מחרוזות – "string" : immutable ו- ordered (ניתן לגשת לאיבר על ידי ציון אינדקס) כל רצף של תו אחד או יותר בפייתון, מוגדר "מחרוזת" (string) שזה בעצם אובייקט מסוג str. הכוונה ב"תו" היא לכל סימן שאפשר לייצר בעזרת המקלדת – מספרים, אותיות אנגליות, רווחים, סימני פיסוק וכו'. מחרוזת כותבים במרכאות כפולות או גרשים יחידים.	
str.capitalize()	מחזיר עותק של מחרוזת עם התו הראשון באות גדולה וכל השאר קטנות. המתודה capitalize תגדיל את האות הראשונה במחרוזת בלבד.
str.title()	מחזיר עותק של מחרוזת עם התו הראשון של כל מילה באות גדולה וכל השאר קטנות. המתודה title תגדיל את האות הראשונה בכל מילה במחרוזת.
str.count()	מחזיר את מספר הפעמים שערך שצוין בסוגריים מופיע במחרוזת.
str.find()	מחזיר את המיקום במחרוזת של התת מחרוזת שצוינה בסוגריים. מחזיר 1- אם התת מחרוזת לא מופיעה במחרוזת.
str.format(*Argument*) str.format(*key=value*)	מחזירה מחרוזת מעוצבת. המחרוזת str מכילה תווים ומצייני מיקום לארגומנטים. הפרמטרים של הפורמט נכתבים בסוגריים מסולסלים בתוך str והם יכולים להיות פרמטרים של עמדה {index} או פרמטרים של הסוג key=value כמו במילון {key}.
str.index()	כמו find אבל מחזיר ValueError אם התת מחרוזת לא קיימת במחרוזת.
str.join()	איחוד רשימת מחרוזות למחרוזת אחת. פרמטר: המחרוזת שאותה רוצים לאחד. הפונקציה מופעלת ע"י המחרוזת שתפריד בין החלקים.
str.replace(*old,new*)	מחזיר עותק של המחרוזת שבה כל 'old' מהמחרוזת המקורית הומר ל-'new'
str.split(*sep,maxsplit*)	פיצול מחרוזת לרשימה של מחרוזות. *sep*: מחרוזת הפרדה (ברירת מחדל: רווחים) *maxsplit* : אינדקס מקסימלי ברשימה המפוצלת (ברירת מחדל: אין הגבלה)
str.lower()	מחזיר עותק של מחרוזת שבה כל האותיות קטנות
str.upper()	מחזיר עותק של מחרוזת שבה כל האותיות גדולות
str.isalpha()	מחזיר True אם כל התווים במחרוזת הם אותיות.
str.isdecimal() str.isnumeric()	מחזיר True אם כל התווים במחרוזת הם מספרים עשרוניים.
str.isdigit()	מחזיר True אם כל התווים במחרוזת הם ספרות.
str.islower()	מחזיר True אם כל האותיות במחרוזת הם אותיות קטנות.
str.isupper()	מחזיר True אם כל האותיות במחרוזת הם אותיות גדולות.
str.isspace()	מחזיר True אם כל התווים במחרוזת הם " ".
str.isalnum()	מחזיר True אם כל התווים במחרוזת הם ספרות או אותיות.
str.endswith(*str1*)	מחזיר True אם str1 נמצא בסוף המחרוזת str.
str.startswith(*str1*)	מחזיר True אם str1 נמצא בתחילת המחרוזת str.
str.zfill(*width*)	מחזיר עותק של מחרוזת עם אפסים (0) בתחילת המחרוזת המקורית, עד להגעה לאורך שצוין (width). אם הערך של פרמטר ה- width הוא פחות מאורך המחרוזת str, לא מתבצע מילוי.

רשימות- [list]: mutable ו- ordered (ניתן לגשת לאיבר על ידי ציון אינדקס) טיפוס נתונים המאפשר למשתנה להכיל ערכים רבים. רשימה היא סדרה של ערכים ויכולה להכיל אברים מטיפוסים שונים. הסימון של רשימות בפייתון הוא באמצעות סוגריים מרובעים – [ ]. הערכים השונים ברשימה מופרדים ע"י פסיקים, וכך גם נכתוב אותם כשנגדיר רשימה עם ערכים. מטרת השימוש ברשימה היא אחזקה של כמה משתנים בצורה מסודרת. רשימה מקוננת- רשימה בתוך רשימה.	
list.append() list[-i].append()	הוספת איבר לסוף הרשימה. אם מוסיפים רשימה היא תתווסף כרשימה לתוך הרשימה. הוספת איבר לסוף האיבר באינדקס i- של הרשימה.
list.clear()	מחיקת כל האיברים מהרשימה. מנקה את כל האברים ברשימה מחזיר רשימה ריקה.
list.copy()	מחזיר העתק של הרשימה
list.count()	מחזיר את מספר הפעמים שערך שצוין בסוגריים מופיע ברשימה.
list.extend()	הוספת איברים של רשימה לסוף הרשימה הנוכחית.
list.index(*i*)	מחזיר את האינדקס של האיבר הראשון ברשימה שערכו i
list.insert(*index, x*)	הוספת x לרשימה באינדקס רצוי
list.pop(*i*)	מחזיר את האיבר שנמצא באינדקס i ומסיר אותו מהרשימה.
list.remove(x)	מוחק את האיבר הראשון ברשימה שערכו x. אם יש עוד x הוא לא מוחק אותו
list.reverse()	הופך את סדר הרשימה
list.sort()	ממיין את הרשימה מהקטן לגדול

טאפלים – (Tuple): immutable ו- ordered (ניתן לגשת לאיבר על ידי ציון אינדקס) טאפל דומה לרשימה. ההבדל ביניהם הוא שלא ניתן לשנות את האיברים בטאפל לאחר שהוגדרו. טפל נוצר על ידי כתיבת הערכים בתוך סוגריים עגולים (), מופרדים באמצעות פסיקים. הסוגריים הם אופציונליים (לא חובה אבל מומלץ)- כתיבה ללא סוגריים נקראת unpacking. טאפל יכול להכיל אברים מטיפוסים שונים, ניתן לגשת לאברים לפי אינדקס.	
tuple.count()	מחזיר את מספר הפעמים שערך שצוין בסוגריים מופיע בtuple.
tuple.index(*i*)	מחזיר את האינדקס של האיבר הראשון בtuple שערכו i

מילון – {dictionary} : mutable ו- not ordered (לא ניתן לגשת לאיבר על ידי ציון אינדקס) מילון הוא טיפוס המאחסן זוגות של ערכים: {key: value}. המילון הוא מיפוי של ערכי key לערכי value , הקשר בניהם הוא באמצעות נקודותיים. הגדרת מילון: { } הגישה לאיבר בתוך המילון היא באמצעות ציון ערך ה-key בתוך סוגריים מרובעים [ ]. ערכי key חייבים להיות immutable ו- unique. לא ניתן להשתמש ברשימה list עבור ערכי key –ביוון שרשימה היא mutable, אך אפשר להשתמש ברשימה עבור ערכי value. ניתן להשתמש ב tuples עבור ערכי key וגם עבור ערכי value.	
`dict[key] = value`	הוספת ערך למילון. מוסיף את value ל-key שהוכנס, אם קיים ערך אחר בkey הוא יידרס ויוחלף בvalue.
`dict.clear()`	מחיקת כל האיברים המילון.
`dict.copy()`	מחזיר העתק של המילון.
`dict.pop(key)`	מחזיר את האיבר (value) שנמצא בkey ומסיר את שניהם מהרשימה.
`dict.popitem()`	מסיר ומחזיר את הזוג האלמנט האחרון (מפתח, ערך) שהוכנס למילון.
`dict.keys()`	מחזיר אובייקט תצוגה המציג רשימה של כל ה-keys במילון.
`dict.values()`	מחזיר אובייקט תצוגה המציג רשימה של כל הערכים במילון. (טיפוס dict_values)
`dict.items()`	מחזיר אובייקט תצוגה המציג רשימה של זוגות הטופלים של מילון (מפתח, ערך)
`dict.get(key)`	מחזירה את הערך עבור המפתח שצוין אם המפתח נמצא במילון.

קבוצות – {set} : mutable ו- not ordered (לא ניתן לגשת לאיבר על ידי ציון אינדקס) קבוצה מתנהגת כמילון המכיל רק ערכי key. אברי הקבוצה חייבים להיות immutable ו- unique. הגדרת קבוצה: באמצעות הפונקציה set, או הגדרת ערכים ב { }. יצרת קבוצה ריקה a=set(). לשים לב שההגדרה a={}, יוצרת מילון ריק ולא קבוצה ריקה.	
`set.add()`	הוספת איבר לקבוצה.
`set.clear()`	מחיקת כל האיברים מהקבוצה.
`set.copy()`	מחזיר העתק של הקבוצה.
`set.difference(set1)`	מחזירה קבוצה חדשה המכילה רק את האיברים שנמצאים ב-set ולא נמצאים ב-set1.
`set.difference_update(set1)`	מחיקת כל האברים ב-set אשר לא נמצאים ב-set1. *משנה את set.
`set.isdisjoint(set1)`	מחזירה True אם אף אחד מהאברים לא מופיע בשתי הקבוצות.
`set.issubset(set1)`	מחזירה True אם set היא תת קבוצה של set1.
`set.issuperset(set1)`	מחזירה True אם כל האברים ב-set1 קיימים ב-set.
`set.pop()`	מחזיר איבר רנדומלי ומסיר אותו מהקבוצה.
`set.remove(x)`	מוחק את x מהקבוצה
`set.union(set1)`	מחזיר קבוצה המכילה את האיחוד של השתי הקבוצות.
`set.update(iterable)`	מעדכן את הסט, מוסיף פריטים מ- iterables אחרים.

קבצים – (files): טיפול בקבצים הוא חלק חשוב בעבודה עם פייתון. קיימות מספר פונקציות ליצירה, קריאה, עדכון ומחיקת קבצים. לצורך עבודה עם קובץ: נשמור את הקובץ באותה תיקיה של קובץ הפייתון שיקרא אותו. שם הקובץ צריך להיות ללא רווח ואם נרצה להפריד בין מילים נעזר בקו תחתון (מומלץ לתת שם באנגלית). פעולת קבצים בתוך פייתון מתבצעת בסדר הבא: 1.פתיחת קובץ. 2.לקרוא או לכתוב (לבצע פעולה). 3.סגירת הקובץ .	
`f= open("file_name.txt","mode")`	מחזירה אובייקט קובץ שניתן להשתמש בו לקריאה, כתיבה ושינוי של הקובץ. בתוך הסוגריים של הפונקציה נכתוב את שם הקובץ (לא לשכוח txt.) ומצב העבודה (mode) בגרשיים. mode: "r"– מצב קריאה (read), "w"- מצב כתיבה (write) מוחק כל מה שיהיה כתוב לפני, "a"- מצב כתיבה לסוף הקובץ מבלי למחוק את הקיים (append).
`text = f.read()`	מחזירה מחרוזת של תוכן הקובץ. *כדי לקרוא קובץ חייבים לפתוח את הקובץ במצב קריאת r.
`text = f.readline()`	מחזירה מחרוזת של שורה אחת (הראשונה) מהקובץ.
`text = f.readlines()`	מחזירה רשימה של השורות בקובץ. * כל מחזרות תסתיים ב- "\n"
`text = f.read().split()` `text = f.read().split('\n')`	מחזירה רשימה של המילים בקובץ. * ניתן להכניס בתוך הסוגריים של הsplit "\n" ואז נקבל רשימה של השורות ללא סיומת "\n" בסוף המחרוזות.
`text = f.write("string")`	כותב את המחרוזת שצוינה בסוגריים לתוך הקובץ . *כדי לכתוב לקובץ חייבים לפתוח את הקובץ במצב קריאת w או a. הפונקציה מחזירה את מספר התווים שהוכנסו לקובץ.
`f.close()`	סגירת הקובץ .

חיתוך – [Slicing] : חיתוך ערכים מתוך רצפים מסודרים (רשימות, מחרוזות וטאפלים), כל ערך ברצף מסודר מקבל מיקום ייחודי שמיוצג ע"י מספר (האינדקס שלו), המספרים מתחילים תמיד מ-0 ולא מ-1. ניתן לגשת לערכים לפי האינדקס שלהם ע"י שימוש בslicing. הרעיון הוא להשתמש במיקומי הערכים כדי "לחתוך" חלקים מהרצפים האלה ולעבוד רק איתם. שימוש בslicing נעשה ע"י כתיבת שם הרצף ולאחריו סוגריים מרובעים, לדוגמה: [ ]str, [ ]list, [ ]tuple. אינדקסים יכולים להיות שליליים [קריאת הרצף מהתו האחרון לראשון]	
מחזיר העתק של הערך שנמצא במקום ה- i ברצף s. גישה לאינדקס יחיד.	`s[i]`
מחזיר העתק של הערכים בטווח שהוגדר. גישה לטווח של ערכים. אפשרויות נוספות: [:]- מחזיר העתק של כל הרצף. [:start] – מחזיר העתק של הרצף מהאינדקס התחלה שהוגדר ועד והסוף. [stop:] – מחזיר העתק של הרצף מההתחלה ועד האינדקס שהוגדר.	`s[start:stop]` `s[:]`
מחזיר העתק של הערכים בטווח שהוגדר לפי קפיצות. גישה לטווח של ערכים ודילוג ערכים מסוימים. ברירת המחדל היא 1.	`s[start:stop:step]` `s[::]`

אופרטורים אריתמטיים: אופרטורים אריתמטיים משמשים לביצוע פעולות מתמטיות כמו הוספה, חיסור, כפל וכו '. עובדים על מספרים		**אופרטורי השוואה:** משמשים להשוואה בין ערכים. מחזיר True או False לפי התנאי	
חיבור של x ו-y	`x + y`	מחזיר True אם x גדול מ-y. אחרת False.	`x > y`
חיסור של x ו-y	`x - y`	מחזיר True אם x קטן מ-y. אחרת False.	`x < y`
כפל של x ב-y	`x * y`	מחזיר True אם x שווה ל-y. אחרת False.	`x == y`
חילוק של x ב-y (תמיד מחזיר float)	`x / y`	מחזיר True אם x לא שווה ל-y. אחרת False.	`x != y`
חילוק של x ב-y ללא שארית (מחזיר Int)	`x // y`	מחזיר True אם x גדול או שווה ל-y. אחרת False.	`x >= y`
מודולו- מחזיר את השארית של החלוקה של x ב-y	`x % y`	מחזיר True אם x קטן או שווה ל-y. אחרת False.	`x <= y`
פועלת חזקה- x בחזקה y.	`x ** y`	מחזיר True אם x ו-y מקושרים לאותו אובייקט בזיכרון. אחרת False	`x is y`
		מחזיר True אם x ו-y לא מקושרים לאותו אובייקט בזיכרון. אחרת False	`x is not y`
אופרטורי השמה: יצירת משתנים		**אופרטורים בוליאניים:**	
יצירת משתנה x שערכו y.	`x = y`	מחזיר True אם גם x וגם y נכונים. אחרת False	`x and y`
יצירת משתנה x שערכו x + y . (שקול: x = x + y)	`x += y`	מחזיר True אם x או y נכונים. אחרת False	`x or y`
יצירת משתנה x שערכו x - y . (שקול: x = x - y)	`x -= y`	מחזיר True אם x הוא False. אחרת False	`not x`
יצירת משתנה x שערכו x * y . (שקול: x = x * y)	`x *= y`	**אופרטורי השתייכות :** משמשים לבדיקה אם ערך או משתנה נמצאים ברצף (מחרוזת, רשימה, כותרת, ערכה ומילון).	
יצירת משתנה x שערכו x / y . (שקול: x = x / y)	`x /= y`		
יצירת משתנה x שערכו x % y . (שקול: x = x % y)	`x %= y`	True אם פריט של y שווה ל-x, אחרת False	`x in y`
יצירת משתנה x שערכו x // y . (שקול: x = x // y)	`x //= y`	False אם פריט של y שווה ל-x, אחרת True	`x not in y`
יצירת משתנה x שערכו x ** y . (שקול: x = x ** y)	`x **= y`	:Bitwise operators	
		חיתוך	`x | y`
		איחוד	`x & y`

פונקציות מובנות	
מחזיר ערך מוחלט של מספר. הארגומנט יכול להיות int או float.	`abs()`
מחזיר מחרוזת של התו שה-Unicode שלו הוא מספר שלם int. למשל chr(97) = 'a'.	`chr(int)`
מקבל מחרוזת המייצגת תו אחד ומחזיר את ה-Unicode שלו. למשל ord('a') = 97	`ord(str)`
מחזיר מספר שלם חיובי או שלילי. מקבל מספר או מחרוזת .	`int()`
מחזיר מספר עשרוני (עם נק') חיובי או שלילי. מקבל מספר או מחרוזת .	`float()`
מחזירה רשימה. יכולה לקבל מחרוזת, tuple, מילון או קבוצה.	`list()`
יוצרת מילון.	`dict()`
מחזירה מחרוזת.	`str()`
מחזירה קבוצה. יכולה לקבל מחרוזת, tuple, מילון או רשימה.	`Set()`
מחזיר את האורך של s.	`len(s)`
מחזיר את הפריט המינימלי של s.	`min(s)`
מחזיר את הפריט המקסימלי של s.	`max(s)`
מדפיסה למסך את האובייקט שהוכנס אליה.	`print()`
מקבלת שורה מהקלט, ממירה למחרוזת ומחזירה.	`input()`
מחזירה זהות (מספר שלם ייחודי) של אובייקט.	`id()`
מחזירה סכום של מספרים	`sum()`
מחזירה רשימה ממוינת. יכולה לקבל מחרוזת, tuple, מילון או קבוצה.	`sorted()`
יוצרת tuple	`tuple()`
מחזירה את סוג הטיפוס.	`type()`
מחזירה סידרה איטרבילית של מספרים שלמים בטווח מוגדר. בשביל ליצור רשימה של המספרים בטווח הרצוי צריך לכתוב list(range())	`range(start, stop, step)`

הערות: חשובות מאוד בכתיבת תוכנית. משמשות לתיאור המתרחש בתוכנית, כך שאדם המתבונן בקוד יכול להבין אותו בקלות.	
כתיבה # לפני הערה. האינטרפרטר מתעלם מהערות.	`# comment`
כתיבת """ או ''' לפני ואחרי הערה. מאפשר לכתוב הערה בכמה שורות.	`"""example of multi-line comments"""`

הדפסות- print():	
מדפיסה את הערכים של הרשימה למסך (בלי פסיקים וסוגריים מרובעים)	`print(*list)`
end מגדיר לפונקציה איזה תו אנחנו רוצים שיופיע בסוף ההדפסה. ברירת המחדל היא ירידת שורה ('\n')..	`print("string",end= )`
sep מגדיר לפונקציה איזה תו אנחנו רוצים שיופיע בין המחרוזות. ברירת המחדל היא רווח (' ') .	`print("str1","str2",sep= )`

מבני בקרה	
משפטי תנאי - משפטי תנאי מאפשרים להתנות ריצה של חלק מסוים בקוד בקיומו (או אי-קיומו) של תנאי מסוים. לשים לב לנקודתיים ולהזחה	
ביצוע פקודות מסוימות אם (`if`) תנאי מסוים מתקיים. ערך התנאי נקבע על ידי אופרטור תנאי או ערך מוחזר של פונקציה. הפקודות statements יתבצעו רק אם לביטוי הבוליאני יש ערך True.	`if <boolean expression>:` `statements`

ביצוע פקודות מסוימות אם (`if`) תנאי מסוים מתקיים ,ופקודות אחרות (`else`) כאשר התנאי לא מתקיים. הפקודות statements1 יתבצעו רק אם לביטוי הבוליאני יש ערך True, אחרת (כלומר לביטוי יש ערך False) הפקודות statements2 יתבצעו.	`if <boolean expression>:`
	`statements1`
	`else:`
	`statements2`

שילוב של else עם תנאי נוסף.	`elif <boolean expression>:` `statements`
לולאות for- מאפשרת לבצע קוד מסוים, מספר ידוע מראש של פעמים. כל מחזור של ביצוע הלולאה נקרא איטרציה. לולאת for משמשת לאיטרציה לאורך רצף (רשימה, tuple, מחרוזת) או אובייקטים איטרטיבילים אחרים.	
var הוא משתנה "רץ " על סידרה sequence לפי סדר האברים בה. על כל אבר ב sequence הפקודות statements 1 מתבצעות פעם אחת. מספר האיטרציות בלולאה הוא כמספר האברים בסדרה.	`for var in sequence:` `statements1`
לולאת while- מאפשרת לבצע קוד מסוים שוב ושוב. לולאת while עובדת לפי העיקרון הבא: כל עוד מתקיים תנאי מסוים, הקוד שבגוף הלולאה ימשיך לרוץ.	

כל עוד לביטוי הבוליאני יש ערך True הפקודות statements1 יתבצעו שוב ושוב. בסיום יתבצעו הפקודות בstatements2 .	`while <boolean expression>:`
	`statements1`
	`statements2`

העתקות:

- **השמה**- y=x אין שום העתקה. זה הצבעה זהה. כל שינוי על y ישפיע על x.
- **העתקה רדודה-** y=x[:] , יוצר רשימה חדשה אבל היא מכילה את אותם רפרנסים, לכן אם יהיה שינוי בתתי רשימות ההשפעה תיהיה גם על x וגם על y. אבל x is not y. אם נשנה את x זה לא ישפיע על y אבל אם נשנה את התתי רשימות אז כן.
- **העתקה "עמוקה"-** לא נמצאת בסיס של פייתון, נמצאת בספריית copy. צריך לייבא אותה. יוצר רשימה נפרדת שאין השפעה של הרשימה אחת מהשנייה. y= copy.deepcopy(x). x ו-y בלתי תלויים.