# פרסום ספריות ב-Pypi

ועכשיו לנושא המרכזי של חלק התכנות של הקורס:   
לאחר שעבדנו קשה על בניית הפרויקט שלנו, נרצה שעוד מתכנתים יוכלו ליהנות מפרי עמלנו, כלומר שגם מתכנתים אחרים יוכלו לעשות pip install לספרייה שבנינו.  
דבר שראוי לציין, ל-python package index אין איזשהו סנן לספריות שמועלות לאתר, מתכנת יכול לעלות מודולים ברמה של numpy או scipy בדיוק כמו שהוא יכול לעלות מודול שמכיל רק הדפסה של ''hello world" למסך.  
כמובן שכמתכנתים נשאף לעלות רק ספריות איכות, אבל לצורך הדוגמא במסמך הבא נעלה ספרייה פשוטה שלא תורמת יותר מידי לקהילה.   
  
מבנה סטנדרטי של ספריית **pypi** –

המבנה הסטנדרטי של הספרייה מכיל כמה קבצים שחייבים להיות בספרייה כדי שיהיה ניתן להוריד אותה מהאתר:   
• קובץ readme עם סיומת '.md' שיסביר על הספרייה בכמה מילים עבור מתכנתים אחרים שנתקלו בספרייה באתר.   
• קובץ setup.py, נסביר עליו בהמשך.  
• קובץ LICENSE גם עליו נסביר בהמשך.  
• הספרייה עצמה בתיקייה נפרדת. מומלץ שהספרייה תכיל קובץ \_\_init\_\_.py כדי להכריז עבור הפרויקט שהספרייה היא לא מרחב שם גרידא אלא תיקייה של הפרויקט ,להסבר מורחב יותר מומלץ לחזור למסמך מג'אווה לפייתון- מבנה של מסמך פייתון, שם תמצאו הסבר מפורט מה בדיוק ההבדל בין תיקייה עם הקובץ ובלעדיה. הקובץ \_\_init\_\_.py לא אמור להכיל איזשהו תוכן, אולי מקסימום להגדיר בו משתנים קבועים וכדו'.   
   
  
**\_\_main\_\_.py:**

משהו שלא יצא לנו לדבר עליו הוא על הרצת ספריית פייתון ולא סקריפט בודד.   
לפעמים נרצה להריץ ספריית פייתון ולא קובץ סקריפט בודד, למשל אם בנינו מנוע גרפי או תוכנה אחרת, ונרצה להריץ אותה על המחשב. אפשרות אחת היא להריץ את הסקריפט הראשי של הספרייה מה שמצריך להיכנס פנימה ולמצוא אותו אפשרות נוחה יותר תהיה להריץ את כל הספרייה משורת הפקודה, נניח לספרייה קוראים my\_package:

python –m my\_packge

כדי להשתמש בדגל '–m ' (על ספריות) נצטרך להגדיר קובץ שממנו רצה התוכנית הראשית (\_\_name\_\_==\_\_main\_\_) , והקובץ צריך להיקרא \_\_main\_\_.py , ואז כאשר מריצים את הפקודה היא מחפשת את הקובץ ומריצה אותו.   
כלומר היא חוסכת כניסה לספרייה והרצה של הקובץ הראשי ידנית.  
אין חובה להגדיר קובץ זה, למעשה מרבית הספריות שהשתמשנו בהן לא מכילות בכלל קובץ \_\_main\_\_.py , אבל חשוב שנכיר אותו.   
הערה: הדגל –m יכול לשמש גם כדי להריץ סקריפטים ולא רק ספריות, ולא צריך להוסיף סיומת '.py' בהרצה של סקריפטים.

**pip requirement –**

עוד נושא שלא יצא לנו לדבר עליו במהלך הקורס הוא על דגלים שונים שיש למערכת ניהול החבילות של פייתון.   
ברמת העיקרון לא היינו צריכים איזשהו דגל מיוחד במהלך הקורס כדי להתקין חבילה ספציפית, וגם אין לנו צורך ממש להכיר את כל האפשרויות להתקנה.   
חשוב להכיר אבל את '-U' שמעדכן גם את החבילות שנצרכות בהתקנת החבילה לגירסה העדכנית שלהם.   
ועוד דגל שחשוב להכיר הוא '-r' או '--requirement'. הדגל הזה מאפשר לנו להתקין יותר מספרייה אחת ע"י שימוש בקובץ טקסט שמכיל רשימה של ספריות שאותן נרצה להתקין.   
למשל אם אנחנו רוצים להתקין את החבילות flask ו-networkx, במקום להתקין כל ספרייה בנפרד, נכין קובץ טקסט שבתוכו מופיעים הקבצים שאותם אנחנו רוצים להתקין (כל חבילה בשורה נפרדת), נניח קוראים לקובץ requirement.txt אז נוכל להקין את הקבצים בבאת אחת כך:

pip install -U -r requirement.txt

איך זה קשור לנושא שלנו? כשאנחנו יוצרים ספרייה נרצה להוסיף קובץ requirement כזה כדי שמתכנתים אחרים שמשתמשים בספרייה יוכלו להתקין אותה עם כל ה-dependencies שלה, כלומר מתי שהם יתקינו את הספרייה של הפרויקט עם pip install כל הספריות הנלוות לפרויקט שלנו יותקנו יחד איתה.   
למעשה אם אנחנו משתמשים בסביבת עבודה וירטואלית, כפי שראינו בשיעור בנושא flask ,אנחנו לא צריכים לעבוד קשה ,ויכולים להגדיר את קובץ requirement בקלות עם הפקודה הבאה:

pip freeze > requirement.txt

מה שעושה הפקודה זה לקחת את כל הספריות שמותקנות דרך pip ולהכניס אותן לקובץ טקסט שקוראים לו requirement.   
התנאינו בשימוש בסביבת עבודה וירטואלית ,אך האמת שגם אם לא היינו משתמשים בסביבה וירטואלית היינו מצליחים לעשות את זה, הבעיה שהיינו מכניסים הרבה מאוד ספריות שלא נצרכות בכלל עבור המתכנתים האחרים.

הכנת החבילה לפני פרסום-

החלק הראשון בתהליך הפרסום, ובין המסובכים בו ,הוא לתת שם (טוב) לספרייה.  
כל הספריות ב-pypi צריכות להיות עם שמות ייחודיים.  
עם כמעט רבע מיליון חבילות באתר סיכוי גדול שהשם של הספרייה שאנחנו רוצים כבר תפוס.   
טיפ: בשביל למצוא אם השם תפוס נוכל להשתמש בשורת החיפוש באתר.

השם של החבילה לא חייב להיות זהה לשם שאתו נייבא את הספרייה- נוכל לקרוא לספרייה my\_package, וניתן לה שם ייחודי באתר, למשל my\_very\_first\_package ואז שיורידו את החבילה מ-pypi יורידו אותה כך:

pip install my\_very\_first\_package

אבל כשנייבא את הספרייה לפרוייקט נייבא אותה כ-my\_package:

import my\_package

כמובן שעדיף לקרוא לחבילה באותו השם שאיתו אנחנו מייבאים אותה, אחרת סתם נבלבל את המשתמשים האחרים.

קונפיגורציה לחבילה:

בכדי שהספרייה תוכל לעלות ל-pypi, צריך לתת קצת אינפורמציה בסיסית עליה.   
את האינפורמציה אנחנו מספקים מקובץ שנקרא setup.py.   
הקובץ אמור להיות בתיקייה הראשית של הפרויקט.  
הקובץ אמור לייבא שני מודולים:   
• pathlib שנועד לייבוא של קבצים מהאזור של התיקייה כמו קובץ readme או requirement וכדו'.   
• מהספרייה setuptools צריך לייבא את הפונקציה setup().   
הספרייה setuptools היא ספרייה שמקלה על המתכנתים לבנות ולהפיץ ספריות פייתון, במיוחד כאלו שמסתמכות על חבילות אחרות.  
שתי הספירות נמצאות בספרייה הסטנדרטית של פייתון כך שאין צורך להתקין שום דבר חיצוני.  
הפונקציה setup() מחייבת כמה פרמטרים:   
• שם(name): כאן אנחנו בעצם מגדירים איך תיקרא החבילה ב-pypi (מחרוזת).  
• גירסה (version): הגירסה הנוכחית של החבילה, מגיע כמחרוזת.   
• חבילות (packages ): התיקיות ותתי התיקיות של הספרייה, הארגומנט אמור להישלח כרשימה של מחרוזות.  
אם יש לנו הרבה תיקיות אפשר להשתמש בפונקציה setuptools.find\_packages() כדי לקבל את כל התיקיות בפרוייקט. אפשר גם להורות לפונקציה אילו תיקיות לא נרצה לייבא עם פרמטר exclude שמקבל tuple או רשימה של שמות של תיקיות שלא נרצה שיילקחו בחשבון בחבילה, למשל טסטים (tests) יכול להיות שלא נרצה וכו':

packages=find\_packages(exclude=("tests",)),

אומנם רק שם, גירסה וחבילות הם פרמטרים שחייבים להוסיף, אבל אם נכניס עוד פרמטרים יהיה הרבה יותר קל למצוא את החבילה שלנו ב-pypi ,לאמת שאכן זאת החבילה שלנו ונקל על מתכנתים אחרים בהתקנת החבילה.  
למשל ממומלץ להגדיר את הקובץ readme בפונקציה כדי שנוכל לראות אותו באתר של pypi .  
או להגדיר את שם המחבר של הספרייה ודרכי התקשרות.   
כמו כן לפעמים הספרייה מצריכה כמה חבילות חיצוניות כמו numpy למשל, שלא מגיעות עם הספרייה הסטנדרטית, במקרה כזה נצטרך פרמטר install\_requires שמכיל רשימה של ספריות שצריך להתקין כדי להשתמש בחבילה שלנו(בהמשך נראה כיצד להשתמש בקובץ requirement.txt שדיברנו עליו קודם לכן).  
עוד משהו מגניב שאפשר להגדיר הוא entry\_points , entry points הם פונקציות שאנחנו מגדירים שיופעלו מתוך שורת הפקודה בשימוש בפקודה מוגדרת מראש. למשל יש לנו בקובץ \_\_main\_\_.py פונקצית main() שנרצה שתופעל בשורת הפקודה כאשר מקלידים my\_package\_main נוכל להגדיר את זה ע"י הוספה של הפרמטר entry\_points שמקבל מילון שהמפתח הוא איפה אמורה להיות מוגדרת הפקודה, והערך הוא רשימה של מחרוזות שמכילות את שם הפקודה שירשור עם '=' שירשור ה-path של קובץ, נקודותיים ושם הפונקציה.   
להמחשה:

  setup(

    ...,

    entry\_points={

      "console\_scripts": ["my\_package\_main =my\_package.\_\_main\_\_:main",]

    }

  )

ולאחר שהמשתמש יוריד את הספרייה הוא יוכל פשוט לכתוב my\_package\_main בטרמינל או ב-cmd וזה יריץ אוטומטית את הפונקציה main() של הקובץ my\_package/\_\_main\_\_.py .

דוקומנטציה:

לפני שמפרסמים את החבילה יש צורך בהסבר על החבילה.   
בהתאם לגודל החבילה הדוקומנטציה יכולה להיות קטנה כקובץ readme פשוט, או גדולה כמו אתר אינטרנט, לרוב חבילות ה-numpy stack למשל יש אתר אינטרנט.   
לכל הפחות יש צורך בקובץ readme עם הפרויקט. קובץ readme טוב אמור לתאר את הפרויקט ולהסביר למשתמש איך להתקין אותו ולהשתמש בו.  
בדרך כלל נרצה להוסיף את קובץ ה-readme עם הפרמטר long\_description של הפונקציה setup(). זה יציג את הקובץ באתר של pypi.  
קבצי readme אמורים להגיע בפורמט markdown (קבצים עם הסיומת '.md'), כמו קבצי ה-readme של גיטהאב, או בפורמט reStructuredText (סיומת '.rst').  
אם צירפתם את קובץ ה-readme לפונקציה צריך להוסיף עוד פרמטר שנקרא long\_description\_content\_type שמתאר את סוג הקובץ שצורף למשל עבור קובץ markdown צריך לפרמטר את הערך 'text/markdown', ועבור קובץ reStructuredText צריך לתת לו את הערך ' text/x-rst'.

כדי להוסיף קובץ מתוך הספרייה לפונקציה נוכל להשתמש במודול pathlib או בקריאת קובץ פשוטה:

with open("README.md", "r") as fh:

    README = fh.read()

או עם הספרייה pathlib כדי לא להשתמש ב-context manager:

# The directory containing this file

HERE = pathlib.Path(\_\_file\_\_).parent

# The text of the README file

README = (HERE / "README.md").read\_text()

אם לסכם את מה שראינו קובץ setup פשוט אמור להיראות בערך כך:

import pathlib

from setuptools import setup, find\_packages

# The directory containing this file

HERE = pathlib.Path(\_\_file\_\_).parent

# The text of the README file

README = (HERE / "README.md").read\_text()

# This call to setup() does all the work

setup(

name="my\_package",

version="1.0.0",

description="An example of a pypi package",

long\_description=README,

long\_description\_content\_type="text/markdown",

url= '...', # link to your github

author="...",

author\_email="...",

packages=find\_packages(exclude=("tests",)),

include\_package\_data=True,

entry\_points={

"console\_scripts": ["my\_package\_main=my\_package.\_\_main\_\_:main",]

},

)

בנוגע ל-requirement : אם יש צורך להוסיף requirement לפונקציה setup() צריך להעביר אותו בצורה של רשימה של מחרוזת, וכל תא ברשימה יהיה שם של ספרייה עם התיאור שלה כפי שמופיע בקובץ requirement.txt (או כל קובץ שבו כתבתם את רשימת החבילות הנלוות לפרויקט).  
בשביל זה נצטרך לקרוא שורה אחר שורה מתוך הקובץ, ולהכניס אותה בנפרד כתא חדש לרשימה:

def parse\_requirements\_file(filename):

    with open(filename) as fid:

        requires = [lin.strip() for lin in fid.readlines()]

    return requires

REQUIRES = parse\_requirements\_file("requirement.txt")

אחר כך בפונקציה setup() נוסיף את המשתנה לפרמטר install\_requires:

setup(

...,  
 install\_requires = REQUIRES  
 ...

טקסט