LOGGING

כשרק למדנו לתכנת, למדנו שיטה פשוטה לנפות שגיאות: הדפסות ביניים. בשיטה הזאת, אנחנו שמים כל מיני הדפסות בקוד בעזרת print או פקודות דומות עד שאנחנו מבינים איפה השגיאה, ואז מוחקים את ההודעות או שמים בהערה. אבל מה קורה אם לאחר זמן שוב מגלים שגיאה? אז צריך שוב להיכנס לקוד ולהכניס הדפסות ביניים או למחוק את ההערה – מאד מבולגן ולא נוח.

בפייתון יש שיטה טובה יותר לנהל את כל העניין של הדפסות מתוך פונקציות: הספריה **logging**. הרעיון הוא להפריד את האחריות בין ההחלטה **מה** לכתוב, לבין ההחלטה **לאן** לכתוב (אם בכלל):

- ההחלטה **מה** לכתוב היא החלטה של מי שכותב את הפונקציה. הוא מכיר את האלגוריתם, ויודע איפה ואיזה הודעות כדאי לשים כדי שמהלך הביצוע של האלגוריתם יהיה ברור.
- ההחלטה **לאן** לכתוב (למסך, לקובץ, או בכלל לא לכתוב) היא החלטה של מי שמשתמש בפונקציה – בד"כ התוכנית הראשית. הוא יודע, לפי השיקולים שלו, האם ועד כמה הוא רוצה לראות הודעות מתוך כל פונקציה.

איתחול

כדי להתחיל לעבוד עם לוגינג, נייבא את הספריה המתאימה (ספריה שמגיעה כחלק מההתקנה של פייתון):

```
import logging
```

עכשיו ניצור עצם בשם "לוגר" (logger) שלתוכו נכתוב את ההודעות. המערכת מזהה לוגרים לפי השם שלהם. לדוגמה, אנחנו יכולים לקבל לוגר בשם "mylogger" באופן הבא:

```
logger = logging.getLogger("mylogger")
```

מקובל ליצור לוגר נפרד עבור כל קובץ או פונקציה, כדי להפריד בין ההודעות השונות. לשם כך אפשר לכתוב בראש הקובץ שלנו:

```
logger = logging.getLogger(__name__)
```

וכך נוצר לנו לוגר ששמו כשם הקובץ הנוכחי.

כתיבת הודעות לוג מתוך הפונקציה

את כל ההודעות מתוך הפונקציות שלנו, נכתוב לתוך הלוגר שיצרנו. לשם דוגמה, נתבונן בפונקציה שמטרתה לחשב את הנוסחה לשורש של משוואה ריבועית:

```
def quadratic_formula(a:float, b:float ,c:float) -> float:
    """ Returns the real solutions to the equation ax^2 + bx + c = 0  """
    logger.info('quadratic_formula(%g,%g,%g)', a, b, c)
    discri = b**2 - 4*a*c
    logger.debug('Compute the discriminant: %g', discri)
    if discri<0:
        logger.warning('Discriminant is negative!')
        return (0,0)
    root_a = (-b + math.sqrt(discri))/(2*a)
    logger.debug('Compute the positive root: %g', root_a)
    root_b = (-b - math.sqrt(discri))/(2*a)
    logger.debug('Compute the negative root: %g', root_b)
    return root_a , root_b
```

ישנן כמה פונקציות של logger המאפשרות לכתוב הודעות. בדוגמה למעלה רואים שלוש פונקציות: warning, info, debug. יש עוד שתי פונקציות שלא רואים שם: error, critical. הפונקציות למעשה מגדירות חמש רמות שונות של הודעות. ככל שהרמה של הודעה גבוהה יותר, כך יש יותר סיכוי שנרצה לראות אותה.

- הרמה הנמוכה ביותר היא debug – הודעה שנועדה לניפוי שגיאות, שברוב המקרים לא נרצה לראות.
- רמה גבוהה יותר היא info – הודעה הנותנת מידע על אופן פעולת הפונקציה; נרצה לראות אותה לפעמים כדי להבין איך הפונקציה עובדת.
- רמה גבוהה יותר היא warning – אזהרה על מצב חריג בפונקציה – שכנראה נרצה לראות כדי לאבחן תקלה אפשרית.
- רמה גבוהה יותר היא error ומעליה critical – הודעות שגיאה המחייבות התייחסות שלנו.

ניתן גם להגדיר רמות מתואמות אישית, אך רמות אלו אמורות להיות מספיקות.

הצגת הודעות לוג מתוך התוכנית הראשית

אם נכתוב עכשיו תוכנית ראשית, ונריץ מתוכה את הפונקציה שלנו, כנראה לא נראה שום דבר. זאת כיוון שלא אמרנו ללוגר לאן לכתוב. אז בואו נגיד לו:

```
if __name__ == "__main__":
    logger = logging.getLogger(__name__)    # נכתוב כאן את שם הלוגר שאנחנו רוצים לשנות
    logger.addHandler(logging.StreamHandler())
```

בשורה השניה, אנחנו מוסיפים ללוגר רכיב שנקרא Handler, שתפקידו לטפל בהודעות. במקרה שלנו, אנחנו מוסיפים StreamHandler, שפשוט שולח את ההודעות למסך. יכולנו גם להוסיף "מטפל" ששולח את ההודעות לקובץ, למשל:

```
logger.addHandler(logging.FileHandler("my_logger.log", mode="w"))
```

עכשיו נריץ את הפונקציה שלנו:

```
print(quadratic_formula(1, 0, -4))
print(quadratic_formula(1, 0, 4))
```

כאן אמורה להיכתב, בשלב זה, רק הודעה אחת – האזהרה על כך שהדיסקרימיננטה שלילית. הסיבה היא, שכברירת מחדל, הלוגר מציג רק הודעות מרמת "אזהרה" ומעלה". כדי לשנות את זה, פשוט נשנה את רמת הלוגר מהתוכנית הראשית:

```
logger.setLevel(logging.INFO)
```

השורה הזאת תגרום ללוגר להדפיס כל הודעה מרמת "מידע" ומעלה (הכל חוץ מדיבאג). אם רוצים לכתוב את כל ההודעות כולל דיבאג, נכתוב:

```
logger.setLevel(logging.DEBUG)
```

עכשיו, כשנריץ שוב את הפונקציות שלנו, נקבל את כל ההודעות ברמות שבחרנו:

```
print(quadratic_formula(1, 0, -4))
print(quadratic_formula(1, 0, 4))
```

שימו לב להפרדה: הפונקציה שכתבנו לא יודעת שום דבר על מסכים או על קבצים; היא רק מנפיקה הודעות. התוכנית הראשית היא שמחליטה, אם ההודעות ילכו למסך או לקובץ, וכן איזה רמה של הודעות להראות.

דבר נוסף שאפשר לשלוט בו מהתוכנית הראשית הוא הפורמט של ההודעות. לשם כך צריך לשנות את הפורמט של ה"מטפל" (handler). כך אפשר, למשל, לקבוע פורמט הודעה שונה למסך ולקובץ. הנה דוגמה:

```
logfile = logging.FileHandler("my_logger.log", mode="w")  # w = replace; r+ = append
```

```
logfile.setFormatter(logging.Formatter('%(asctime)s: %(levelname)s: %(name)s: Line
%(lineno)d: %(message)s'))
console = logging.StreamHandler()  # writes to stderr (= cerr)
console.setFormatter(logging.Formatter('Line %(lineno)d: %(message)s'))
logger.handlers = [console, logfile]
```

כאן יצרנו שני רכיבי טיפול, נתננו לכל אחד מהם פורמט אחר, וקבענו שהלוגר ישלח את ההודעות לשני המטפלים. כך אותן הודעות יגיעו למסך בפורמט אחד, ולקובץ בפורמט אחר.

איך להחליט איזה הודעות לכתוב?

תחשבו שאתם צריכים להסביר למישהו איך עובד האלגוריתם שכתבתם. כנראה תרצו להסביר את זה על-ידי דוגמה: תראו לו דוגמה לקלט, ותסבירו לו את שלבי הריצה של האלגוריתם.

הודעות הלוג שאתם כותבים מתוך הפונקציה למעשה מבצעות את אותו תפקיד: הן מסבירות לקורא את שלבי הריצה של האלגוריתם.

למעשה, בעזרת ההודעות שאתם כותבים מתוך הפונקציה, אתם יכולים ליצור דוגמאות-הרצה מוסברות בקלות; זה לא רק כלי לניפוי שגיאות, אלא גם כלי לימודי, ללימוד אלגוריתמים חדשים. הודעות אלו יש לכתוב ברמה **info**.

מעבר לזה, תוך כדי פיתוח הפונקציה, בוודאי תגלו מקומות שבהם תצטרכו יותר מידע על אופן פעולת הפונקציה במקרים מסויימים; במקרה זה הוסיפו הודעות ברמה **debug** . רוב המשתמשים לא יצטרכו לראות אותן, אבל תהיה אפשרות להפעיל אותן במקרה הצורך כדי להבין יותר טוב מצבי שגיאה.

אם האלגוריתם נתקל בבעיה אפשרית שעלולה להעיד על שגיאה בקלט, יש להוציא הודעה ברמה **warning**.

הודעות ברמה גבוהה יותר (error, critical) בדרך-כלל לא כותבים, כי במקרה של שגיאה עדיף לזרוק חריגה (raise).