מה זה פייתון?

רקע –

פייתון היא שפת תכנות עילית, המשתמשת במודל ביצוע מפורש (interpreter) והיא שפה רב תכליתית.
הפילוסופיה של עיצוב השפה היא שהקוד יהיה קריא כמה שאפשר.
כלומר שהקוד יהיה קרוב יותר לשפת בני האדם: נקי, לוגי ומצומצם יותר עבור פרויקטים גדולים וקטנים כאחד.
הפילוסופיה של השפה ,המסוכמת בקובץ "The zen of python" כולל מימרות (היגדים) כגון:

- יפה עדיף על מכוער.
- מפורש עדיף על מרומז (implicit)
- פשוט עדיף על מורכב.
- מורכב עדיף על מסובך.
- קריאות נחשב כמעלה.

יותר משהיא מכילה את הפונקציונליות שלה בהתבסס על הפילוסופיה של השפה, פייתון גם עוצבה להיות שפה הניתנת להתרחבות (extensible) מה שהופך אותה לפופולרית במיוחד כאמצעי להוספת ממשקים ניתנים לתכנות לממשקים קיימים.

פייתון היא שפה מרובת פרדיגמות המאפשרת תכנות מונחה עצמים, תכנות פרוצדורלי, ואפשר גם לומר שתכנות פונקציונלי.
כפי שצויין קודם פייתון היא שפה רב תכליתית, עם פייתון ניתן לבנות... אממ... אפשר לומר שהכל: אפליקציות רשת, אפליקציות לאנדרואיד, משחקים , יישומים מדעיים, AI ועוד.
חלק מהספריות של פייתון עומדות בחזית המרכזית של תחומים שלמים בתעשייה בניהם: הנדסת נתונים, ראייה ממוחשבת בפרט ולמידת מכונה בכלל, הקמת שרתים, בניית רכיבי תוכנה לחומרות, אוטומציה וכו'.
כחלק מהחברות שנבנו בשימוש השפה ניתן למנות את: youtube, spotify reddit השתמשו בשפה כדי לבנות את השרתים שלהם, dropbox , quora , גוגל השתמשה בשפה עוד מיום הקמת החברה והיא נחשבת השפה הראשית של החברה.
על פי האתר learnworthy.net פייתון היא השפה השנייה הכי פעילה בגיטהאב וגם השנייה הכי רווחית (המתכנתים של פייתון הם במקום השני במדד השכר מבין מתכנתים בשפות השונות), ובמקום הראשון בתחום ניתוח הנתונים שצובר תאוצה בשנים האחרונות.

את השפה יזם חידו ואן רוסו בשלהי שנות השמונים, והתחיל לממשה בתחילת שנות התשעים כממשיכת דרכה של השפה ABC.
שם השפה הוא מחווה לתוכנית הטלוויזיה "הקרקס המעופף של מונטי פייתון".

מהדר ומפרש-

ישנם שני סוגים של מימושי קוד - מימוש קוד באמצעות מהדר (compiler) ומימוש באמצעות מפרש (interpreter).
מימוש מהדר הוא מימוש הקוד ע"י "הידורו" לשפת מכונה, כלומר הקוד המתקבל עובר תהליך של קומפילציה שמחזיר קוד הכתוב בשפת מכונה שמתבצע.
ומימוש מפרש הוא מימוש שעובר שורה אחר שורה בקוד ,מתרגם אותה, ושמור את הנתונים בזיכרון לשימוש מאוחר יותר.

הקוד מופעל ישירות ולא עובר איזשהו תהליך סריקה מקדים מה שמאפשר ביצוע של קטע קוד ספציפי מתוך התוכנית, מבלי לעבור על כל הקוד קודם לכן.
את ההבדלים המרכזיים בין מפרש למהדר ניתן לסכם כך:

מהדר	מפרש
מעביר את כל הקוד בבאת אחת אחר סריקה.	מריץ שורה אחר שורה של קוד המקור.
מייצר קוד מכונה.	לא מייצר קוד מכונה.
משתמש ביותר זיכרון בהפעלת התוכנית (כי הוא מייצר את הקוד מכונה).	משתמש בפחות זיכרון בהפעלת התוכנית, אך יכול להיות שמשתמש ביותר זיכרון במדידה לאורך כל התוכנית.
מקמפל פעם אחת את הקוד ואז ניתן להשתמש בו בכל זמן.	קוד המקור מתורגם כל פעם מחדש.
מהיר.	איטי.

פייתון אומנם נחשבת לשפה המשתמשת במודל ביצוע מפורש, אך יש לה כמה אלמנטים שמשותפים לשפות מהודרות-
לפייתון קיים "מימוש-יחוס", מערכת שמגדירה את התנהגות הקוד הנכתב בשפה בשם Cpython.
מימוש הייחוס פועל בשני שלבים מקבילים: שלב "ההידור" ושלב ההרצה.
1. בשלב הראשון. קוד פייתון "מהודר" לשפת ביניים נמוכה מבוססת מחסנית והתוצאה נשמרת בקבצים עם הסיומות '.pyc' (סקריפטים של פייתון נכתבים עם הסיומת '.py').
2. בעת ההרצה המפרש מריץ את קוד הביניים, כלומר הקוד שהוא הגיע אליו לאחר שהוא רץ שורה שורה.
וכמובן שמשום שפייתון היא שפה מפורשת ניתן גם להריץ את הקוד בצורה אינטראקטיבית (ללא ה-"קומפילציה" של מימוש-הייחוס).

תכנות דינאמי ותכנות סטטי-

פייתון היא שפת תכנות דינאמית.
ההבדל בין שפות תכנות דינאמיות לשפות תכנות סטאטיות הוא כזה:
שפות תכנות סטאטיות הכוונה שהטיפוסים של המשתנים מוגדרים מראש, כך שהם נבדקים עוד לפני הרצת התוכנית, פעולה שנעשית בד"כ ע"י הקומפיילר(המהדר).
דוגמא לשפות תכנות סטאטיות הן כל אותן שפות שיורשות מ-C למשל ג'אווה, C++ C# וכו'.
הסינטקס שלהן בד"כ נראה כך:

```
public void foo() {
    int x = 5;
    boolean b = true;
}
```

כאן לפני שנותנים ערך ל-x או b מגדירים את הטיפוס שלהם (int ,bool,float…).
היתרון המשמעותי ביותר שיש לשפות כאלה הוא שהרבה יותר קל לזהות שגיאות תכנות לפני שהן קוראות בפועל, כלומר לזהות אותן עוד בזמן קומפילציה ולא בזמן ריצת התוכנית.
זה מקל גם על ה-IDE (integrated development environment) לסייע לנו לזהות שגיאות אף לפני שקימפלנו את הקוד, וככל שסביבת הפיתוח מכירה יותר את המשתנים יהיה לה קל יותר להגיד לנו אם השתמשנו במשתנה שלא הוגדר או שחייב להיות מוגדר עם יצירת אינסטנס שלו(למשל משתנים שהם const או מצביעים).
וכמובן זמן ריצה- מתי שהקומפיילר כבר מכיר את הקוד והטיפוסים השונים שבו הוא יכול להסיק מראש כמה זיכרון יש להקצות ואין צורך בחישובים מיותרים בזמן אמת, מה שמפחית את זמן ריצת התוכנית משמעותית.

ומהצד השני יש לנו שפות תכנות דינאמיות שבהן טיפוס המשתנים מוגדר בזמן ריצת התוכנית
למשל הקוד הבא בשפות דינאמיות הוא לגיטימי:

```
x = 1
x = 'one'
```

למרות שהגדרנו ל-x להיות משתנה מטיפוס int אנחנו משנים אותו אח"כ לטיפוס מחרוזת, על אף שהוא כבר הקצה משאבים לint.
שגיאות בשפות דינאמיות נבדקות רק כאשר התוכנית הגיעה לשורה שבה מתרחש הקוד עם השגיאה.
למשל בשפות סטטיות השגיאה הבאה תיתפס עוד בזמן קומפילציה אם לא ע"י ה-ide :

```
_zero = 1 - 'one'
```

בשפות דינאמיות נדע על השגיאה רק כאשר נגיע לאותה שורה בקוד וכנראה שהאופרטור '-' יזרוק אותה בזמן ריצה.
אך עם זאת יש גם הרבה יתרונות לשפות דינמיות על שפות סטאטיות:
השפות הדינמיות הרבה יותר תמציתיות- שפות סטטיות בדר"כ דורשות הרבה מאוד 'קוד מוקדם' עד שנוכל להשתמש במשתנה למשהו שימושי, למשל הקטע קוד הבא שנכתב בג'אווה לעומת אותה פונקציונליות רק בפייתון:

```
// Java Hello World
package helloworld;

import java.util.Scanner;


public class HelloWorld {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Please enter your name: ");
        String name = scanner.nextLine();
        System.out.println("Hello " + name);
    }
}


# Python Hello World
name = input("Please enter your name: ")
print(f"Hello {name}")
```

קל לראות כאן את ההבדל בין כמות השורות הדרושות בשביל לקבל קלט ולהדפיסו בשפת ג'אווה לעומת הקלות שבה ניתן לבצע את אותה הפעולה בדיוק רק בפייתון.
קוד קצר יותר הוא גם הרבה יותר קריא, לאו דווקא יותר מובן, אלא יותר קל לעקוב אחרי השתלשלות האירועים בקוד, מה שיכול להקל על מתכנתים חדשים "להיכנס" אליו.
גם יותר פשוט ללמוד שפות דינאמיות, השפה הרבה פחות עמוסה בחוקים והסינטקס הרבה יותר קרוב לצורת מחשבה אנושית.
עוד יתרון שיש לשפות דינאמיות שניתן להריץ את הקוד שלהן בחלקים ואין צורך לקמפל את כל התוכנית או ליצור קוד במיוחד כדי להריץ קטע קוד ספציפי.
וכמובן שימוש פשוט יותר בספריות חיצוניות (API וכו'). לשפות דינאמיות יש בדר"כ יותר ספריות חיצוניות שניתן לייבא לפרויקט ככל הנראה בשל הקלות שבה ניתן לתכנת בהן, אך זה גם יכול להוות חסרון במובן מסוים, שאומנם יש יותר מבחר לעומת ספריות של שפות סטטיות שבדר"כ רמת המתכנתים בהן יותר גבוהה.
,אך רמת המתכנתים שיש לספריות בשפות דינאמיות היא לא חד משמעית בשל הקלות ללמוד אותן

ניהול הזיכרון בפייתון-

שפות כמו C או ++C דורשות מהמתכנתים לנהל את הזיכרון של התוכנית, למשל לבקש מהערימה הדינמית שתקצה זיכרון חדש עבור אובייקט מטיפוס מסוים ע"י פקודה כלשהי (malloc או new למשל), ואז צריך להפעיל פקודה נוספת כדי להחזיר את אותו זיכרון שהוקצה למערכת(free או delete).
בשפות כאלה האחריות על המתכנת גוררת לא פעם "זליגת זיכרון" ,כלומר שהזיכרון שהוקצה לא חוזר לזיכרון המערכת למרות שאין בו כבר שימוש, דבר שיכול לגרום למערכת לאבד את כל הזיכרון ככל שהתוכנית תרוץ יותר זמן מבלי להחזיר את הזיכרון שלא בשימוש.
יותר מאוחר (או במקביל) קמו שפות חדשות המנהלות לעצמן את הזיכרון, דבר שפתר בצורה חלקית את בעיית "זליגת הזיכרון" , אך לא לחלוטין.
המנגנונים שבהם משתמשות השפות כדי לפתור את בעיית זליגת הזיכרון הם Garbage collector ו- Reference count.
Garbage collector – הוא בעצם סוג של דימון(daemon) – תרד(thread) שמופעל פעם בכמה זמן, ותפקידו הוא למצוא אובייקטים שלא ניתן לגשת אליהם ומחזיר אותם למערכת.
Reference count - הוא מנגנון ששומר ליד כל אובייקט מספר שמציין כמה משתנים מתייחסים לאובייקט, כשהמספר יורד לאפס אוטומטית הזיכרון חוזר למערכת ההפעלה.
פייתון משתמשת בשני המנגנונים, ב-Reference count היא משתמשת כדי לזהות מידית אובייקטים שאין להם מצביעים, וב- Garbage collector עבור מקרים של מעגלים בזיכרון- קבוצת אובייקטים שמצביעים אחד על השני, או אובייקט שמצביע על עצמו.

PIP מה זה?

מערכת ניהול חבילות היא תוכנה המאפשרת לבצע אוטומצית התקנה והסרת חבילות תוכנה וכן עדכונן והגדרתן.
מנהל החבילות מהווה יתרון בעת התקנת תוכנה בשל טיפולו בתחומי תאימות התוכנה לפלטפורמה עליה היא מתקנת, גרסתה ומניעת בעיות נוספות.
המערכת מתכתבת לרוב עם בסיס נתונים תכנותי העשוי להכיל אלפי אם לא מאות אלפי חבילות תוכנה.

למי שיצא להשתמש במערכות הפעלה מבוססות לינוקס בוודאי יצא להשתמש במנהל החבילות של המערכת.
למשל למערכת בהפצת דביאן ונגזרותיה (למשל אובונטו או מינט) יש את apt.

לפייתון יש מעל ל130,000 ספריות (רשומות) שלחלקן לא ניתן למצוא מקבילה בשום שפת תכנות אחרת.
pip היא מערכת ניהול חבילות שנכתבה בפייתון ומשמשת להתקנה וניהול של חבילות תוכנה עבור קבצי השפה.
היא מחוברת לספרייה מקוונת של חבילות ציבוריות ופרטיות (בתשלום) הנקראות Python package index.
רוב ההפצות של פייתון מגיעות עם pip מותקן מראש ,והחל מפייתון 2.7.9 (בסדרת פייתון 2) ומפייתון 3.4 (בסדרת פייתון3) היא מוגדרת כברירת מחדל.
היתרון המשמעותי של pip הוא הפשטות בממשק שורת הפקודה שמאפשר להתקין חבילות תוכנה של פייתון בשורת קוד אחת בלבד:

```
pip install some-package-name
```

וגם מחיקה של ספרייה באותה הקלות:

```
pip uninstall some-package-name
```

כברירת מחדל pip יתקין את החבילה בגרסה החדשה ביותר של פייתון שמותקן במחשב , אך ניתן גם להגדיר באיזו גרסה להשתמש:

```
pip${python-version-number} install some-package-name
```

ניתן למצוא חבילות ולפרסמן באתר הרשמי של Python package index-
https://pypi.org/

-JUPYTER NOTEBOOK

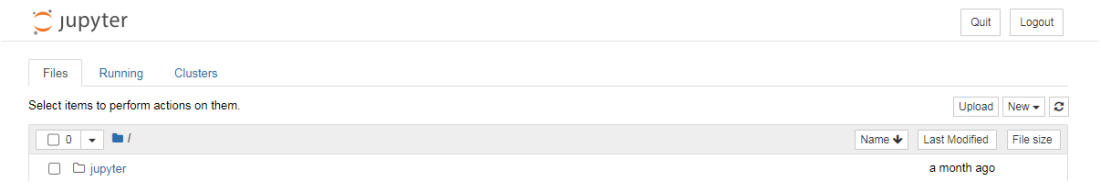
Jupyter notebook (ג'ופיטר) הוא פרוייקט קוד פתוח ללא מטרות רווח מבוסס פייתון שמאפשר לפרק קוד פייתון לכמה חלקים (sections), ולהריץ כל חלק בנפרד.
הפרויקט תומך כתיבת חלקי קוד פייתון וחלקי טקסט הנכתבים בפורמט דומה לכתיבת html.
את מרבית קבצי הקוד של הסיכומים העלנו לקובצי ג'ופיטר (קבצים עם סיומת '.ipynb') וניתן למצוא אותם בתיקייה '0.code' בכל תיקייה של שיעור בקורס.
כדי להפעיל קבצי ג'ופיטר יש להתקין את הספרייה.
במידה ועדיין לא התקנתם פייתון3 אתם מוזמנים להסתכל בפרק הבא בנושא התקנה של פייתון, אבל אם כבר יש לכם פייתון ההתקנה פשוטה ביותר- משורת הפקודה (טרמינל ללינוקס, cmd לווינדוס וכו') נכניס את הפקודה הבאה:

```
pip install notebook
```

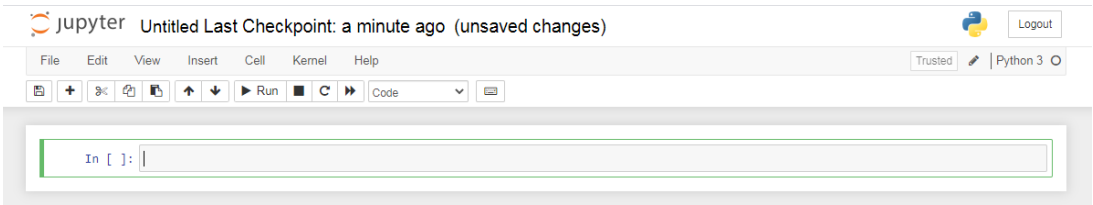
ברכותי התקנתם את הספרייה! לא קשה. עכשיו כדי להפעיל אותה נפתח חלון חדש במסוף ניכנס לתיקייה שבה נרצה לשמור את הקוד, או להריץ קוד קיים, ונקליד את הפקודה:

```
jupyter notebook
```

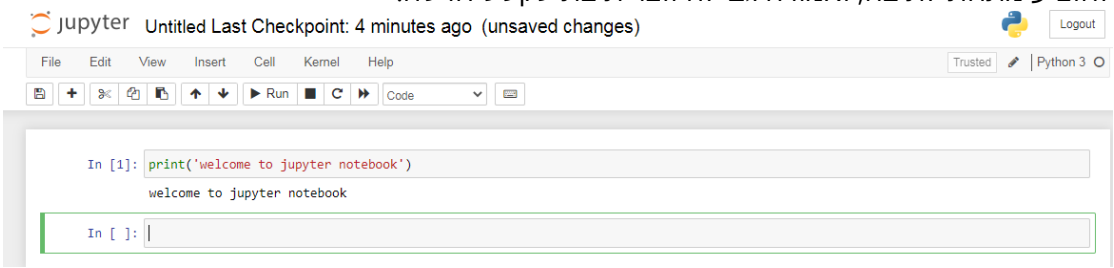
אם אתם משתמשים במערכות ווינדוס אפשר להיכנס לתיקייה ובשורת החיפוש (איפה שכתוב ה-path של התיקייה) לכתוב jupyter notebook.
זה אמור להקים שרת ולהפעיל אוטומטית את האתר שהוא מקים. האתר אמור להציג את התיקייה שממנה הפעלנו את הקוד, אני הפעלתי מתיקייה שמכילה תיקייה נוספת שנקראת jupyter :



בשביל ליצור מסמך חדש נלחץ על new -> Python 3 או 'python roots' והוא יפתח דף חדש במחברת ג'ופיטר:



ועכשיו כל מה שצריך זה לכתוב קוד פייתון בתיבת הטקסט, וכדי להריץ את הקוד נלחץ shift+enter . הפלט אמור להופיע מתחת לתיבה, ואמורה גם להיווצר תיבת טקסט חדשה:



כדי להוסיף תיבות בלי להריץ קודים אפשר להתשמש ב-+ מצד שמאל למעלה.
כמו כן ניתן ליצור תיבות טקסט ולא תיבות קוד אם נעמוד של הכיתוב מצד שמאל In []: ונלחץ על M .
התיבת טקסט אמורה להכתב כמו קובץ md. (קבצי readme של גיטהאב) או קבצי html.
כדי לשמור אפשר עם ctrl+s או באייקון של הדיסקט שמופיע מצד שמאל למעלה. ונוכל להכינס לקובץ מתי שנרצה מהתיקייה הראשית.
וזהו, זה פחות או יותר כל מה שצריך לדעת על jupyter notebook, לעוד פיצ'רים שיש לספרייה להציע ניתן למצוא בסרטון הזה או באתר של הפרויקט.