מונחה עצמים, מדעי המחשב – **פתרון לדוגמא**

מבחן מועד א' – 24/1/2021

שי אהרן, בעז בן משה

הנחיות כלליות:

- בבחינה זו 4 שאלות כולן חובה, כל שאלה 25 נקודות.
- משך הבחינה הוא שעתיים וחצי (2.5 שעות, 150 דקות).
- אסור חומר עזר: אסור שימוש כלשהו ברשת, בפרט אסור להשתמש באינטרנט, במחשבים, בספרים או בכל חומר כתוב.
- הבחינה נערכת על דף: מילוי התשובות יעשה על גבי מחברת הבחינה בלבד.
- לבחינה זו מצורפים קטעי קוד, אם לא נאמר בפירוש אחרת, ניתן להשתמש בהם לפתרון כל אחד מהסעיפים.

בבחינה זו 6 עמודים:

- עמוד 1: הנחיות כלליות לבחינה
- עמוד 2-3: שאלות 1,2,3,4
- עמודים 4-6 נספח קוד (פורסם מראש)

נספח קוד (עמודים 4-6) מצורף לבחינה (פורסם מראש):

- ממשק GraphInterface ב java וב python שמייצג גרף מכוון ממושקל.
  ממשק GraphAlgoInterface ב java וב python שמייצג מספר אלגוריתמים על גרפים מכוונים וממושקלים.
- דוגמאת קוד של המחלקה Point2D: ב java, ומחלקת בדיקה בסיסית.
- דוגמאת קוד של המחלקה Point2D: ב python, וקובץ בדיקה בסיסי.
- דוגמאת קוד בסיסית ב python לשימוש במבני נתונים קיימים.

בהצלחה!!

שאלה 1 (25 נקודות):

בשאלה זו נעסוק בייצוג מידע בפורמט JSON,

1.1 (5 נקודות) הסבירו בקצרה (ללא קוד) מהו המבנה של הפורמט JSON, ואילו סוגי טיפוסי נתונים ניתן לשמור בעזרת הפורמט.

1.2 (5 נקודות) הסבירו באופן כללי (ללא קוד) דרך כיצד ניתן לבדוק אם שני אובייקטים מסוג JSON מייצגים אותו מידע.

1.3 (10 נקודות) השלימו את הפונקציה is_eq ב Python שבודקת האם שני אובייקטים מסוג JSON הם שווים. משמע, מייצגים את אותו מידע, הדרכה: ניתן בהחלט לכתוב פונקציות עזר.

```
import json
# JSON string
json_1 = '{"1" : "a", "2" : "b"}'
json_2 = '{"2" : "b" ,"1" : "a"}'
# Convert string into Python dictionary
json1_dict = json.loads(json_1)
json2_dict = json.loads(json_2)
# add your code below



def is_eq(json1,json2):  # add your code below



is_eq(json1_dict, json2_dict) # should be true
```

1.4 (5 נקודות) כתבו את מחלקת הבדיקה: Test_json_eq(unittest.TestCase) שמאפשרת בדיקת נכונות לפונקציה is_eq שכתבתם.

תשובה:

1.1 פורמט JSON הינו פורמט טקסטואלי שמבוסס על סוגריים מאוזנים וזוגות של שם נתון וערך ומפרדים ע"י נקודתיים, לדוגמא:

```
{
 "course": "OOP",
 "metadata": {
        "University": "Ariel",
        "number": 1234,
        },
 "links": [
       "www.github.com",
       "python.org"
    ]
}
```

הפורמט מאפשר שמירת נתונים כגון: מספרים שלמים, ממשיים, אמת או שקר, מחרוזות, מערכים ואובייקטים – שמייצגים היררכיה של נתונים (ראו למשל את המערך "links" ואת האובייקט "metadata" מעלה). המימוש עצמו של הפורמט הוא לרוב בעזרת מבנה נתונים אסוציאטיבי כגון dictionary.

1.2 נציג שני פתרונות אפשריים (כל אחד מהם הוא פתרון מלא):

פתרון ראשון: הכלה הדדית: נכתוב פונקציה שבודקת האם אובייקט אחד מכיל את המידע של השני, ונחזיר "הכללה הדדית (דו כיוונית). כדי לבדוק הכלה: נעבור על כל שם נתון באובייקט json הראשון, ונוודא שהוא קיים באובייקט השני, אם מדובר בנתון פשוט נבצע השוואה ערכים פשוטה, אם מדובר באובייקט – נקרא רקורסיבית על כל אחד מהשדות שלו. ואם מדובר על מערך ניתן להתייחס אליו כרשימה אותה נמיין ונקרא רקורסיבית עם כל אחד מהאיברים שלה.

פתרון שני: מיון האובייקטים: נמיין כל אחד מהאובייקטים לפי שמות השדות שלהם, ואז נשווה את הערכים של כל אחד מהשדות, אם מדובר בשדה "פרימיטיבי" – פשוט נשווה, אם מדובר באובייקט – נקרא רקורסיבית, ואם מדובר על מערך נמיין אותו ונקרא רקורסיבית על כל אחד מהשדות – ראו מימוש ב 1.3

1.3

```
def ordered(obj):
    if isinstance(obj, dict):
        return sorted((k, ordered(v)) for k, v in obj.items())
    if isinstance(obj, list):
        return sorted(ordered(x) for x in obj)
    else:
        return obj

def is_eq(json1,json2):
  a = ordered(json1)
  b = ordered(json2)
  return (a == b)
```

1.4

```
class TestQ1(unittest.TestCase):
    def setUp(self):
        j1a = '{"data" : {"Name":"1a" ,"arr":["A","B","C"]}, "v": 1}'
        j1b = '{"v": 1, "data" : {"Name":"1a" ,"arr":["C","A","B"]}}'
        j2a = '{"data" : {"Name":"2a" ,"arr":["C","A","B"]}, "v": 1}'
        j2b = '{"data" : {"Name":"1a" ,"arr":["C","A","B"]}, "v": 2}'
        self._j1a = json.loads(j1a)
        self._j1b = json.loads(j1b)
        self._j2a = json.loads(j2a)
        self._j2b = json.loads(j2b)

    def test_eq(self):
        a1 = is_eq(self._j1a, self._j1a)
        self.assertTrue(a1)
        ab1 = is_eq(self._j1a, self._j1b)
        self.assertTrue(ab1)
    def test_neq(self):
        a12 = is_eq(self._j1a, self._j2a)
        self.assertFalse(a12)
        b12 = is_eq(self._j1a, self._j2b)
        self.assertFalse(b12)
```

שאלה 2 (25 נקודות):

במטלה הראשונה בקורס עסקנו ב"מעליות חכמות", בהן המשתמש נדרש להקיש מחוץ למעליות את קומת היעד (destination) ואז המערכת צריכה לשבץ (להקצות) לו מעלית מסויימת אשר מוגדר לה כבר לעצור בקומת היעד. בהינתן קריאה למעלית מקומת המקור לקומת היעד - המערכת תרצה לשבץ את המעלית שתצמצם למינימום את זמן ההגעה (זמן ההגעה מוגדר להיות משך הזמן בשניות שבין הקריאה למעלית ובין ההגעה לקומת היעד). באופן יותר כללי נאמר שבהינתן בניין בעל מספר מעליות ואוסף קריאות בזמן נרצה להגדיר אסטרטגיית שיבוץ מעלית ספציפית לכל קריאה כך שסך משך זמן ההגעה עבור כלל הקריאות יצומצם למינימום.

שימו לב בשאלה זו אתם לא נדרשים לכתוב קוד, אלא להסברים בלבד.

2.1 (8 נקודות) הסבירו (ללא קוד) מהן המחלקות העיקריות במערכת.

2.2 (9 נקודות) הציגו אלגוריתם (online) יעיל ככול יכולתכם לשיבוץ מעלית לכל קריאה.

2.3 (8 נקודות) הסבירו כיצד ניתן להוסיף למערכת ממשק גרפי, התייחסו לתבנית העיצוב MVC, וכן הקפידו להסביר האם יש צורך ביותר מתהליך אחד לכלל המערכת.

תשובה:

2.1 המחלקות העיקריות במערכת הן:

בניין: שכולל מספר קומות (נניח מ 2- עד קומה 10, סה"כ 13 קומות), וכן מכיל מספר מעליות.

מעלית: כוללת מידע על מהירות, זמן האצה והאטה, זמן פתיחה וסגירה, של דלתות, מיקום נוכחי, ויכולת לקבל "קריאה" לקומה (יעד או מקור) מסויימת.

קריאה: מייצגת הזמנה של מעלית בזמן מסויים מקומת מקור לקומת יעד.

קריאות: אוסף של קריאות מסודרות לפי הזמן.

אלגוריתם שיבוץ: מאפשר לשבץ את המעלית המיטבית עבור כל קריאה (**online**). האלגוריתם כולל מידול של כל אחת מהמעליות בבניין כדי לאפשר לה למעלית לבצע את כל הקריאות ששובצו לה.

סימולטור: מערכת שמאפשרת הפעלה של אוסף קריאות ע"ג בניין מסויים בהתאם לשיבוץ שמבוצע ע"י האלג'.

הסימולטור מאפשר קבלה של מידע לגבי ביצועי האלג' ביחס לתרחיש (בניין + קריאות).

2.2 כיוון מדובר באלג' **online** וכיוון שהשיבוץ למעלית לא יכול לחכות אלא צריך להתבצע ברגע שמתקבלת הקריאה וכיוון שאסור לשנות את השיבוץ למעלית לאחר שניתן. האלגוריתם מיטבי יפעל כדלקמן:

1. עבור על כל הקריאות לפי הסדר:
2. בהינתן קריאה, בדוק עבור כל מעלית את תוספת הזמן שתגרם לכלל השיבוצים הנוכחיים במעלית כתוצאה משיבוץ הקריאה (הנוכחית) למעלית.
3. שבץ את הקריאה למעלית שגורמת לתוספת מינימאלית.

2.3 הוספת ממשק גרפי תתבצע לפי עקרונות ה **MVC**, משמע אנו נחלק את המערכת לשלושה מרכיבים העיקריים:

מרכיב **Model** שיכלול את כלל המידע של המערכת לרבות נתוני הבניין המעליות הקריאות וכו'.

מרכיב **View** שכולל את הממשקים הגרפים: שמאפשר לטעון בניין ,אוסף קריאות ולהריץ את המערכת על אלג' שונים ולראות הצגה של השיבוצים ולבסוף של הנתונים.

מרכיב **Controller** שמאפשר חיבור בין המידע שנמצא ב **Model** למערכת התצוגה (**View**), רכיב זה יכולת אפשריות של גישה למידע וקריאה לפונקציות הצגה שלו, בפרט תכלול את הסימולטור.

הערה: המיקום המדויק של האלגוריתמים הוא לרוב בחבילה נפרדת שאינה נמצאת באף אחד מהרכיבים – אם זאת, תשובה שבה האלגוריתמים מוקמו ברכיב שאינו **View** – היא סבירה.

המערכת תכלול מספר תהליכים: תהליך מרכזי שיריץ הסימולטור שיאפשר הצגה בזמן של הקריאות, השיבוצים, ותנועת המעליות. תהליך זה ירוץ לפי קצב הצגה קבוע + אירועים שונים כמו הגעת מעלית לקומה קבלת קריאה כו'. מעבר לכך המערכת תכלול תהליך של ממשק גרפי שאפשר לעצור לעדכן או להמשיך את הסימולאטור מבחוץ.

שאלה 3 (25 נקודות):

במטלות 2,3 עסקנו במידול גרפים מכוונים וממושקלים, את הגרפים שמרנו כקובצי json. סטודנט שם לב ששמירת גרף תופסת מקום רב בזיכרון. בשאלה זו נפתח שיטה לשמירה קומפקטית של גרפים.

3.1 (6 נקודות) הסבירו באופן כללי מדוע שמירה של גרף כקובץ json יכולה להיות בזבזנית במקום.

3.2 (12 נקודות) הסבירו כיצד ניתן לשמור (ב java) גרפים באופן הקומפקטי ביותר – התייחסו לאופן השמירה של המידע, שימ לב: בסעיף זה אין צורך לכתוב קוד, אבל חייבים להתייחס לאופן השמירה ב java מבחינת מבנה נתונים, ושיטת שמירת המידע.

3.3 (7 נקודות) הניחו שקיימת לכם המחלקה Graph שמממשת את הממשק GraphInterface, ולמחלקה מימשו את השיטות saveJson(String f), saveCompact(String f), loadJson(String f), loadCompact(String f) כתבו את מחלקת בדיקה GtaphTest.java שמאפשרת לבדוק את נכונות הקוד ואת החיסכון בגודל הקובץ בין שני הפורמטים, ביחס לקובץ G1.json.

הדרכה:

- ניתן להניח שהקובץ G1.json שמייצג גרף קשיר בעל 1000 קדקודים ו8000 צלעות (בפורמט json) קיים לכם ורק עליו אתם צריכים לעשות את הבדיקות.
- למחלקה File שמייצגת קובץ קיים בנאי שמקבל מחרוזת (שם הקובץ), וכן קיימת שיטה int getTotalSpace() שמחזירה את גודל הקובץ בזיכרון ב bytes.
- בשאלה זו יש לממש בקוד רק את סעיף 3.3 ורק את המחלקה GtaphTest.java

תשובה:

3.1 כיוון ש **json** הוא פורמט טקסטואלי, מספרים ישמרו כטקסט בקובץ, ולפיכך מספר כמו 12345678 ידרוש נפח של 8 תווים (כ 16 **bytes**) בעוד ששמירה שלו כמספר שלם (**int**) היתה דורשת 4 **bytes** בלבד. מעבר לכך מבנה הקובץ שכולל סוגריים מאוזנים ושמות שדה לכל ערך ורווחים יגרמו לבזבוז מקום רב.

3.2 ניתן לייצג קודקוד בגרף לפי ה **id** שלו – מיוצג כמספר שלם (ולפי הצורך גם מיקום – אם קיים). צלע בגרף תיוצג כזוג סדור (מקור יעד) (שני שלמים) ומשקל (ממשי) לפיכך ניתן לשמור הגרף בצורה קומפקטית כמערך של קודקודים + מערך של צלעות. את הגרף נשמור בעזרת **serializable** שמאפשר מיפוי הזיכרון לקובץ ובנייה שלו בהתאם (הקובץ אומנם שומר גם את הייצוג של המחלקות – אבל יש רק 3 מחלקות – שיחד תופסות מקום זניח ביחס למידע עצמו של הגרף (בהנחה שהגרף גדול).

הערה: כמובן שניתן לשמור את כל המידע ישירות כמערכים של שלמים וממשיים ובכך לצמצם עוד יותר את גודל קובץ ה **serializable** - אך תשובה שכזו לא נדרשה כדי לקבל את מלוא הנקודות בשאלה זו.

3.3

```
class GtaphTest {
	@Test
	void testSize() {
		Graph  g1 = new Graph(), g2 = new Graph();
		g1.loadJson("G1.json");
		g1.saveCompact("G1_com.bin");
		g2.loadCompact("G1_com.bin");
		int s1 = new File("G1.json").getTotalSpace();
		int s2 = new File("G1_com.bin").getTotalSpace();
		assetTrue(s2<s1);
		double c1 = 0.5;  // or any relevant value smaller than 1.0
		assetTrue(s2<s1*c1);
	}
```

שאלה 4 (25 נקודות):

בשאלה זו נניח שקיימת לכם מחלקה GraphAlgo שמממשת את הממשק GraphAlgoInterface (ב python).
בשאלה זו נתכנן ונבדוק את המחלקה GraphAlgoMem שמממשת את GraphAlgoInterface, ומאפשרת שימוש בזיכרון כך שכאשר שיטה (בממשק) נקראת פעם ראשונה היא תחושב, אבל בפעם הבאה ששיטה תופעל עם אותו קלט (על אותו גרף) תוחזר התשובה הקודמת ללא צורך בחישוב מחדש.
4.1 (13 נקודות) הסבירו (אין חובה לכתוב קוד) כיצד נכון לממש מחלקה זו, ציינו באילו מבנה נתונים נכון להשתמש וכיצד תממשו כל אחת מהשיטות ב GraphAlgoInterface
4.2 (12 נקודות) כתבו מחלקת בדיקה TestGraphAlgoMem(unittest.TestCace) שבודקת את השיטות השונות במחלקה.

הדרכה חשובה:

- בשאלה זו הניחו שהשיטה get_graph מחזירה עותק חדש (deep copy semantics) של הגרף.
- ניתן להניח שהקובץ G1.json שמייצג גרף קשיר בעל 1000 קודקודים ו8000 צלעות קיים לכם ורק עליו אתם צריכים לעשות את הבדיקות

תשובה

4.1 נממש את המחלקה GraphAlgoMem ע"י הכלה של המחלקה GraphAlgo (יתקבלו גם תשובות של שימוש בירושה).

- נוסיף למחלקה dictionary שכולל קלט: שם הפונקציה (או קוד שלה) + רשימת הפרמטרים (נניח בפורמט מחרוזת או json) ופלט: הערך המוחזר של הפונקציה.
- כאשר יש קריאה לאחת מהפונקציות האלגוריתמיות (centerPoint, shortest_path)
  נבדוק האם יש כבר ב dictionary כניסה מתאימה לקלט:
  אם כן, נחזיר את התשובה (מבלי לחשב)
  אם לא, נחשב את הפונקציה (ע"י הפעלה של הפונקציה המתאימה מרכיב המוכל של GraphAlgo) ונעדכן את ה dictionary.
- מבחינת מימוש הפונקציות של שמירה לקובץ ניתן לשמור את השדה הנוסף (dictionary) כמידע json – מיפוי טבעי, ובהתאם בטעינה לאתחל את ה dictionary מהקובץ שנשמר. נשים לב שבשאלה זו אין צורך להוסיף שדה של ה MC הנוכחי של הגרף שכן מבחינת ממשק לא ניתן לשנות את הגרף – מקבלים עותק חדש, אם זאת חובה להחליף את ה dictionary כאשר טוענים GraphAlgoMem חדש.

4.2

```
class TestGraphAlgoMem(unittest.TestCace):
    def setUp(self):
        self._gam = GraphAlgoMem()
        self._gam.load_from_json("G1.json")
        self._ga = GraphAlgoMem()
        self._ga.load_from_json("G1.json")

    def test_performance_center(self):
        cm = self._gam.centerPoint()
        sm = time.time()
        cm = self._gam.centerPoint()
        em = time.time()
        dtm = em - sm

        s = time.time()
        c = self._gam.centerPoint()
        e = time.time()
        dt = e - s
        self.assertEqual(c, cm)
        self.assertTrue(dtm < dt)
```

```
        c2 = 0.1
        self.assertTrue(dtm < dt * c1)

    def test_performance_shortest_path(self):
        for x in range(100):
            spm = self._gam.shortest_path(x, x+100)

        sm = time.time()
        for x in range(100):
            spm = self._gam.shortest_path(x, x+100)
        em = time.time()
        dtm = em - sm

        s = time.time()
        for x in range(100):
            sp = self._ga.shortest_path(x, x+100)
        e = time.time()
        dt = e - s

        self.assertEqual(sp, spm)
        self.assertTrue(dtm < dt)
        c2 = 0.01
        self.assertTrue(dtm < dt * c1)
```

בהצלחה!!!

נספח: חומר עוזר (פורסם מראש):

Graph: GraphInterface.java, GraphAlgoInterface.java, GraphInterface.py GraphAlgoInterface.py
Basics: Point2D.java, Point2DTest.java, Point2D.py, TestPoint2D.py

```python
class GraphInterface:
    """ Abstract class representing a directed graph."""
    def v_size(self) -> int:
        """ @return: The number of vertices"""
        raise NotImplementedError
    def e_size(self) -> int:
        """ @return: The number of edges"""
        raise NotImplementedError
    def get_all_v(self) -> dict:
        """return a dictionary of all the nodes"""
    def all_in_edges_of_node(self, id1: int) -> dict:
        """ return a dictionary of all the in edges """
    def all_out_edges_of_node(self, id1: int) -> dict:
        """ return a dictionary of all the out edges"""
    def get_mc(self) -> int:
        """ @return: The Mode Counter """
        raise NotImplementedError
    def add_edge(self, id1: int, id2: int, weight: float) -> bool:
       """ @return: True if the edge was added successfully, """
        raise NotImplementedError
    def add_node(self, node_id: int, pos: tuple = None) -> bool:
        """ @return: True if the node was added successfully """
        raise NotImplementedError
    def remove_node(self, node_id: int) -> bool:
        """ Removes a node from the graph."""
        raise NotImplementedError
    def remove_edge(self, node_id1: int, node_id2: int) -> bool:
        """ Removes an edge from the graph."""
        raise NotImplementedError
```

```python
from typing import List
import GraphInterface

class GraphAlgoInterface:
    """Abstract class representing algorithms on graphs"""
    def get_graph(self) -> GraphInterface:
        """ returns: the underlaying directed graph """
    def load_from_json(self, file_name: str) -> bool:
        """returns: True if the loading was successful"""
        raise NotImplementedError
    def save_to_json(self, file_name: str) -> bool:
        """@return: True if the save was successful, """
        raise NotImplementedError
    def shortest_path(self, id1: int, id2: int) -> (float, list):
        """ :returns: shortest path, and distance """
        raise NotImplementedError
    def centerPoint(self) -> (int, float):
        """:returns The nodes id, min-maximum distance
        """
```

```java
package moed_a;
import java.util.Iterator;
/** This interface represents a directional weighted graph. */
public interface GraphInterface {
    /** @return true iff there is a node.id==key in the Graph. */
    public boolean hasNode(int key);
    /** @return the weight of the edge (src,dest), -1 if none.*/
    public double getEdge(int src, int dest);
    /** Adds a new node to the graph with the given id=n. */
    public void addNode(int n);
    /** Connects an edge with weight w between node src-->dest. */
    public void connect(int src, int dest, double w);
    /** @return an Iterator over all the Nodes (ID). */
    public Iterator<Integer> iterAllV();
    /** @return an Iterator over all the out edges of node id.*/
    public Iterator<Integer> iterOutNodes(int id);
    /** Deletes the node from the graph (and all related edges).*/
    public void removeNode(int key);
    /** Deletes and return the weight(src,dest), -1 is none. */
    public double removeEdge(int src, int dest);
    /** @return the number of vertices (nodes) in the graph. */
    public int nodeSize();
    /** @return the number of edges (assume directional graph). */
    public int edgeSize();
    /** @return the Mode Count: for testing changes in the DS.*/
    public int getMC();
}
```

```java
package moed_a;
/** This interface represents few Graph Algorithms
 * (on directed weighted graphs).*/
public interface GraphAlgoInterface {
    /** updates the underlying graph on which the
     * algorithms work on. */
    public void init(GraphInterface g);
    /** returns the graph (interface) on which the
     * algorithm works in */
    public GraphInterface getGraph();
    /** @return a new and empty (no nodes) graph*/
    public GraphInterface getEmptyGraph();
    /** returns true iff the underlying graph is
     * strongly connected (as a directed graph).*/
    public boolean isConnected();
    /** returns the distance of the shortest path
     * between src and the dest (-1 if none)/ */
    public double shortestPath(int source, int dest);
}
```

```java
package moed_a;
/** This class represents a 2D point in the plane. */
public class Point2D {
    public static final double EPS = 0.00001;
    public static final Point2D ORIGIN = new Point2D( x: 0, y: 0);
    private double _x,_y;
    public Point2D(double x,double y) {_x=x; _y=y;}
    public Point2D(Point2D p) {this(p.x(), p.y());}
    public double x() {return _x;}
    public double y() {return _y;}
    public Point2D add(Point2D p) {
        Point2D a = new Point2D( x: p.x()+x(), y: p.y()+y());
        return a; }
    public String toString() {return _x+","+_y;}
    /** Return the 2D distance from this point to (0,0). */
    public double distance() {return this.distance(ORIGIN);}
    /** @return the 2D (Euclidean) distance between this and p2.
     */
    public double distance(Point2D p2) {
        double dx = this.x() - p2.x();
        double dy = this.y() - p2.y();
        double t = (dx*dx+dy*dy);
        return Math.sqrt(t);
    }
    /**
     * Check if this point equels to the other (p) point.
     * @param p the other point
     * @return true iff this point equels exactly to (p).
     */
    public boolean equals(Object p) {
        if(p==null || !(p instanceof Point2D)) {return false;}
        Point2D p2 = (Point2D)p;
        return ( (_x==p2._x) && (_y==p2._y));
    }
}
```

```java
import moed_a.Point2D;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import static java.util.concurrent.TimeUnit.MILLISECONDS;
import static org.junit.jupiter.api.Assertions.*;

public class Point2DTest {
    @Test
    void testDistance() {
        Point2D p1 = new Point2D( x: 3, y: 4);
        double d = p1.distance(Point2D.ORIGIN);
        double e = Math.abs(d-5);
        assertTrue( condition: e<Point2D.EPS);
    }
    @Timeout(value = 100,  unit = MILLISECONDS)
    @Test
    void add() {
        int size = 10000;
        Point2D p0 = new Point2D(Point2D.ORIGIN);
        Point2D p1 = new Point2D( x: 1, y: 2);
        for(int i=0;i<size;i=i+1) {
            p0 = p0.add(p1);
        }
        assertEquals(p0.x(), size, Point2D.EPS);
        assertEquals(p0.y(),  actual: 2*size, Point2D.EPS);
        assertNotEquals(p0.y(),  actual: 2.001*size, Point2D.EPS);
    }
}
```

```python
from numpy import sqrt
class Point2D:
  def __init__(self, x: float, y: float):
    self._x = x
    self._y = y
  def add(self, p):
    self._x += p._x
    self._y += p._y
  def __add__(self, p):
    return Point2D(self._x + p._x, self._y + p._y)
  def __sub__(self, p):
    return Point2D(self._x - p._x, self._y - p._y)
  def __eq__(self, other):
    if isinstance(other, Point2D):
      return self._x==other._x and self._y==other._y
    return False
  def __repr__(self):
    return f"Point2D ({self._x},{self._y})"
  def dist2(self):
    return self._x*self._x + self._y*self._y
  def dist(self, other):
    return sqrt((self-other).dist2())
  def __lt__(self, other):
    if isinstance(other, Point2D):
      return self.dist2() < other.dist2()
    return False
import unittest
from Point2D import Point2D

class TestPoint2D(unittest.TestCase):
    def setUp(self):
        self._p1 = Point2D(1,2)
        self._p0 = Point2D(0, 0)
        self._p2 = Point2D(4, 6)
    def test_eq(self):
        p1 = Point2D(1,2)
        self.assertEquals(p1, self._p1)
        self.assertNotEquals(self._p1, self._p2)
    def test_dist(self):
        self.assertEquals(self._p1.dist(self._p2),5)
    def test_dist2(self):
        self.assertEquals(self._p0.dist2(), 0)
        self.assertNotEquals(self._p1.dist2(), 0)

if __name__ =='__main__':
    unittest.main()
```

```python
# motivated by www.geeksforgeeks.org/
List1 = []
for i in range(0, 4):
    List1.append(i)
List1.insert(3, len(List1)*3)
List1.insert(-1, 'G 4 G')
List1.remove(2)
List1.pop(1)
List1.extend([5,6,7])
List1.insert(-2, [8,9,10])
print(List1)
# [0, 12, 'G 4 G', 3, 5, [8, 9, 10], 6, 7]

set1 = set()
for i in range(-2, 3):
    set1.add(i)
set1.add((3, 4))
set1.add(1)
set1.add(1)
set1.remove(1)
print(set1) #{0, 2, (3, 4), -1, -2}

Dict=dict({1:'Geeks',2:'For',3:'Geeks'})
Dict[5] = "five"
Dict.pop(2)
print(Dict)# {1: 'Geeks', 3: 'Geeks', 5: 'five'}
```