

## מג'אוה לפייתון חלק א'

### מבנה של סקריפט פייתון-

בדומה לתוכניות של ג'אוה, לכל פרויקט יש כמה מסמכים או סקריפטים, ולכל מסמך יש את הסימנים שלו- בג'אוה: '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 `module` או `path` שלו (במידה והטרמינל לא נמצא באותה תיקייה של ה-module). **הערה:** כדי לצאת ממצב האינטראקטיבי צריך להפעיל את הפונקציה `exit()` שיוצאת מתהליך וחוזרת לטרמינל, ניתן גם ללחוץ `ctrl+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 `python` או `python3` ואז שם הסקריפט בסימנים '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 כ `modules`. בזכות המשתנה `__name__` נוכל לקבוע אם נרצה להריץ את הקובץ כסקריפט או לייבא אותו כ `modules`. כשאנחנו מריצים את הקובץ כסקריפט המשתנה `__name__` יהיה שווה למחרת `"__main__"` אבל כשמריצים את הקוד כ `modules` 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 module תקבע באילו משתנים היא רוצה להשתמש מתוך הקובץ.

### 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. הסוגרים 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, מה 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 כל הקוד בכללותו, עוזרת במציאת שגיאות בקוד 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 שתי סוגי הערות המרכזיות).



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 שפייתון-הערות בלוק:  
 בשביל הערות של בלוק כותבים בתחילת הבלוק " " " (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.  
 בד"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.  
 הוספנה לה docstring ועכשיו נוכל לבדוק מה הפונקציה עושה מבלי להשתמש בה:

```
>>> print add.__doc__
Calculate the sum of value1 and value2.
```

