**סיכום פייתון למבחן**

|  |  |  |
| --- | --- | --- |
| **מה** | **עמוד** | **הערות** |
| תוכן עניינים | 1 |  |
| מודולים | 2 - 6 | Import math: 2 – 4  Import string: 4 – 5  Import random: 5 – 6 |
| פונקציות built in | 6 – 13 |  |
| כללי | 13 – 29 | דיני חלוקה, הדפסה בטור, תנאים מורכבים 13  איטרבל, אינטרטורים, range, מחרוזות 14  מטריצות 15  רשימות 15 – 16  tuple 16  קבוצות 17 – 18  מילונים 18 – 20  טבלת השוואת טיפוסים 20  טבלת טווחים, תנאים 21  פקודות לולאות 21 – 22  בוליאני, משתנים גלובליים, פונ' עם מספר פרמטרים לא ידוע 23  טיפול בקבצים 24  דוגמאות 25 – 29 |

**אופרטורי השוואה Comparison Operators**   **+ דוגמאות**





**תנאים מורכבים**

**טבלאות אמת לאופרטורים הלוגיים and, or, not**



* A and B שקול ל not ( not A or not B )
* A or B שקול ל not ( not A and not B )
* not ( not ( A )) שקול ל A

**חוקי פילוג:**

* A or ( B and C ) שקול ל ( A or B ) and (A or C)
* A and ( B or C ) שקול ל ( A and B ) or (A and C)

**על תנאים לוגיים (ביטוי בוליאני)**

* תנאי הוא ביטוי שמחזיר אחד מהערכים הבאים: True או False.
* בפייתון, כל ערך מספרי שונה מ-0, נחשב לנכון (True) ו-0 נחשב לשקר (False).
* כנ"ל כל מחרוזת או רשימה שאורכה שונה מאפס, נחשבת לנכון (True).

מחרוזת או רשימה באורך 0 נחשבת לשקר (False).

**טבלאות אמת**

**and:**

|  |  |
| --- | --- |
| **ביטוי** | **תוצאה** |
| **true and true** | **True** |
| **true and false** | **False** |
| **false and true** | **False** |
| **false and false** | **False** |

|  |  |
| --- | --- |
| **ביטוי** | **תוצאה** |
| **true or true** | **True** |
| **true or false** | **True** |
| **false or true** | **True** |
| **false or false** | **False** |

**מודולים חשובים**

Import math

מודול המכיל מספר פונקציות מתמטיות .מומלץ לשים אותו בתחילת כל תרגיל רלוונטי, ולחסוך כאב ראש אח"כ. 

|  |  |  |  |
| --- | --- | --- | --- |
| **שם** | **תיאור** | **דוגמא** | |
| math.pi | פאי |  | |
| math.exp() | אקספוננט |  | |
|  | מחזיר x (כ float) עם הסימן של y |  | |
| math.gcd(a,b) | מחלק משותף גדול ביותר של a ו b |  | |
| math.isqrt(n) | נותן את השורש הריבועי השלם הקרוב ביותר למספר השלם n |  | |
| math.isfinite() | מחזיר אמת אם המספר סופי |  | |
| math.isclose() | מחזיר אמת אם שני המספרים קרובים ושקר אחרת |  | |
| math.ceil(a) | מעגל כלפי מעלה |  | |
| math.floor(a) | מעגל כלפי מטה |  | |
| math.trunc(x.yz) | מעגל למספר השלם שמרכיב את המספר שניתן לו (x) |  | |
| math.modf() | מחזיר את המספר השלם והשבר |  | |
| math.fmod() | מחזיר את שארית החלוקה |  | |
| math.ldexp(x,y) | מחזיר את התוצאה של  x\*(2^y) |  | |
| math.sin() | סינוס (ברדיאנים, תקף גם לשאר הפונקציות הטריגונומטריות) |  | |
| math.acos() | פונקציה טריגונומטרית הפוכה  (ברדיאנים, תקף גם לשאר הפונקציות הטריגונומטריות ההפוכות) |  | |
| math.factorial(a) | עצרת של a |  | |
| math.remainder() | שארית |  | |
| math.fsum() | מחשב סכום ומעגל למעלה |  | |
| math.comb(n,k) | מקדמי בינום לפי הנוסחא  0 | |  |
| math.log10() | לוגריתם בבסיס 10 (עשר בחזקת מה ייתן לי את הביטוי בסוגריים) |  | |
| math.log() | לוגריתם (ברירת המחדל היא האקספוננט) |  | |
| math.log(a,b) | הלוג של a לפי בסיס b |  | |
| math.log2(x) | לוגריתם של x לפי בסיס 2 |  | |
| math.degrees(x) | ממיר את x למעלות דיגריז () |  | |
| math.radians(x) | ממיר את x למעלות רדיאנים (2) |  | |

Import string

מודול המכיל מספר פונקציות לטיפול במחרוזות. 

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **שם** | **תיאור** | | | | | **דוגמא** | |
| string.ascii\_uppercase | הדפסת האותיות הגדולות באלף בית האנגלי | | | | |  | |
| string.ascii\_lowercase | הדפסת האותיות הקטנות באלף בית האנגלי | | | | |  | |
| string.ascii\_letters | | הדפסת אותיות האלף בית האנגלי | | |  | | |
| string.ascii\_letters[] | הדפסת אותיות בחלוקות משתנות  (שימו לב שההוראה האחרונה משאירה שורה ריקה) | | |  | | | |
| string.digits | | | מחרוזת של 10 הספרות | | | |  |

Import random

מודול המכיל מספר פונקציות רנדומליות. 

טיפ לסטודנט העצלן ו/או הממהר/סתם זה שאין לו כוח לרשום random כל הזמן:

ניתן ליצור קיצור בעת הייבוא.



|  |  |  |
| --- | --- | --- |
| **שם** | **תיאור** | **דוגמא** |
| random.random() | מספר אקראי בין אפס לאחת. |  |
| random.randint() | מספר אקראי בין שני מספרים, כולל מספרי הגבול עצמם. |  |
| random.uniform() | מספר ממשי אקראי בין שני מספרים, כולל מספרי הגבול עצמם |  |
| random.randrange(n) | מספר שלם אקראי בין 0 ל n – 1 |  |
| random.randrange(a,b,n) | מספר שלם אקראי בין a ל b – 1  בקפיצות של n |  |
| random.sample | מוציא k ערכים אקראיים מרשימה (/קבוצה וכו') נתונה |  |
| random.choise(seq) | מוציא תו אקראי ממחרוזת נתונה או מחרוזת |  |
| random.shuffle() | מערבב איברי רשימה באופן אקראי ולא מחזיר אותם אח"כ לסידור המקורי |  |

**פונקציות built-in**

פונקציות מוכנות שאפשר להשתמש בהם מבלי לזמן מודולים.

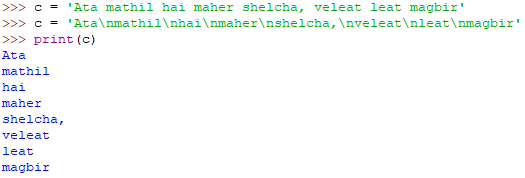
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **שם (ועל מה)** | | | **תיאור** | | | **דוגמא** | | | | |
| אופרטור in(רק על מחרוזת)  -בדיקת שייכות | | | האם אות או ביטוי נמצא במחרוזת | | |  | | | | |
| אופרטור not (מחרוזת) | | | מאפשר לבדוק את אי-שייכות (ההיפוך הלוגי של in) | | |  | | | | |
| pow(a,b)  (אריתמטית) | | | העלאת a בחזקתb | | |  | | | | |
| abs  (אריתמטית) | | | מחזיר ערך מוחלט | | |  | | | | |
| max(a,b)  \*הפונקציה גם יכולה לקבל רשימה ולבדוק את הערך הגבוה ברשימה.  ***max(grades)*** | | | מחזיר את הערך הגבוה יותר | | |  | | | | |
| min(a,b)  \*הפונקציה גם יכולה לקבל רשימה ולבדוק את הערך הנמוך ברשימה.  ***min(grades)*** | | | מחזיר ערך נמוך יותר | | |  | | | | |
| int() | | | משנה משתנה למשתנה מסוג int | | |  | |  | | |
| floor() | | | משנה מספר למספר מסוג float | | |  | |  | | |
| round | | | עיגול למספר השלם הקרוב ביותר | | |  | | | | |
| bin(a) | | | ממיר מספר עשרוני a למספר הבינארי התואם לו. | | |  | | | |  | |
| complex(a,b) | | | | ממיר למספר מדומיין, j מייצג את i המספר המדומה | | | |  | | | |
| zip | | | ריצ'רץ' – שילוב של שני מערכים | | |  | | | | |
| map | | | הפעלת פונקציה על איברים ברשימה | | |  | | | | |
| clear() | | | מרוקן מהאיברים | | |  | |  | | |
| sort() | | | ממיין מהקטן בשמאל לגדול בימין | | |  | | | | |
| insert(a,b)  (list)  myList.insert(pos,x) | | | הכנסת פריט (b) לתוך הרשימה (במיקום a). | | |  | | | | |
| append  (list)  myList.append(x) | | | מוסיף פריט יחיד לסוף הרשימה.  \*אפשר להוסיף גם רשימה לתוך רשימה באמצעות הפונקציה. | | |  | | | | |
| Extend  (list) | | | הוספת איברים לסוף הרשימה. | | |  | | | | |
| remove | | | מחיקת פרטים לפי מופע | | |  | | |  | |
| reverse  (רשימה) | | | פונקציה שהופכת את סדר הרשימה | | |  | | | | |
| שימוש ב sort ו reverse לצורך מיון הפוך |  | | | |  | | | | | |
| sum(l) | | | מחשב סכום האיברים של l | | |  | |  | | | |
| len(a) | | | מחשב אורך של a(מחרוזות, רשימות)  הספירה מתחילה ב 1. | | |  | |  | | | |
|  | | | |
| pop | | | מחיקת איבר והדפסתו, איבר אחרון כברירת מחדל אם לא ניתן איבר ספציפי בסוגריים | | |  | |  | | | |
|  | | | | |
| split() | | | פיצול | | |  | | | | |
|  | | | | |
|  | | | | |
| partition() | | | פיצול מחרוזת לשלוש מחרוזות לפי תת מחרוזת שתינתן על ידנו | | |  | | | | |
| list | | | המרת טיפוסים לרשימה | | |  | | | | |
| type(x) | | | קבלת הסוג של x | | |  | | | | |
| str(x) | | | הופכת ביטוי שאינו מחרוזת למחרוזת | | |  | | | | |
| tuple() | | | ממירה ל tuple | | |  | | | | |
| x.update(y)  (קבוצות) | | | הוספת איברים ל x שאינם נמצאים ב x  מ y | | |  | | | | |
| ord  (יוניקוד) | | | מציג את הערך המספרי של איבר לפי טבלת Unicode | | |  | | | | |
| chr  (יוניקוד) | | | מציג את הערך של מספר בטבלת Unicode | | |  | | | | |
| count() | | | סופר כמה מופעים יש לאיבר מסוים בתוך משתנה. | | |  | | | | |
| Isinstance  isinstance(object,type or class) | | | מקבל אובייקט וטיפוס ומחזיר אמת אם יש תאום בניהם ושקר אחרת | | | ניתן גם להשתמש בעבור  כמה טיפוסים.  לדוגמא האם אובייקט הוא מספר שלם או עשרוני  ***>>> n=23.7***  ***>>> isinstance(n,(int, float))***  ***True*** | | | | |
| all() | | | בודק האם כל המפתחות במילון (או האיברים בטיפוס אחר) הם True (לא אפס) ומחזיר אמת אם כן, שקר אחרת | | |  | | | | |
| any() | | | בודק האם יש מפתח במילון (או איבר בטיפוס אחר) שהוא True (לא אפס) ומחזיר אמת אם כן, ושקר אחרת | | |  | | | | |
| Eval  eval(expression) | | | פונקציית  ***eval()***מקבלת **מחרוזת**, מפענחת אותה כביטוי בשפת פייתון, מחשבת את ערך הביטוי הזה, ומחזירה את ערכו.  \*באמצעות פונקציות EVAL אפשר להכניס כמה משתנים לקלט של פונקציה  a,b,c=eval(input("please enter a,b,c: ")) | | | L1 = input('Enter the first list to add: ')  L2 = input('Enter the second list to add: ')  # the input for each list will be like [23,56,78,-34] or ['aa','bb']  L1 = eval(L1)  L2 = eval(L2)  Lresult = []  for i in range(len(L1)):  Lresult.append(L1[i] + L2[i])  print(Lresult)  תוצאה:  [1, 2, 3, 4] + [4, 5, 6, 7] → [5, 7, 9, 11] | | | | |
| find() | | | מחפשת תת מחרוזת בתוך מחרוזת, ומחזירה את מיקום התו הראשון של המחרוזת, או 1 – אם לא נמצא. במידה ומופיע יותר מפעם אחת, תחזיר את המיקום של המופע השמאלי ביותר. | | | |  | | | |
| rfind() | | | כמו find, רק עובדת מימין לשמאל | | |  | | | | |
| replace() | | | החלפה של תו בתו אחר | | |  | | | | |
| startswith() | | | מחזירה אמת אם מחרוזת מתחילה ברצף נתון ושקר אחרת | | |  | | | | |
| endswith() | | | מחזירה אמת אם מחרוזת מסתיימת ברצף נתון ושקר אחרת | | |  | | | | |
| upper  (גופן) | | | משנה אותיות קטנות לגדולות | | | | | | | |
|  | | | | | | | | | | |
| lower  (גופן) | | משנה אותיות גדולות לקטנות | | | |  | | | | |
| strip()  )גופן) | | | מחיקת איברים נבחרים במחרוזת (או בתחילתה או בסופה), ויצירת מחרוזת חדשה בלעדיהם. | | |  | | | | |
| capitalize()  )גופן) | | | המרת האות הראשונה באות גדולה | | |  | | | | |
| title()  )גופן) | | | המרת האות הראשונה בכל מילה באות גדולה | | |  | | | | |
| islower() | | | מחזיר אמת אם כל האותיות קטנות ושקר אחרת | | |  | | | | |
| isupper() | | | מחזיר אמת אם כל האותיות גדולות ושקר אחרת | | |  | | | | |
| **enumerate**  (טיפוסים שאינם משתנים) | | | מחזירה זוגות של טפלים כשהראשון הוא פריט שניתן לספירה, והשני ערך מטיפוס שאינו משתנה. | | |  | | | | |
| join() | | | צירוף תווים למחרוזת | | |  | | | | |
| swapcase() | | | מחליפה אותיות גדולות בקטנות ולהיפך | | |  | | | | |
| isalpha()  isnumeric() | | | מחזיר אמת אם כל התווים נמצאים באלפבית האנגלי (isalpha())/מספרים (isnumeric()) ושקר אחרת | | |  | |  | | |
| אחרת. False, מכילה ספרות בלבד str אם True מחזירה **str.isdigit()הפונקציה**  אחרת.False, מכילה אותיות בלבד str אם True מחזירה **str.isalpha()הפונקציה**  , מכילה שילוב של ספרות ואותיות, str אם True מחזירה **str.isalnum()הפונקציה**  כאותיות גדולות. str מחזירה מחרוזת עם האותיות של **str.upper()** **הפונקציה** | | | | | | | |  | | |
| isalnum() | | | מחזיר אמת אם כל התווים נמצאים באלפבית האנגלי /מספרים (ללא סימנים מיוחדים ורווחים) ושקר אחרת | | | |  | | | |

כללי

* דיני חלוקה:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

* הדפסה בטור נעשית בעזרת n\:



Escape sequences

מאפשרים למתכנת להשחיל תווים מיוחדים בתוך המחרוזות השולטים על איך הטקסט יוצג על המסך.

כל escape sequence בנוי מ- 2 תווים: התו הראשון הוא backslash ואח"כ באה אות לועזית מסוימת.

התו \

| **Escape Sequence** | **משמעות** |
| --- | --- |
| \*n* | ירידת שורה |
| \\ | Backslash (\) |
| \' | Single quote (') |
| \" | Double quote (") |
| \t | Horizontal Tab (TAB) |

מחרוזות המסומנות ע"י שלושה גרשיים, **יכולות להתפרס על פני מספר שורות** :

***>>> message = """This message will***

***span several***

***lines."""***

***>>> message***

***'This message will\nspan several\nlines.'***

***>>> print( message)***

***This message will***

***span several***

***lines.***

**איטרבל**

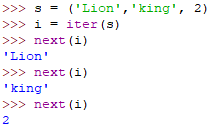
Iterable הוא טיפוס שניתן לעבור עליו לפי סדר הנוצר באמצעות iter או range, ניתן להגדיר אותו ככזה שאפשר להפעיל לולאת for כך שהיא תעבור על כל אובייקט של הטיפוס.

דוגמאות: str, list, tuple, set, dict, file, range, frozenset.

דוגמאות ללא: int, כל טקסט התא של האובייקטים בשורה למעלה (להוציא str).

איטרטורים

איטרטור הוא אובייקט שניתן לעשות עליו איטרציות באמצעות הפונקציות (iter, defval)next, המחזירה בכל קריאה איבר הבא בתור. לאחר מכן הוא נמחק מהאיטרטור.



range

פונקציית טווח.

|  |  |
| --- | --- |
| range(5) = [0,1,2,3,4]  \*נשים לב! יוצרת רשימה שלא כוללת את 5 | range(9,5) = [] |
| range(5,9) = [5,6,7,8]  \*נשים לב! יוצרת רשימה שלא כוללת את 9 | range(5,20,5) = [5,10,15] |
| שימושים בפונקציה range | |
| יצירת לולאת FOR שתרוץ כמות פעמים לפי בחירה שלנו. | for i in range(3):  print('a')   #a #a #a |

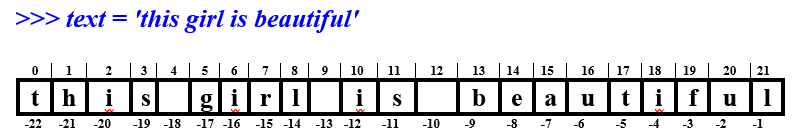
מחרוזות

התו \

| **Escape Sequence** | **משמעות** |
| --- | --- |
| \*n* | ירידת שורה |
| \\ | Backslash (\) |
| \' | Single quote (') |
| \" | Double quote (") |
| \t | Horizontal Tab (TAB) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1.הגדרות בסיס>> | סדור? כן | ניתן לשינוי?  לא!(Immutable) | טיפוס אחד?  כן! (רק תווים) | סימן היכר?  מירכאות או גרשיים |
| 2.חיבור מחרוזות בעזרת הסימן + | שימוש?  יצירת מחרוזת חדשה. |  |  | דוגמא |
| ניתן לכפיל מחרוזות | יצירת מחרוזת חדשה. |  |  | דוגמא |
| יצירת מחרוזת ריקה |  |  |  | ***>>> emptyStr=""***  ***>>> len(emptyStr)***  ***0*** |
| קריאת מחרוזת מהמסך |  |  |  | שם משתנה=input("Your string please ") |

Slices- תתי-מחרוזות



# שיטות (Methods) על מחרוזות

|  |  |  |
| --- | --- | --- |
| **השיטה** | **הסבר** | **דוגמא** |
| slice-notation**-**  הבטוי בפייתון המאפשר לנו לשלוף תת-מחרוזת מתוך מחרוזת נתונה**.** | text[a:b] תשלוף תת-המחרוזת שבין המקום a לבין המקום b-1, ללא התו text[b].  \*אם b גדול ממספר התו האחרון במחרוזת, נשלף הרצף עד כולל התו האחרון. | ***>>> text[0:3]***  ***'thi'***  ***>>> text[0:4]***  ***'this'*** |
| מותר להשמיט את ערך האינדקס הראשון [b:] | ה-slice מתחיל מהמקום 0, התו הראשון של המחרוזת. | ***>>> text[:3]***  ***'thi'*** |
| מותר להשמיט את ערך האינדקס השני [:a] | ה-slice מסתיים בתו האחרון של המחרוזת | ***>>> text[3:]***  ***'s girl is beautiful'*** |
| מותר להשמיט את שני האינדקסים [:] | ערך האינדקס הראשון של ה-slice יהיה 0 וערך האינדקס השני יהיה המיקום של התו האחרון של המחרוזת | ***>>> text[:]***  ***'This girl is beautiful'*** |
| **בחירת גודל הצעד (ברירת המחדל היא 1)**  **[ : : : ]**  String[a,b,jump] | הערך השלישי בסגוריים נותן אפשרות לבחור את גודל הצעד.  \* ציון צעד שלילי מסמן שהקפיצה היא מהסוף להתחלה. | ***>>> s = 'abcdefg'***  ***>>> s[2:7:2]***  ***'ceg'***  ***\* >> s[-2:-7:-2]***  ***'fdb'*** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| המשך שיטות | | הסבר | | | דוגמא | |
| Count-  st.count(sub)  st.count(sub, start, end)  מותר להגדיר טווח לחיפוש:  str.count(s,beg,end) | | השיטה סופרת ומחזירה את מספר המופעים של תת מחרוזת בתוך מחזורת. אם לא מוגדר טווח לחיפוש, החיפוש מתבצע בכל המחרוזת. | | | בדומה לשיטה find, מותר להגדיר טווח מתוך המחרוזת str ואז הספירה של המופעים של s, נעשית רק בתוך הטווח:    ***name.count("i",4,11)***  התוצאה היא 1 | |
| Find-  st.find(sub)  st.find(sub,start, end)  \*פונקצית  ***rfind()***היא כמו ***find()***, אך היא מתחילה את החיפוש מצד ימין אל צד שמאל  \*\*מותר להגדיר טווח לחיפוש:  str.find(s,beg,end) | | פונקציה לחיפוש תת-מחרוזת בתוך מחרוזת.  \*מחפשת משמאל לימין.  \*\* במקרה שיש כמה מופעים של sub, הפונקציה מחזירה את המיקום של המופע השמאלי ביותר. | | | ***>>> seq = 'agcgccttgaattcggcaccaggcaaatctta'***  ***>>> seq.find('gcc')***  ***3***  ***>>> seq.find('gc',2,6)***  ***3*** | |
| str.index(s) | | זהה לשיטה find מלבד ההבדל הבא: כאשר מחפשים תת מחרוזת s שאינה נמצאת במחרוזת str, מתקבלת שגיאה. | | |  | |
| replace()-  str.[replace(old, new [, max])](http://www.tutorialspoint.com/python/string_replace.htm) | | מחזיר מחרוזת ובה מחליף את כל המופעים של old ב-new. ניתן להגביל ל-max פעמים. המחרוזת המקורית אינה משתנה. | | |  | |
| str.upper()  \*  str.lower() | מחזיר מחרוזת בה כל האותיות גדולות.  \* מחזיר מחרוזת בה כל האותיות קטנות. | | |  | | |
| **פונקצית split**  str.split()  str.split(sep, maxsplit)  **פונקציית rsplit**  str.rsplit()  str.rsplit(sep)  str.rsplit(sep, maxsplit***)*** | | | פונקציה זו מפרידה את המחרוזת על ידי תו שאנחנו בוחרים ויוצרת מבנה של **רשימה** של מחרוזות.  \*\* על ידי שימוש בפרמטר האופציונאלי maxsplit, אפשר לציין בכמה "מפרידנים sep" להשתמש. ואז מספר המילים שנקבל לא יעלה על maxsplit+1 | | | ***>>> string = "I live in Jerusalem"***  ***>>> string***  ***'I live in Jerusalem'***  ***>>> string.split()***  ***['I', 'live', 'in', 'Jerusalem']***  ***>>>***  \*\*  ***>>> 'banana'.split('an',1)***  ***['b', 'ana']*** |
| **join()**  'a'.join(['string1','string2']) | | | צירוף תווים **למחרוזת**.  מדביקה את כל המחרוזות **מתוך הרשימה**, כאשר "המפרידן" הוא המחרוזת אשר מופיעה לפני הנקודה | | |  |
| eval  eval(expression) | | | **מקבלת מחרוזת**, מפענחת אותה כביטוי בשפת פייתון, מחשבת את ערך הביטוי הזה, ומחזירה את ערכו. | | | ***>>> x = 5***  ***>>> y = 6***  ***>>> s = input('enter an operation to do with x and y: ')***  ***enter an operation to do with x and y: x\*\*y***  ***>>> eval(s)***  ***15625*** |

# מחרוזות - האופרטור in

האופרטור in בודק אם תת מחרוזת נמצאת בתוך מחרוזת, מחזיר True אם כן ו-False אם לא. הבדיקה היא Case sensitive.

# *>>> "a" in name*

# *True*

# *>>> "ax" in name*

# *False*

האופרטור not מאפשר לבדוק אם תת מחרוזת אינה נמצאת בתוך מחרוזת:

***>>> 'g' not in 'gtagtt'***

***False***

מטריצות

|  |  |
| --- | --- |
|  |  |

**רשימות**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| רשימה - List | סדור? כן | ניתנת לשינוי? כן!  Mutable  (להוסיף, למחוק, לשנות איבר ספציפי ולמיין. | טיפוס אחיד? לא!! מכל סוג(מספרים, מחרוזות, תמונות, קבצי שמע ועוד..) | איך מסמנים?  סוגריים מרובעים |

|  |  |  |  |
| --- | --- | --- | --- |
| החלפת איבר (אם היינו מפנים למקום שלא קיים הייתה מתקבלת שגיאה) |  | | |
| החלפה של כמה איברים ברשימה | כאשר אנחנו רוצים להחליף שני איברים ברשימה באיברים חדשים.  \*זה גם יכול להיות להחליף שני איברים ב3 איברים חדשים או אפילו איבר בודד. | | ***>>> listush = ['iosef', 7, fluor',8,'ruon',5]***  ***>>>istush[0:2]=['johny','frankie']***  ***>>> listush***  ***['johny', 'frankie', 'fluor', 8, 'ruon', 5]*** |
| הוספה של תוכן בין שני איברים של רשימה קיימת | הגבולות של ה slice הנמצא באגף השמאלי של ההשמה מבטא את המיקום בו אנחנו מעוניינים להכניס את תוכן הרשימה הנמצאת באגף הימני. | | ***>>> listush = ['iosef', 7, 'ety',8,'ruon',5]***  ***>>> listToInsert = ['alon','ronit']***  ***>>> listush[1:1] = listToInsert***  ***>>> listush***  ***['iosef', 'alon', 'ronit', 7, 'ety', 8, 'ruon', 5]*** |
|  | | הוספה לאמצע רשימה:  ***dug = [1,2,3]***  ***dug= dug[:1]+[66]+dug[1:]***  ***dug***  ***[1, 66, 2, 3]***  או  ***dug = [1,2,3]***  ***dug[1:1]=[99]***  ***dug***  ***1, 99, 2, 3]*** |
| השיטה insert(index, obj)  מכניסה את obj לפני המקום index | | ***dug=[1,2,3]***  ***dug.insert(2,-7)***  ***dug***  ***[1, 2, -7, 3]***  ***dug.insert(len(dug),100)***  ***dug***  ***[1, 2, -7, 3, 100]*** |
| גישה לאיבר/ים לפי מיקום |  | | |
| **Enumerate(list)**  יוצר רשימה של זוגות טאפלים עם האיברים ברשימה המקורית והאינדקס שלהם | L=['a','b','c','d','e', 'f', 'e']  for i in enumerate(L):  print(i) >>>> (0, 'a') (1, 'b') (2, 'c') (3, 'd') (4, 'e') | | |
| חיבור רשימות |  | | |
| חיבור רשימה עם מחרוזת |  | | |
| קינון רשימות |  | | |
| מחרוזת נחשבת בפייתון לרצפים, לכן אפשר לומר שכל מילה היא רשימה של תווים. | |  | |
| הפונקציה list()  כשיטה ליצירת רשימות  (הפונקציה הופכת אובייקט קיים לרשימה) | | וניתן להמיר tuple לרשימה:  ***person = ("Efrat", 27, "Beit Hakerem", 17, "Jerusalem", "Student")***  ***l=list(person)***  ***l***  ***['Efrat', 27, 'Beit Hakerem', 17, 'Jerusalem', 'Student']*** | |
| List Comprehension  שיטה נוספת ליצירת רשימות.  הדוגמא השנייה על מילון.  השלישית על מחרוזת. | |  | |
| sort()  ממיין מהקטן בשמאל לגדול בימין | | ***>>> my\_list = [ 34,67,89,15,9,23]***  ***>>> my\_list.sort()***  ***>>> my\_list***  ***[9, 15, 23, 34, 67, 89]***  ***>>> list1 = [ 'moshe','cohen']***  ***>>> list1.sort()***  ***>>> list1***  ***['cohen', 'moshe']*** | |
| sortedNewList=sorted(oldList)  מחזירה רשימה ממיונת חדשה! בלי לשנות את הרשימה הקודמת | | ***a = ("h", "b", "a", "c", "f", "d", "e", "g")***  ***x = sorted(a)***  ***print(x)***  ***<<<['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']***  x = sorted(a, reverse=True)  print(x) <<<['h', 'g', 'f', 'e', 'd', 'c', 'b', 'a'] | |

**הוספת איברים לזנב הרשימה**

|  |  |  |
| --- | --- | --- |
| באמצעות הפונקציה append | myList.append(x) | ***>>> listush = ['iosef', 7, 'ety',8,'ruon',5]***  ***>>>listush.append("baloon")***  ***>>> listush***  ***['iosef', 7, 'ety', 8, 'ruon', 5, 'baloon']*** |
| באמצעות שירשור | חיבור של רשימות | ***>>> listush = ['iosef', 7, 'ety',8,'ruon',5]***  ***>>> listush += ["baloon"]***  ***>>> listush***  ***['iosef', 7, 'ety', 8, 'ruon', 5, 'baloon']*** |
| באמצעות הפונקציה insert(a,b) | myList.insert(pos,x)  הכנסת פריט (b) לתוך הרשימה (במיקום a). | ***>>> listush.insert(len(listush),"ball")***  ***>>> listush***  ***['iosef', 7, 'ety', 8, 'ruon', 5, 'baloon', 'ball']*** |
| באמצעות הפונקציה  Extend  אם נרצה להוסיף רשימה אחת לתוך רשימה קיימת תוסיף את איברי הרשימה כבודדים בניגוד לפונקציה append שאם נשתמש בה על הרשימה שאותה אנו רוצים להוסיף לרשימה הקיימת, ה- append פשוט תוסיף איבר חדש יחיד שהוא יהיה כל הרשימה. | myList.extend(x)  \*\*\*אם נרצה להוסיף איבר בודד באמצעות EXTAND נקבל שגיאה, EXTAND מוסיפה רק איברים שהן Iterbale | ***>>> listush = ['iosef', 7, 'ety',8,'ruon',5]***  ***>>> listush.extend(["baloon", "ball"])***  ***>>> listush***  ***['iosef', 7, 'ety', 8, 'ruon', 5, 'baloon', 'ball']*** |

\*\*רק כאשר מוסיפים באמצעות append ניתן להוסיך רשימה שלמה, אם ננסה באמצעות הפונקציות האחרות יתוופסו רק האיברים בתוך הרשימה שאנחנו מוסיפים.

**מחיקת איברים מהרשימה**

|  |  |  |
| --- | --- | --- |
| באמצעות הוראת del | אחרי המילה del כותבים את האיבר שאנו רוצים למחוק מהרשימה. | מחיקת איבר בודד:  ***>>> listush = ['iosef', 7, 'ety',8,'ruon',5]***  ***>>> del listush[2]***  ***>>> listush***  ***['iosef', 7, 8, 'ruon', 5]***  מחיקת תת רשימה(slice)  ***>>> listush = ['iosef', 7, 'ety',8,'ruon',5]***  ***>>> del listush[:2]***  ***>>> listush***  ***['ety', 8, 'ruon', 5]*** |
| תוך שימוש ברשימה ריקה *[ ]* | איפוס רשימהאו מחיקת תת-רשימה(slice) | איפוס רשימה  ***>>> listush = ['iosef', 7, 'fluor',8,'ruon',5]***  ***>>> listush=[]***  ***>>> listush***  ***[]***  *מחיקת תת רשימה*  ***>>> listush = ['iosef', 7, 'fluor',8,'ruon',5]***  ***>>> listush[0:2]=[]***  ***>>> listush***  ***['fluor', 8, 'ruon', 0]*** |
| - באמצעות הפונקציה remove  \*שימו לב שכאן החיפוש הוא לפי ערך ולא לפי מיקום, ו x חייב להיות ברשימה.  אם x לא יהיה ברשימה נקבל שגיאה  \*\*כאשר יש שני ערכים זהים, הפונקציה תוריד את האיבר הראשון שמופיע. | myList.remove(x)  כאשר יודעים איזה איבר אנחנו רוצים למחוק | ***>>> listush = ['iosef', 7, 'fluor',8,'ruon',5]***  ***>>> listush.remove('fluor')***  ***>>> listush***  ***['iosef', 7, 8, 'ruon', 5]*** |
| **השיטה pop()** מסירה את האיבר האחרון ברשימה ומחזירה אותו. אפשר גם להעביר אינדקס כפרמטר. | Mylist.pop() | ***ls=[4,6,2,9]***  ***ls.pop()***  ***9***  ***ls [4,6,2]***  ***ls.pop(1)***  ***6***  ***Ls [4,2]*** |

**קינון רשימות List nesting))**

כאשר יש לנו רשימה, כל איבר יכול להיות בעצמו רשימה.

למציאות הזאת אנחנו קוראים לה קינון (***nesting***).

|  |  |  |
| --- | --- | --- |
| דוגמא פשוטה |  | ***>>> a = [1, 2, 3]***  ***>>> b = [5, a, 6]***  ***>>> a***  ***[1, 2, 3]***  ***>>> b***  ***[5, [1, 2, 3], 6]***  ***>>> b[1]***  ***[1, 2, 3]*** |
| ניתן לשנות את הרשימה של a ואז גם התכולה של b תשתנה בהתאם. |  | ***>>> a[1] = 'yes'***  ***>>> a***  ***[1, 'yes', 3]***  ***>>> b***  ***[5, [1, 'yes', 3], 6]*** |
| גישה לאיבר בתוך רשימה פנימית, מקוננת |  | ***>>> a = [1,2,3]***  ***>>> b = [5,a,6]***  ***>>> a[1] = 'yes'***  ***>>> b[1][1]***  ***'yes'*** |

tuple

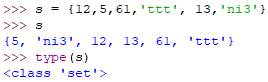
* לא ניתן לשינוי (Immutable)
* מסומן בסוגריים עגולים.
* ברירת המחדל בפייתון:



|  |  |  |
| --- | --- | --- |
| גישה לאיברים – באמצעות אינדקסים (כמו במחרוזות וברשימות): |  | ***person = ("Efrat", 27, "Beit Hakerem", 17, "Jerusalem", "Student")***  ***person[-2:]***  ***('Jerusalem', 'Student')*** |
| לא ניתן לשנות איברים של tuple: |  | ***person = ("Efrat", 27, "Beit Hakerem", 17, "Jerusalem", "Student")***  ***>>> person[4]="Tel Aviv"***  ***Traceback (most recent call last):***  ***File "<pyshell#139>", line 1, in <module>***  ***person[4]="Tel Aviv"***  ***TypeError: 'tuple' object does not support item assignment*** |
| שיטות הפועלות על tuple: index ו- count: |  | ***person.index("Student")***  ***5***  ***person.count(7)***  ***0***  ***person.count(17)***  ***1*** |
| חיבור tuples: |  | ***grades + person***  ***(89, 76, 100, 87, 'Efrat', 27, 'Beit Hakerem', 17, 'Jerusalem', 'Student')*** |
| כפל tuples: |  | ***grades\*3***  ***(89, 76, 100, 87, 89, 76, 100, 87, 89, 76, 100, 87)*** |
| האופרטור IN: |  | ***>>> "efrat" in person***  ***False***  ***>>> "Efrat" in person***  ***True*** |
| ההוראה del, ניתן למחוק רק tuple שלם: |  | ***del*** ***person*** |
| ניתן להמיר כל סוג איטרטיבי, ל-tuple וניתן להמיר tuple לרשימה:  **(סוג איטרטיבי הוא סוג המכיל איברים שניתן לעבור עליהם אחד-אחד באיזשהו סדר).** |  | ***l=list(person)***  ***l***  ***['Efrat', 27, 'Beit Hakerem', 17, 'Jerusalem', 'Student']***  ***list1 = [89,76,100, 87]***  ***grades = tuple(list1)***  ***grades***  ***(89, 76, 100, 87)*** |
| ניתן לפתוח tuple: |  | ***>>> grades***  ***(89, 76, 100, 87)***  ***>>> a,b,c,d = grades***  ***>>> a***  ***89***  ***>>> b***  ***76***  ***>>> c***  ***100***  ***>>> d***  ***87*** |

קבוצותset-

* אוסף איברים בו אין חשיבות לסדר, כל ערך יופיע פעם אחת בלבד:



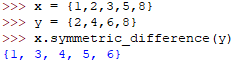
* מוגדר ע"י סוגריים מסולסלים.
* המחשב יכול לסדר את האיברים כרצונו.
* טוב לצורך היפטרות מערכים כפולים:



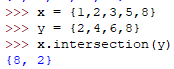
* להיזהר מבלבול עם מילון המוגדר גם הוט ע"י סוגריים מסולסלים.

|  |  |
| --- | --- |
| גישה לאיברים – באמצעות אינדקסים (כמו במחרוזות וברשימות): | ***person = ("Efrat", 27, "Beit Hakerem", 17, "Jerusalem", "Student")***  ***person[-2:]***  ***('Jerusalem', 'Student')*** |
| לא ניתן לשנות איברים של tuple: |  |
| פונקציות ושיטות בSET |  |
| חיסור בין שתי קבוצות.  \*\* לחילופין, ניתן להשתמש במתודה difference | ***\*\*\*\*\**** |
| איחוד  איברים ב-fruit או ב- fruitILikeהאפרטור | : | ***fruit={'orange', 'pear', 'apple', 'banana'}***  ***fruitILike = {'plum', 'apple', 'grapes', 'banana'}***  ***fruit | fruitILike***  ***{'apple', 'plum', 'pear', 'grapes', 'orange', 'banana'}***  זהה לשיטהunion***:***  ***fruit.union(fruitILike)***  ***{'apple', 'plum', 'pear', 'grapes', 'orange', 'banana'}*** |
| חיתוך  איברים הנמצאים בשני הסטים, גם ב-fruit וגם ב- fruitILikeהאופרטור & | ***fruit & fruitILike***  ***{'apple', 'banana'}***  זהה לשיטהintersection***:***  ***fruit.intersection(fruitILike)***  ***{'apple', 'banana'}*** |
| איברים ב-fruit או ב- fruitILike אך לא בשניהם האופרטור ^ : | ***fruit ^ fruitILike***  ***{'plum', 'pear', 'grapes', 'orange'}***  זהה לשיטה symmetric\_difference:  ***fruit.symmetric\_difference(fruitILike)***  ***{'plum', 'pear', 'grapes', 'orange'}*** |
| **הוספת איבר ל- set, השיטה add** | ***>>> set2.add(8)***  ***>>> set2.add("hello")***  ***>>> set2.add(9.9)***  ***>>> set2.add((1,2,3,4))***  ***>>> set2***  ***{1, 2, 3, 4, (1, 2, 3, 4), 8, 9.9, 'hello'}***  ***\*\*\*\**** אבל, לא ניתן להכניס ל-set איברים שניתנים לשינוי:  ***>>> set2.add([8,9,10])***  ***Traceback (most recent call last):***  ***File "<pyshell#37>", line 1, in <module>***  ***set2.add([8,9,10])***  ***TypeError: unhashable type: 'list'***  ***>>> set2.add({1,7,9})***  ***Traceback (most recent call last):***  ***File "<pyshell#38>", line 1, in <module>***  ***set2.add({1,7,9})***  ***TypeError: unhashable type: 'set'*** |
| **מחיקת איבר מ- set****השיטה remove** **השיטה discard** | ***>>> set2***  ***{1, 2, 3, 4, (1, 2, 3, 4), 8, 9.9, 'hello'}***  ***>>> set2.remove(4)***  ***>>> set2***  ***{1, 2, 3, (1, 2, 3, 4), 8, 9.9, 'hello'}***  *ניסיון למחוק איבר שאינו קיים בעזרת השיטה remove, יגרום לשגיאה:*  ***>>> set2.remove(10)***  ***Traceback (most recent call last):***  ***File "<pyshell#42>", line 1, in <module>***  ***set2.remove(10)***  ***KeyError: 10*** \*\*\* **השיטה discard** אין שגיאה אם האיבר לא נמצא ב-set  ***>>> set2***  ***{1, 2, 3, (1, 2, 3, 4), 8, 9.9, 'hello'}***  ***>>> set2.discard(9.9)***  ***>>> set2***  ***{1, 2, 3, (1, 2, 3, 4), 8, 'hello'}***  ***>>> set2.discard(9.9)***  ***>>> set2***  ***{1, 2, 3, (1, 2, 3, 4), 8, 'hello'*** |
| יחסים על sets | ***sp={1,2,3,5,7,11,13,17, 19}***  ***sa={7, 19, 3, 2}***  ***>>> sa<sp***  ***True***  ***>>> sa<=sp***  ***True***  ***>>> sp<sp***  ***False***  ***>>> sp<=sp***  ***True***  ***>>> sp==sp***  ***True*** |
|  |  |
|  |  |

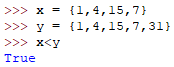
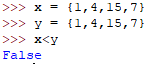
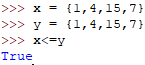
* אם רוצים לקחת את כל האיברים הנמצאים בקבוצה אחת ולא בשנייה, נעזרים ב symmetric\_diffrence:



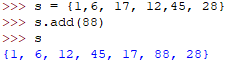
* אם רוצים לקחת את כל האיברים הנמצאים בשתי הקבוצות, נעזרים ב intersection:



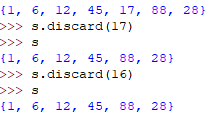
* יחזיר True אם כל האיברים ב x מוכלים ב y, ו False אם לא.

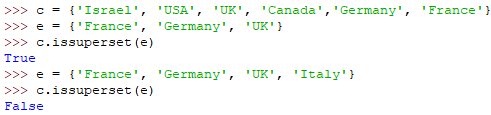
* ניתן לשנות קבוצה, אבל אי אפשר לשנות איברים בתוכו.
* הוספת איבר בעזרת add:



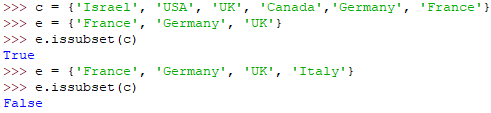
* ניתן לחסר ערך עם remove, אך עדיף עם discard, כדי לא לקבל שגיאה עם האיבר לא קיים בקבוצה:



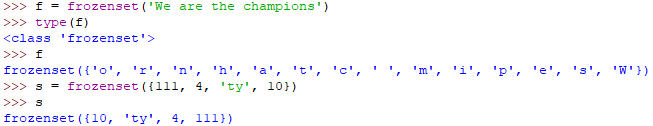
* אם סט ערכים מכיל סט שלם אחר הוא מוגדר כסופר סט ביחס לסט השני. המתודה issuperset תבדוק זאת עבורנו:



* - אם סט ערכים נכלל בשלמותו בסט שלם אחר הוא מוגדר כת סט ביחס לסט השני. המתודה issubset תבדוק זאת עבורנו:



* קבוצה קפואה היא קבוצה שלא ניתנת לשינוי לאחר שנוצרה:



מילונים

1. מילון מורכב מזוגות של ערכים הנקראים מפתח וערך ובאנגלית, key and value.

ניתן ליצור מילון ע"י הצבה:

name\_of\_dict = {key1:value1, key2:value2, …….}

1. הזוגות חסומים בסוגריים מסולסלים, בין הזוגות מפרידים פסיקים, בין מפתח לערך מפרידים נקודתיים.

דוגמא:

***dict1 = {1:"one",2:"two",3:"three"}***

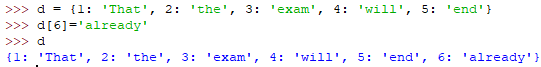
1. אין סדר בין הזוגות ולכן אברי המילון יוצגו בסדר כלשהו בלתי צפוי.

|  |  |
| --- | --- |
| **מילון הוא mutable – כלומר ניתן לשינוי** |  |
| ניתן **להוסיף** איברים למילון: | ***>>> dict1[4]='four'***  ***>>> dict1***  ***{1: 'another', 2: 'two', 3: 'three', 4: 'four'}*** |
| ניתן **לעדכן** ערך של מפתח קיים: | ***>>> dict1[1]='one'***  ***>>> dict1***  ***{1: 'one', 2: 'two', 3: 'three', 4: 'four'}*** |
| ניתן **למחוק** איבר באמצעות ההוראה **del**: | ***>>> del dict1[3]***  ***>>> dict1***  ***{1: 'one', 2: 'two', 4: 'four'}*** |
| ניתן למחוק גם באמצעות השיטה **pop** כשהפרמטר הוא המפתח של הזוג למחיקה: | ***>>> dict1.pop(3)*** |

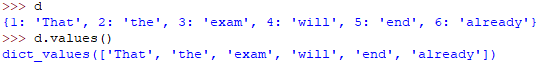
**שיטות של מילונים**

|  |  |
| --- | --- |
| שיטה |  |
| dict.values() | השיטה מחזירה נתון מסוג dict.values. אפשר לעבור עליו בלולאת for ולקבל את ערכי המילון. |
| dict.keys() | השיטה מחזירה נתון מסוג dict. keys. אפשר לעבור עליו בלולאת for ולקבל את מפתחות המילון |
| דרך נוספת לגשת לערך היא באמצעות השיטה **get**:  dict.get(key[default])  key הינו פרמטר חובה המכיל את המפתח שאת ערכו רוצים לקבל, default הינו פרמטר רשות שיכיל ערך שיוחזר, במידה והמפתח key אינו נמצא במילון dict. | ***dict1***  ***{1: 'one', 2: 'two', 3: 'three', 4: 'four'}***  ***dict1.get(2)***  ***'two'***  ***dict1.get(2,"NO SUCH KEY")***  ***'two'***  ***dict1.get(5,"NO SUCH KEY")***  ***'NO SUCH KEY'*** |
| הפונקציה zip מאפשרת ליצור רשימה של tuples משתיים או יותר רשימות.  **מרשימה של tuples אפשר ליצור מילון.**  **\*\*** אם הרשימות באורך שונה, תהיה התוצאה באורך הרשימה הקצרה. | ***l1=[1,2,3,4]***  ***l2=['a', 'b', 'c', 'd']***  ***t1=zip(l2,l1)***  **ליצור מילון:**  ***d1=dict(t1)***  ***{'a': 1, 'c': 3, 'b': 2, 'd': 4}*** |
| איך יודעים אם איבר יכול להיות מפתח במילון???  בגלל אופן הפעולה של מילון, ערכי המפתח (בדומה לערכים השייכים לקבוצה) חייבים להיות כאלה שניתן להפעיל עליהם פונקציית hash; כלומר, חייבים להיות hashable values:  ערכים אלו הם ערכים אשר לא ניתנים לשינוי(מחרוזת,מספרים, רשומות) | **>>> import collections.abc**  **>>> isinstance([1,2], collections.abc.Hashable)**  **False**  **>>> isinstance((1,2), collections.abc.Hashable)**  **True** |
| dict.items() | השיטה מחזירה נתון מסוג dict. items. אפשר לעבור עליו בלולאת for ולקבל tuples של מפתח וערך.  ***for i in spanish.items():***  ***print(i)***    ***('one', 'uno')***  ***('two', 'dos')***  ***('three', 'tres')***  אפשר לפתוח את ה-tuples  ***for k,v in spanish.items():***  ***print("key=", k, "value =", v)***    ***key= one value = uno***  ***key= two value = dos***  ***key= three value = tres*** |
| dict.update(anotherDict) | מוסיף את המילון בפרמטר, למילון dict. |
|  |  |

* הוספת ערך למילון:



* הצגת הערכים של המילון:



* הצגת המפתחות של המילון:
* 
* גריעת ערך ממילון:



* גישה לערך לפי מפתח:

-

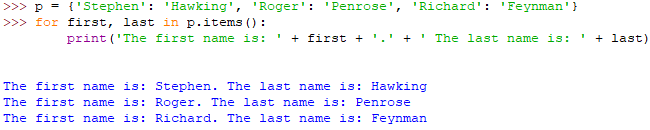
- מפתח יכול להכיל מספר ערכים:



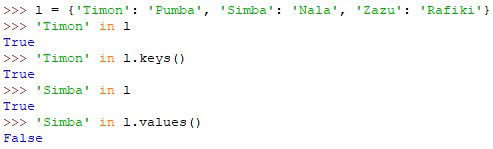
* ניתן לשנות ערכים במילון, לא מפתחות:



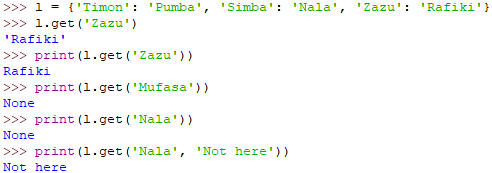
* המתודה items יוצרת רשימה של צמדים מסודרים בתוך טאפלים לצורך עבודה עם המפתחות והערכים באותה הלולאה (לא לשלב בין מחרוזת למספרים, זה יעשה בעיות):



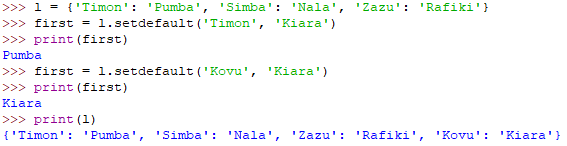
* בדיקה אם ערך נמצא במילון:



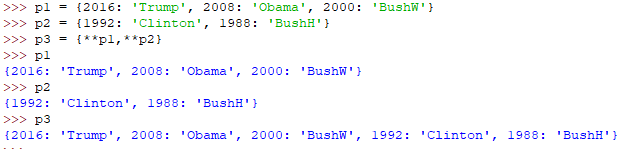
* שימוש במתודה get לבדיקה האם מפתח נמצא במילון כדי לא לקבל שגיאה אם לא:



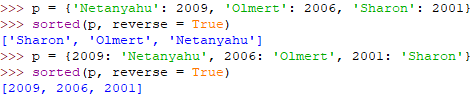
* שימוש ב setdefault מאפשר להוסיף מפתח שאינו קיים:



* איחוד מילונים תוך שמירה על המילונים המקוריים בדרך הבאה:



* הפיכת סדר המיון:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **טיפוס** | **סדור** | **ניתן לשינוי** | **סימון** | **אחיד** | **גישה לאיברים** |
| **string** | כן | לא | " ", ' ' | כן תווים בלבד | כן |
| **list** | כן | לא | [,] | לא | כן |
| **Tuple** | כן | לא | (,) | לא | כן |
| **Set** | לא | כן | {,} | לא, חייב להיות immutable | לא |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **סימון** | **מתייחס לראשון** | **מתייחס לאחרון** | **הערות** |
| **len** | len(object) | כן, מתחיל מ1 | כולל |  |
| **טווח** | X[first, last] | כן | לא כולל |  |
| **טווח** | [2:] | כן | כולל |  |
| **string** | str.find("s",beg,end) | כן | לא כולל | s ="a,c,c"  s.find("b",8,9)  -1(לא נמצא) |
| **A,range** | range(1,3) | כן | לא כולל | ייתן 21, |

תנאים

|  |  |
| --- | --- |
| (בדיקה האם מספר חיובי, שלילי או אפס) |  |

**לולאות**

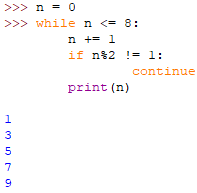
|  |  |  |
| --- | --- | --- |
| For | מאפשרת יצירת פקודות שרצות מספר פעמים בהתאם לאינדקס.  המבנה הכללי:  ***for משתנה (index) = רצף (index range):***  ***statement(s)***  ה-for מורכב מ*רצף* (sequence) של***n***  איברים, ומשתנה שעובר על כל אחד מהם, לפי הסדר. בלוק הפקודות יתבצע ***n*** פעמים, כמספר אברי הרצף. | 1.מהו סכום המספרים הבא:  1 + 3 + 5 + … + 99  Nseq = range(1,100,2)  seqSum = 0  for i in Nseq:  seqSum += i  print('1 + 3 + 5 + … + 99 = ', seqSum) |
| **עוד על for**  כאמור, הפעולה רצה על רשימה. נבחין בין ריצה על איברי הרשימה עצמה לבין ריצה על האינדקס של איברי הרשימה. | בהינתן רשימה – זו ריצה על איברי הרשימה.  ***>>> animals = ['dog','cat','bird','snake']***  ***>>> for a in animals:***  ***print(a)***    ***dog***  ***cat***  ***bird***  ***snake***  זו ריצה על האינדקס של איברי הרשימה.  ***>>> for c in range(len(animals)):***  ***print(animals[c])***    ***dog***  ***cat***  ***bird***  ***snake***  תוצאת הריצה זהה. |
| יצירת לולאת FOR שתרוץ כמות פעמים לפי בחירה שלנו. | for i in range(3):  print('a')   #a #a #a |
| **לולאות מקוננות:** for בתוך for | דוגמא 1-  for k in range(1,11):  for m in range(1,k+1):  print(k\*m,end=' ')  print()  שימו לב לפלט:  1  2 4  3 6 9  4 8 12 16  5 10 15 20 25  6 12 18 24 30 36  7 14 21 28 35 42 49  8 16 24 32 40 48 56 64  9 18 27 36 45 54 63 72 81  10 20 30 40 50 60 70 80 90 100  \*אם הרשימה בנויה מרשימות מקוננות מאותו אורך, ניתן לרוץ על איברי הרשימות המקוננות כך:  ***>>> mylist = [[23,45],[78,19],['aaa','bbb'],[True,False]]***  ***>>> for a,b in mylist:***  ***print(a+b)***    ***68***  ***97***  ***aaabbb***  ***1***  דוגמא 2-  גישה לאיברים של tuple בתוך רשימה:  רשימות של tuples:  for i in birthdays:  for j in i:  print (j)  print('\*\*\*\*\*\*\*')  [4, 7, 2]  19  3  1968  \*\*\*\*\*\*\*  6  7  2011  \*\*\*\*\*\*\*  13  1  1972  \*\*\*\*\*\*\*  8  7  2005  \*\*\*\*\*\*\*  28  7  2003  \*\*\*\*\*\*\* |
| If | **ביצוע מותנה:** הוא התניה לפיו תבוצע או לא תבוצע פעולה.  הביטוי הבוליאני המופיע אחרי מילת המפתח if בדוגמה, נקרא התנאי. אם ורק אם התנאי מתקיים (כלומר ערך הביטוי הוא True), מקטע הקוד המוזח שמופיע מתחת לשורת ה-if מתבצע.  מבנה כללי:  **if** condition:  **statement**(s)  else:  statement(s) | דוגמא 1  **grade = int(input('what is your grade? '))**  **if grade > 60:**  **print('You passed')**  **else:**  **print('You failed')**  דוגמא 2  \*אם בלוק ההוראות מסתכם בהוראה אחת**.**  **if x>y : x,y = y,x** |
| **קינון הוראות if**  מבנה:  **if condition1:**  **statement(s)**  ***if condition2:***  ***statement(s)***  ***else:***  ***statement(s***) | grade = int(input('what is your grade? '))  if grade > 60:  print('You passed')  if grade >= 85:  print('You are also an excellent student')  else:  print('You failed') |
| else | במקרה שבו נירצה להפריד את התנאי לשתי קבוצות של פקודות נשתמש בelse, אם התנאי הראשון (if) הוא true מתבצעת הראשונה, ואם ערך התנאי הוא false, השנייה.  Else תפריד בין שתי הקבוצות. |  |
| **Elif**  בהוראת if אחת יכולות להופיע אחת אחרי השנייה מספר שרירותי של elif.  משתמשים במבנה הזה של הוראת if כשבבעיה הנדונה ניתן לזהות יותר משני מקרים שונים, שכל אחד מהם דורש טיפול (או פתרון) יחודי משלו, שונה מהאחרים.  הפתרון של כל אחד מהמקרים האלה יתבטא כבלוק של הוראות לביצוע.  (למשל, אם לכל ערך אפשרי של משתנה מסוים יש צורך לבצע רצף מסוים של הוראות, נצטרך לכתוב בלוק הוראות מתאים לכל אחד מערכי המשתנה. לכן, אנו נהיה מעוניינים לדעת מהו הערך הנוכחי של אותו משתנה על מנת לדעת איזה בלוק חייב להתבצע.) | my\_list = eval(input(**"enter a list: "**)) s = 0 **for** r **in** my\_list:  **if** isinstance(r,bool):  s += 0  **elif** isinstance(r,int):  s += r  **elif** isinstance(r,float):  s += r  **elif** isinstance(r,range):  s += sum(r)  **elif** isinstance(r,list):  s += sum(r)  print(**'the sum is'**,s) |
| **else של לולאות**  מבנה else קיים גם עבור לולאות. ההוראות שבמבנה ה-else מבוצעות רק אם הלולאה סיימה את עבודתה בצורה "נורמאלית", ולא על ידי הפעלת מנגנון break. | for x in [1,2,3]:  print( x)  else:  print( 'all the list elements were printed')  הפלט יהיה:  1  2  3  all the list elements were printed |
| break | הפסקת האיטרציה הנוכחית של הלולאה ויציאה ממנה.  נשים לב שלפני הוראה זו תבוא בד"כ הוראת תנאי שנועדה לפקח על ביצוע הוראה זו. | \*הדוגמא היא ממאטלב. |
| continue | פירושה של הוראת continue: הפסקת האיטרציה הנוכחית של הלולאה וקפיצה חזרה להתחלת איטרציה חדשה. | סכום המספרים: 12 + 22 + 42 + 52  ***my\_sum = 0***  ***for n in range(1,6):***  ***# skip 3***  ***if c == 3: continue***  ***my\_sum += n\*n***  ***print(my\_sum)*** |
| while | מבנה ההוראה:  ***while condition:***  ***statements***  חוזרת ומבצעת קבוצת פקודות, כל עוד ערך ביטוי התנאי שלה הוא true.  יש לוודא שבשלב כלשהו, הוראות הלולאה יגרמו לתנאי להפוך ל-False אחרת נוצרת לולאה אינסופית.  (באופן כללי, נשתמש בלולאת for כאשר ידועה רשימת הערכים עליהם צריך לעבוד. בלולאת while נשתמש כאשר אין לנו רשימה כזו וביצוע ההוראות מוגבל בתנאי קיום של הלולאה.) | דוגמה שימושית, לולאת while לבדיקת קלט והבטחת קבלת קלט תקין.  ***getNum = True***  ***while getNum:***  ***n = int(input("enter an integer even number between 4 and 20: "))***  ***if 4 <= n <= 20 and n%2 == 0:***  ***getNum = False*** |
|  | **עוד על While**  ניתן להשתמש ביצירת ערך משתנה אשר סופר את הפעמים שהלולאה רצה מחדש. | import random number = random.randint(1,100) guess = eval(input('guess a number between 1 and 100')) c=0 while guess != number:  c+=1  if number > guess:  guess = eval (input("My number is bigger, try agian "))  else:  guess = eval (input("My number is smaller, try agian "))  if guess == number:  print('you guessed it! the number is', number)  print('number of tries', c) |

While

while condition

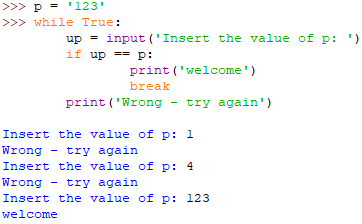
פקודת continue:

מאפשרת לדלג על ריצה אחת של הלולאה. דוגמא – הדפסת מספרים אי זוגיים:

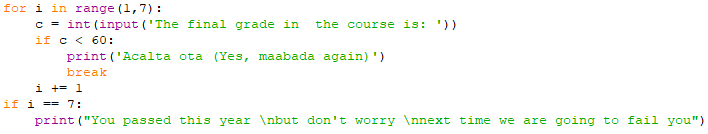


פקודת break:

מאפשרת לדלג על הלולאה בשלמותה ולעזוב אותה. דוגמא – קליטת קוד, ויציאה כשהוא נכון:

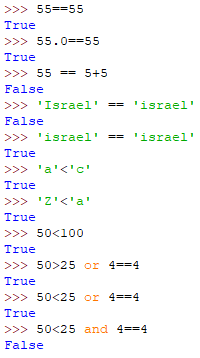


ניתן גם בלולאת for. דוגמא: נניח שיש לנו שישה קורסים לעבור, והאוניברסיטה החליטה להיות נבזית יותר מבדרך כלל, כך שמספיק שנכשלת בקורס אחד ואתה לא עובר את הסמסטר ונאלץ לחזור על השנה כולה. אבל יש catch – אתה יכול להיכשל בקורס מסוים ולעבור, אם תעזור לאוניברסיטה לכתוב קטע קוד שיגיד לכל תלמיד אם הוא עובר או לא (ולפי איך שהאפליקציה של האוניברסיטה עובדת, הגיוני שהם יצטרכו עזרה בכך). מכיוון שמספיק שנכשלת פעם אחת, ניתן לצאת מהלולאה באמצע:



|  |  |
| --- | --- |
|  |  |
| (מה יקרה בפועל) |  |

ביטויים בוליאניים:



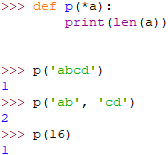
משתנים גלובליים

משתנה חיצוני לפונקציה, שכדי להשתמש בו נדרש להפעיל מתודה global (בניגוד למשתנים מקומיים המוגדרים בתוך הפונקציה).

|  |  |
| --- | --- |
| משתנה מקומי | משתנה גלובלי |
|  |  |

פונקציה עם מספר לא ידוע של פרמטרים

חוקי, להלן דוגמא:



טיפול בקבצים

ניתן ליצור, לקרוא ולכתוב קבצים בפייתון. בשביל ליצור קשר עם קובץ קיים, או לפתח קובץ חדש, נשתמש בפונקציה open(filepath, mode).

filepath הוא מחרוזת עם הכתובת של הקובץ במחשב, או שם הקובץ אם הקובץ נמצא בתיקייה בה נמצא קובץ הפייתון עליו אנו עובדים, יחד עם הסיומת המציינת מהו סוג הקובץ.

mode הוא פרמטר המציין מה יהיה אופן הפעולה עם הקובץ, ומקבל אחד מהערכים הבאים:

|  |  |
| --- | --- |
| "r" קריאה | "r+" קריאה וכתיבה |
| "w" כתיבה | a"" כתיבה בזנבו של קובץ קיים, או יצירת קובץ חדש אם לא קיים |

mode אופציונלי, ברירת המחדל היא "r". עבור mode w אם הקובץ שנפתח היה קיים, הוא יימחק ויוצר קובץ חדש במקומו באותו השם.

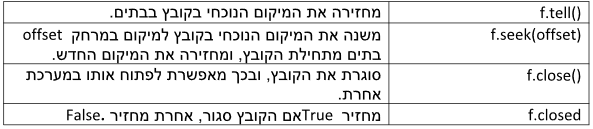
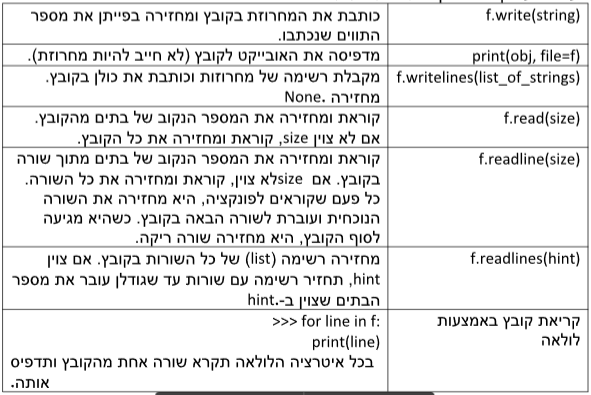
דוגמא לאיך נראית שורה (בפועל הרצתה כשלעצמה תיתן שגיאה):



השיטות הבאות יעזרו לנו לעבוד עם קבצים:

***שיטות לקריאה מקובץ***

|  |  |  |
| --- | --- | --- |
| file.read([size]) | השיטה מחזירה מחרוזת ובה כל תוכן הקובץ כולל סימון לסוף שורה. ללא הפרמטר size, נקרא כל הקובץ אחרת, נקרא (בערך) מספר התווים שנקבע ב-size | לדוגמא:  ***s1=f.read(10)***  ***s=f.read()*** |
| file.readlines() | השיטה מחזירה רשימה בה כל איבר הוא מחרוזת המכילה שורה מהקובץ. בסוף כל שורה, יופיע הסימון לסוף שורה ("n\"). | ***ls=f.readlines()*** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |



# פונקציות (functions)

## **סיבות לכתיבת פונקציות**

1) מאפשר לתת שם לקבוצת הוראות ובכך הופך את התכנית למובנת יותר.

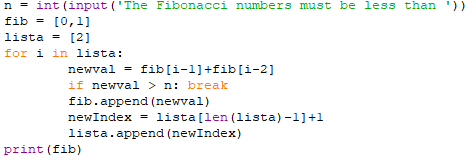
2) מאפשר שימוש חוזר בקוד.

3) מקל על תחזוקת תכניות.

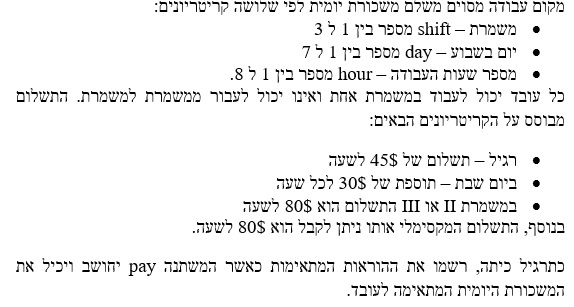
|  |  |  |
| --- | --- | --- |
|  | **מבנה פונקציה המוגדרת ע"י המשתמש:** כותרת הפונקציה מתחילה במילה השמורה def אחריה שם הפונקציה וסוגריים עגולים ובתוכם שמות לפרמטרים שהפונקציה מצפה לקבל מופרדים בפסיקים. במידה והפונקציה אינה מצפה לקבל פרמטרים, הסוגריים יהיו ריקים. אחריהם יופיעו נקודתיים. | סינטקס:  def function\_name(param1, param2, ...) :  statement\_1   statement\_2     .... |
| פונקציה יכולה להחזיר ערך באמצעות ההוראה return. כשמתבצעת הוראת return, הפונקציה מסתיימת והוראות הכתובות לאחר הוראה זו, אינן מתבצעות. | ***def mySqrt(x):***  ***return x\*\*0.5***  ***### Call the function***  ***print (mySqrt(9))***  ***x= mySqrt(49)*** |
|  |  |
|  |  |
|  |  |  |
|  |  |
|  |  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

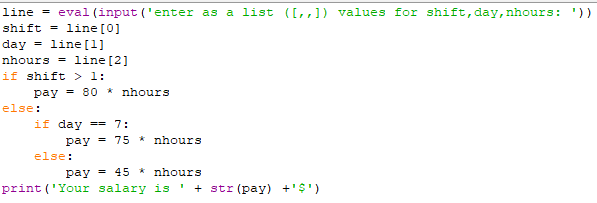
דוגמאות

סדרת פיבונאצ'י:









|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר ועצרתו: | | | | קינון לולאות: |
| פונקציה שבודקת האם מספר ראשוני ומחזירה אמת אם כן ושקר אחרת. | | | |  |
| החלפת האיבר 20 הראשון שמופיעה ברשימה ב12: | | | |
| פונקציה המקבלת מספר זוגי גדול משתיים, ומוצאת שני מספרים ראשוניים שסכומם שווה לו, על בסיס הפונקציה הקודמת. | | | |  |
| פונקציה המחזירה אמת אם כל איברי tuple שווים ושקר אחרת | | | |
| תוכנית המקבלת string ארוך ומדפיסה רשימת שכיחויות ממוינת של כל האותיות | | | |  |
| עוד דרך להיפוך רשימה, מחרוזת וכו': | | | |
| מספרים חברים – זוג מספרים שונים שסכום המחלקים של אחד שווה לשני ולהיפך. התוכנית תמצא זוגות נוספים עד שתקבל הוראה לעצור. | | | |  |
| החלפת משתנים: | | | |
| קליטת מספר:  קליטת מחרוזת | | | |
| פתרון משוואה ריבועית | | | |  |
| קליטת מספר טבעי וחישוב סכום ספרותיו. | | |  | |
| התוכנית קולטת מספר טבעי ורושמת את כל המספרים הראשוניים הקטנים ממנו או שווים לו (אלגוריתם של ארטוסטנס) | |  | | |
| תוכנית המקבלת רשימה של חברים ומתי הם באו לבקר, והשליש בחודש בו התרחשו הכי הרבה ביקורים. |  | | | |
| התוכנית מקבלת רשימה של ערכים שמספרם לא קבוע ולא ידוע. התוכנית תצמיד לכל ערך שיכול להיות מפתח (hashable) ערך שאינו יכול להיות מפתח. אם לא ניתן ליצור זוג כי נגמרו הערכים ו/או המפתחות, יש לסיים את יצירת המילון. בסיום יש להדפיס זוגות של המילון בצורה מסודרת: מפתח וערך. | | |  | |

