# ל"ע – טיפוסי נתונים

במהלך הל"ע אתם הולכים ללמוד על הטיפוסים השונים שקיימים בפייתון, חלק מהנושאים הועברו בחוברות ההטרמה ומצויינים פה לשם חזרה וחידוד.

כל משתנה בפייתון הוא טיפוס נתונים מסוג מסויים, כלומר כל משתנה מכיל מידע שמסוגל להשתנות תוך כדי הריצה אבל **סוג המידע** בכל משתנה הוא אחר. למשל, משתנה מסוג מסויים ישמור ערך מספרי ומשתנה מסוג אחר ישמור ערך מחרוזתי. על כל טיפוס נתונים נוכל לבצע פעולות מסויימות. למשל, ניתן לחבר שני מספרים אחד עם השני אך בלתי אפשרי לחבר בין מספר לבין ערך מחרוזתי (מילה למשל).

חשוב להבין שבשפות שונות קיימים טיפוסי נתונים שונים, וכן צורת העבודה מול כל טיפוס נתונים משתנה בהתאם לשפה. בל"ע זה נעבור על טיפוסי הנתונים ודרכי העבודה איתם בשפת פייתון.

**טיפוסי נתונים בפייתון**

**מספרים שלמים**

מספרים שלמים, integer אוint , הם הסוג הראשון שנלמד עליו.   
מספר שלם הוא כל מספר שאינו שבר, בין אם חיובי או שלילי. לדוגמה : 0 , 1 , 5 , 12-.



**type() –** פונקציה בפיתון המקבלת פרמטר אחד.   
**type** מחזירה את סוג טיפוס הנתונים שמכיל המשתנה. בדוגמא, number מכיל מספר שלם כאמור, int, ולכן **type** של numberמחזירה לנו int''.  
כאשר נרצה לוודא מהו טיפוס הנתונים של משתנה מסוים נשתמש בפונקציה type.

הבדלים בין גרסאות פייתון

בפייתון 2.7 ל-**int** הייתה מגבלה אחת, והיא גודל המספר שאותו הוא מסוגל להכיל. וזאת תלוי במספר הסיביות (ביטים) שבהם תומך המחשב עליו רצה המערכת.   
כך שבמערכות 32-ביט, המספר הגדול ביותר שint- מסוגל להכיל הוא 2,147,483,647. כאן נחלץ לעזרתנו טיפוס נתונים הנקרא **Long**. בפייתון 2.7 **Long** הוא טיפוס נתונים המסוגל להכיל כל מספר שלם ללא מגבלה.

**דוגמאות מפייתון 2.7**



אם היינו רוצים ליצור טיפוס נתונים מסוג long שערכו ההתחלתי נמצא בטווח ש-int יכול להכיל נעשה את הדבר הבא:



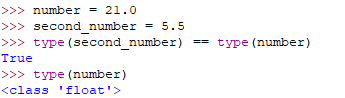
שימו לב שבשפות מסוימות ל-**Long** יש גודל מקסימלי.

**בגרסאות 3 ומעלה**

טיפוס הנתונים long אינו קיים יותר בפייתון בגרסאות החדשות שלה, ונעשית הפרדה רק בין מספרים שלמים (בכל הגדלים) לבין מספרים עשרוניים.

**מספרים עשרוניים**

גם מספרים עשרוניים מיוצגים בפייתון, אך בדרך שונה ממספרים שלמים.   
על מנת ליצור מספר עשרוני נעשה זאת באמצעות נקודה(.) מספר עשרוני בפיתון נקרא  **float.**

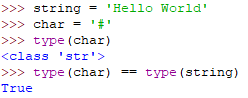


המושג **float** מגיע מ-**floating-point number**. כלומר, "נקודה צפה" זאת מפני שבמחשבים שונים מספר הספרות אחר הנקודה עשוי להשתנות בעקבות החומרה השונה ולכן נקרא כך.

**מחרוזות**

מחרוזת היא רצף תווים (לדוגמא: hello). מחרוזות יכולות להכיל צירופי תווים שונים.

המחרוזת הקצרה ביותר היא מחרוזת ללא תווים כלל (ריקה).   
המחרוזת הארוכה ביותר תמלא את כל הזיכרון הפנוי (למרות שלא באמת נצליח לעשות אחת כזאת).   
המחרוזות הנפוצות ביותר הן מחרוזות המכילות את אותיות האלפבית (a-z, A-Z) הספרות (0-9) והסמנים (+,-,!, \*, (, ), % וכו').



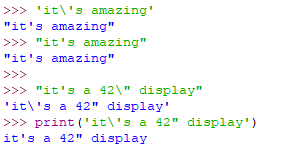
על מנת להגיד מחרוזת אנחנו צריכים לפתוח אותה באחד משלושת הסימנים הבאים:

מרכאות – "

גרש – '

או מרכאות משולשות – """

בין גרש למרכאות אין כל כך הבדל אבל היתרון הוא שלפעמים נרצה להשתמש בגרש בתוך המחרוזת עצמה כלומר בתוך משפט וזאת אפשר לעשות בכמה דרכים :



נוכל להשתמש ב Backslash "\" על מנת לציין לפיתון שהתו הבא הולך להיות חלק מהמחרוזת. דרך נוספת היא פשוט להגדיר את המחרוזת בעזרת מרכאות ואז פיתון יודעת שעד שהיא לא תפגוש עוד מרכאות המחרוזת לא מסתיימת.

במקרה השני שהוא קצת יותר מסובך היינו צריכים את 2 סוגי המרכאות ולכן פה כבר פיתון מציינת לנו גם כן על ידי "\" מתי מגיע תו שיודפס כרגיל. אך על מנת לראות את המחרוזת בצורה התקינה נצטרך להשתמש בפעולת הדפסה.

לאחר ה - Backslash יכולים לבוא כל מני תווים שבצירוף ה - Backslash יש להם משמעות מיוחדת.

להלן טבלה המכילה חלק מצירופי התוויים עם Backslash ומשמעותם:

|  |  |
| --- | --- |
| **המשמעות** | **הרצף** |
| רד שורה | \n |
| Tab | \t |
| Backslash | \\ |
| גרש | \' |

**פונקציות על מחרוזות**

* **index & find –**



Index ו find הן פונקציות די זהות. הן מקבלות פרמטר אחד ומחזירות את המיקום הראשון שלו במחרוזת, כלומר באיזה אינדקס הערך הזה נמצא בפעם הראשונה. מה ששונה בין 2 הפונקציות הללו הוא שבמקרה ש-find לא תמצא איזשהו ערך היא פשוט תחזיר לנו מ 1- . לעומת זאת Index תזרוק לנו שגיאה.

* **len –**

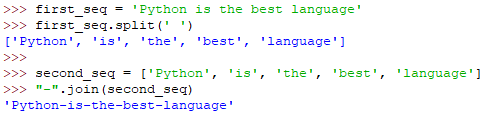
הפונקציה len מקבלת פרמטר אחד שהוא מחרוזת(בהמשך נראה שlen יודעת לעבוד עם כל טיפוס נתונים שהוא מסוג רצף) ומחזירה את אורכה.

* **split & join**

Split ו join הן מתודות מאוד מיוחדות שבעזרתן יהיה לנו הרבה יותר קל לפצל ולחבר איברי רצף שונים.

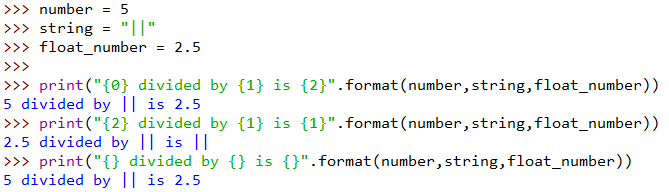
**split** –פועלת על מחרוזת ומקבלת כפרמטר תו. הפונקציה מפצלת את המחרוזת כך שהתו שהיא קבלה הוא זה שיפריד את המחרוזת. בדוגמא למטה שמנו כפרמטר את המחרוזת " "(כלומר רווח) ולכן הפונקציה החזירה לנו רשימה, שבה כל איבר הוא מה שהיה בין 2 רווחים. בשפה פשוטה יותר לפצל את המחרוזת על ידי הרווחים

**join** –היא בדיוק הפוכה מsplit.join פועלת על מחרוזת וגם מקבלת כפרמטר טיפוס רצף.join תיקח את טיפוס הרצף שאותו היא קיבלה כפרמטר ותוסיף לו בין כל איבר ואיבר את המחרוזת.



**פרמוט מחרוזות**

יהיו מקרים שבהם נרצה להדפיס כמה טיפוסי נתונים שונים בהדפסה אחת. בשפות אחרות סביר להניח שפשוט היינו משתמשים ב + בין האיברים השונים ומדפיסים (שרשור). בפייתון אנחנו נשתמש בformat .

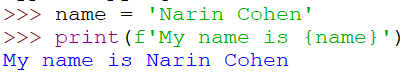


על מנת להשתמש בformat אנחנו נצטרף לפתוח סוגריים מסולסלים "{}" במקומות שבהם נרצה להכניס את האובייקטים השונים. כל סוגריים כאלו יכילו אובייקט אחר שנרצה להכניס לתוך המחרוזת. לאחר המחרוזת שאותה אנו רוצים לפרמט נשים את format ואחריו את כל האובייקטים שאותם אנו רוצים להכניס אל המחרוזת.

נהוג לסמן בתוך הסוגריים 0 ,1 ,2 וכך הלאה בהתאם למיקום הפרמטרים.

בסוגריים המסולסלים שבתוכם 0 יכנס הפרמטר הראשון ובסוגריים המסולסלים שבתוכם 1 יכנס הפרמטר השני וכך הלאה... מפני שבדוגמא אנו מכניסים את הפרמטרים לפי סדר אין סיבה למספר.

בפייתון 3 התווספה דרך חדש לפרמט מחרוזות. ניתן לשים את האות f לפני המחרוזות, ובמקום להשתמש בפונקציה format ניתן לשים את המשתנים אותם נרצה לפרמט ישר בתוך הסוגריים המסולסלות כך:



מומלץ לעבוד כך.

דרך ישנה יותר לפירמוט מחרוזות **שאינה** **בשימוש** היום היא דרך פירמוט המחרוזות שהייתה בשימוש בשפת C:



**טיפוסים מתקדמים**

**List**

רשימה היא טיפוס נתונים מיוחד. רשימה מסוגלת להכיל כמה איברים יחד והיא מאפשרת לנו לגשת לכל איבר בתוכה. רשימה בפיתון , בניגוד לרשימות אחרות ברוב השפות, מאפשרת לרשימה לשמור את כל סוגי הנתונים בכל איבר של הרשימה :



כך נגדיר רשימה פשוטה. בדוגמא הראשונה הכנסנו לרשימה רק מספרים אבל אפשר לראות שהרשימה יודעת להתמודד עם כל סוג איבר שנכניס אליה גם אם האיברים ביניהם אינם זהים כלל. כמובן שאיברי הרשימה מסודרים אחד אחרי השני בסדר קבוע. האיבר הראשון נמצא במקום 0 וכך הלאה.

כך ניתן לגשת לאיבר ספציפי ברשימה:



נוכל לגשת לכל איבר לפי האינדקס (index = מיקום) שלו ברשימה. אם יש לנו איבר שמכיל רצף (sequence) כמו רשימה או מחרוזת נוכל לגשת גם לכל איבר שלהם בנפרד (בהתאם לדוגמאות הקודמות):



האיבר באינדקס 4 הוא רשימה. כאשר נרצה לגשת לאיבר מתוך הרשימה נפתח עוד סוגריים מרובעים ופיתון תדע שאנחנו רוצים לגשת לאיבר בתוך האיבר.

גם מחרוזות הן איבר רצף , ולכן גם בהן אנו יכולים לגשת לאיבר פנימי ספציפי.

כאשר ננסה לגשת לאיבר שלא קיים ברשימה כמובן שתקפוץ לנו שגיאה:



**הכנסת\שינוי איברים ברשימה**

לאחר שיצרנו את הרשימה אנו יכולים להכניס אליה איברים בשתי דרכים שונות :



בעזרת הפונקציה **append** נכניס איבר לסוף הרשימה כלומר אחרי האיבר האחרון. אך, אם נרצה להכניס את האיבר למקום ספציפי ולא בהכרח לסוף, נוכל לעשות זאת בעזרת **insert**, שמקבלת גם אינדקס שאליו יכנס האיבר ואת האיבר עצמו. Insert תדחוף את כל שאר האיברים ברשימה צעד אחד קדימה.

אז אם אני רוצה לשנות איבר ברשימה ולא להוסיף ? נעשה זאת כך :



**Len**

את len הכרנו כבר במחרוזות, כמובן שהיא עובדת גם פה. נוכל לדעת כמה איברים יש לנו ברשימה:



**מציאת איבר ברשימה**

פיתון מאפשר לנו למצוא האם איבר נמצא ברשימה שלנו בצורה מאוד פשוטה ונוחה :



בעזרת **in** אנחנו יכולים לבדוק אילו איברים קיימים לנו ברשימה. במקום לרוץ על הרשימה ולבדוק in עושה לנו את החיים קלים. ביטוי שנכתוב בעזרת in יחזיר לנו true אם האיבר קיים ו-false אם לא. כמובן שנוכל להכניס את הביטוי לתנאי ובכך להשתמש בו בתכנית. גם in עובדת על כל אובייקט רצף ולכן תעבוד גם על מחרוזות.

**הסרת איבר מהרשימה**

יש כמה דרכים להסיר איברים מהרשימה :

1)**pop**



בעזרת **pop** נשלוף את האיבר האחרון ברשימה וpop גם תחזיר לנו אותו.

אופציה נוספת היא להכניס לpop אינדקס של איבר כפרמטר וכך pop תמחק את האיבר לפי אינדקס שניתן.

2)**remove**

**remove** מקבלת ערך מסוים ומחפשת אותו ברשימה. הערך הראשון שתואם לפרמטר שניתן היא תמחק אותו. remove לא מחזירה כלום.

3)**del**

****

בעזרת **del** אנחנו יכולים למחוק אובייקטים מהזיכרון.

del מאפשרת לנו למחוק גם איבר ספציפי של איבריי רצף מהזיכרון.

(ניתן להשתמש בdel עם או בלי הסוגריים)

**Slicing**

פיתון מאפשר לנו עבודה מאוד מאוד נוחה עם אובייקטים מסוג רצף. ונותנת לנו לעבוד איתם באמצעות slicing. בעזרת slicing אני אוכל לחתוך את האובייקט שלי בצורה קלה נוחה ומיוחדת. לדוגמה, ניצור רשימה של מספרים בעזרת range ונעשה עלייה חיתוכים:

את החיתוך נעשה בעזרת סוגרים וביניהם B:A

A יהיה המיקום שממנו נרצה להתחיל את החיתוך ו B יהיה המיקום שבו יסתיים החיתוך כמובן שנוכל לוותר על אחד מהם ואז החיתוך יהיה מהתחלה או עד הסוף. אופציה נוספת שיש לנו עם slicing היא לחתוך את האובייקט עם קפיצות על אינדקסים כך :

הוספנו עוד נקודתיים ואחריהם יגיע מספר הקפיצות שנעשה בין איבר לאיבר. בדוגמא , התחלנו באיזה הראשון כי לא ציינו כלום , עד האיבר האחרון כי גם בו לא ציינו כלום ובקפיצות של 3.

**תרגול :**

**תגדירו רשימה עם range(20) מצאו בה את הפרטים הבאים בעזרת חיתוכים:**

* **כל האיברים הזוגיים.**
* **כל האיברים האי-זוגיים.**
* **הצגה של האיברים בקפיצות של 3.**
* **האיבר האחרון ברשימה (2 דרכים).**
* **הרשימה בסדר הפוך.**
* **כל האיברים עד האיבר ה-6.**

**Mutable & Immutable**

בפייתון, המשתנים שלנו לא באמת מכילים את האובייקט שאותו אנו מיישמים להם, כל משתנה מכיל מצביע לזיכרון שבו האובייקט שמור.אז מה שיש לנו כרגע זה את a שהוא מצביע בזיכרון למקום שבו המספר 400 שמור וכך גם b מכיל מצביע לזיכרון למקום שבו המספר 415 שמור.

|  |  |
| --- | --- |
| 400 |  |
| 415 |  |

|  |  |
| --- | --- |
| a |  |
| b |  |

בפיתון, יש לנו 2 סוגים שונים של משתנים והם ה-mutables וה-immutables.  
בואו נבין קודם אילו משתנים שייכים לכל קבוצה ולאחר מכן מה ההבדל בין הקבוצות הללו.

המשתנים: int, float, tuple, string הם משתנים הנחשבים "מורכבים" ומכונים **immutable**.  
המשתנים: list, set, dict מכונים **mutable**.  
  
(נלמד על set ו tuple בהמשך הקובץ ועל dict בהמשך המקצוע)

**ההבדל העיקרי בין 2 הסוגים הנ"ל הוא שמרגע ההשמה (הגדרת המשתנה) לזיכרון, immutable - לא מסוגל להשתנות לעולם, ואילו mutable הוא דינאמי ומסוגל להשתנות בזיכרון בכל רגע נתון.**

על מנת להבין את הנושא הזה יותר טוב, נשתמש במתודות id ו is .

**id –** פונקציה המציגה לנו את המיקום בזיכרון אליו מצביע המצביע. המיקום יהיה מוצג כמספר. אותו מספר הוא לא באמת המיקום של האובייקט בram אלא הוא פשוט דרך ייצוג של פייתון.

**is –** אופרטור שמשווה בין 2 id-ים של משתנים ומחזיר True במידה והם שווים ו False במידה והם לא.



במקרה הזה יש לנו את a ו b ששניהם immutable כי הם מכילים מספרים. כמובן שהשניים לא מצביעים לאותו מיקום בזיכרון מסיבה שהם מכילים ערכים שונים ולכן כל אחד מצביע למקום אחר.

אז מה קרה פה? אמרנו ש immutableלעולם לא משתנים.  
תחילה, הגדרנו ל a את המספר 25 ובדקנו מה המקום שלו בזיכרון.  
לאחר מכן הוספנו 4 ל 25. במקום שהערך השמור במיקום בזיכרון שאליו מצביע a פשוט ישתנה מ25 ל29, פייתון יוצרת מיקום חדש בזיכרון אליו היא משימה 29 ומשנה גם את המצביע של a אליו.

**זהו העיקרון המרכזי ביותר ב immutables.**  
נראה מה קורה עם הקבוצה השנייה - mutables:

במקרה הזה יצרנו רשימה, שבניגוד למספרים (int) שייכת ל mutables ולכן היא מסוגלת להשתנות במיקום שלה בזיכרון. כמו שניתן לראות גם לאחר ששינינו אותה היא עדיין נמצאת באותו מקום בזיכרון

|  |
| --- |
| a |
| b |

|  |
| --- |
| 3 |
|  |

על מנת להבין את המקרה הזה, נעזר בסרטוט. תחילה השמנו 3 ל a ואמרנו ל b = a. בפועל זה מה שקרה :

כרגע גם a וגם b מצביעים על 3 ולא b מצביע על a שמבציע על 3.  
ברגע שהגדרנו ש b=a בעצם אמרנו לפייתון שb יצביע אל המיקום בזיכרון שa מצביע אליו (תזכורת: המשתנה לא "מכיל" את הערך, הוא מספק הצבעה אל המיקום בו הערך שמור)  
עכשיו מפני שאמרנו שמספרים הם immutable לאחר שהוספנו 2 ל a , אז כמו בדוגמה למעלה פייתון תפתח מיקום חדש בזיכרון ותכניס אליו 5 (כלומר 2 ועוד 3).  
בנוסף לכך ישנה הצבעה של a אל מיקום הזיכרון החדש. מכיוון שb- מצביע על המקום הקודם אליו הצביע a ולא על a עצמו, b לא ישתנה וזה יראה כך :

|  |
| --- |
| a |
| b |

|  |
| --- |
| 3 |
| 5 |

כמובן אם נעשה את אותו הדבר לרשימה גם a וגם b ישתנו :

זאת קורה בדיוק בגלל מה שאמרנו שרשימה היא Mutable והיא מסוגלת להשתנות כל הזמן אז במקרה הזה a מצביעה על רשימה מסיומת ועלייה מצביע גם b וכשאנו משנים אותה גם a וגם b משתנים מפני שהם מצביעים על אותה רשימה שמשתנה.

יש כאן אבל בעיה קטנה, מה יקרה כשאני ארצה להעתיק רשימה? או כל איבר אחר שהוא mutable? בשביל זה נשתמש ב slicing שלמדנו קודם לכן. אמרנו ש slicingיודעת להחזיר לנו **עותק** של רשימה עם חיתוכים שונים :



אם נשתמש בslicing מהמקום הראשון עד לאחרון הכוונה כך [:] למעשה נעתיק את הרשימה במלואה ונוכל להכניס אותו למשתנה אחר.

**Tuple**

Tuple הוא טיפוס נתונים שמאוד מזכיר רשימה אך בניגוד לרשימה, tuple לא יכול  
להשתנות לעולם. זאת אומרת מרגע שהגדרנו אותו לא נוכל לעשות בו שינויים (immutable).  
על מנת להגדיר tuple נשתמש בסוגריים עגולים () בצורה הבאה:



העבודה עם tuples היא זהה כמו עם רשימות. כמובן שאין לנו את כלל הפונקציות של רשימה אך עדיין נוכל להשתמש בslicing ובפונקציות המובנות של tuple שהם index ו count. count מקבל פרמטר וכפלט מחזיר כמה פעמים הפרמטר שהכנסנו נמצא בtuple וindex יחזיר לנו את הindex הראשון של הערך שהוא מקבל כפרמטר.

בנוסף ניתן לחבר tuple-ים בדיוק כמו עם רשימות :

**Tuple Assignment**

כאשר מבצעים השמה בין 2 tuple-ים פייתון מבצעת השמה אוטומטית בין כל 2 איברים תואמים כך:

בגלל שפייתון מכירה את השיטה הזו לא נהיה חייבים להשתמש בסוגריים.



**Set**

**Set** (קבוצה) דומה ל**רשימה** (list), מלבד העובדה ש**לא** ניתן לאחסן **פעמיים** את אותו האיבר. Set הוא טיפוס **שאינו מסודר** – כלומר, איבר שאנחנו מכניסים אליו לאו דווקא יכנס לSet בתור האיבר האחרון.

1. קראו ב-**help** על הפונקציות הבאות של הטיפוס Set ו**התנסו** בהן:
   * Add.
   * Discard.
   * Remove. (מה ההבדל בין Discard ל-Remove?)
   * Clear.
   * Pop.
   * Difference. (שימו לב – הפונקציה מקבלת Set אחד או יותר, בתור קלט)
   * Intersection.
   * Union.
   * קיימות כמובן קיימות פונקציות נוספות, תוכלו לקרוא גם עליהן.
2. אם נרצה, נוכל גם ליצור set מרשימה או מ-Tuple **קיים**, וכך לקבל את אותה רשימה, **ללא** כפילויות:



**תרגול**

##### 1. קלוט 10 מספרים ממשתמש והדפס אותם בסדר הפוך לקליטה.

##### 2. הגדר שתי רשימות בעלות 10 תאים. השתמש בלולאה על מנת לבדוק האם הרשימות זהות או לא. במידה ונמצא תא אחד שונה, נפסיק מיד את הבדיקה, נודיע שהרשימות לא זהות, ונדפיס באיזה אינדקס עצרנו (קראו על else בלולאות for).

##### 3. הגדר רשימה בעלת 10 איברים מספריים. הדפס את סכום האיברים של הרשימה, ממוצע הערכים, הערך המקסימאלי והערך המינימאלי.

##### 4. הגדר שתי רשימות בגודל 10 איברים, חבר את ערך התאים בשתי הרשימות לתוך רשימה שלישית לפי הindex בהתאם (התא הראשון ברשימת התוצאה יכיל את התא הראשון ברשימה הראשונה + התא הראשון ברשימה השנייה).

##### 5. הגדר רשימה באורך 10 תאים המכילה ציונים. מצא את התאים שהערך בהם נמוך מממוצע הציונים שברשימה (ניתן להניח כי אין ציונים זהים) והדפס אותם.

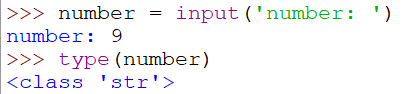
##### 6. נתונה רשימה בגודל 20 איברים. החלף את כל האיברים במקום האי-זוגי במחרוזת "XXX".

**שאלת בונוס:**

צור רשימה שתכיל כתובות ip ותבדוק אם יש תקשורת אל אותן כתובות. תדפיס "Yes" אם כן ו""No אם לא לצד כתובת הip.

**המרת טיפוסי נתונים**

קיימים מספר סוגי מידע שניתן להתייחס אליהם כמספר טיפוסי נתונים שונים. למשל, ניתן להתייחס למספר בתור ערך מחרוזתי, ערך מספרי שלם (int) או ערך מספרי חלקי (float). למשל, כל קלט שנקבל מהמשתמש (בעזרת הפונקציה input) תמיד יהיה מחרוזתי (string), גם אם המידע בו הוא מספרי למשל:



לכן, נרצה **להמיר** את טיפוס הנתונים לטיפוס נתונים אחר, כדי שנוכל להתייחס למידע אחרת. למשל, אם נרצה לבצע חישוב על מספר שמתקבל כקלט מהמשתמש נצטרך להמיר את הקלט שבצורה דיפולטית תמיד יהיה מחרוזתי (string) לערך מספרי מתאים (int). כדי לעשות זאת קיימות בפייתון מספר פונקציות שממירות טיפוס נתונים אחד לאחר:

* **Str** – ממירה לערך מחרוזתי. ניתן להמיר כמעט כל טיפוס נתונים למחרוזת.
* **Int** – ממירה לערך מספרי שלם. ניתן להמיר רק ערך המכיל מספר בלבד (ניתן להמיר גם מספר עם נקודה עשרונית). לרוב נשתמש בפונקציה זו כדי להמיר קלט מהמשתמש לערך מספרי.
* **Float** – ממירה לערך מספרי עם נקודה עשרונית. ניתן להמיר רק ערך המכיל מספר בלבד (ניתן להמיר גם מספר שלם).
* **List** – ממירה לרשימה. ניתן להמיר רק tuple ואובייקטים ספורים נוספים (למשל range). אם נמיר מחרוזת נקבל רשימה בה כל תו במחרוזת יהיה איבר נפרד.
* **Tuple** – ממירה לtuple. ניתן להמיר רק רשימה ואובייקטים ספורים נוספים (למשל range). אם נמיר מחרוזת נקבל tuple בה כל תו במחרוזת יהיה איבר נפרד.
* **Set** – ממירה לset. ניתן להמיר רשימה וtuple.

קיימות מספר פונקציות נוספות לטיפוסי נתונים מורכבים יותר (למשל מילון, שנלמד בהמשך המקצוע).

חשוב לשים לב לטיפוס הנתונים איתו אנו עובדים ולהמיר לטיפוס הנתונים המתאים במידת הצורך.

**תרגול**

כתוב תוכנית המקבלת מהמשתמש את גילו כמספר עשרוני כאשר המספר השלם מייצג את גילו בשנים והמספר אחרי הנקודה בחודשים. הדפס בצורה מילולית את גילו בשנים ובחודשים.

קלוט מהמשתמש רשימה של מספרים מופרדים בעזרת רווח, לאחר מכן הכנס אותם לרשימה תוך המרה לטיפוס הנתונים המתאים. כלומר אם המספר הוא עשרוני המספר יומר לfloat ואם הוא מספר שלם המספר יומר לint.

כתוב תוכנית שמקבלת מהמשתמש שני זמנים בפורמט של שעה:דקה (12:00 למשל), וחשב את ההפרש בינהם בשעות ובדקות. (ניתן להניח שהשעה שנקלטה ראשונה מגיעה לפני השעה השנייה שנקלטה ביום)