ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

2η ΟΜΑΔΙΚΗ ΕΡΓΑΣΙΑ ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΚΑΙ ΕΜΠΕΙΡΑ ΣΥΣΤΗΜΑΤΑ – ΕΡΩΤΗΜΑ 1ο

ΓΚΟΛΕΜΙ ΚΡΙΣΤΙΑΝ, Π18029

ΑΛΕΞΑΝΔΡΟΣ ΓΚΙΝΕΤΣΙ, Π18028

1. Αναπτύξτε πρόγραμμα επίλυσης του προβλήματος χρωματισμού γράφου με χρήση γενετικών αλγορίθμων και γλώσσα προγραμματισμού της επιλογής σας. Χρησιμοποιείστε τυχαίο αρχικό πληθυσμό με πλήθος της δικής σας επιλογής.

Παρακάτω παρατίθεται ο κώδικας επίλυσης του προβλήματος, ο οποίος θα αναλυθεί στη συνέχεια:

```
from random import randint, choices, randrange, random
from collections import namedtuple
import time
import matplotlib.pyplot as plt
import numpy as np

Box = namedtuple('Box', ['name', 'value'])

"""
The closest the distance a box has from a box that we want to be filled, the higher is its value.
distance: 0 => value: 5
distance: 1 => value: 0
distance: 2 => value: 0
distance: 3 => value: 0
distance: 4 => value: 0
distance: 5 => value: 0

In order to calculate the distance between the boxes we use the Manhattan formula.
"""


boxes_K = [
    Box('1', 5),  Box('2', 0),  Box('3', 0),  Box('4', 0),  Box('5', 0),  Box('6', 0),  Box('7', 5),
    Box('8', 5),  Box('9', 0),  Box('10', 0), Box('11', 0), Box('12', 0), Box('13', 5), Box('14', 0),
    Box('15', 5), Box('16', 0), Box('17', 0), Box('18', 0), Box('19', 5), Box('20', 0), Box('21', 0),
    Box('22', 5), Box('23', 0), Box('24', 0), Box('25', 5), Box('26', 0), Box('27', 0), Box('28', 0),
    Box('29', 5), Box('30', 0), Box('31', 5), Box('32', 0), Box('33', 0), Box('34', 0), Box('35', 0),
    Box('36', 5), Box('37', 5), Box('38', 0), Box('39', 0), Box('40', 0), Box('41', 0), Box('42', 0),
    Box('43', 5), Box('44', 0), Box('45', 5), Box('46', 0), Box('47', 0), Box('48', 0), Box('49', 0),
    Box('50', 5), Box('51', 0), Box('52', 0), Box('53', 5), Box('54', 0), Box('55', 0), Box('56', 0),
    Box('57', 5), Box('58', 0), Box('59', 0), Box('60', 0), Box('61', 5), Box('62', 0), Box('63', 0),
    Box('64', 5), Box('65', 0), Box('66', 0), Box('67', 0), Box('68', 0), Box('69', 5), Box('70', 0),
    Box('71', 5), Box('72', 0), Box('73', 0), Box('74', 0), Box('75', 0), Box('76', 0), Box('77', 5)
]
```

```
boxes_G = [
    Box('1', 0),  Box('2', 0),  Box('3', 0),  Box('4', 0),  Box('5', 0),  Box('6', 0),  Box('7', 0),
    Box('8', 0),  Box('9', 5),  Box('10', 5), Box('11', 5), Box('12', 5), Box('13', 5), Box('14', 0),
    Box('15', 0), Box('16', 5), Box('17', 0), Box('18', 0), Box('19', 0), Box('20', 0), Box('21', 0),
    Box('22', 0), Box('23', 5), Box('24', 0), Box('25', 0), Box('26', 0), Box('27', 0), Box('28', 0),
    Box('29', 0), Box('30', 5), Box('31', 0), Box('32', 0), Box('33', 0), Box('34', 0), Box('35', 0),
    Box('36', 0), Box('37', 5), Box('38', 0), Box('39', 0), Box('40', 0), Box('41', 0), Box('42', 0),
    Box('43', 0), Box('44', 5), Box('45', 0), Box('46', 0), Box('47', 0), Box('48', 0), Box('49', 0),
    Box('50', 0), Box('51', 5), Box('52', 0), Box('53', 0), Box('54', 0), Box('55', 0), Box('56', 0),
    Box('57', 0), Box('58', 5), Box('59', 0), Box('60', 0), Box('61', 0), Box('62', 0), Box('63', 0),
    Box('64', 0), Box('65', 5), Box('66', 0), Box('67', 0), Box('68', 0), Box('69', 0), Box('70', 0),
    Box('71', 0), Box('72', 5), Box('73', 0), Box('74', 0), Box('75', 0), Box('76', 0), Box('77', 0)
]


def generate_solution():
    """
    Function to generate a possible solution.
    """
    # 1: the box is filled
    # 0: the box is empty
    random_solution = []

    for _ in range(77):
        random_solution.append(randint(0, 1))

    return random_solution


def generate_population(size):  # size represents how many random solutions we want to create
    """
    Function to generate a list of random solutions.

    size: Represents the number of random solutions we are creating
    """
    population = []

    for _ in range(size):
        rnd = generate_solution()
        population.append(rnd)

    return population
```

```python
def fitness(sol, boxes):
    """
    Function to evaluate the solutions.
    """
    filled = 0  # nr of boxes currently filled
    value = 0

    for i, box in enumerate(boxes):
        if sol[i] == 1:
            filled += 1
            value += box.value

            if filled > 77:
                return 0

    return value  # the higher the value, the better the solution is


def select(population, boxes):
    """
    Function to select the solutions to be paired in order to create the next generation.
    """
    possibility = [fitness(sol, boxes) for sol in population]

    # "weights = possibility" means that solutions with a better fitness score have a higher possibility to be selected as parents
    # "k = 2" means that we draw twice from the populations as two parents are needed
    return choices(population = population, weights = possibility, k = 2)


def crossover(sol_a, sol_b):
    """
    Crossover function - defines how two solutions will be paired.
    """
    p = randint(1, 76)  # the length of each solution is 77, so we randomly pick a position between 1 and 76 to cut the solution into two parts

    return sol_a[0:p] + sol_b[p:], sol_b[0:p] + sol_a[p:]
```

```
def mutation(sol):
    """
    Mutation function - takes as a parameter a possible solution and with a certain probability
    changes 0's to 1's and 1's to 0's in random positions.
    """
    probability = 0.5
    index = randrange(77)
    sol[index] = sol[index] if random() > probability else abs(sol[index] - 1)

    return sol


def run_evolution(boxes, fitness_limit):
    """
    Function that actually runs the evolution, using the predefined methods.
    """
    generation_limit = 10
    population = generate_population(100)

    for _ in range(generation_limit):
        while (fitness(population[0], boxes) != fitness_limit):
            # sorting the current population based to the fitness of each solution so that solutions with better fitness score are first
            population = sorted(
                population,
                key = lambda solution: fitness(solution, boxes),
                reverse = True
                )

            # check if we have reached the fitness_limit, which would mean that the goal is accomplished
            if fitness(population[0], boxes) >= fitness_limit:
                break

            # elitism - keep the best two solutions from the previous generation
            next_generation = population[0:2]

            # time to generate all the other new solutions
            for _ in range(int(len(population) / 2) - 1):
                parents = select(population, boxes)
                offspring_a, offspring_b = crossover(parents[0], parents[1])
                offspring_a = mutation(offspring_a)
                offspring_b = mutation(offspring_b)
                next_generation += [offspring_a, offspring_b]

            population = next_generation
```

```
        population = sorted(
            population,
            key = lambda solution: fitness(solution, boxes),
            reverse = True
        )

    return population


def print_board(sol, boxes):
    """
    Function to print the board with the wanted letter.
    """
    board = np.array(sol)

    for i, box in enumerate(boxes):
        if box.value == 0:
            board[i] = 0

    new_board = board.reshape(11, 7)    # turn the 1d array to 2d

    plt.figure(figsize=(4, 4))
    plt.imshow(new_board, cmap='Greys')
    plt.axis(False)
    plt.show()


def choose():
    """
    Function that gives the user the ability to choose the letter he wants to print.
    The user can choose between the letter 'K' and the greek letter 'G'.
    """
    press = input("---CHOOSE THE LETTER YOU WANT TO PRINT. ENTER 'K' OR 'G' IN ORDER TO PRINT THE CORRESPONDING LETTER: ")
    # take only the first letter from the user's input
    press = press[0].lower()

    # check for invalid input
    while press != 'k' and press != 'g':
        press = input("Invalid input! Try again: ")
        press = press[0].lower()

    if press == 'k':
        return True
    else:
        return False
```

```
def main():
    """
    Main function.
    """
    choice = choose()
    if choice:
        population = run_evolution(boxes_K, 110)
        print_board(population[0], boxes_K)
    else:
        population = run_evolution(boxes_G, 70)
        print_board(population[0], boxes_G)


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

Ανάλυση των παραπάνω συναρτήσεων αναλυτικά:

def choose():

Η παραπάνω συνάρτηση δίνει στον χρήστη τη δυνατότητα να επιλέξει ποίο γράμμα θέλει να τυπωθεί χρησιμοποιώντας τον γενετικό αλγόριθμο που έχει υλοποιηθεί. Ο χρήστης έχει δύο επιλογές – το γράμμα ‘Κ’ και το γράμμα ‘Γ’. Εφόσον ο χρήστης πιέσει k ή K στο πληκτρολόγιο του, η συνάρτηση επιστρέφει την τιμή True και τυπώνεται το γράμμα ‘Κ’, ενώ εάν πιέσει g ή G, η συνάρτηση επιστρέφει False και τυπώνεται το γράμμα ‘Γ’.

def generate_solution():

Η παραπάνω συνάρτηση δημιουργεί τυχαίες λύσεις για το πρόβλημα. Οι τυχαίες λύσεις αναπαρίστανται με δυαδικά ψηφία και επιστρέφονται σε μορφή μιας λίστας με 77 στοιχεία. Ο γράφος που χρησιμοποιείται για να τυπωθούν τα γράμματα είναι διαστάσεων 11*7, επομένως για αυτό τον λόγο υπάρχουν 77 στοιχεία στη λίστα με τις τυχαίες λύσεις. Κάθε στοιχείο είναι είτε 0 είτε 1 – το 0 σημαίνει ότι το κουτάκι στην αντίστοιχη θέση είναι λευκό, ενώ το 1 σημαίνει ότι το κουτάκι στην αντίστοιχη θέση είναι χρωματισμένο.

def generate_population(size):

Η συνάρτηση *generate_population()* χρησιμοποιείται για τη δημιουργία ενός πληθυσμού από τυχαίες λύσεις, καλώντας διαδοχικά όσες φορές ορίσουμε εμείς μέσω της μεταβλητής *size* τη συνάρτηση generate_solution(). Επιστρέφει μια λίστα με έναν αριθμό απο τυχαίες λύσεις, την *population[].*

def fitness(sol, boxes):

Η συνάρτηση *fitness()* χρησιμοποιείται για την εκτίμηση της καταλληλότητας κάθε πιθανής λύσης. Κάθε ένα από τα ''κουτάκια'' του γράφου έχει μια συγκεκριμένη αξία με βάση την απόσταση που βρίσκεται από το κοντινότερο ''κουτάκι'' που θα θέλαμε να είναι χρωματισμένο για την επίτευξη του στόχου. Όσα κουτάκια θα θέλαμε να είναι τελικά χρωματισμένα έχουν ενδεικτικά αξία 5, ενώ τα υπόλοιπα κουτάκια έχουν αξία 0.

```
for i, box in enumerate(boxes):
    if sol[i] == 1:
        filled += 1
        value += box.value

        if filled > 77:
            return 0

return value  # the higher the value, the better the solution is
```

Με το παραπάνω κομμάτι κώδικα η συνάρτηση ελέγχει διαδοχικά αν το κάθε κουτάκι στην τυχαία λύση που της δίνεται ως όρισμα είναι χρωματισμένο. Σε περίπτωση που το κουτάκι που ελέγχεται είναι χρωματισμένο, η μεταβλητή *value* αυξάνεται κατά 5 που είναι και η αξία για κάθε χρωματισμένο κουτάκι ενώ γίνεται παράλληλα έλεγχος αν έχουν χρωματιστεί παραπάνω κουτάκια από όσα υπάρχουν μέσω της μεταβλητής *filled.* Τελικά επιστρέφεται η μεταβλητή *value* η οποία αναπαριστά την καταλληλότητα της λύσης – όσο πιο κοντά είναι η μεταβλητή value στον επιθυμητό αριθμό *, τόσο καλύτερη είναι η λύση.

* για το γράμμα Κ ιδανικά *value = 110* αφού θα πρέπει να χρωματιστούν 22 κουτάκια (22 * 5 = 110), ενώ για το γράμμα Γ ιδανικά *value = 70* (14 * 5 = 70).

def select(population, boxes):

Η συνάρτηση select() παίρνει ως όρισμα έναν τυχαίο πληθυσμό από τυχαίες λύσεις καθώς και τη λίστα με τα κουτάκια που αντιστοιχούν είτε στο γράμμα ''Κ'' είτε στο γράμμα ''Γ'' ανάλογα με το τι επιλέξει ο χρήστης. Από αυτόν τον πληθυσμό επιλέγει δύο λύσεις και τις επιστρέφει με σκοπό αυτές οι λύσεις αργότερα να συνδυαστούν ακολουθώντας τη διαδικασία του γενετικού αλγορίθμου.

```
return choices(population = population, weights = possibility, k = 2)
```

Οι λύσεις δεν επιλέγονται εντελώς τυχαία αλλά ανάλογα με το σκορ που έχει η κάθε λύση από τη συνάρτηση καταλληλότητας fitness(). Για την επιλογή των τυχαίων γονέων χρησιμοποιείται η έτοιμη συνάρτηση random.choices() με ορίσματα τον πληθυσμό population που δίνεται,

```
possibility = [fitness(sol, boxes) for sol in population]
```

weights = possibility που ορίζει ότι είναι πιο πιθανό να επιλεχθούν λύσεις με μεγαλύτερο fitness score και k = 2, που ορίζει ότι θα επιλεχθούν μόνο δύο λύσεις αφού χρειάζονται δύο γονείς για να γίνει αναπαραγωγή.

def crossover(sol_a, sol_b):

Η παραπάνω συνάρτηση παίρνει ως όρισμα δύο λύσεις και ορίζει το πώς θα αναπαραχθούν, ώστε να προκύψουν οι απόγονοι τους για την εκτέλεση του γενετικού αλγορίθμου. Αρχικά, επιλέγει έναν τυχαίο αριθμό απο 1 εώς 76 – όσα είναι και τα στοιχεία μιας τυχαίας λύσης και επιστρέφει δύο καινούργιες λύσεις. Η πρώτη περιέχει το πρώτο κομμάτι της $1^{ης}$ αρχικής λύσης που δόθηκε ως όρισμα μέχρι και το στοιχείο που επιλέχθηκε τυχαία και τα υπόλοιπα στοιχεία της $2^{ης}$ αρχικής λύσης από το τυχαίο στοιχείο και μετά. Αντίστοιχα, αλλά με αντίστροφο τρόπο δημιουργείται και ο $2^{ος}$ απόγονος.

def mutation(sol):

Η συνάρτηση *mutation(),* παίρνει ως όρισμα μια λύση και εκτελέι τη μετάλλαξή της. Επιλέγει ένα τυχαίο στοιχείο της λύσης και αν η πιθανότητα (ένας αριθμός που ορίζουμε εμείς *float = 0.5*) είναι μεγαλύτερη από έναν

τυχαίο αριθμό που δημιουργούμε, αλλάζει το στοιχείο απο 0 σε 1 και αντίστροφα, ενώ αν η πιθανότητα είναι μικρότερη από τον τυχαίο αριθμό αφήνει το στοιχείο ως έχει.

def print_board(sol, boxes):

Η συνάρτηση *print_board()* είναι αυτή που εμφανίζει τον πίνακα με βάση τη λύση που της δίνεται ως όρισμα. Ουσιαστικά μετατρέπει τα δυαδικά ψηφία της λύσης που της δίνεται σε χρωματισμένα ή μη κουτάκια για την τύπωση του επιθυμητού γράμματος. Για να γίνει αυτό χρησιμοποιούνται οι έτοιμες βιβλιοθήκες της python, matplotlib και numpy.

def run_evolution(boxes, fitness_limit):

Συνάρτηση η οποία ουσιαστικά εκτελεί τον γενετικό αλγόριθμο με βάση τα βήματα του.

```
generation_limit = 10
population = generate_population(100)
```

Αρχικά, ορίζουμε ότι θέλουμε να υπάρξουν μέχρι 10 γενιές για να λυθεί το πρόβλημα (μεχρι 10 δημιουργίες νέων πληθυσμών) και ορίζουμε έναν αρχικό πληθυσμό με 100 διαφορετικές τυχαίες λύσεις καλώντας τη συνάρτηση generate_population(100).

Όσο δεν έχουμε ξεπεράσει τις 10 γενιές και όσο δεν έχουμε φτάσει ακόμα το επιθυμητό fitness (110 και 70 για Κ και Γ αντίστοιχα) εκτελούνται διαδοχικά τα παρακάτω βήματα:

```
    population = sorted(
      population,
      key = lambda solution: fitness(solution, boxes),
      reverse = True
      )
```

- Σορτάρισμα του πληθυσμού με βάση το fitness σκορ κάθε λύσης – οι λύσεις με μεγαλύτερο σκορ θα έρθουν πρώτες.

```
    if fitness(population[0], boxes) >= fitness_limit:
      break
```

- Έλεγχος αν το πρώτο στοιχείο του συγκεκριμένου πληθυσμού (που θα έχει και το καλύτερο σκορ) είναι λύση του προβλήματος – ο έλεγχος γίνεται με βάση το σκορ του. Αν το σκορ του είναι ίδιο με αυτό που θέλουμε να πετύχουμε σημαίνει ότι λύσαμε το πρόβλημα και βγαίνουμε από τον βρόγχο επιστρέφοντας τη λύση.

```
next_generation = population[0:2]
```

- Κρατάμε τις δύο καλύτερες λύσεις του προηγούμενου πληθυσμού και τις μεταφέρουμε στον επόμενο (ελιτισμός ώστε να μην χάσουμε τυχόν καλές λύσεις ενώ μεταβαίνουμε στην επόμενη γενιά).

```
    # time to generate all the other new solutions
    for _ in range(int(len(population) / 2) - 1):
        parents = select(population, boxes)
        offspring_a, offspring_b = crossover(parents[0], parents[1])
        offspring_a = mutation(offspring_a)
        offspring_b = mutation(offspring_b)
        next_generation += [offspring_a, offspring_b]
```

- Δημιουργία νέου πληθυσμού μέσω αναπαραγωγής γονέων που επιλέγονται από τη συνάρτηση select με τον τωρινό πληθυσμό και έπειτα μετάλλαξης των απογόνων που προκύπτουν.
- Τέλος σορτάρουμε και πάλι τον νέο πληθυσμό που προκύπτει με τον ίδιο τρόπο που σορτάραμε αρχικά και τον επιστρέφουμε. Σε περίπτωση που δεν έχει βρεθεί η λύση που θέλουμε ο γενετικός αλγόριθμος θα εκτελείται με τα παραπάνω βήματα μέχρις ώτου να βρεθεί η επιθυμητή λύση.

def main():

Η main συνάρτηση του προβλήματος που καλεί τη συνάρτηση *run_evolution()* με τα κατάλληλα ορίσματα ανάλογα με το αν ο χρήστης επέλεξε να τυπώσει το γράμμα Κ ή το Γ.

```
Box = namedtuple('Box', ['name', 'value'])
```

Ορίζουμε ότι κάθε κουτί του γράφου θα έχει ένα όνομα (ανάλογα με την αρίθμηση) καθώς και μια αξία. Υπάρχουν δύο λίστες με 77 κουτάκια, μια για το γράμμα Κ και μια για το γράμμα Γ. Ανάλογα με την επιλογή του χρήστη κάθε φορά καλείται η αντίστοιχη λίστα.

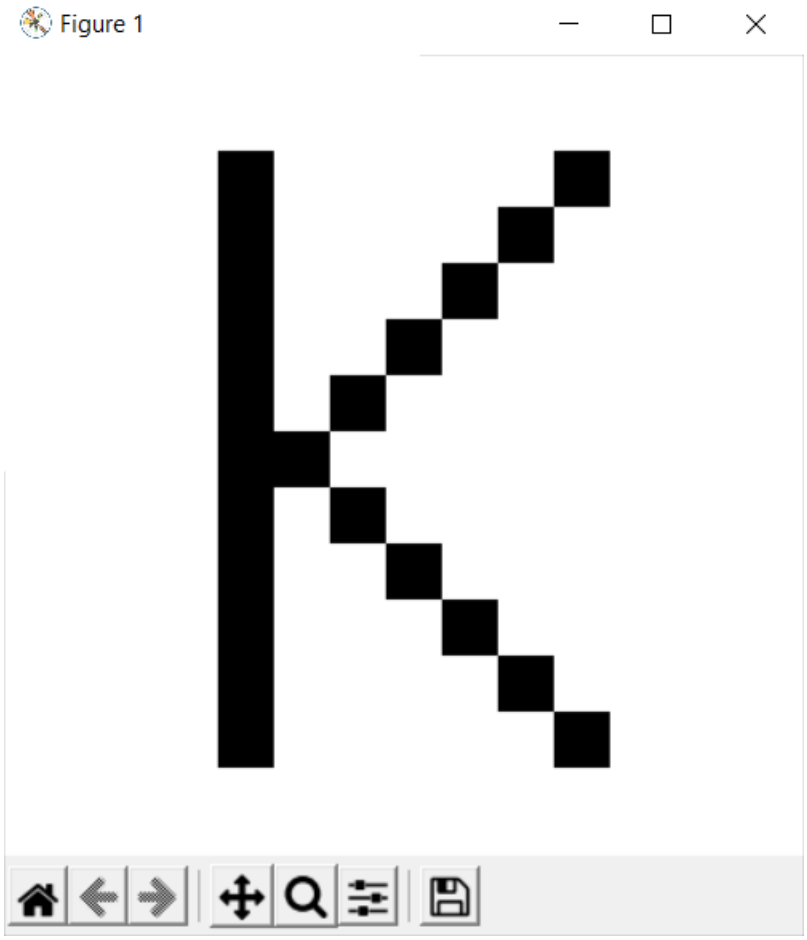
Παρακάτω παρατίθονται η λίστες για τα στοιχεία του γράμματος Κ και Γ αντίστοιχα.

```
boxes_K = [
    Box('1', 5),  Box('2', 0),  Box('3', 0),  Box('4', 0),  Box('5', 0),  Box('6', 0),  Box('7', 5),
    Box('8', 5),  Box('9', 0),  Box('10', 0), Box('11', 0), Box('12', 0), Box('13', 5), Box('14', 0),
    Box('15', 5), Box('16', 0), Box('17', 0), Box('18', 0), Box('19', 5), Box('20', 0), Box('21', 0),
    Box('22', 5), Box('23', 0), Box('24', 0), Box('25', 5), Box('26', 0), Box('27', 0), Box('28', 0),
    Box('29', 5), Box('30', 0), Box('31', 5), Box('32', 0), Box('33', 0), Box('34', 0), Box('35', 0),
    Box('36', 5), Box('37', 5), Box('38', 0), Box('39', 0), Box('40', 0), Box('41', 0), Box('42', 0),
    Box('43', 5), Box('44', 0), Box('45', 5), Box('46', 0), Box('47', 0), Box('48', 0), Box('49', 0),
    Box('50', 5), Box('51', 0), Box('52', 0), Box('53', 5), Box('54', 0), Box('55', 0), Box('56', 0),
    Box('57', 5), Box('58', 0), Box('59', 0), Box('60', 0), Box('61', 5), Box('62', 0), Box('63', 0),
    Box('64', 5), Box('65', 0), Box('66', 0), Box('67', 0), Box('68', 0), Box('69', 5), Box('70', 0),
    Box('71', 5), Box('72', 0), Box('73', 0), Box('74', 0), Box('75', 0), Box('76', 0), Box('77', 5)
]
```

```
boxes_G = [
    Box('1', 0),  Box('2', 0),  Box('3', 0),  Box('4', 0),  Box('5', 0),  Box('6', 0),  Box('7', 0),
    Box('8', 0),  Box('9', 5),  Box('10', 5), Box('11', 5), Box('12', 5), Box('13', 5), Box('14', 0),
    Box('15', 0), Box('16', 5), Box('17', 0), Box('18', 0), Box('19', 0), Box('20', 0), Box('21', 0),
    Box('22', 0), Box('23', 5), Box('24', 0), Box('25', 0), Box('26', 0), Box('27', 0), Box('28', 0),
    Box('29', 0), Box('30', 5), Box('31', 0), Box('32', 0), Box('33', 0), Box('34', 0), Box('35', 0),
    Box('36', 0), Box('37', 5), Box('38', 0), Box('39', 0), Box('40', 0), Box('41', 0), Box('42', 0),
    Box('43', 0), Box('44', 5), Box('45', 0), Box('46', 0), Box('47', 0), Box('48', 0), Box('49', 0),
    Box('50', 0), Box('51', 5), Box('52', 0), Box('53', 0), Box('54', 0), Box('55', 0), Box('56', 0),
    Box('57', 0), Box('58', 5), Box('59', 0), Box('60', 0), Box('61', 0), Box('62', 0), Box('63', 0),
    Box('64', 0), Box('65', 5), Box('66', 0), Box('67', 0), Box('68', 0), Box('69', 0), Box('70', 0),
    Box('71', 0), Box('72', 5), Box('73', 0), Box('74', 0), Box('75', 0), Box('76', 0), Box('77', 0)
]
```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ:

```
S C:\Users\golem\OneDrive - unipi.gr\Python> & C:/Users/golem/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/golem/OneDrive - unipi.gr/Python/printLetter_GA.py"
--CHOOSE THE LETTER YOU WANT TO PRINT. ENTER 'K' OR 'G' IN ORDER TO PRINT THE CORRESPONDING LETTER: k
```



```
PS C:\Users\golem\OneDrive - unipi.gr\Python> python -u "c:\Users\golem\OneDrive - unipi.gr\Python\printLetter_GA.py"
---CHOOSE THE LETTER YOU WANT TO PRINT. ENTER 'K' OR 'G' IN ORDER TO PRINT THE CORRESPONDING LETTER: g
```