פרק 3 עבודה עם משתנים וטיפוסי משתנים

בפרק 3 של תוכנית הלימודים להנדסאים בשפת פייתון רשום:

פרק 3 : עבודה עם משתנים וטיפוסי נתונים

יעדים

הבנה והעמקת הידע של עבודה עם משתנים.

תכנים

1. התלמיד יכיר את המושג משתנה (Variable)
2. חוקיות שמות משתנים
3. מספרים מרוכבים (Complex Numbers)
4. בסיסי משתנים Binary, Octal and Hexadecimal
5. משתנים מטיפוס שלם
6. משתנים מטיפוס מחרוזתי
7. משתנה מטיפוס ממשי
8. משתנה בוליאני (Boolean)
9. ההבדל בין שמות משתנים לפקודות שמורות בזיכרון

סיכום שעות ההוראה : 8 שעות עיוניות.

3.1 מטרה : הבנה והעמקת הידע בעבודה עם קבועים ומשתנים

כשנכתוב תוכניות בפייתון נרצה להדפיס לבצע קלט ולבצע עיבוד של נתונים. משיגים זאת באמצעות קבועים ומשתנים, ונלמד כמה מושגים בסיסיים נוספים בפסקה הזו.

3.2 כיצד נכתוב הערות - Comments ?

הערות הן כל טקסט מימין לסמל # והן שימושיות בעיקר כהערות עבורנו ולקוראי התוכנית שלנו.

לדוגמה:

```
print('hello world')   # Note that print is a function
```

או

```
# Note that print is a function
print('hello world')
```

מומלץ להשתמש בכמה שיותר הערות שימושיות בתוכנית שלנו כדי:

- להסביר הנחות
- להסביר החלטות חשובות

- הסבר של פרטים חשובים
- הסבר בעיות שמנסים לפתור
- להסביר בעיות שמנסים להתגבר עליהן בתוכנית וכו'.

שורות/הצהרות הקוד אומרות איך לבצע , ההערות מסבירות למה.

ההערות שימושיות עבור הקוראים של התוכנית, כך שיוכלו להבין בקלות מה התוכנית עושה. כדאי לזכור שהאדם שקורא את התוכנית יכול להיות אתה/אני אחרי זמן מסוים וההערות יעזרו לנו להבין מה כתבנו .

3.3 קבועים מילוליים Literal Constants

דוגמה של קבוע מילולי היא מספר כגון 127 או 3.14 , או מחרוזת כגון "My String" .

משתמשים בערך פשוטו כמשמעו. המספר הוא ***קבוע*** כי הערך שלו לא ניתן לשנוי. מילולי הכוונה כיצד הוא נשמע בדיבור.

3.4 מספרים Numbers

המספרים הם בעיקר משני סוגים - **מספרים שלמים - integers ומספרים עם נקודה צפה floats** .

דוגמה למספר שלם היא 175 שהוא רק מספר שלם והוא מטיפוס int .

דוגמאות למספרים עם נקודה צפה float הן 3.1432 או בעזרת E המציין חזקה של 10 כמו 45.3E-4 שאומר $45.3*10^{-4}$ או 0.00453 .

בפייתון אין את הטיפוס long כמו בשפות אחרות . הטיפוס int הוא מספר שלם בכל גודל !

3.5 מחרוזות Strings

מחרוזת היא *רצף תווים*. מחרוזות הן בעצם חבורה של מילים.

נשתמש במחרוזות כמעט בכל תוכנית פייתון שנכתוב.

3.6 מרכאות בודדות (גרשים בודדים) Single Quote

אפשר לציין מחרוזות באמצעות מרכאות בודדות או גרשים בודדים כגון 'Hello Friends' או 'מה שלומך' .

כל התווים הלבנים כמו רווח ו tab , בין המרכאות/ הגרשים נשמרים כפי שהם.

3.7 מרכאות כפולות (גרשיים) - Double Quote

מחרוזות במרכאות כפולות (גרשיים) פועלות בדיוק באותו אופן כמו מחרוזות במרכאות בודדות. דוגמה לכך היא "What's your name?" או "מה שלומך ? " .

3.8 מרכאות משולשות (גרשיים משולשים) - Triple Quote

באפשרותך לציין מחרוזות של מספר שורות באמצעות מרכאות משולשות """ או ''' . ניתן להשתמש במרכאות בודדות ובמרכאות כפולות בחופשיות בתוך המרכאות המשולשות. דוגמה לכך היא :

```
''' This is a multi-line string. This is the first line.
This is the second line.
"What's your name ? " I asked.
He said "Bond, James Bond."
'''
```

3.9 מחרוזות הן בלתי ניתנת לשינוי - Strings Are Immutable

מחרוזות אינן ניתנות לשינוי בפייתון. משמעות הדבר היא שברגע שיוצרים מחרוזת, אין אפשרות לשנות אותה. למרות שזה אולי נראה כמו דבר רע - זה באמת לא. בהמשך נראה מדוע לא מדובר במגבלה בתוכניות השונות .

הערה למתכנתי C/C++ : בפייתון אין טיפוס char . אין צורך אמיתי בזה .

3.10 המתודה format( )

לפעמים אולי נרצה לבנות מחרוזות ממקומות אחרים ואז נשתמש במתודה format( ) .

שמור את השורות הבאות כקובץ str_format.py :

```
age = 20
name = 'Swaroop'
print('{0} was {1} years old when he wrote this book'.format(name, age))
print('Why is {0} playing with that python?'.format(name))
```

ההדפסות שנקבל :

```
$ python str_format.py
Swaroop was 20 years old when he wrote this book
Why is Swaroop playing with that python?
```

איך זה עובד בדוגמה ?

מחרוזת יכולה להשתמש במאפיינים מסוימים ולאחר מכן, ניתן לקרוא למתודה format כדי להחליף מאפיינים אלה בארגומנטים תואמים.

שים לב לשימוש הראשון שבו אנו משתמשים {0} וזה תואם לשם המשתנה name שהוא הארגומנט הראשון במתודה format( ) . באופן דומה, המאפיין השני הוא {1} המתאים לגיל שהוא הארגומנט השני במתודה format( ).

יש לשים לב שפייתון מתחיל לספור מ- 0 מה שאומר שהמיקום הראשון נמצא באינדקס 0, המיקום השני נמצא באינדקס 1 וכן הלאה.

יכולנו להשיג את אותן ההדפסות בעזרת שרשור (חיבור) של מחרוזות :

```
name + ' is ' + str(age) + ' years old'
```

אבל זה הרבה יותר מכוער ויותר מועד לטעויות. שנית, ההמרה למחרוזת תיעשה באופן אוטומטי על-ידי המתודה format( ) במקום ההמרה המפורשת למחרוזות הדרושות במקרה זה. שלישית, בעת שימוש במתודת ה format( ) , אנו יכולים לשנות את ההודעה מבלי להתמודד עם משתנים ולהיפך.

ניתן להשתמש גם במספרים . לדוגמה :

```
age = 20
name = 'Swaroop'
print('{} was {} years old when he wrote this book'.format(name, age))
print('Why is {} playing with that python?'.format(name))
```

ונקבל את אותה ההדפסה שקיבלנו ממקודם.

ניתן גם לתת שמות לפרמטרים :

```
age = 20
name = 'Swaroop'
print('{name} was {age} years old when he wrote this book'.format(name=name, age=age))
print('Why is {name} playing with that python?'.format(name=name))
```

פייתון 3.6 מציגה דרך קצרה יותר לתת שמות לפרמטרים הנקראת "f-strings" :

```
age = 20
name = 'Swaroop'

print(f'{name} was {age} years old when he wrote this book')  # notice the 'f'
before the string
print(f'Why is {name} playing with that python?')  # notice the 'f' before the string
```

ונקבל את אותן ההדפסות כמו בדוגמאות הקודמות.

פייתון מחליפה במתודה כל ערך של ארגומנט במקום המפורט בשורה. יש אפשרות לפירוט נוסף. לדוגמה : format

```
# decimal (.) precision of 3 for float '0.333'
print('{0:.3f}'.format(1.0/3))
# fill with underscores (_) with the text centered
```

```
# (^) to 11 width '___hello___'
print('{0:_^11}'.format('hello'))
# keyword-based 'Swaroop wrote A Byte of Python'
print('{name} wrote {book}'.format(name='Swaroop', book='A Byte of Python'))
```

ההדפסה שנקבל :

```
0.333
___hello___
Swaroop wrote A Byte of Python
```

היות ואנחנו עוסקים בפורמט אז מקבלים שורה חדשה למרות שלא כותבים את התו "new line" (\n בשפות אחרות) כך שקריאה ל print מתחילה בשורה חדשה. כדי להימנע מכתיבה של התו new line יש לציין בשורת ה print לא לעבור לשורה חדשה יש לציין זאת . לדוגמה :

```
print('a', end='')
print('b', end='')
```

ההדפסה שנקבל : ab

אם רוצים רווח בין כל תו נרשום :

```
print('a', end=' ')
print('b', end=' ')
print('c')
```

ההדפסה שנקבל : a b c

3.11 רצפים של Escape - Escape Sequences (שימוש בתו \ back slash)

נניח שרוצים מחרוזת המכילה את התו גרש (') בתוך המחרוזת . כיצד נציין מחרוזת כזו ?

אם נרשום "What's your name?" עם גרשיים (מרכאות כפולות) ולא גרשים (מרכאות בודדות) פייתון תדפיס את המחרוזת ללא בעיה.

אם נרשום 'What's your name? ' פייתון לא תדע מתי מתחילה המחרוזת ומתי מסתיימת .

ניתן לעשות זאת בעזרת המושג **Escape Sequences** שבו רושמים את התו back slash (\ ) לפני הגרש בצורה הבאה: \' . ואז המחרוזת נראית כך : 'What\'s your name? .

כתיבת התו \ מסייעת בהרבה מקרים לכתיבה נכונה של מחרוזת.

דוגמאות :

1. אם נרצה להדפיס את המחרוזת : he said : "hello" נוכל לעשות זאת כך : print('he said : "hello"') .
2. אפשרות נוספת היא בעזרת התו \ : print ("he said : \"hello\"") .
3. אם נרצה להדפיס את התו \ עצמו : print("character backslash : \\")
4. ניתן גם להשתמש בתו \ לציין שהמשפט לא הסתיים ויש לעבור לשורה הבאה.

"This is the first sentence. \

... This is the second sentence."

ההדפסה שנקבל : 'This is the first sentence. This is the second sentence.'

הוא שווה למשפט : print("This is the first sentence. This is the second sentence.")

התו \ עוזר לנו גם בהדפסה מסודרת לפי רצוננו.

\t דומה ללחיצה על tab שמעבירה את הסמן לשדה ההדפסה הבא.

\n מעבירה את הסמן לתחילת השורה הבאה.

דוגמאות:

1. אם נרשום : 'This is the first line\nThis is the second line'

ההדפסה שנקבל :

This is the first line
This is the second line

2. דוגמה עם \t :

```
print("num 1\tnum 2\nnum 3")
```

ההדפסה שנקבל :

num 1 num 2

num 3

3.12 מחרוזת "גולמית" - Raw String אם יש לנו מספר מחרוזות שבהן אין צורך

בעיבוד מיוחד כמו escape sequences או \ אז ניתן לציין שהמחרוזת גולמית (פשוטה ללא עיבוד) אז בתחילת המחרוזת נרשום את התו r או R .

דוגמה : נרשום את הפקודה r"Newlines are indicated by \n"

ונקבל : 'Newlines are indicated by \\n'

3.13 משתנה Variable

כמו בכל שפת תכנות, גם בפייתון יש משתנים.

שימוש בקבועים מילוליים Literal Constants בלבד איננו מעשי. אנחנו צריכים דרך כלשהי לאחסן כל מידע ולבצע עיבוד על המידע הזה. כאן **משתנים variables** נכנסים לתמונה. משתנים הם בדיוק מה שהשם מרמז - הערך שלהם יכול להשתנות, כלומר, ניתן לאחסן כל דבר באמצעות משתנה. משתנים הם חלקים מזיכרון המחשב שבהם נאחסן מידע מסוים. היות והערך יכול להשתנות אז משתנה נמצא בכתובת/כתובות בזיכרון ה RAM של המחשב כי ניתן גם לקרוא וגם לכתוב אליו.
בניגוד לקבועים מילוליים, צריך שיטה כלשהי של גישה למשתנים אלו ולכן נותנים להם שמות.
משתנה הוא מקום בזיכרון המחשב שמכיל ערך. המילה משתנה מרמזת שניתן לשנות את הערך של המשתנה.
יצירת משתנה בפייתון (או הגדרת משתנה) היא פשוטה . מציינים את שם המשתנה ומבצעים השמה של ערך התחלתי אליו.
הפעולה זו נקראת השמה כי שמים ערך במשתנה.

דוגמאות :

```
>>> myName="Arik"
>>> x=25
>>> num1=12.543
```

כדאי לשים לב שלא כמו בשפות אחרות כמו C , בפייתון אין צורך לציין את טיפוס המשתנה !

טיפוס המשתנה נקבע לפי הערך ששמנו בו !

המשתנה myName יהיה מטיפוס מחרוזתי , המשתנה x יהיה מטיפוס שלם - int והמשתנה num1 יהיה מטיפוס ממשי float .
אם רוצים לדעת מאיזה טיפוס משתנה כלשהו נרשום :

```
>>>type (myName)
```

ניתן להגדיר מהו טיפוס המשתנה בצורה הבאה : **(ערך) טיפוס המשתנה = שם המשתנה** .

דוגמאות: num1=int (74) או num2=complex(5j+10)

3.14 מתן שם למזהה - Identifier Naming

משתנים הם דוגמאות של מזהים. מזהים הם שמות שניתנו כדי לזהות משהו. יש כמה כללים שצריך לשים לב כשנותנים שם למזהה (משתנה) :

- התו הראשון של המזהה חייב להיות אות של ה אלפא בית (אות רישית - גדולה - uppercase - של ASCII או אותיות קטנות של ASCII - lowercase - או תו של Unicode - שהוא תקן בינלאומי לייצוג טקסט - או קו תחתון _ underscore).
- שאר השם של המזהה יכול לכלול אותיות (אותיות רישיות ב ASCII או תו קטן ב ASCII או תווי Unicode) או קו תחתון או ספרות (0-9).
- שמות מזהים הם case sensitive . לדוגמה המשתנים myname או Myname או MyName הם מזהים שונים.
  דוגמאות לשמות מזהים *חוקיים* הן name_2_3 , i .
- דוגמאות לשמות מזהים לא חוקיים הם 2good (אסור מספר בתחילת השם) , או my-name (מקף הוא לא חוקי) או a1> (גדול מ הוא לא חוקי) או x@ (שטרודל איננו חוקי) או my name (אסור רווח בשם) וכו'.

לסיכום : השם יכול להיות כל שם שרוצים כל עוד הוא מתחיל באות (אנגלית) או קו תחתון. לאחר התו הראשון השם יכול להכיל מספרים, אותיות וקווים תחתונים. בפיתון נהוג ששמות משתנים יכילו אותיות קטנות בלבד אבל אפשר לכלול גם אותיות גדולות. ההשמה של ערך התחלתי למשתנה מתבצעת ע"י התו '=' .

נהוג לרשום שם משתנה המורכב מ 2 מילים מחובר על ידי קו תחתון. לדוגמה : zero_counter . אם כי אפשר גם zeroCounter .

3.15 טיפוסי משתנים

משתנים יכולים להכיל ערכים מטיפוסים שונים הנקראים טיפוסי נתונים. הטיפוסים הבסיסיים הם מספרים ומחרוזות שהזכרנו מקודם . בהמשך הפרק נראה כיצד ליצור טיפוסים שלנו בעזרת מחלקות – classes .

בסעיפים הבאים נתאר על קצה המזלג את נושא המשתנים וטיפוסי הנתונים השונים.

הרחבה בנושא משתנים ניתן למצוא בקישור : variables.pdf (arikporat.com)

הרחבה בנושא טיפוסי נתונים ניתן למצוא בקישור : data types.pdf (arikporat.com)

3.15.א משתנה מטיפוס שלם integer

כאשר רשמנו מקודם את השורה x=25 הגדרנו/יצרנו משתנה בשם x ועשינו השמה של הערך 25 אליו. במקרה הזה נוצר באופן אוטומטי משתנה מטיפוס שלם integer או בשפת פייתון int .

להבדיל משפות אחרות כמו למשל C אין למשתנה מטיפוס int טווח ערכים. כל ערך שנרשום הוא טוב.

דוגמה : נרשום מספר שלם בן 22 ספרות

```
>>> y=9876543212345678909876
>>> print(y)
9876543212345678909876
>>>
```

ניתן לבצע חישובים מתמטיים ולוגיים על המשתנה בדומה לשפות אחרות.

לדוגמה:

```
>>> x=10
>>> print(x+10+x*10)
120
```

3.15.ב משתנה מטיפוס ממשי float

משתנה מטיפוס ממשי float מכיל מספרים שיש להם גם חלק שלם וגם חלק של שבר . כאשר רשמנו מקודם num1=12.543 הגדרנו משתנה מטיפוס ממשי וביצענו אליו השמה של הערך 12.543 . גם כאן אין טווח ערכים.

גם כאן ניתן לבצע על המשתנה חישובים מתמטיים ולוגיים כמו בשפות אחרות.

3.15.ג משתנה מטיפוס מחרוזתי - string

משתנה מטיפוס מחרוזתי רושמים בין גרשים (מרכאות בודדות) או גרשיים (מרכאות כפולות) .

ההגדרה "str1="hello זהה להגדרה 'str1='Hello . אין הבדל בין סוגי המרכאות .

ניתן לשרשר בין מחרוזות (לרשום מחרוזת אחת בסיום מחרוזת שנייה) ועוד פעולות בעזרת מתודות שנראה בפרקים הבאים.

דוגמה : שרשור מחרוזות

```
>>> myFirstName="Arik"
>>> myLastName='Porat'
>>> print(myFirstName + ' ' + myLastName)
Arik Porat
```

החישוב החשבוני היחיד שניתן לעשות על מחרוזות הוא הכפלה.

דוגמה : הדפסת המחרוזת hello 8 פעמים:

```
>>> str='Hello'
>>> print(str*8)
HelloHelloHelloHelloHelloHelloHelloHello
```

3.15.ד משתנה מטיפוס מספר מרוכב Complex Number

שפת פייתון תומכת במספרים מרוכבים (מספרים קומפלקסים) . ההגדרה x=4j יוצרת משתנה מטיפוס מרוכב.

דוגמה: נגדיר 2 מספרים קומפלקסים ונחבר ביניהם .

```
>>> x=4j+2
>>> y=-2j+7
>>> print(x+y)
(9+2j)
```

3.15.ה משתנה מטיפוס בוליאני

ניתן להגדיר משתנה מטיפוס בוליאני והוא יכול להכיל ערכים של True או False (האות הראשונה היא אות גדולה והשאר קטנות !) .

דוגמה :

```
>>> bool_variable=True
>>> print(bool_variable)
True
```

3.15.ו ייצוג מספרים בבסיסים שונים

בפייתון ניתן לייצג מספרים ב 4 סוגי חשבון :

חשבון בינארי . נשתמש ב 0b או 0B כדי לציין שהערך בבינארי . דוגמה : x=0b10101010 או x=0B10101010.

חשבון אוקטלי - חשבון בו 8 ספרות בסיסיות מ 0 עד 7 . נשתמש בצמד התווים 0O או 0o . דוגמה : y=0O17 או y=0o17.

חשבון עשרוני – ספרות 0 עד 9 . אם לא מציינים מהו סוג החשבון אז מדובר בחשבון עשרוני.

חשבון הקסה דצימלי - חשבון שבו 16 ספרות ואותיות בסיסיות. הספרות הן מ 0 עד 9 והאותיות הן מ A עד F. נשתמש בצמד התווים 0x או 0X . דוגמה : num=0x17 או num=0X17 .

הרחבה בנושא ייצוג מספרים בבסיסים שונים נמצא בפרק הקודם – פרק 2 בספר בסעיף 2.4 בקישור:

chapter 2 background and basics.pdf (arikporat.com)

3.16 אובייקט Object

פייתון מתייחסת לכל דבר שיש בפרויקט כאובייקט . באופן כללי , במקום לומר "משהו" אנחנו נגיד האובייקט.

פייתון היא Object Oriented - מונחית עצמים - במובן הזה שכל דבר הוא אובייקט, כולל מספרים, מחרוזות ופונקציות.

3.17 כיצד לכתוב תוכנית בפייתון ?

נרשום מספר כללים שישמשו אותנו מעכשיו והלאה כסטנדרט לשמירה והרצה של תוכנית בפייתון .

3.17.א עבור PyCharm

1. נפתח את PyCharm .
2. ניצור קובץ חדש – new file – עם השם הרצוי לך.
3. נרשום את קוד התוכנית .
4. לחיצה על המפסק הימני בעכבר – right click – והרצת הקובץ הנוכחי.

הערה : כאשר יש להכניס ארגומנטים בשורת הפקודה נקליק על Run -> Edit Configuration ונקיש את הארגומנטים במקטע ה Script parameters ונלחץ על אישור – OK .

3.17.ב עבור עורכים אחרים

1. נפתח את העורך שרוצים לעבוד איתו.
2. נקליד את קוד התוכנית.
3. נשמור אותו כקובץ עם שם הקובץ שהוזכר.
4. נפעיל את המפרש interpreter עם פיתון הפקודה program.py להרצת התוכנית.

3.17.ג דוגמה לשימוש בקבועים ומשתנים

נקליד את התוכנית הבאה ונריץ אותה:

```
# Filename : var.py
i = 5
```

```
print(i)
i = i + 1
Basics
30
print(i)
s = '''This is a multi-line string.
This is the second line.'''
 print(s)
```

ההדפסה שנקבל :

```
5
6
This is a multi-line string.
This is the second line.
```

איך התוכנית עובדת ?

ראשית, אנו מקצים ערך קבוע מילולי 5 למשתנה i באמצעות האופרטור (=) .

שורה זו נקראת הצהרה מכיוון שהיא קובעת שיש לעשות משהו ובמקרה זה, אנו מחברים את שם המשתנה i לערך 5.

לאחר מכן אנו מדפיסים את הערך של i באמצעות ההצהרה ( )print שמדפיסה את הערך שיש במשתנה i למסך.

לאחר מכן נוסיף 1 לערך המאוחסן ב - i ונאחסן אותו בחזרה ב i .

לאחר מכן אנו מדפיסים אותו ומקבלים את הערך 6 .

באופן דומה, אנו מקצים את המחרוזת המילולית למשתנה s ולאחר מכן מדפיסים אותה.

הערה : כדאי לשים לב שהעברנו לכל משתנה ערך מבלי לציין מאיזה טיפוס המשתנה כמו בשפת תכנות אחרת. למשל בשפת C או C++ היינו צריכים לרשום int i =5; . בפייתון המשתנה מקבל את הטיפוס לפי הערך שהעברנו לו. כאן i הוא מטיפוס שלם ו s הוא מטיפוס מחרוזתי.

3.18 שורה לוגית ופיזית Logical And Physical Line

שורה פיזית היא מה שרואים בעת כתיבת התוכנית. שורה לוגית היא מה פייתון רואה כהצהרה אחת. פייתון מניח במרומז שכל שורה פיזית מתאימה לשורה לוגית.

דוגמה לשורה לוגית היא משפט כמו print('hello world') - אם זו היה בשורה בפני עצמה (כפי שאתה רואה את זה בעורך) אז זה גם מתאים לשורה פיזית.

פייתון מעודדת שימוש בהצהרה אחת לכל שורה וזה הופך את הקוד לקריא יותר.

אם ברצוננו לציין יותר משורה לוגית אחת בשורה פיזית אחת, יש לציין זאת במפורש באמצעות נקודה-פסיק (; ;) המציין את סופו של שורה/הצהרה לוגיים. לדוגמה : שתי השורות

```
i = 5
print(i)
```

מבצעת אותה פעולה כמו השורה : i = 5; print(i); או כמו השורה i = 5; print(i)

עם זאת, *מומלץ* לדבוק בכתיבת קו לוגי יחיד לכל היותר בכל קו פיזי בודד. הרעיון הוא שאסור להשתמש בנקודה-פסיק. יש מצב אחד שבו מושג זה הוא באמת שימושי. אם יש שורה ארוכה של קוד, ניתן לשבור אותה למספר קווים פיזיים באמצעות הקו הנטוי הנגדי. זה נקרא צירוף שורה מפורש - explicit line joining . לדוגמה :

```
s = 'This is a string. \                    הקו הנטוי "שובר" את השורה
This continues the string.'
print(s)
```

ההדפסה שנקבל : This is a string. This continues the string.

בצורה דומה :

```
i = \
5
```

זהה ל i = 5

לפעמים, יש הנחה מרומזת שבה אתה לא צריך להשתמש בקו נטוי הפוך (\ ) backslash. זה המקרה שבו לשורה הלוגית יש התחלה עם סוגריים , התחלה עם סוגריים מרובעים או סוגריים מסולסלים מתחילים אך ללא סוגריים מתאימים סוגרים . פעולה זו נקראת צירוף שורה מרומז implicit line joining .
נראה זאת בפעולה בעת כתיבת תוכניות באמצעות list - רשימה - בפרקים מאוחרים יותר.

3.19 הזחה Indentation

מקש הרווח space חשוב בפייתון. למעשה, הרווח בתחילת השורה חשוב. פעולה זו נקראת הזחה **Indentation.**
הזחה היא הזזה של הצהרה מהשוליים או בשפת פייתון הזזה יחסית להצהרה שמעליה.
רווח מוביל (רווחים ו tab) בתחילת השורה הלוגית משמש לקבוע את רמת ההזחה של השורה הלוגית ומשמש גם כדי לקבוע את קבוצת/בלוק ההצהרות (דומה לבלוק של משפטים בין סוגריים מסולסלים בשפת C או C++).
משמעות הדבר היא כי הצהרות אשר הולכות יחד חייבות להיות באותה הזחה. כל קבוצה כזו של הצהרות נקראת בלוק. אנחנו ניראה דוגמאות לאופן שבו בלוקים חשובים בפרקים מאוחרים יותר.
דבר אחד שחייבים לזכור הוא שהזחות שגויות יכולות לגרום לשגיאות.
לדוגמה: נרשום תוכנית עם הזחה לא נכונה. השורה השנייה צריכה להיות בדיוק מתחת לראשונה ולכן נקבל שגיאה.

```
i = 5
  print('Value is', i)
print('I repeat, the value is', i)
```

נריץ את התוכנית ב PyCharm ונקבל את השגיאה הבאה (ההדגשה - Bold - היא שלנו ולא של התוכנה):

```
File "<input>", line 2
  print('Value is', i)
```

IndentationError: unexpected indent

כדאי לשים לב שיש רווח בתחילת השורה השנייה. הודעת השגיאה אומרת כי בשורה השנייה יש שגיאה של הזחה לא צפויה .

דוגמה נוספת :

```
a = 133
b = 200
if b > a:
  print("b is greater than a")
  print ("200 >133")
```

במקרה זה שמנו בכוונה הזחה אחרי משפט התנאי if , לשתי השורות הבאות. זה אומר ששתי השורות הבאות הן קבוצת הוראות אחת (כמו הוראות שנמצאות בין סוגריים מסולסלים בשפות אחרות).

אם לא היינו עושים הזחה אלא רושמים את אותה תוכנית ללא הזחה :

```
a = 133
b = 200
if b > a:
print("b is greater than a")
print ("200 >133")
```

היינו מקבלים את הודעת השגיאה הבאה :

```
File "<input>", line 4
    print("b is greater than a")
    ^
IndentationError: expected an indented block
```

שאומרת שיש לבצע הזחה בשורה 4 .

3.19.א כיצד לבצע הזחה ?

הזחה מבצעים עם מקש הרווח Space Bar . רווח של 4 תווים היא המלצה רשמית של שפת פייתון ויש עורכים שהמשתמש יכול לקבוע את כמות הרווחים. גם הזחה של תו אחד תעבוד כראוי. כדאי להשתמש במספר עקבי של רווחים אחרת יש חשש שהתוכנית לא תעבוד או תעבוד בצורה לא צפויה.

בפרק זה עברנו על הרבה פרטים ובפרקים הבאים נעסוק באופרטורים, הצהרות של בקרת זרימת (משפטי תנאי) , פונקציות ועוד.