מג'אווה לפייתון-מבנה של תוכנית פייתון

בדומה לתוכניות של ג'אווה, לכל פרויקט יש כמה מסמכים או סקריפטים, ולכל מסמך יש את הסיומת שלו-
בג'אווה: 'java.' או 'py.' בפייתון.
לתוכניות בשפות סטטיות יש כמה קבצים וקובץ ראשי המכיל פונקציית main שרץ בתחילת התוכנית "ומנהל" אותה.
בפייתון לעומת זאת המפרש מריץ סקריפטים החל מהמסמך הראשון ועד המסמך האחרון לפי סדר מסוים ,ואין צורך בפונקציה ראשית שפועלת בתחילת התוכנית,
אך על כל פנים, ניתן להגדיר פונקציה ספציפית כנקודת תחילת התוכנית, זה שימושי בעיקר כדי להבין כיצד התוכנית עובדת.
הדרך הנהוגה להגדרת פונקציה ראשית היא ע"י "__if __name__ == "__main :

```
def main():
    print("Hello World!")


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

סקריפט ו-module -

כדי להבין מה זה __name__ נצטרך להבין תחילה איך להריץ קבצי פייתון-
ישנם שתי דרכים (עיקריות) להורות לפייתון להריץ את קבצי הפייתון:
פייתון היא שפה שמפעילה "מפרש" שעובר שורה אחר שורה של הקוד ומבצע אותה או שומר בזיכרון את הפעולה של אותה שורה.
עם התקנת השפה על המחשב אנחנו מתקינים גם מצב "אינטראקטיבי", שהוא תוכנית שרצה בזמן אמת, מחכה לפקודות בשפת פייתון ומבצעת אותם.
כדי להפעיל את המצב האינטראקטיבי, צריך רק להפעיל את הפקודה python או python3 בטרמינל, והוא יקרה למצב הנ"ל.
נוכל לייבא למצב הקיים גם modules של פייתון שכתבנו מראש וכך להפעיל אותם מתוך התוכנית האינטראקטיבית ע"י הפקודה import ושם הmodule או הpath שלו (במידה והטרמינל לא נמצא באותה תיקייה של ה-module).
הערה: כדי לצאת ממצב אינטראקטיבי צריך להפעיל את הפונקציה exit() שיוצאת מתהליך וחוזרת לטרמינל, ניתן גם ללחוץ ctr+d.

דרך נוספת להריץ את הקוד היא כסקריפט, ואז המשתמש צריך להזין בטרמינל, בתיקייה של הקבצים את הפקודה:
'python3 name_of_the_file.py' (הפקודה היא python3 או python ואז שם הסקריפט בסיומת 'py.').

סקריפט הוא קובץ טקסט של פייתון(קובץ 'py.') שמכיל קוד פייתון שמטרתו לרוץ ישירות ע"י המשתמש.
לעומתו module הוא קובץ טקסט (קובץ 'py.') המכיל קוד פייתון ומטרתו להיות תוכנית שמיובאת לתוכניות פייתון אחרות, אז בעצם ההבדל בניהם הוא שהראשון נועד להרצה והשני נועד לייבוא לתוכנית רצה.

לפעמים נרצה להשתמש בסקריפט אחד בתוך סקריפט אחר כmodule.
בזכות המשתנה __name__ נוכל לקבוע אם נרצה להריץ את הקובץ כסקריפט או לייבא אותו כmodule.
כשאנחנו מריצים את הקובץ כסקריפט המשתנה __name__ יהיה שווה למחרוזת " "__main__ אבל כשמריצים את הקוד כmodule בתוך תוכנית אחרת, אז ערך המשתנה הוא שם הקובץ.

עכשיו מה שיקרא אם נריץ את הסקריפט הוא שהמפרש יבדוק מה ערכו של __name__ ואם המשתנה יהיה שווה __main__ הוא יבין שהקוד נקרא כסקריפט, ויריץ אותו החל מנקודה שאותה צוינה כנקודת ההתחלה, למשל

למעלה קבענו שהתוכנית תתחיל מהפונקציה main.
אך אם המשתנה לא שווה ל- __main__ המפרש יבין שזהו module בתוך תוכנית אחרת, ובמקרה כזה לא יהיה צורך להגדיר נקודת התחלה, כי התוכנית שמריצה את הmodule תקבע באילו משתנים היא רוצה להשתמש מתוך הקובץ.
ראו דוגמא ב-3.0 ← מבנה של מסמך פייתון.code ← with_or_without_main.ipynb .

בלוקים וסיומות של פקודה-

בשפות כמו ג'אווה אנחנו מציינים סיום פקודה בנקודה פסיק ';', ותחילת קטע וסיומו עם סוגרים מסולסלים '{ }'.
כל קטע קוד המתחיל בסוגר מסולסל (פותח) ומסתיים בסוגר מסולסל (סוגר) נקרא בלוק.
הסוגריים מאפשרים לתוכנה לזהות היכן נגמר הקטע, אך הם לא מחייבים לשמור על איזשהו סדר, מה שבהרבה מקרים גורם למתכנתים מתחילים ,אבל לא רק, ליצור קוד מבולגן שקשה לעקוב אחר הלוגיקה שלו, דבר שמקשה על מתכנתים חדשים "להיכנס" לתוכנית.
פייתון היא שפה שבנויה בהתאם לאיזשהו מניפסט שמחייב אותה, לכן כדי להקפיד על עיקרון "קוד נקי" וכדי שהשפה תהיה דומה ככל הניתן לשפה אנושית, בלוק בפייתון מצוין בנקודתיים ,ירידת שורה ובהזחות במקום בסוגרים עגולים ונקודה פסיק.
סיומות של פקודה לא נגמרות עם איזשהו סימן מיוחד אלא פשוט בירידת שורה, מה שמקנה לשפה מראה של כותרת ופירוט או רשימת סופר, שהיא בהחלט יותר אנושית מהמבנה המוכר של שפות תכנות כמו ג'אווה.
דוגמא:

```
public class Test { public static void main(String args[]) {
        String array[] = {"Hello, World", "Hi there, Everyone", "6"};
        for (String i : array) {System.out.println(i);}}}
```

קוד חוקי בג'אווה שמדפיס למסך כל אחת מהמחרוזות במערך array, הקוד לא מחויב לחוקי אסתטיקה קפדניים במיוחד.
ואותו הפונקציונליות בדיוק בפייתון:

```
stuff = ["Hello, World!", "Hi there, Everyone!", 6]
for i in stuff:
    print(i)
```

כל שורה היא פקודה נפרדת, וכל בלוק בנוי מכותרת (במקרה הזה ההצהרה על לולאה), נקודתיים, ותחילת הבלוק בשורה מתחת עם רווח מתחילת מיקום הכותרת.
במבנה כזה נוצרת איזושהי היררכיה- כל הפקודות שנחשבות שוות אחת לשנייה, כלומר מוכלות באותו הבלוק, יתחילו מאותה נקודה רק בשורות נפרדות, כך שבמקרה כמו הקוד המצוין לעיל, אם נרצה להוסיף פקודה שתבוא בסוף הלולאה, נוכל לזהות אותה בקלות גם אם אנחנו לא כותבי הקוד, כי היא פשוט תתחיל מאותה נקודה שהתחילה הכותרת של הלולאה:

```
 stuff = ["Hello, World!", "Hi there, Everyone!", 6]
for i in stuff:
    print(i)
print("end")
```

הזחות הן דבר מרכזי בפייתון ואם הפקודה שבאה באותו הבלוק לא זהה ברווח לשאר הפקודות בבלוק, או שהיא לא בדיוק במרחק המתאים מהכותרת המפרש לא יכול לזהות את הפקודה, או שהוא יזהה אותה בבלוק אחר.

הערות (comments)-

כתיבת הערות בקוד עוזרת לתאר את תהליך החשיבה של המתכנת, עוזרת לו ולאחרים להבין יותר מאוחר את כוונתו בכתיבת שורות ספציפיות או הקוד בכללותו, עוזרת במציאת שגיאות ותיקונם, שיפור הקוד ושימוש בו (אינטגרציה) בפרויקטים אחרים.
בג'אווה יש שני סוגים של הערות: הערת שורה שאותה אנחנו מציינים עם '//' והיא מגדירה שכל מה שבא מהסימון של שני הקווים האלכסוניים ועד סוף השורה יחשב כהערה ולא יקומפל ע"י המהדר;
או הערת בלוק (הערה של כמה שורות) שאותה אנחנו מסמנים עם /* בתחילת בלוק ,הערה, ובסוף הבלוק אנחנו סוגרים עם */ (יש עוד כמה סוגים כמו הערות javadocs אך אלו שני סוגי הערות המרכזיים).

בפייתון סימון הערות הוא בצורה שונה, כשרוצים לעשות הערות של שורה אחת משתמשים בתו '#' .

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

במדריך "style guide for python code" כתוב שגודל השורה המומלץ הוא כ 72 תווים.
במידה ואנחנו חורגים מהגודל מומלץ לפצל את הערות לכמה שורות של הערות או לבלוק הערות , מה שמוביל אותנו לסוג השני של הערות של פייתון-הערות בלוק:
בשביל הערות בלוק כותבים בהתחלה " " " (שלושה מרכאות) ומסיימים את בלוק גם בשלושה מרכאות:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

טיפ: נהוג להתחיל קובץ פייתון בכמה שורות של הערות, שורות אלה מציינות מידע אודות הפרויקט, מטרת הקובץ, מי המתכנת ורישיון התוכנה.
בדר"כ הערות כאלה מסוגננות בצורה הבאה:

```
#-----------------------------------------------------------------
#demonstrates how to write ms excel files using python-openpyxl
#
#(C) 2015 Frank Hofmann, Berlin, Germany
#Released under GNU Public License (GPL)
#email email@email.com
#-----------------------------------------------------------------
```

יש עוד סוג של הערות הפייתון והוא docstring .
docstring היא הערה שמוסיפים מתחת לכותרת של פונקציה, מחלקה, שיטה של מחלקה או module,
והיא מסייעת לצרף הערות לחלקים בפרויקט כך שגם מחוץ לפרויקט יהיה ניתן לקרוא עליהם:

```
def add(value1, value2):

    """Calculate the sum of value1 and value2."""

    return value1 + value2
```

add היא פונקציה שמקבלת שני ערכים ומחזירה את הסכום שלהם.
הוספנו לה docstring ועכשיו נוכל לבדוק מה הפונקציה עושה מבלי להשתמש בה:

```
>> print (add.__doc__)

   Calculate the sum of value1 and value2.
```

יבוא ספריות חיצוניות-

אחד ההבדלים הבולטים בין ג'אווה לפייתון הוא בהתייחסות ל-modules שונים.
בג'אווה כל מסמך מוגדר כמחלקה חדשה, ורק המסמך הראשי מכיל קובץ main.
זה שכל מסמך נחשב לאובייקט עוזר למנוע בעיות של ambiguous בקוד, כלומר שימוש בשתי מתודות או יותר עם אותו השם, אבל ממחלקות שונות.
פייתון יותר דומה ל-c/c++ בדבר הזה, וניתן להגדיר פונקציות שלא נחשבות למתודות של מחלקות ספציפיות, ואז כאשר מייבאים את ה-module של הפונקציה ניתן לקרוא לפונקציה בשמה אבל צריך להגדיר מאיפה הגיע הפונקציה,
לצורך הדוגמא נניח שהמודול cow מכיל רק פונקציה אחת והיא moo שמדפיסה למסך את המילה moo :

```
#cow.py
>>> def moo():
...   print('moo')
```

עכשיו כדי להשתמש בפונקציה נצטרך לייבא את ההספרייה ולהגדיר באיזו פונקציה של הספרייה נרצה להשתמש:

```
>>> import cow
>>> cow.moo()
'moo'
```

לפעמים יהיו modules גדולים שנרצה מהם רק פונקציה ספציפית, למשל מהספרייה math נרצה להשתמש רק במשתנה pi, ולא בכל הפונקציות בספרייה.
במקרה כזה נוכל להגדיר אילו אובייקטים ספציפיים לקחת מהספרייה עם המילים השמורות from ו- import :

```
>>> from math import pi
```

במקרה כזה במשתנה pi יהפך להיות משתנה גלובלי בפרוייקט הנוכחי שלנו, ולא נכיר אותו כ-math.pi כפי שהיה קורה במקרה בו היינו מייבאים את כל הספרייה:

```
>>> from math import pi
>>> pi
3.141592653589793
>>> math.pi
NameError: name 'math' is not defined
```

נוכל גם להגדיר את השימוש בספריות או אובקיטים של הספרייה במרחב שם חדש ע"י שימוש במילה as:

```
>>> import math as m
>>> m.pi
3.141592653589793
>>> from math import pi as PI
>>> PI
3.141592653589793
```

לפעמים נרצה להגדיר בעצמנו ספריות בתוך הפרויקט, למשל אם יש לנו פרויקט שמורכב מפונקציות על מסדי נתונים, פונקציות חישוביות ופונקציות שמטפלות ב-UI, נרצה לפרק אותן לכמה תיקיות נפרדות כדי שיהיה קל יותר לשלוט בקוד. ברמת העיקרון ניתן להגדיר מרחבי שם בתוכנית ע"י הכנסה של כל הסקריפטים שקשורים באותו נושא לאותה התיקייה, ואז כשנרצה לייבא מודול ספציפי נוכל לייבא אותו מהתיקייה בצורה דומה לייבוא פונקציות מתוך מודולים גדולים. לדוגמא יש לנו תיקייה שנקראת UI ויש לה מודול שממונה על הכפתורים buttons.py , נוכל לייבא אותו מתוך התיקייה כך:

```
>>> form UI import buttons
```

ואם יש בו מחלקה או פונקציה ספציפית נוכל לייבא אותה כך:

```
>>> from UI.buttons import Fbutton
```

יכול להיות שחלקכם ראיתם תוכניות פייתון שבהן היו תיקיות עם קובץ שנקרא __init__.py . ברמת העיקרון בעבר (לפני פייתון 2.3) כדי להגדיר תתי ספריות או תיקיות בתוכנית פייתון, היה הכרחי להשתמש בקובץ כדי שהתיקייה תהיה חלק מהתוכנית, אבל עדיין משתמשים בקובץ והסיבה העיקרית כדי להפריד בין תיקיות שהן מרחב שם גרידא ותיקיות שהן ממש חלק מהפרויקט.
מה הכוונה? כשקוד פייתון מייבא ספרייה כלשהי הוא דבר ראשון מחפש את כל הספריות שנמצאות באזור שבו השפה מותקנת, כלומר הוא מחפש את אותו הקוד ב-PATH של השפה. אם הוא לא מוצא שם הוא מחפש בסביבה הקרובה אליו, התיקייה בה נמצא הסקריפט.
עכשיו, לצורך הדוגמא, נניח יש לנו שתי תיקיות בשם my_package. הראשונה נמצאת בתיקייה בה נמצא הקוד אותו אנחנו כותבים ומכילה את הסקריפט my_module1.py, והשנייה נמצאת בתיקיה בה מותקנת השפה במחשב ומכילה את הקובץ my_module2.py. עכשיו נניח אנחנו רוצים לייבא את הסקריפטים my_module1.py ו-my_module2.py . אם לא היינו מגדירים קובץ __init__.py לא היינו מקבלים שגיאה במקרה זה:

```
>>> from my_package import my_module1
>>> from my_package import my_module2
```

מה שיקרה בפועל זה שמתי שנייבא את המודול הוא יחפש אותו בתוך תיקייה שקוראים לה my_pakage , וכפי שהסברנו קודם אם הוא מצא תיקייה בשם הזה באזור שבו מותקנת השפה ויש לה את המודול שאנחנו מחפשים הוא ישתמש בה, אבל אם הוא לא מצא אחת כזאת הוא יחפש באזור שבו נכתב הקוד(באותה תיקייה שבה נכתב הקוד). במקרה שלנו משום שהגדרנו תיקייה בשם הזה בתיקייה של הקוד, ויש לה סקריפט בשם שאותו אנחנו מחפשים הוא ישתמש בתיקייה הזאת (אם לא קיימת תיקייה כזאת עם המודול הזה ב-path של השפה) , ואז כשהוא יחפש את הסקריפט השני הוא יבצע את אותו התהליך מההתחלה, ומשום שיש סקריפט בשם my_mpdule2.py בתיקייה my_package שבתוך התיקייה שבה מותקנת השפה הוא יוכל להשתמש בסקריפט שלה.
במילים אחרות אין לנו הגבלה לתיקייה ספציפית אלא לשם של תיקייה ספציפי.
לעומת זאת אם נשים מודל __init__.py בתיקייה my_package שנמצאת היכן שנכתב הקוד לא נוכל לייבא את my_module2.py , או הפוך- נכתוב בתיקייה שב-path של השפה את הקובץ __init__.py אז לא נוכל לייבא את my_module1.py , כי הקובץ מגדיר לתוכנה לקחת מהתיקייה הספציפית my_package עם הקובץ __init__.py ,כלומר הוא מגביל את השימוש בתיקייה ספציפית ולא במרחב שם ספציפי.
הקובץ לא צריך להכיל שם קוד למעשה בשביל האפקט הזה, אבל בדר"כ נוהגים להשתמש בו בתור מסוף לאובייקטים שמשתמשים בהם בכל הספרייה, למשל אם יש לנו תיקייה עבור database מקובל להגדיר את האובייקט שלו בתוך קובץ ה-__init__.py .
לכן אם תרצו להגדיר תיקייה בפרויקט פייתון שלכם, מומלץ להגדיר אותה עם קובץ __init__.py.

מוסכמות-

אלו רק מוסכמות, אין חובה לציית להם, אבל אם אתם מתכננים לעבוד עם אנשים אחרים שאמורים לקרוא את הקוד שלכם מומלץ להסכים על מוסכמות בניכם.
במדריך הרשמי של פייתון מצוינות כמה מוסכמות בנוגע לכתיבת קוד נכון בפייתון:
* לא להשתמש באות l (האות א"ל קטנה)או I (האות אי"י גדולה)או באות O (האות או"ו גדולה או קטנה) כייצוג שם של משתנה, היות ובחלק מהפונטים קשה להבדיל בין האותיות האלה למספרים אחד או אפס.

* כל המזהים חייבים להיות כתובים בasciii ואמורים להיות כתובים באנגלית בלבד.

* שמות של modules אמורים להיות כתובים באותיות קטנות בלבד, להשתמש בקו תחתון במקרה שרוצים שהשם שלו יהיה בנוי מכמה מילים למשל my_first_project.py

* שמות של מחלקות אמורות להיות במבנה של CapWord כלומר להתחיל באות גדולה, וכל פעם שרוצים להוסיף מילה חדשה לשם המחלקה נוסיף אותה עם אות גדולה למשל: MyClass .

* חריגות הן מחלקות בפייתון (נראה בהמשך) ולכן שמן יהיה כשם של מחלקה. במידה והחריגה היא שגיאה נהוג להוסיף את המילה Error לסיפא של שמה, למשל: ZeroDevisionError .

* שמות של פונקציות צרכים להיות באותיות קטנות עם הפרדה של '_' בין מילים, למשל: " def print_hello():"

* שמות של משתנים (גלובליים או לוקלים) צריכים להיות כמו שמות של פונקציות, יוצא דופן הוא משתנה גלובלי של module שכדאי לסמן שהוא לא לשימוש הכלל אלא משתנה פרטי של המודול, במקרה כזה נסמן את המשתנה ב-"__<name>__", למשל כדי להגדיר גירסה לפרוייקט נשמור משתנה בשם __version__ בקובץ מתאים.

* משתנים קבועים מציינים באותיות גדולות עם ' _ ' שמפריד בין מילים: MAX_VALUE, TOTAL וכו'.

* משתנים פרטיים של מחלקות (אמורים להיות פרטיים) מציינים עם '_' אחד לפני שם המשתנה, ולא עם שני קווים תחתונים, שני קווים תחתונים משמשים לדבר אחר, נראה את זה בשיעור בנושא מונחה עצמים בפייתון.
דוגמא: _height_ , size וכו'.