 3))
        for i in range(9):
            lst = [
                grid[row][i],
                grid[i][col],
                grid[ subgrid_coords[0] + i//3 ][ subgrid_coords[1] + (i%3) ]
            ]
            for itm in lst:
                if itm in valids:
                    valids.remove(itm)
```

Code du Générateur de grilles



```python
        if row == 8 and col == 8:
            if len(valids) == 0:
                return False
            else:
                grid[row][col] = valids[0]
                return True
        next_case = (row, col+1) if col<8 else (row+1, 0)

        random.shuffle(valids)
        for itm in valids:
            grid[row][col] = itm
            if add_next_item(grid, next_case[0], next_case[1]):
                return True
        grid[row][col] = 0
        return False

    add_next_item(grid, 1, 0)
    return grid
```