**חוברת הפעלה והסבר**

**תוכן עניינים**

[הסבר כללי 3](#_Toc73889380)

[סדר פעולת התוכנית 3](#_Toc73889381)

[סוגי קבצים 4](#_Toc73889382)

[קבצי מחלקה ראשיים 4](#_Toc73889383)

[קבצי מחלקה משניים 4](#_Toc73889384)

[קבצי בנייה 5](#_Toc73889385)

[סוגי פונקציות ומתודות 5](#_Toc73889386)

[פונקציות בנייה 5](#_Toc73889387)

[מתודות 'דחיפה' 5](#_Toc73889388)

[מתודות 'הוספה' 5](#_Toc73889389)

[חלקי התוכנית 6](#_Toc73889390)

[Main\_Diagram 6](#_Toc73889391)

[Direction 17](#_Toc73889392)

[טבלה 19](#_Toc73889393)

[בניה 19](#_Toc73889394)

[מחלקה 19](#_Toc73889395)

[ת"ז 19](#_Toc73889396)

[בניה 19](#_Toc73889397)

[מחלקה 19](#_Toc73889398)

# הסבר כללי

ג'אנק (JUNC – קיצור של Junction) היא תוכנית הכתובה בשפת פייטון ומטרתה להפיק תרשים ויזואלי של ניתוחי רמות שירות בצמתים. התוכנית עובדת בשילוב עם התוכנית Phaser, אשר מטרתה לחשב ולנתח רמות שירות בצמתים. הפלט של ג'אנק מבוסס על הפלט של פייזר.

לאחר הרצת הג'אנק, נקבל תיקיה על שולחן העבודה בשם ×JUNC×. התיקייה מכילה בתוכה שלושה סוגי קבצים –

* **Volume calculator –** קובץ אקסל ששימש לניתוח הצומת הנוכחי. ג'אנק מייצר העתק של קובץ זה על מנת להקל במידה ויש צורך בהרצה חוזרת של הניתוח (עבור תיקונים). במסמך זה נתייחס לקובץ כ- VC, מטעמי נוחיות.
* **×Diagram×, ×ID×, ×Table× -** קבצי תמונה, אלו קבצי הפלט של ג'אנק:
  + **×Diagram× -** תרשים הצומת. התרשים מציג את הצומת עליה בוצע הניתוח וכולל בתוכו את שמות הדרכים, מספר הזרועות בצומת, נפחים לשעת שיא בוקר ואחר הצהריים, ניתוב עבור כל זרוע וכן סימון במידה וקיים רק"ל או מטרו אשר עוברים בצומת (או תחתיו).
  + **×Table× -** טבלת הנפחים. הטבלה מציגה את החלוקה לתמונות עבור שעת שיא בוקר ואחה"צ. החלוקה מוצגת לפי הפלט והחישוב שבוצע ע"י פייזר. הטבלה מציגה את סה"כ הנפחים, את יחס קיבולת לנפח (VOC), את רמת השירות בצומת (LOS) וכן את קיבולת הרק"ל, במידה וקיימת בצומת.
  + **×ID× -** תעודת זהות עבור הריצה הנוכחית. מסמך זה מכיל בתוכו מידע כללי לגבי הניתוח מסמך זה מיועד לנוחיות המשתמש, לצורכי תיעוד ומעקב אחר ניתוחי צמתים שונים עבור פרויקטים שונים. המסמך מציין את שמות הדרכים בצומת, תאריך ושעת ההרצה של הניתוח בJUNC, שם המריץ (כפי שמופיע ביישומי Office), מספר הרצה (לצורך מעקב הרצות עבור אותו הצומת), שם הפרויקט ומספר הפרויקט (יצוין ע"י המשתמש בקובץ VC) וכן הערות נוספות. אין צורך לצרף אותו בדו"חות אשר מפיקים, אלא להשאיר אותו בתיקייה המקורית ביחד עם הקבצים האחרים.
* **×pptx files× -** תיקייה אשר מכילה שלושה קבצי PowerPoint – קבצי הפלט המקוריים של ג'אנק. קבצים אלו מצורפים לפלט הסופי על מנת לאפשר למשתמש לערוך או לתקן באופן נקודתי בפלט. מומלץ להימנע משינויים ידניים בחישובים או בתוצאות החישוב וזאת כדי לא לפגוע באמינות התוצאות.

# סדר פעולת התוכנית

**הרצת הג'אנק**

הרצת ג'אנק מתבצעת דרך התוכנה PyCharm (כל עוד אין ממשק אשר מחליף את ההרצה – תיכנון עתידי). פאיצ'ארם מריצה את הקוד אשר כתוב בפייתון.

לפני הרצת ג'אנק יש לוודא:

* קובץ הVC הנכון נמצא בתיקייה בה JUNC מותקן.
* אין קבצי פאוורפוינט אחרי שפתוחים ברקע.
* הקובץ אותו מריצים בפאיצ'ארם נקרא Main\_ID (הסבר מפורט על משמעות הקובץ בהמשך מסמך זה).

במהלך הריצה יתבצעו החישובים הנחוצים, התוכנה פאוורפוינט תיפתח ותיסגר מספר פעמים וקבצים חדשים ייווצרו על שולחן העבודה. מהלך הריצה לוקח בין 20 ל30 שניות (תלוי בסוג המחשב עליו הוא רץ); מומלץ להימנע משימוש במחשב במהלך הריצה ולהמתין לסיום הריצה במלואה לפני המשך העבודה.

מהלך הריצה של ג'אנק פועל "מהסוף להתחלה". הכוונה היא, שהקובץ האחרון שנוצר (JUNC\_ID) קורא לקובץ שלפניו (JUNC\_TABLE); JUNC\_TABLE קורא לקובץ JUNC\_DIAGRAM – הוא הקובץ הראשוני.

הסיבה לכך היא שכל קובץ צריך את הקובץ שלפניו כדי להיבנות.

נציג (באופן כללי) את סדר הפעלת הקבצים הקשורים לג'אנק במהלך הריצה, כדי להבהיר נקודה זו:

Main\_ID 🡪 Main\_Table 🡪 Main\_Diagram 🡪 Building\_Diagram 🡪 Phaser 🡪 Main\_Diagram 🡪 Building\_Diagram 🡪 Main\_Table 🡪 Building\_Table 🡪 Main\_ID 🡪 Building\_ID 🡪 Main\_ID

## סוגי קבצים

הקבצים השונים אשר משמשים לריצה של ג'אנק מחולקים ל3 סוגים:

1. קבצי מחלקה ראשיים
2. קבצי מחלקה משניים
3. קבצי בנייה

### קבצי מחלקה ראשיים

קיימים 3 קבצים מסוג זה בהתאם לשלושת הפלטים של ג'אנק – Diagram, Table, ID. שמות קבצים אלו מתחילים במילה Main בצירוף סוג הפלט.

לקובץ מחלקה ראשי מספר תפקידים:

1. לאחד ולהכיל את כל קבצי המחלקה המשניים השייכים לו.
2. להכיל את כל מתודות המחלקה השייכות לו. מתודות אלו, קשורות באופן ישיר **למידע ולא לקובץ** (לדוגמא, חישוב מספר התמונות בטבלה, חישוב כמות הזרועות בצומת או הכנסת מידע למאפייני המחלקה).
3. לקרוא לפונקציות בנייה בהתאם לשלב הבנייה הנוכחי בריצה.
4. להחזיק את המידע אשר הגיע מהפלט של פייזר.

בהמשך המסמך נפרט על מבנה ותוכנו של כל קובץ מחלקה ראשי.

### קבצי מחלקה משניים

קבצי מחלקה משניים הם הקבצים המייצגים את תתי המחלקות של כל חלק בפלט של ג'אנק.

צורת הכתיבה בה ג'אנק נכתב (מבחינת קוד) היא של "תכנות מונחה עצמים" (Object Oriented programming - OOP).

לא נרחיב באופן מלא על צורת כתיבה זו, אך נעבור באופן כללי על מאפייניה וכיצד באים לידי ביטוי בג'אנק.

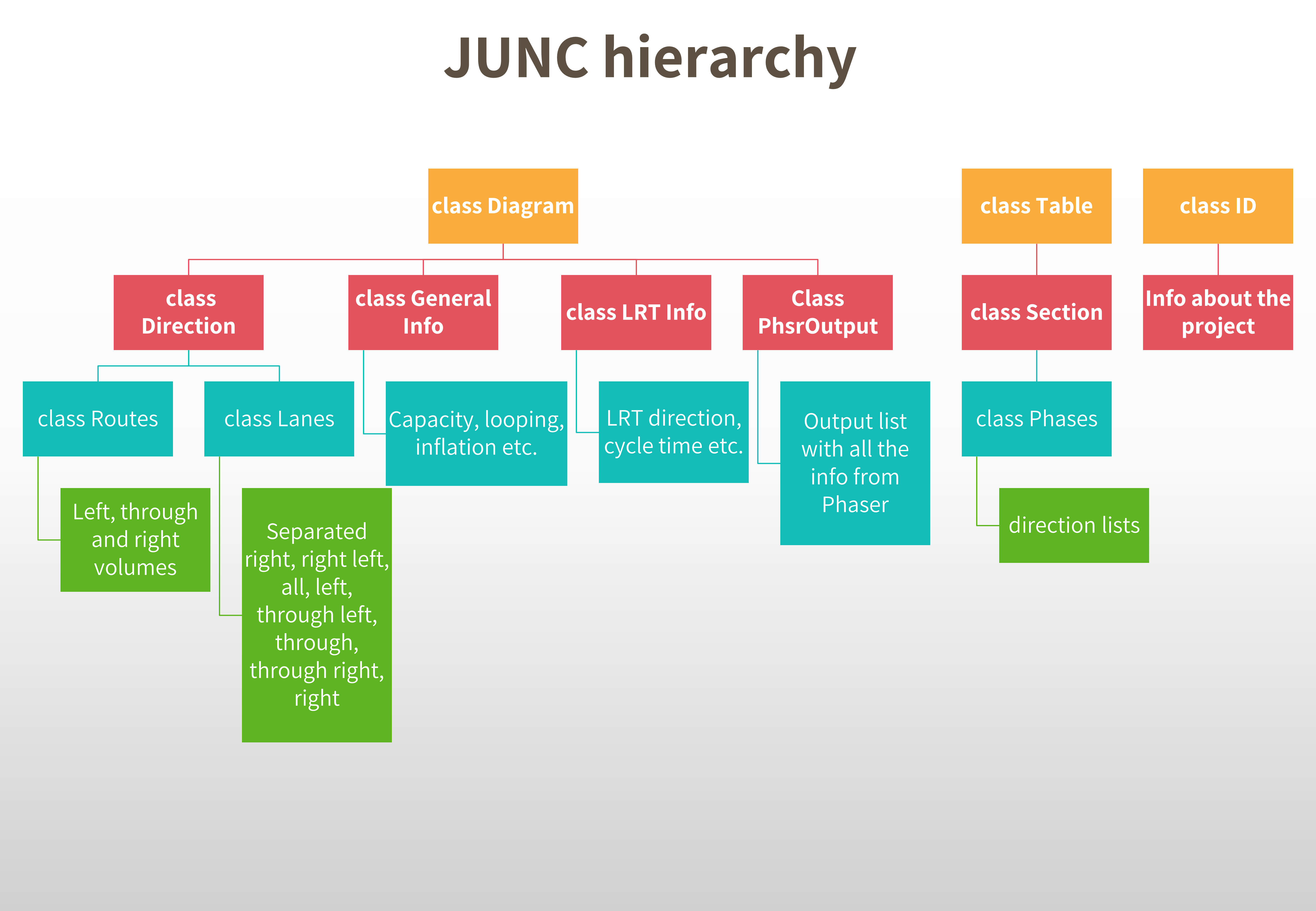
תכנות מונחה עצמים או OOP, מחלק את 'העולם' למחלקות או משפחות שונות. מחלקת העל, תכיל את כל תתי המחלקות. לדוגמא, מחלקת "חיות" תהיה מחלקת העל, בתוכה תהיה מחלקת "יונקים"; בתוך מחלקת יונקים תהיה מחלקת "חתולים". כל מחלקה יכול להכיל מאפיינים משלה ומתודות השייכות לה או למחלקה שמעליה.

נבחין בין 'פונקציה' ל'מתודה': פונקציה אחראית על פעולה כללית שלא קשורה לאובייקט מסוים – למשל, הדפסת המספרים מ1 עד 200; מתודה היא פעולה הקשורה באופן ישיר לאובייקט – למשל, כלב תופס מקל, מכונית נוסעת.

כאשר יוצרים "מספר דורות" של מחלקות, ניתן לגשת אליהם באופן הבא; נניח ונרצה לגשת למאפיין צבע חתול, שנמצא תחת מחלקת חתול, שהוא חלק ממחלקת יונקים, שהוא חלק ממחלקת חיות: Animal.Mammal.Cats.cat\_color

(זו דוגמה לאופן בה ניגש למאפיין בתוך תתי מחלקות; בפועל צורת הכתיבה שונה מהמתואר מעלה).

כדי להבין את הצורה בה ג'אנק מחלק את חלקי המידע השונים למחלקות, מאפיינים ומתודות, נסתכל על התרשים הבא:



בהמשך המסמך נפרט על מבנה ותוכנו של כל קובץ מחלקה משני.

### קבצי בנייה

קיימים 3 קבצים מסוג זה בהתאם לשלושת הפלטים של ג'אנק – Diagram, Table, ID. שמות קבצים אלו מתחילים במילה Building בצירוף סוג הפלט. קבצי בנייה הם קבצים אשר מכילים פונקציות הקשורות לכל תהליך בניית המצגת.

דוגמאות לפונקציות בקובץ בנייה – סידור ויצירת תיקייה חדשה על שולחן העבודה, מחיקת קבצים זמניים שנוצרו במהלך הריצה, ייצוא של המצגת לקובץ תמונה.

בהמשך המסמך נפרט על מבנה ותוכנו של כל אחד מקבצי הבנייה.

## סוגי פונקציות ומתודות

את סוגי הפונקציות והמתודות בג'אנק ניתן לחלק 3 קבוצות עיקריות ועוד קבוצה משנית נוספת:

### פונקציות בנייה

פונקציות אלו אחראיות על יצירה ובנייה של המצגת עצמה. הן לא מסומנות בצורה מסוימת ולרוב שייכות לקבצי הבנייה של כל חלק בג'אנק.

### מתודות 'דחיפה'

מתודות אלו אחראיות על הכנסת המידע אשר מגיע כפלט מפייזר לתוך תת המחלקות הנכונים בחלקים השונים של ג'אנק. מתודות אלו לרוב מתחילות במילה push והן שייכות למחלקות העל בהתאם לחלק בג'אנק.

### מתודות 'הוספה'

מתודות אלו אחראיות על הוספת המידע אשר מאוכלס במחלקות השונות בג'אנק – לתוך המיקומים הנכונים בקובץ המצגת שנבנה. מתודות אלו לרוב מתחילות במילה add והן שייכות למחלקות העל בהתאם לחלק בג'אנק.

שלושת סוגי הפונקציות יוצר שרשרת פעולות של הכנת המצגת:

בניית המצגת 🡨 דחיפת נתונים מפייזר לתוך ג'אנק 🡨 הוספת הנתונים מג'אנק למצגת.

# חלקי התוכנית

בפרק זה נעבור על כל אחד מחלקי הקוד בהתאם לתרשים מעלה, בכל פעם נעבור דרך ענף אחר בתרשים.

## Main\_Diagram

חלקי הimport-

import Phaser  
from Directions import \*  
from General\_Info import \*  
from LRT\_Info import \*  
from Phaser\_Output import \*  
from Building\_Diagram import \*  
from pptx.dml.color import RGBColor  
from pptx.enum.text import MSO\_AUTO\_SIZE  
from pptx.enum.lang import MSO\_LANGUAGE\_ID

לפי הסדר: קובץ פייזר, תת מחלקה כיוונים, תת מחלקה מידע כללי, תת מחלקה מידע לגבי רק"ל, תת מחלקה פלט פייזר, תת מחלקה תרשים - קובץ בנייה, ערכת צבעים לשימוש במספרי RGB, מאפיין להגדרת גודל משתנה לטקסט בתוך בצורה במצגת, מאפיין הגדרת שפה לטקסט בתוך צורה במצגת.

class Diagram:  
 *"""  
 A class used to represent all the info about the junction that is needed for creating the Diagram presentation file.  
 """* def \_\_init\_\_(self, phsrlist, xlprop):  
 *""" The constructor of the Diagram class,called when a new instance of a class is created.  
 To initialize, it needs the output of Phaser and the info from the excel"""* self.\_\_North = Direction(  
 "North") # A property representing the north direction. Initialized with the name "North"  
 self.\_\_South = Direction(  
 "South") # A property representing the south direction. Initialized with the name "South"  
 self.\_\_East = Direction("East") # A property representing the east direction. Initialized with the name "East"  
 self.\_\_West = Direction("West") # A property representing the west direction. Initialized with the name "West"  
 self.\_\_GenInfo = General\_Info() # A property representing general info about the junction.  
 self.\_\_LRTInfo = LRT\_Info() # A property representing LRT info about the junction.  
 self.\_\_InfoFromPhaser = PhsrOutput(  
 phsrlist) # A class that holds the output from Phaser. That is the data the will be pushed to  
 # the different class in Diagram and Table.  
 self.\_\_ExcelProperties = xlprop # A list with info about the creator of the volume\_calculator excel file. It is  
 # used in the ID file

מחלקת דיאגרם היא מחלקת העל של התרשים של ג'אנק. היא מכילה בתוך את כל המאפיינים הקשורים לתרשים וכן את כל המתודות הקשורות לתרשים של ג'אנק. כדי ליצור מופע חדש של דיאגרם, חייבים לתת שני משתנים – רשימת פלט של פייזר ורשימה עם מידע לגבי קובץ הVC שהכין המשתמש.

**פירוט מאפייני המחלקה**

* ארבע מאפיינים מסוג כיוון (Direction).   
  אלו מכילים בתוכם את כל המידע הקשור לכל זרוע. זוהי תת מחלקה אשר יפורט עליה בהמשך.
* מאפיין מסוג מידע כללי (General\_Info).   
  מכיל בתוכו מידע כללי לגבי הצומת. זוהי תת מחלקה אשר יפורט עליה בהמשך.
* מאפיין מסוג מידע רק"ל (LRT\_Info).   
  מכיל בתוכו מידע כללי לגבי הצומת. זוהי תת מחלקה אשר יפורט עליה בהמשך.
* מאפיין מסוג פלט פייזר (PhsrOutput).   
  מכיל בתוכו את כל המידע שמגיע מפייזר, מחולק לפי מאפיינים מתאימים. זוהי תת מחלקה אשר יפורט עליה בהמשך.
* מאפיין מסוג מאפייני משתמש אקסל (xlprop).  
  רשימה אשר מכילה מידע לגבי קובץ האקסל VC – תאריך יצירת הקובץ, שם היוצר וכו'.

**Getter / Setter**

על מנת לאפשר גישה מהירה ונוחה למחלקות ולתת המחלקות בכל אחד מהחלקים של ג'אנק, נשתמש בפונקציות גט וסט. פונקציות אלו מאפשרות גישה בצורה מוסדרת לקבלה של מאפיין או השמה למאפיין במחלקה או בתת מחלקה (כפי שמוסבר בקבצי מחלקה משניים). ברשימה זו נפרט את הגטרים והסטרים ביחד עם המאפיין אליו הם קשורים.

**נשים לב, כי לא ניתן לגשת למאפיין ללא שימוש בגטר או סטר המתאים. אפשרות זאת חסומה על מנת לא ליצור בלבול ועומס בכתיבת הקוד.**

|  |  |  |
| --- | --- | --- |
| **Getter / Setter** | **Property** | **info** |
| NO | self.\_\_North | גישה אל מאפיין צפון מסוג כיוון |
| SO | self.\_\_South | גישה אל מאפיין דרום מסוג כיוון |
| EA | self.\_\_East | גישה אל מאפיין מזרח מסוג כיוון |
| WE | self.\_\_West | גישה אל מאפיין מערב מסוג כיוון |
| G\_INF | self.\_\_GenInfo | גישה אל מאפיין מידע כללי מסוג מידע כללי |
| LRT\_INF | self.\_\_LRTInfo | גישה אל מאפיין מידע רק"ל מסוג מידע רק"ל |
| phsr\_lst | self.\_\_InfoFromPhaser | גישה אל מאפיין רשימת פייזר מסוג מידע מפייזר |
| Xlprop | self.\_\_ExcelProperties | גישה אל מאפיין מאפייני אקסל מסוג מאפייני אקסל |

**פירוט מתודות המחלקה**

def push\_arr(self):  
 *"""the method uses the output arrows of Phaser to push them into the right subclass of each direction,  
 divided to regular arrows and public transport arrows. """* arr\_list = [self.phsr\_lst.ARROW\_REG, self.phsr\_lst.ARROW\_PT]  
 SR\_cur\_arrow\_input = [0, 0]  
 orig\_lanes = ["R", "TR", "T", "TL", "L", "A", "RL"]  
 directions = [self.NO.LAN, self.SO.LAN, self.EA.LAN, self.WE.LAN]  
 for direc in directions:  
 for lan in orig\_lanes:  
 if arr\_list[0]:  
 SR\_cur\_arrow\_input = [0, 0]  
 cur\_arrow\_input = [arr\_list[0][0], arr\_list[1][0]]  
 if lan == "R" and arr\_list[0][0] == 9:  
 SR\_cur\_arrow\_input = [0, 0]  
 SR\_cur\_arrow\_input[0] = 1  
 cur\_arrow\_input[0] = 0  
 if lan == "R" and arr\_list[1][0] == 9:  
 SR\_cur\_arrow\_input[1] = 1  
 cur\_arrow\_input[1] = 0  
 setattr(direc, lan, cur\_arrow\_input)  
 setattr(direc, "SR", SR\_cur\_arrow\_input)  
 arr\_list[0].pop(0)  
 arr\_list[1].pop(0)

**סוג**

מתודת דחיפה

**תפקיד**

דחיפה של סוגי החצים (ניתוב) בכל זרוע בצומת.

**אופן פעולת המתודה**

המתודה מאחסנת רשימה של הניתוב הקיים עבור נתיב רגיל ורשימה של נת"צ (הגיעו כפלט מפייזר). כל רשימה מורכבת מ28 תווים, כאשר כל שבעה תווים מייצגים כיוון לפי המפורט ב orig\_lanes. המתודה עוברת על כל אחד מכיווני הצומת, בפנים עוברת על כל כיוון. המתודה 'מרכיבה' את הסטר המתאים בעזרת setattr ומכניסה לתוכו רשימה שמורכבת משני תאים – אחד מייצג כמות של נתיבים רגילים מהכיוון בו נמצאים ואחד מייצג כמות של נתצים.

עבור כיוון ימינה (“R”), המתודה בודקת אם מסומן בספרה 9 – זו מייצגת ימינה חופשי בצומת. במידה וקיים, מכניסה לתא המתאים (“SR”). בסוף הלולאה, המתודה מוציאה את האיבר הראשון ברשימה עד שהרשימה מתרוקנת לגמרי.

**תיקונים אופציונליים**

* הוספת סוגי ניתוב (לדוגמא, נתיב אופניים)
* הוספת סוגי חצים (סימון פרסה)

def push\_vol(self):  
 *"""the method uses the output volumes of Phaser to push them into the right subclass of each direction,  
 divided to morning and evening"""* vol\_list = [self.phsr\_lst.MOR\_VOL, self.phsr\_lst.EVE\_VOL]  
 directions\_mor = [self.NO.MOR, self.SO.MOR, self.EA.MOR, self.WE.MOR]  
 directions\_eve = [self.NO.EVE, self.SO.EVE, self.EA.EVE, self.WE.EVE]  
 count = -1  
  
 for vol in vol\_list:  
 count += 1  
 if count == 0:  
 directions = directions\_mor  
 else:  
 directions = directions\_eve  
 for direc in directions:  
 routes = ["R", "T", "L"]  
 for rou in routes:  
 if vol:  
 value\_to\_push = int(vol[0])  
 setattr(direc, rou, value\_to\_push)  
 vol.pop(0)

**סוג**

מתודת דחיפה.

**תפקיד**

דחיפה של נפחי התנועה לזרועות השונות תוך חלוקה לשעת שיא בוקר ואחה"צ.

**אופן פעולת המתודה**

המתודה מאחסנת רשימה של נפחי שעת שיא בוקר ושעת שיא אחה"צ (הגיעו כפלט מפייזר). כל רשימה מורכבת מ12 מספרים, כאשר כל שלושה מספרים מייצגים זרוע לפי הסדר המופיע בdirections\_mor / directions\_eve - .

המתודה עוברת על כל אחד מזרועות הצומת, בפנים עוברת על כל כיוון. המתודה 'מרכיבה' את הסטר המתאים בעזרת setattr ומכניסה לתוכו את הנפח המתאים. בסוף הלולאה, המתודה מוציאה את האיבר הראשון ברשימה עד שהרשימה מתרוקנת לגמרי.

def push\_general\_info(self):  
 *"""the method uses the output general information of Phaser to push it into G\_INF subclass and to each matching property in that subclass. For specific info that related to the LRT, it pushes it to LRT\_INF """* phaser\_gen\_info\_list = self.phsr\_lst.GEN\_INFO  
 info\_list = [self.G\_INF, self.LRT\_INF]  
 inf\_counter = 0  
 lrt\_types = [0, 0]  
 info\_types = ["CAP", "NLSL", "ELWL", "IMG5", "IMG6", "GEONS", "GEOEW", "LOOP", "LRT\_Orig", "LRT\_Orig", "INF"]  
 while inf\_counter < len(info\_types):  
 if inf\_counter == 8:  
 curr\_inf = info\_list[1]  
 lrt\_types[0] = phaser\_gen\_info\_list[inf\_counter]  
 data\_to\_push = lrt\_types  
 if inf\_counter == 9:  
 curr\_inf = info\_list[1]  
 lrt\_types[1] = phaser\_gen\_info\_list[inf\_counter]  
 data\_to\_push = lrt\_types  
 else:  
 curr\_inf = info\_list[0]  
 data\_to\_push = phaser\_gen\_info\_list[inf\_counter]  
 setattr(curr\_inf, info\_types[inf\_counter], data\_to\_push)  
 inf\_counter += 1  
 self.LRT\_INF.lrt\_orig\_to\_dir()

**סוג**

מתודת דחיפה

**תפקיד**

דחיפה של מידע כללי הנוגע למבנה הצומת או אופן הניתוח של הנתונים.

**אופן פעולת המתודה**

המתודה מאחסנת רשימה של מידע כללי על הצומת (הגיעו כפלט מפייזר). כל נתון מאוחסן תחת אותה מחלקה חוץ משני נתונים – כיוון הרק"ל בצומת. אלו מסומנים באינדקס 8 ו9 ועוברים להיות מאוחסנים במחלקה האחראית למידע על הרק"ל. המתודה 'מרכיבה' את הסטר המתאים בעזרת setattr ומכניסה לתוכו את המידע המתאים. בסוף הלולאה, המתודה קוראת למתודה של מחלקת מידע רק"ל (LRT\_INF) אשר מתרגמת את כיווני הרקל שהוזנו לסוג המתאים באיחסון המידע במאפיין.

def push\_lrt\_info(self):  
 *"""the method uses the output LRT information of Phaser to push it into LRT\_INF subclass and to each matching property in that subclass."""* phaser\_lrt\_info\_list = self.phsr\_lst.LRT\_INFO  
 phaser\_lrt\_info\_list.pop(0)  
 lrt\_info\_types = ["CYC\_TIME", "LRT\_LOST\_TIME", "LRT\_HDWAY", "LRT\_MCU", "GEN\_LOST\_TIME"]  
 for lrt\_inf in lrt\_info\_types:  
 setattr(self.LRT\_INF, lrt\_inf, phaser\_lrt\_info\_list[lrt\_info\_types.index(lrt\_inf)])

**סוג**

מתודת דחיפה

**תפקיד**

דחיפה של מידע הנוגע לרק"ל בצומת.

**אופן פעולת המתודה**

המתודה מאחסנת רשימה של מידע על רק"ל בצומת (הגיעו כפלט מפייזר). המתודה 'מרכיבה' את הסטר המתאים בעזרת setattr ומכניסה לתוכו את המידע המתאים.

def push\_street\_names(self):  
 *"""the method uses the output street names of Phaser to push it into phsr\_lst subclass and to the  
 matching property in that subclass (STREET)."""* phaser\_street\_names\_list = self.phsr\_lst.STREETS  
 dir\_list = {"NO": self.NO, "SO": self.SO, "EA": self.EA, "WE": self.WE}  
 dir\_keys = list(dir\_list.keys())  
 for cur\_dir in dir\_keys:  
 the\_dir = dir\_list[cur\_dir]  
 the\_name = phaser\_street\_names\_list[dir\_keys.index(cur\_dir)]  
 setattr(the\_dir, "NAME", the\_name)

**סוג**

מתודת דחיפה

**תפקיד**

דחיפה של שמות הרחובות או הדרכים בכל זרוע.

**אופן פעולת המתודה**

המתודה מאחסנת רשימה של שמות הרחובות בצומת (הגיעו כפלט מפייזר). המתודה 'מרכיבה' את הסטר המתאים בעזרת setattr ומכניסה לתוכו את המידע המתאים, ע"י מעבר על מפתחות dir\_list – זרועות הצומת.

def get\_type\_of\_junc\_for\_choosing\_slide(self):  
 *"""This method checks about each direction in the junction whether it's empty or not (empty: no lanes or  
 volumes); It creates a string that is later translated into a type of junction ＋,⊢,⊤,⊣,⊥ The method also  
 checks for LRT in the junction (and it's direction), and for a metro around the junction. The method return a  
 number that matches the info about the junction and represents a matching slide in the diagrams template  
 file. If the junction contains only three directions, the method updates a property that later will be  
 used for the oneway function.  
 """* north = str(self.NO.empty\_direction())  
 south = str(self.SO.empty\_direction())  
 east = str(self.EA.empty\_direction())  
 west = str(self.WE.empty\_direction())  
 dir\_exist = north + south + east + west  
 types = {'1111': 1, '1110': 2, '0111': 3, '1101': 4, '1011': 5}  
 if self.LRT\_INF.LRT\_Dir > 0:  
 lrt\_type = 1  
 else:  
 lrt\_type = 0  
 if self.LRT\_INF.Metro\_Dir > 0:  
 metro\_type = 2  
 else:  
 metro\_type = 0  
 junc\_type = metro\_type \* 5 + lrt\_type \* 5 + types[dir\_exist]  
 if types[dir\_exist] > 1:  
 self.G\_INF.ONEWAY = types[dir\_exist]  
 return junc\_type

**סוג**

מתודת בנייה

**תפקיד**

קביעת סוג הצומת לצורך בחירת השקופית הנכונה.

**אופן פעולת המתודה**

מטרת המתודה היא לאסוף חלקי מידע לגבי הצומת ולתרגם אותם למספר שקופית אשר מכילה את הצומת הנדרש כבסיס לבניית התרשים. חלקי המידע הם מספר הזרועות של הצומת (3 או 4), האם קיימת רק"ל בצומת והאם קיים מטרו בצומת. המתודה בודקת לגבי כל זרוע האם היא ריקה: זרוע ריקה הינה זרוע ללא נפחים וללא ניתוב. המתודה מרכיבה מחרוזת בעלת 4 תווים, המכילה 0 או 1 ומייצגת זרועות ריקות או מלאות. את המחרוזת מתרגמים למספר אשר מייצג את סוג הצומת. בנוסף בודקים אם קיים רק"ל או מטרו. המתודה מחברת את כל המידע למספר אשר תואם את מספר השקופית. במידה והצומת הוא צומת T, המתודה מאחסנת את סוג הצומת במאפיין המתאים לכך (לצורך בדיקה מאוחרת יותר אם קיימת שם זרוע חד סטרית).

**תיקונים אופציונליים**

* הוספת סוגי צמתים חדשים (יותר זרועות, פחות זרועות)
* הוספת מידע נוסף על הצומת (סימון לאמצעי תחבורה אחרים אשר עוברים או ממוקמים בצומת, כגון רק"ל).

def add\_street\_name\_and\_lrt(self, pres):

street\_placeholders = {"NORTH\_NAME": self.NO.NAME, "SOUTH\_NAME": self.SO.NAME, "WEST\_NAME": self.WE.NAME, "RAKAL": self.LRT\_INF.LRT\_Dir}  
 lrt\_type\_to\_string = {0: "", 1: "צפון⇋דרום", 2: "מזרח⇋מערב", 3: "צפ⇋דר, מז⇋מע"}  
 for slide in pres.slides:  
 for shape in slide.shapes:  
 if shape.name in street\_placeholders.keys():  
 text\_frame = shape.text\_frame  
 text\_frame.clear()  
 text\_frame.word\_wrap = True  
 text\_frame.auto\_size = MSO\_AUTO\_SIZE.TEXT\_TO\_FIT\_SHAPE  
 if shape.name == "RAKAL":  
 text\_frame = shape.text\_frame  
 text\_frame.auto\_size = MSO\_AUTO\_SIZE.TEXT\_TO\_FIT\_SHAPE  
 p = text\_frame.paragraphs[0]  
 run = p.add\_run()  
 font = run.font  
 font.bold = False  
 font.size = Pt(18)  
 font.color.rgb = RGBColor(228, 223, 211)  
 font.language\_id = MSO\_LANGUAGE\_ID.HEBREW  
 text\_frame.text\_wrap = True  
 text\_frame.auto\_size = MSO\_AUTO\_SIZE.TEXT\_TO\_FIT\_SHAPE  
 text\_frame.text\_wrap = True  
 font.name = 'Assistant'  
 run.text = str(lrt\_type\_to\_string[street\_placeholders[shape.name]])  
 else:  
 text\_frame = shape.text\_frame  
 text\_frame.auto\_size = MSO\_AUTO\_SIZE.TEXT\_TO\_FIT\_SHAPE  
 p = text\_frame.paragraphs[0]  
 run = p.add\_run()  
 font = run.font  
 font.bold = True  
 font.size = Pt(24)  
 font.language\_id = MSO\_LANGUAGE\_ID.HEBREW  
 font.color.rgb = RGBColor(89, 89, 89)  
 font.name = 'Assistant'  
 if street\_placeholders[shape.name] == 0:  
 run.text = ""  
 else:  
 run.text = str(street\_placeholders[shape.name])  
 text\_frame.text\_wrap = True  
 text\_frame.auto\_size = MSO\_AUTO\_SIZE.TEXT\_TO\_FIT\_SHAPE  
 if self.G\_INF.ONEWAY > 1:  
 self.is\_oneway(pres)  
 pres.save("Street\_Diagram.pptx")

**סוג**

מתודת הוספה

**תפקיד**

הוספת שמות הרחובות ואת כיוון הרק"ל במידה וקיימת בצומת

**אופן פעולת המתודה**

המתודה מחזיקה מילון עם שמות הרחובות ועם כיוון הרק"ל. המתודה עוברת על כל הצורות הקיימות בשקופית במצגת של התרשים, היא 'מחפשת' צורות אשר השם שלהן קיים כמפתח במילון. במידה וקיימת כזו צורה, המתודה 'מנקה' את הצורה מטקסט קודם; במידה והצורה אשר זוהתה היא צורת RAKAL, נתאים הגדרות מסוימות (מבחינת צבע הטקסט, גודל וכו'), אחרת נתאים הגדרות לשמות הרחובות. במידה וסוג הצומת הוא T, נקרא למתודה is\_oneway לבדיקת זרוע חד סטרית.

בסוף הריצה, המתודה שומרת את המצגת החדשה שנשמרה (יוצרת קובץ זמני שנמחק בסוף ההרצה).

**תיקונים אופציונליים**

* תיקון עיצוב של שמות הרחובות.

def add\_morning\_volumes(self, pres):  
 *"""  
 The method goes through all the shapes in Diagram pptx file and checks if it represents a volume of the  
 morning counts; It adds the matching volume to each direction, based on the route property of each direction(  
 L,T,R).  
 """* volume\_placeholders = {"NORTH\_MOR\_R": self.NO.MOR.R, "NORTH\_MOR\_T": self.NO.MOR.T, "NORTH\_MOR\_L": self.NO.MOR.L,  
 "SOUTH\_MOR\_R": self.SO.MOR.R, "SOUTH\_MOR\_T": self.SO.MOR.T, "SOUTH\_MOR\_L": self.SO.MOR.L,  
 "EAST\_MOR\_R": self.EA.MOR.R, "EAST\_MOR\_T": self.EA.MOR.T, "EAST\_MOR\_L": self.EA.MOR.L,  
 "WEST\_MOR\_R": self.WE.MOR.R, "WEST\_MOR\_T": self.WE.MOR.T, "WEST\_MOR\_L": self.WE.MOR.L  
 }  
 for slide in pres.slides:  
 for shape in slide.shapes:  
 if shape.name in volume\_placeholders.keys():  
 text\_frame = shape.text\_frame  
 text\_frame.clear()  
 p = text\_frame.paragraphs[0]  
 run = p.add\_run()  
 font = run.font  
 font.language\_id = MSO\_LANGUAGE\_ID.HEBREW  
 font.bold = True  
 font.italic = None  
 font.size = Pt(16)  
 font.color.rgb = RGBColor(255, 192, 0)  
 font.name = "Assistant"  
 if volume\_placeholders[shape.name] == 0:  
 data\_to\_push = "-"  
 else:  
 data\_to\_push = str(volume\_placeholders[shape.name])  
 run.text = data\_to\_push  
 pres.save("Morn\_Diagram.pptx")

**סוג**

מתודת הוספה

**תפקיד**

הוספת נפחי בוקר למצגת

**אופן פעולת המתודה**

המתודה מחזיקה מילון עם נפחי הצומת עבור שעת שיא בוקר. המתודה עוברת על כל הצורות הקיימות בשקופית במצגת של התרשים, היא 'מחפשת' צורות אשר השם שלהן קיים כמפתח במילון. במידה וקיימת כזו צורה, המתודה 'מנקה' את הצורה מטקסט קודם; המתודה מכניסה את הנפח למקום הנכון לאחר קביעת העיצוב לנפח. במידה והנפח שווה ל-0, המתודה מכניסה את הסימן "-" במקום.

בסוף הריצה, המתודה שומרת את המצגת החדשה שנשמרה (יוצרת קובץ זמני שנמחק בסוף ההרצה).

מתודה זו זהה כמעט לחלוטין להוספת נפחי אחר הצהריים ולכן לא יפורט על מתודה זו בנפרד.

**תיקונים אופציונליים**

* תיקון עיצוב של נפחי הבוקר

def add\_direction\_arrows(self, pres):  
 *"""  
 The method goes through all the shapes in Diagram pptx file and checks if it represents a string of arrows;  
 It adds the matching arrow string after organizing them in the right logical order,  
 using Organize\_arrows\_order method.  
 """* match\_colors\_to\_type = {"White": RGBColor(255, 255, 255), "Yellow": RGBColor(250, 201, 49)}  
 arrows\_placeholders = {"NORTH\_ARROWS": self.NO.LAN.Organize\_arrows\_order(),  
 "SOUTH\_ARROWS": self.SO.LAN.Organize\_arrows\_order(),  
 "EAST\_ARROWS": self.EA.LAN.Organize\_arrows\_order(),  
 "WEST\_ARROWS": self.WE.LAN.Organize\_arrows\_order()}  
 for slide in pres.slides:  
 for shape in slide.shapes:  
 if shape.name in arrows\_placeholders.keys():  
 text\_frame = shape.text\_frame  
 text\_frame.clear()  
 arrows\_list = arrows\_placeholders[shape.name]  
 for arrow in arrows\_list:  
 text\_frame = shape.text\_frame  
 p = text\_frame.paragraphs[0]  
 run = p.add\_run()  
 font = run.font  
 font.bold = False  
 font.size = Pt(50)  
 font.color.rgb = match\_colors\_to\_type[arrow[2]]  
 font.name = 'Traffic Arrows 2 Med normal'  
 run.text = arrow[0] \* arrow[1]  
 slide\_layouts = prs.slide\_master.slide\_layouts  
 pres.save("Dirc\_Diagram.pptx")

**סוג**

מתודת הוספה

**תפקיד**

הוספת ניתוב

**אופן פעולת המתודה**

המתודה מחזיקה מילון עם ניתוב הצומת לאחר סידור לוגי (קוראת למתודה שמסדרת משמאל לימין את החצים). המתודה עוברת על כל הצורות הקיימות בשקופית במצגת של התרשים, היא 'מחפשת' צורות אשר השם שלהן קיים כמפתח במילון. במידה וקיימת כזו צורה, המתודה 'מנקה' את הצורה מטקסט קודם; המתודה מכניסה את החץ למקום הנכון לאחר קביעת העיצוב לחץ. העיצוב לכל חץ נקבע בצורה הבאה: כל חץ מורכב ממערך עם 3 תאים: תו תואם לחץ הנדרש, כמות הופעת החץ, צבע החץ בהתאם לסוג הנתיב.

בסוף הריצה, המתודה שומרת את המצגת החדשה שנשמרה (יוצרת קובץ זמני שנמחק בסוף ההרצה).

**תיקונים אופציונליים**

* הוספת סוגי ניתוב חדשים.

def is\_oneway(self, pres):  
 *"""the method is being called when the junction has three directions; It checks if the fourth direction is a oneway direction. If it is, it adds a matching oneway road to the final diagram. """* opt\_oneway = self.G\_INF.ONEWAY  
 src = os.getcwd() + r"\Oneway\\"  
 type\_dict = {2: "NoWest", 3: "NoNorth", 4: "NoEast", 5: "NoSouth"}  
 OneWay = ""  
 if type\_dict[opt\_oneway] == "NoNorth":  
 if (self.EA.MOR.R > 0) or (self.EA.EVE.R > 0) \  
 or (self.WE.MOR.L > 0) or (self.WE.EVE.L > 0) \  
 or (self.SO.MOR.T > 0) or (self.SO.EVE.T > 0):  
 OneWay = "North"  
  
 if type\_dict[opt\_oneway] == "NoSouth":  
 if (self.WE.MOR.R > 0) or (self.WE.EVE.R > 0) \  
 or (self.EA.MOR.L > 0) or (self.EA.EVE.L > 0) \  
 or (self.NO.MOR.T > 0) or (self.NO.EVE.T > 0):  
 OneWay = "South"  
  
 if type\_dict[opt\_oneway] == "NoEast":  
 if (self.SO.MOR.R > 0) or (self.SO.EVE.R > 0) \  
 or (self.NO.MOR.L > 0) or (self.NO.EVE.L > 0) \  
 or (self.WE.MOR.T > 0) or (self.WE.EVE.T > 0):  
 OneWay = "East"  
  
 if type\_dict[opt\_oneway] == "NoWest":  
 if (self.NO.MOR.R > 0) or (self.NO.EVE.R > 0) \  
 or (self.SO.MOR.L > 0) or (self.SO.EVE.L > 0) \  
 or (self.EA.MOR.T > 0) or (self.EA.EVE.T > 0):  
 OneWay = "West"  
 if OneWay != "":  
 for slide in pres.slides:  
 oneway\_prop = {"North": [2915380, 0],"South": [2915380,4635945], "East": [4635945, 2915380], "West": [0, 2915380]}

# North: [(slide.width-pic.width)/2, 0]

# South: [(slide.width-pic.width)/2,(slide.height-pic.height)]

# East: [(slide.width - pic.width), (slide.width - pic.width)/2]

# West: [0, (slide.width - pic.width)/2]

img\_path = src + OneWay + "\_one\_way.png"  
 slide.shapes.add\_picture(img\_path, oneway\_prop[OneWay][0], oneway\_prop[OneWay][1])

**סוג**

מתודת בנייה

**תפקיד**

לבדוק האם קיימת בצומת זרוע שמתפקדת כחד סטרי, אם כן – להוסיף סימון חד סטרי במיקום הנכון (תמונה על המצגת).

**אופן פעולת המתודה**

הקריאה למתודה במידה וקיים צומת בעל 3 זרועות. המתודה מחזיקה מילון עם שמות מתאימים לזרוע החסרה בצומת. המתודה בודקת האם הזרוע החסרה היא זרוע של חד סטרי: במידה וקיים נפחים מובילים אליה מ3 הזרועות האחרות.

המשתנה OneWay מחזיק את שם הזרוע שהיא חד סטרי – במידה וקיימת כזו.

במידה וקיים חד סטרי, נוצר מילון עם קואורדינטות מתאימות לכל כיוון. קואורדינטות אלו יעזרו למיקום התמונה שמייצגת חד סטרי. חישוב הקואורדינטות מוסבר בשורה שמתחת למילון.

## Direction

חלקי הimport-

from Routes import \*  
from Lanes import \*

לפי הסדר: תת מחלקה פניות, תת מחלקה נתיבים.

class Direction:  
 *"""  
 A class used to represent a direction containing it's counts and lanes  
 The counts (volumes) are divided to two different properties: Morning route and Evening route.  
 """* def \_\_init\_\_(self, name):  
 *""" The constructor of the Direction class, called when a new instance of a class is created. Assigns values into the instance.* ***:param*** *name: Represents the name of the Direction (street / way)* ***:type*** *name : str* ***:return****: none  
 """* self.\_\_Morning\_route = Routes() # A property representing the volumes for the morning rush hour  
 self.\_\_Evening\_route = Routes() # A property representing the volumes for the evening rush hour  
 self.\_\_All\_lanes = Lanes() # A property representing the lanes for the specific direction  
 self.\_\_Name = name # A property representing the name or the way

מחלקת דיירקשן היא תת מחלקה של מחלקת העל - דיאגרם. מחלקה זו מייצגת את זרועות הצומת. היא מכילה מאפיינים של נפחי בוקר ואחר הצהריים וכן את הניתוב עבור הזרוע. כדי ליצור מופע חדש של זרוע, חייבים לתת שם לזרוע.

**פירוט מאפייני המחלקה**

* שני מאפיינים מסוג פניות (Routes).   
  אלו מכילים בתוכם את כל המידע הקשור לנפחי הזרוע. זוהי תת מחלקה אשר יפורט עליה בהמשך.
* מאפיין מסוג נתיבים (Lanes).   
  מכיל את הניתוב עבור הזרוע. זוהי תת מחלקה אשר יפורט עליה בהמשך.
* מאפיין מסוג שם (Name).   
  שם הזרוע כפי שנקבע בעת יצירה המופע.

**Getter / Setter**

הסבר לגבי גטרים וסטרים מופיע בחלק Main Diagram.

**נשים לב, כי לא ניתן לגשת למאפיין ללא שימוש בגטר או סטר המתאים.**

|  |  |  |
| --- | --- | --- |
| **Getter / Setter** | **Property** | **info** |
| MOR | self.\_\_Morning\_route | גישה אל מאפיין בוקר מסוג פניות |
| EVE | self.\_\_Evening\_route | גישה אל מאפיין אח"צ מסוג פניות |
| LAN | self.\_\_All\_lanes | גישה אל מאפיין כל הנתיבים מסוג נתיבים |
| NAME | self.\_\_Name | גישה אל מאפיין שם מסוג מחרוזת |

**פירוט מתודות המחלקה**

def empty\_direction(self):  
 *"""The method checks whether a direction is empty (has no volumes AND lanes);  
 It returns 0 if the direction is empty, and 1 if the direction is not empty"""* if self.LAN.Empty\_lanes() is False or self.MOR.Empty\_routes() is False or self.EVE.Empty\_routes() is False:  
 return 1  
 else:  
 return 0

**סוג**

מתודת בנייה

**תפקיד**

בדיקה האם זרוע מוגדרת כריקה – אין ניתוב השייך לה וכל הנפחים בה שווים ל-0.

**אופן פעולת המתודה**

המתודה קוראת למתודות אשר בודקות האם הנפחים בזרוע ריקים והאם לא קיים ניתוב בזרוע.

במידה ויש נפחי בוקר או ערב או ניתוב – הזרוע אינה ריקה והמתודה מחזירה 1, אחרת מחזירה 0.

## Routes

לא קיימות ספריות לייבוא ב-import.

class Routes:  
 *"""  
 A class used to represent the counting of a route with right, left and through  
 """* def \_\_init\_\_(self, left=0, right=0, through=0):  
 self.\_\_left = left  
 self.\_\_right = right  
 self.\_\_through = through

מחלקת רוטס היא תת מחלקה של מחלקת דיירקשן. מחלקה זו מייצגת את הפניות בזרוע, כלומר את הנפחים לכל פנייה בזרוע.

**פירוט מאפייני המחלקה**

* שלושה מאפיינים מסוג מספר, עבור נפחים שמאלה, ימינה וישר.

**Getter / Setter**

הסבר לגבי גטרים וסטרים מופיע בחלק Main Diagram.

**נשים לב, כי לא ניתן לגשת למאפיין ללא שימוש בגטר או סטר המתאים.**

|  |  |  |
| --- | --- | --- |
| **Getter / Setter** | **Property** | **info** |
| L | self.\_\_left | גישה אל מאפיין שמאלה מסוג מספר |
| R | self.\_\_right | גישה אל מאפיין ימינה מסוג מספר |
| T | self.\_\_through | גישה אל מאפיין ישר מסוג מספר |

**פירוט מתודות המחלקה**

def Empty\_routes(self):  
 *"""The method returns if True if the routes (volumes) == 0, and False if has volumes"""* all\_routes = [self.L, self.T, self.R]  
 return sum(all\_routes) == 0

**סוג**

מתודת בנייה

**תפקיד**

בדיקה האם הפניות ריקות - כל הנפחים בה שווים ל-0.

**אופן פעולת המתודה**

המתודה בודקת את סכום הנפחים ומחזירה אמת אם הנפחים שווים ל0.

## Lanes

לא קיימות ספריות לייבוא ב-import.

class Lanes:  
 *""" A class used to represent the lanes of the counter, divided to [regular,p\_transport] for each lane."""*

def \_\_init\_\_(self):  
 self.\_Separated\_Right = [0, 0]  
 self.\_Right\_Left = [0, 0]  
 self.\_All = [0, 0]  
 self.\_Left = [0, 0]  
 self.\_Through\_Left = [0, 0]  
 self.\_Through = [0, 0]  
 self.\_Through\_Right = [0, 0]  
 self.\_Right = [0, 0]

מחלקת ליינס היא תת מחלקה של מחלקת דיירקשן. מחלקה זו מייצגת את הניתוב בזרוע. כל סוג חץ מכיל מערך כאשר האיבר הראשון הוא כמות חצים רגילים והאיבר השני הוא חצי נת"צ.

**פירוט מאפייני המחלקה**

* שמונה מאפיינים מסוג רשימה, עבור כמות חצים מכל סוג (רגיל ונת"צ)

**Getter / Setter**

הסבר לגבי גטרים וסטרים מופיע בחלק Main Diagram.

**נשים לב, כי לא ניתן לגשת למאפיין ללא שימוש בגטר או סטר המתאים.**

|  |  |  |
| --- | --- | --- |
| **Getter / Setter** | **Property** | **info** |
| SR | self.\_Separated\_Right | גישה אל מאפיין ימינה-חופשי מסוג רשימה |
| RL | self.\_Right\_Left | גישה אל מאפיין ימינה-שמאלה מסוג רשימה |
| A | self.\_All | גישה אל מאפיין ימין-שמאל-ישר מסוג רשימה |
| L | self.\_Left | גישה אל מאפיין ישר מסוג רשימה |
| TL | self.\_Through\_Left | גישה אל מאפיין ישר-שמאלה מסוג רשימה |
| T | self.\_Through | גישה אל מאפיין ישר מסוג רשימה |
| TR | self.\_Through\_Right | גישה אל מאפיין ישר-ימינה מסוג רשימה |
| R | self.\_Right | גישה אל מאפיין ימינה מסוג רשימה |

**פירוט מתודות המחלקה**

def Empty\_lanes(self):  
 *"""The method returns if True if the lanes == 0, and False if has lanes"""* lanes = [self.SR, self.RL, self.A, self.L, self.TL, self.T, self.TR, self.R]  
 sum\_lanes = [sum(lane) for lane in lanes]  
 return sum(sum\_lanes) == 0

**סוג**

מתודת בנייה

**תפקיד**

בדיקה האם הניתובים ריקים – סכום כל רשימות החצים בזרוע שווים ל-0.

**אופן פעולת המתודה**

המתודה בודקת את סכום כמות החצים בזרוע ומחזירה אמת אם הנפחים שווים ל0.

def Organize\_arrows\_order(self):  
 *"""The method organizes the logical order of the arrows (left to right) and adds an arrow type to indicate whether the color of the arrow. The method takes care of complex arrows: arrows that contain 2 types, meaning - regular and PT. """* match\_arrow\_type = {0: "White", 1: "Yellow"}  
 complex\_arrows = {"D": 1, "A": 2, "W": 3, "S": 4}  
 complex\_arrows\_full = {12: [['U', 1, 'White'], ['F', 1, 'Yellow']],  
 13: [['U', 1, 'Yellow'], ['F', 1, 'White']],  
 23: [['N', 1, 'White'], ['U', 1, 'Yellow']],  
 24: [['N', 1, 'Yellow'], ['U', 1, 'White']],  
 32: [['B', 1, 'White'], ['F', 1, 'Yellow']],  
 33: [['N', 1, 'White'], ['U', 1, 'Yellow'], ['F', 1, 'White']],  
 34: [['N', 1, 'Yellow'], ['H', 1, 'White']],  
 35: [['N', 1, 'White'], ['H', 1, 'Yellow']],  
 36: [['N', 1, 'Yellow'], ['U', 1, 'White'], ['F', 1, 'Yellow']],  
 37: [['B', 1, 'Yellow'], ['F', 1, 'White']],  
 42: [['J', 1, 'White'], ['K', 1, 'Yellow']],  
 43: [['J', 1, 'Yellow'], ['K', 1, 'White']]  
 }  
 arranged\_arrows = []  
 lane\_arrows = [["L", self.L], ["A", self.TL], ["T", self.T], ["W", self.A], ["S", self.RL], ["D", self.TR],  
 ["R", self.R], ["E", self.SR]]  
 for arrow in lane\_arrows:  
 if arrow[0] in complex\_arrows.keys() and arrow[1][1] > 1:  
 complex\_arrow\_symbol = complex\_arrows[arrow[0]] \* 10 + arrow[1][1]  
 build\_arrow = complex\_arrows\_full[complex\_arrow\_symbol]  
 for arr in build\_arrow:  
 arranged\_arrows.append(arr)  
 else:  
 for arrow\_type in range(len(arrow[1])):  
 arranged\_arrows.append([arrow[0], arrow[1][arrow\_type], match\_arrow\_type[arrow\_type]])  
 return arranged\_arrows

**סוג**

מתודת בנייה

**תפקיד**

סידור וארגון החצים, כך שיהיו מסודרים לפי סדר לוגי וכן ארגון מידע נחוץ לעיצוב החץ במצגת (צבע, אות מתאימה).

**אופן פעולת המתודה**

המתודה מרכיבה מכל החצים רשימה של תת רשימות, כאשר כל תת רשימה מכילה את האות המתאימה לחץ (בפונט החצים של ג'אנק, כל אות מייצגת חץ אחר), כמות מאותו חוץ, צבע החץ. המתודה עוברת על מאפייני המחלקה ובדוקת עבור כל חץ כמה פעמים מופיע בזרוע. המתודה מפצלת בין ניתוב אחיד (רק רגיל או רק נת"צ) לבין ניתוב משולב (למשל, ישר רגיל וימינה נת"צ). במידה וחץ הוא משולב, המתודה מרכיבה את שני חלקי החץ; אופן הגדרת הניתוב הוא בהתאם למוסבר בקובץ CV.

במידה וחץ אינו משולב, הוא מוכנס גם כן כתת רשימה לרשימה הראשית arranged\_arrows. בסוף הריצה, המתודה מחזירה את הרשימה הראשית המכילה את כל תתי הרשימות.

## General\_Info

לא קיימות ספריות לייבוא ב-import.

*""" A class used to represent the general info of the junction"""*def \_\_init\_\_(self):  
 self.\_\_Capacity = 1800  
 self.\_\_NLSL\_Allowed = False  
 self.\_\_ELWL\_Allowed = False  
 self.\_\_5th\_Image = False  
 self.\_\_6th\_Image = False  
 self.\_\_Geometry\_N\_S = 3  
 self.\_\_Geometry\_E\_W = 3  
 self.\_\_Inflation = 1  
 self.\_\_Looping = False  
 self.\_\_OneWay = 0

מחלקת ג'נרל אינפו היא תת מחלקה של מחלקת העל - דיאגרם. מחלקה זו מייצגת מידע כללי של הצומת.

**פירוט מאפייני המחלקה**

* מאפיין קיבולת מסוג מספר. ברירת מחדל היא 1800.
* שני מאפיינים המאפשרים פניות שמאלה במקביל צפ' -ד ר'/ מז' – מע' מסוג בוליאני.   
  שימוש במאפיין זה מצריך בדיקה של גיאומטרית הצומת.
* שני מאפיינים המאפשרים תמונה 5 ו6 בחישוב עבור הטבלה.
* שני מאפייני גיאומטריה, כרגע ללא שימוש.
* מאפיין 'אינפלציה' לצורך הגדלת נפחים.  
  ניתן להגדיל את הנפחים באחוז מסוים, למשל לשנות ל1.2 עבור 20%.
* מאפיין 'לופינג', כרגע ללא שימוש.
* מאפיין חד סטרי עבור הצומת, מסוג מחרוזת.

**Getter / Setter**

הסבר לגבי גטרים וסטרים מופיע בחלק Main Diagram.

**נשים לב, כי לא ניתן לגשת למאפיין ללא שימוש בגטר או סטר המתאים.**

|  |  |  |
| --- | --- | --- |
| **Getter / Setter** | **Property** | **info** |
| CAP | self.\_\_Capacity | גישה אל מאפיין קיבולת מסוג מספר |
| NLSL | self.\_\_NLSL\_Allowed | גישה אל מאפיין פניות שמאלה במקביל צפ' – דר' מסוג בוליאני |
| ELWL | self.\_\_ELWL\_Allowed | גישה אל מאפיין פניות שמאלה במקביל מז' – מע' מסוג בוליאני |
| IMG5 | self.\_\_5th\_Image | גישה אל מאפיין מאפשר תמונה חמישית מסוג בוליאני |
| IMG6 | self.\_\_6th\_Image | גישה אל מאפיין מאפשר תמונה שישית מסוג בוליאני |
| GEONS | self.\_\_Geometry\_N\_S | גישה אל מאפיין גיאומטריה צפון דרום, כרגע ללא שימוש |
| GEOEW | self.\_\_Geometry\_E\_W | גישה אל מאפיין גיאומטריה מזרח מערב, כרגע ללא שימוש |
| INF | self.\_\_Inflation | גישה אל מאפיין אינפלציה מסוג מספר עשרוני |
| LOOP | self.\_\_Looping | גישה אל מאפיין 'לופ', כרגע ללא שימוש |
| ONEWAY | self.\_\_OneWay | גישה אל מאפיין חד סיטרי מסוג מספר |

## LRT\_Info

לא קיימות ספריות לייבוא ב-import.

class LRT\_Info:  
 *"""  
 A class used to represent the LRT info of the junc  
 """* def \_\_init\_\_(self):  
 self.\_\_LRT\_Dir = 0 # 0= no LRT, 1 = North & South, 2 = East & West , 3 = North, South, East & West  
 self.\_\_cycle\_time = 120  
 self.\_\_train\_lost\_time = 25  
 self.\_\_train\_headway = 4  
 self.\_\_MCU = 1  
 self.\_\_lost\_time = 0  
 self.\_\_Metro\_Dir = 0 # 0= no metro, 1 = North & South, 2 = East & West  
 self.\_\_LRT\_orig\_dir = [0, 0] # [0] = North & South, [1] = East & West

מחלקת אל.אר.טי אינפו היא תת מחלקה של מחלקת העל - דיאגרם. מחלקה זו מייצגת מידע לגבי רק"ל ומטרו, במידה וקיימים בצומת.

**פירוט מאפייני המחלקה**

* מאפיין כיוון הרק"ל מסוג מספר.  
  המספר שנקבע משמש לבחירת מחרוזת להוספה לתרשים (בהתאמה למפורט בתיעוד).
* מאפיין זמן מחזור מסוג מספר.
* מאפיין זמן רק"ל אבוד מסוג מספר.
* מאפיין זמן בין רכבות מסוג מספר.
* מאפיין מקדם ית"ן מסוג מספר.
* מאפיין זמן אבוד מסוג מספר.
* מאפיין כיוון מטרו מסוג מספר.
* מאפיין כיוון רק"ל מקורי מסוג רשימה.  
  מאפיין זה מגיע כרשימה, הוא מתורגם למספר בעזרת מתודה של המחלקה המיועדת לכך.

**Getter / Setter**

הסבר לגבי גטרים וסטרים מופיע בחלק Main Diagram.

**נשים לב, כי לא ניתן לגשת למאפיין ללא שימוש בגטר או סטר המתאים.**

|  |  |  |
| --- | --- | --- |
| **Getter / Setter** | **Property** | **info** |
| LRT\_Dir | self.\_\_LRT\_Dir | גישה אל מאפיין כיוון רק"ל מסוג מספר. |
| CYC\_TIME | self.\_\_cycle\_time | גישה אל מאפיין זמן מחזור מסוג מספר. |
| LRT\_LOST\_TIME | self.\_\_train\_lost\_time | גישה אל מאפיין זמן רק"ל אבוד מסוג מספר. |
| LRT\_HDWAY | self.\_\_train\_headway | גישה אל מאפיין זמן בין רכבות. |
| LRT\_MCU | self.\_\_MCU | גישה אל מאפיין מקדם ית"ן מסוג מספר. |
| GEN\_LOST\_TIME | self.\_\_lost\_time | גישה אל מאפיין זמן אבוד מסוג מספר. |
| Metro\_Dir | self.\_\_Metro\_Dir | גישה אל מאפיין כיוון מטרו מסוג מספר. |
| LRT\_Orig | self.\_\_LRT\_orig\_dir | גישה אל מאפיין כיוון רק"ל מקורי מסוג רשימה. |

**פירוט מתודות המחלקה**

def lrt\_orig\_to\_dir(self):  
 *"""The method translates the received LRT direction to a number that represents that LRT direction, stored in LRT\_Dir.  
  
 [North & South , East & West] -> 0/1/2/3  
 0 = no LRT  
 1 = North & South  
 2 = East & West  
 3 = North, South, East & West  
 """* lrt\_options = {'[0, 0]': 0, '[1, 0]': 1, '[0, 1]': 2, '[1, 1]': 3}  
 self.LRT\_Dir = lrt\_options[str(self.LRT\_Orig)]

**סוג**

מתודת בנייה

**תפקיד**

המרת מאפיין כיוון רק"ל אשר מגיע כרשימה, למספר אשר מייצג את כיווני הרק"ל.

**אופן פעולת המתודה**

המתודה הופכת את המאפיין למחרוזת אשר משמש כמפתח למספר המתאים. המתודה מכניסה את המספר המתאים למאפיין LRT\_Dir.

## טבלה

### בניה

### מחלקה

## ת"ז

### בניה

### מחלקה