Python

TomerBu

נושאים:

ריבוי ארגומנטים לפונקציות
חריגות
מבני נתונים

קלט של רשימה:

```
def input_of_an_array():
    word = input("Enter a word: ")

    lst = []
    for letter in word:
        lst.append(letter)

    return lst

def input_array_of_numbers():
  input_string = ""
  lst = []
  while input_string != ".":
    input_string = int(input("Enter a number or type '.' to finish: "))
    if input_string != ".":
      lst.append(input_string)


def input_of_an_array_v2():
    word = input("Enter a word: ")
    return [letter for letter in word]

def input_of_an_array_v3():
    return list(input("Enter a word: "))
```

קלט של מחרוזת עם מפריד:

"3,4,5,6,7"

הפונקציה split:
מחלקת מחרוזת לרשימה לפי מפריד:

```
def input_array_of_numbers():
    lst = []
    while True:
        num = input("Enter a number: ")
        if num == "done":
            break
        lst.append(int(num))
    return lst



def input_array_of_numbers_v2():
    sequence = input("Enter a sequence of numbers separated by comma: ").strip()
    lst = sequence.split(",")
    return [int(num) for num in lst]
```

הדפסה של לוח הכפל:

```
def mult_table():
    for row in range(1,11):
        for col in range(1,11):
            print(row * col, end="\t")
        print("\n")

mult_table()
```

לולאה פנימית שמדפיסה עמודות

ירידת שורה

רשימה של רשימות: (לוח הכפל)

```
def mult_table_lists():
    mult_table = []
    for row in range(1,11):
        mult_table_row = []
        for col in range(1,11):
            mult_table_row.append(row * col)
        mult_table.append(mult_table_row)
    return mult_table
```

במקום להדפיס:
מייצרים רשימה וממלאים אותה עם ערכי העמודות

במקום ירידת שורה:
נכניס את הרשימה לרשימה-של-רשימות

הדפסה של רשימה דו מימדית

```
def print_2d_array(two_d_array):
    for row in two_d_array:
        for col in row:
            print(col, end="\t")
        print("\n")
```

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

פונקציות: מאגר של ידע ופונקציונליות

```
from string import ascii_lowercase, ascii_uppercase, digits, punctuation
from random import choice

def random_lower():
    return choice(ascii_lowercase)

def random_upper():
    return choice(ascii_uppercase)

def random_digit():
    return choice(digits)

def random_punctuation():
    return choice(punctuation)

def get_letter(message = "Enter a single letter: "):
   #loop until we get a single letter
   while True:
       char = input(message).strip()
       if len(char) == 1 and char.isalpha():
           return char
       print("Invalid input, please try again")
```

*args
פונקציה יכולה לקבל מספר משתנה של ארגומנטים:

```
def cool_print(*words):
    for word in words:
        print("❄️" + word + "❄️")

cool_print("elsa", 'anna', 'olaf') # ❄️elsa❄️ ❄️anna❄️ ❄️olaf❄️


print()
```

הפונקציה מקבלת את words
כאוסף של מילים (אפשר לרוץ על המילים בלולאה)

הפונקציה יכולה לקבל כמה מחרוזות שנרצה בפרמטר words

*args
פונקציה יכולה לקבל מספר משתנה של ארגומנטים:

```
def my_sum(*numbers):
    return sum(numbers)

my_sum(1,2,3,4,5) # 15

print(min(100, 20, 40, 1, 30)) # 1
```

מתי נרצה להשתמש בargs:

*פונקציות שמקבלות מספר לא ידוע של מספרים:
כפל של כל המספרים
חיבור של כל המספרים
מציאת האיבר הקטן ביותר
מציאת האיבר הגדול ביותר
מיון של אוסף מספרים

מתי נרצה להשתמש בargs:

*פונקציות שמקבלות קלט לא ידוע להדפסה:
הדפסה של טקסט ללוג (קונסולה)
הדפסה של טקסט לקובץ (file)

בSQL יהיה נוח לקבל values לinsert
שיהיו *args

כתבו את הפונקציות הבאות:

1. פונקציה שמקבלת args של מספרים:
הפונקציה תחזיר את הסכום של כל המספרים

2. פונקציה שמקבלת args של מספרים:
הפונקציה תחזיר את המספר הקטן ביותר

3. פונקציה שמקבלת args של מספרים:
הפונקציה תחזיר את המספר הגדול ביותר

4. פונקציה שמקבלת args של מספרים:
הפונקציה תחזיר את ההפרש הכי גדול בין המספרים

קיימות פונקציות מובנות:

sum, min, max

אחרי ההפסקה נפתור ונציג Tuple

אשמח לפתרונות בצ׳אט כדי לדעת איפה אתם נמצאים
ולתת לכם טיפים

פתרון:

```
def my_sum(*numbers):
    return sum(numbers)

def my_min(*numbers):
    return min(numbers)

def my_max(*numbers):
    return max(numbers)

def max_diff(*numbers):
    return max(numbers) - min(numbers)
```

Tuple:

```
def stats(*numbers):
    return sum(numbers), max(numbers), min(numbers), len(numbers), sum(numbers) / len(numbers)


result = stats(100, 20, 30, 40, 50)

print("sum", result[0])
print("max", result[1])
print("min", result[2])
print("count", result[3])
print("average", result[4])
```

אפשר להחזיר Tuple מפונקציה
וכך להחזיר הרבה תוצאות מהפונקציה.

Tuple מבנה נתונים מסודר לפי אינדקס
בדומה ל list

ההבדל הוא ש tuple הוא immutable
כלומר אחרי שהוא נוצר לא ניתן לשנות את הערכים שלו.

לא ניתן להוסיף לשנות או למחוק איברים אחרי יצירתו

Tuple:

```
m_tuple = (1, 2, True, 4, 5)
print(m_tuple)
print(m_tuple[0])


tuple_from_list = tuple([1, 2, 3, 4, 5])

m_tuple_with_single_element = (1,)
m_tuple_with_single_element = (1) # this is not a tuple

# We cannot change the value of a tuple
m_tuple[0] = 100 # this will throw an error
```

Tuple:

```
tuple_1 = (1, 2, 3)
tuple_2 = (4, 5, 6)

tuple_3 = tuple_1 + tuple_2 # (1, 2, 3, 4, 5, 6)


# tuple unpacking
tuple_demo = ("John", "Doe", 30)

first_name, last_name, age = tuple_demo
print(first_name, last_name, age)
```

חיבור של tuples:

פיזור של tuple למשתנים בודדים:

השוואה בין tuple לרשימה:

יצירה:	קריאה:	כתיבה:	פירוק:
lst = [1, 2, 3]	lst[0]	lst[0] = 3	first = lst[0]
tup = (1, 2, 3)	tup[0]	~~tup[0]~~	last = lst[1]
			first, last = tup

2 הבדלים מהותיים:

*יעילות - tuple יותר יעיל ומהיר

*לא ניתן לשינוי = Immutable

*** מי שקורא את הקוד - מבין שהערך מיועד להיות קבוע.**

אם נרצה לשקף read_only

tuple לא ניתן לשינוי

Prefer immutable:

ככל שאפשר - נעדיף להשתמש בקבועים:

ארגון קוד כזה:

ישקף את כוונת המתכנת - שערך מסוים לא ניתן לשנות.
ישפר את היעילות
ישפר קריאות - ברור מה אמור להשתנות ומה לא.

3_data_structures.py U ×

3_data_structures.py > ...

```
(parameter) numbers: tuple
```

```python
1  def stats(*numbers):
2      return sum(numbers), max(numbers), min(numbers)
```

פונקציה שמקבלת args למעשה מקבלת tuple

Tuple:
כמו רשימה אבל בלי פעולות של שינוי:

```
data = ('John', 'Doe', 30)
data.
```

- count
- index

פחות פעולות - אבל יותר יעיל
יותר מובן - יותר קל להבנה

```
data = ('John', 'Doe', 30)

first_name, last_name, age = data
```

Tuple:
כמו רשימה אבל בלי פעולות של שינוי:

```
def gas_prices():
    return (7.1, 7.2, 8.0)

benzin, diesel, octan = gas_prices()


def card_suits():
    return ("❤️", "♦️", "♠️", "♣️")
```

מבני נתונים:

:tuple

מבנה נתונים מסודר לפי **אינדקס** - לא ניתן לעריכה

יעילות חיפוש לפי אינדקס

:list

מבנה נתונים מסודר לפי **אינדקס** - ניתן לעריכה

יעילות חיפוש לפי אינדקס

:set

מבנה נתונים לא ממוין - ניתן לעריכה - מונע כפילויות

מונע כפילויות, פעולות על קבוצות

:dictionary

מבנה נתונים מסודר לפי מפתח - ניתן לעריכה
אוסף של key-values שניתן לעריכה

יעילות חיפוש לפי מפתח

הכנסה והסרה וחיפוש ביעילות O(1)

```
{
    "name": "moe" ,
    "age": 30
}
```

בשונה מJS
יש להשתמש ב"" למפתחות

מילונים ישמשו אותנו בעבודה עם JSON

מבני נתונים:

set:

מבנה נתונים שמונע כפילויות:

יש להגריל 3 זוכים **שונים** להגרלה

עוזר לסנן רשימות - כדי למצוא ערכים יחודיים.

יעילות חיפוש לפי אינדקס

עוזר ליצור אוספים - כשכל איבר שונה מהשני.

סט של מיילים יחודיים

מונע כפילויות

יעילות חיפוש לפי מפתח

מיני תרגיל:

```
# יש להגריל 3 מספרים שונים בין 1 ל10

(מבלי לחזור על אותו מספר) רנדום עלול להגריל את אותו המספר פעמיים

from random import randint
```

נסו לחשוב כמה פעולות יבוצעו

מי שסיים - כדאי לשמור בבנק הפונקציות:
פונקציה שמקבלת פרמטר - כמה מספרים רנדומליים רוצים
ואת הטווח

ומחזירה מספרים רנדומליים **שונים** בטווח הרצוי.

מיני תרגיל:

```
# יש להגריל 3 מספרים שונים בין 1 ל10

from random import randint

def distinct_random_numbers(n, start = 0, end = 10):
    winners = []

    while len(winners) < n: #o(n*n)
        num = randint(start, end)
        if num not in winners:
            winners.append(num)
    return winners
```

קיים פתרון ביעילות של O(n)

קבוצה - Set
מבנה נתונים שמונע כפילויות:

```
def distinct_random_numbers(n, start = 0, end = 10):
    winners = set()

    while len(winners) < n: #o(n)
        num = randint(start, end)
        winners.add(num)
    return winners
```

בעית חיפוש **ברשימה**:

נניח שמבקשים למצוא את האות e ברשימה

חיפוש יצרוך n פעולות:

כלומר O(n)

0	1	2	3
'a'	'b'	'a'	'd'

קבוצה - Set
מבנה נתונים שמונע כפילויות:

בעית חיפוש **ברשימה**:

נניח שמבקשים למצוא את האות e ברשימה

חיפוש יצרוך n פעולות:

כלומר O(n)

0	1	2	3
'a'	'b'	'a'	'd'

סט יוצר אינדקס מראש לכל הערכים.

כשאנו מחפשים אות מסוימת
החיפוש ידרוש פעולה אחת

החיפוש ביעילות O(1)

'a'	'b'	'd'

קבוצה - Set
מבנה נתונים שמונע כפילויות:
יצירת Set

```
# creating sets:

# create empty set:
s1 = set() # new empty set

# create set with data - removes duplicates:
s2 = {1, 1, 4, 4, 5} # {1, 4, 5}

# creating sets from list:
my_list = [1, 1, 4, 4, 5] # [1, 1, 4, 4, 5]
s3 = set(my_list) # {1, 4, 5}

# creating sets from string:
s4 = set('hello') # {'h', 'e', 'l', 'o'}
```

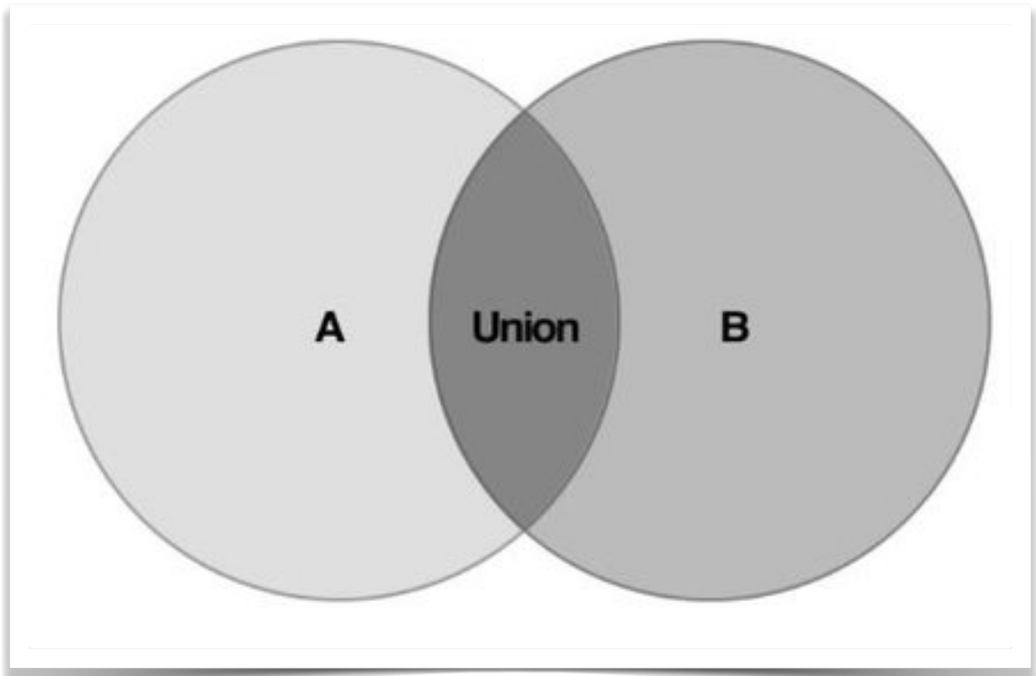
קבוצה - Set
מבנה נתונים שמונע כפילויות:
פעולות:

```
s = {1, 2, 3, 4, 5}

s.add(6) # {1, 2, 3, 4, 5, 6}
s.add(6) # {1, 2, 3, 4, 5, 6} # no change
s.update({7, 8, 9}) # {1, 2, 3, 4, 5, 6, 7, 8, 9}

# can't remove element that not exists (error)

if 7 in s:
    s.remove(7) # remove raises KeyError if item does not exist:


#does not raise errors:
s.discard(7) # {1, 2, 3, 4, 5, 6} # no error
```

קבוצה - Set
מבנה נתונים שמונע כפילויות:
פעולת איחוד:

שחקנים ששיחקו בסרט אמריקן

שחקנים ששיחקו בסרט באטמן

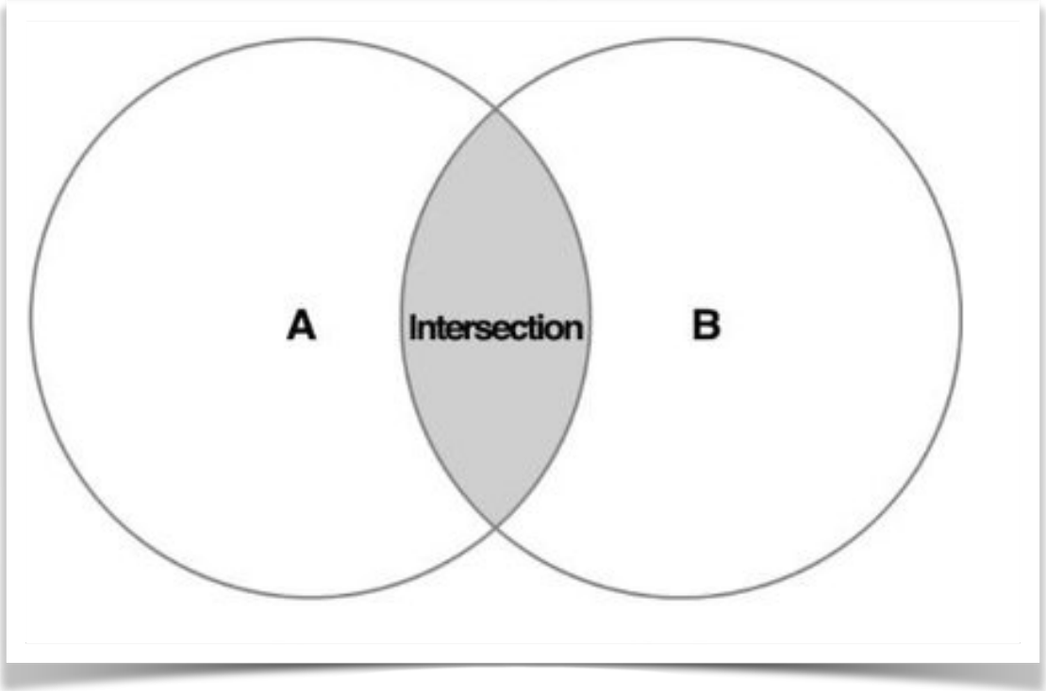



```
batman = {'Christian Bale', 'Michael Keaton', 'Ben Affleck'}
american = {'Christian Bale', 'Ben Affleck', 'Tom Hanks'}

# union
union = batman.union(american) # {'Christian Bale', 'Michael Keaton', 'Ben Affleck', 'Tom Hanks'}
union = batman|american # {'Christian Bale', 'Michael Keaton', 'Ben Affleck', 'Tom Hanks'}
```

קבוצה - Set
מבנה נתונים שמונע כפילויות:
פעולת חיתוך:

שחקנים ששיחקו
בסרט אמריקן

שחקנים ששיחקו
בסרט באטמן



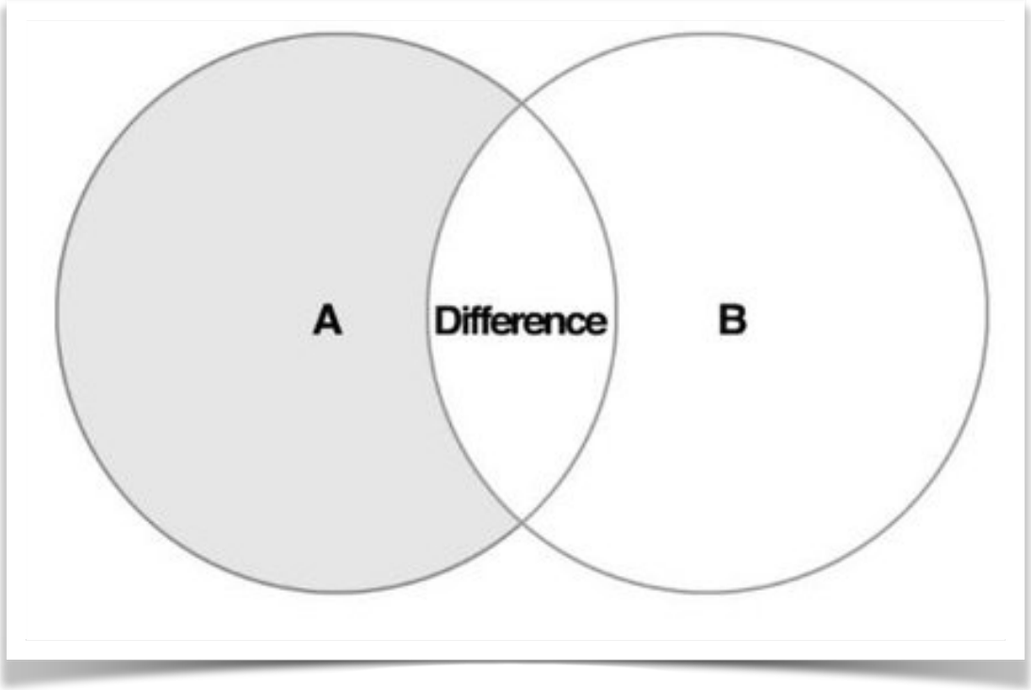
חיתוך - אילו שחקנים
שיחקו ב-2 הסרטים
(אלמנטים שקיימים ב-2 הקבוצות)

```
batman = {'Christian Bale', 'Michael Keaton', 'Ben Affleck'}
american = {'Christian Bale', 'Ben Affleck', 'Tom Hanks'}

# intersection
intersection = batman.intersection(american) # {'Christian Bale', 'Ben Affleck'}
intersection = batman & american # {'Christian Bale', 'Ben Affleck'}
```

קבוצה - Set
מבנה נתונים שמונע כפילויות:
פעולות:

שחקנים ששיחקו בסרט אמריקן

שחקנים ששיחקו בסרט באטמן



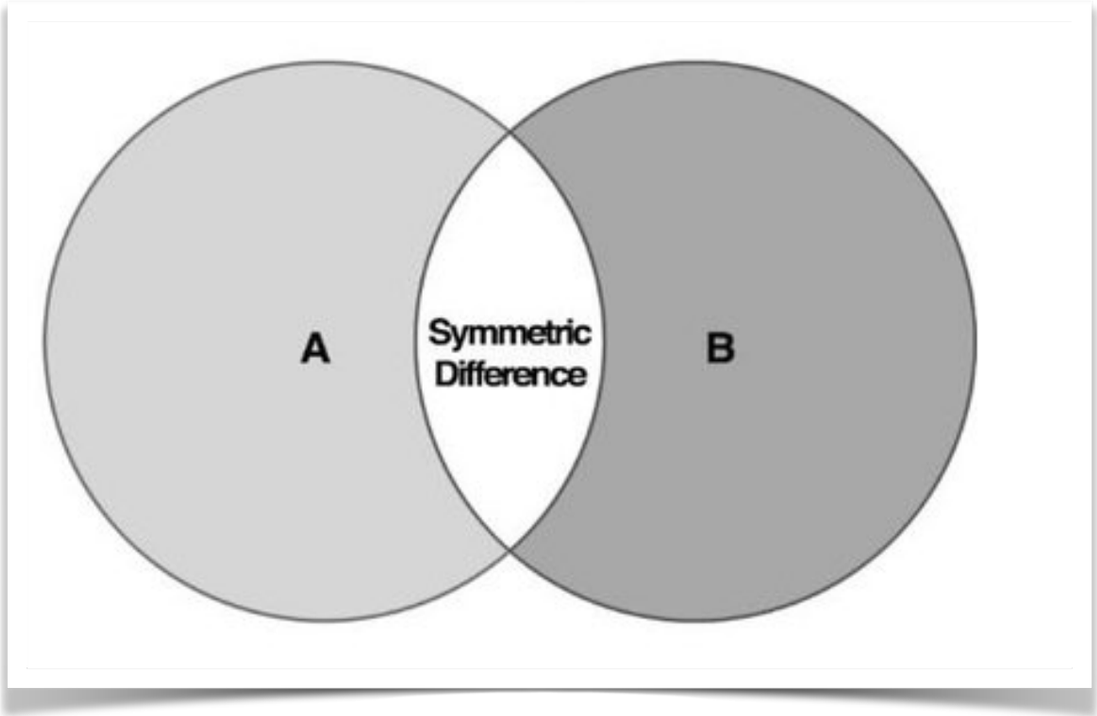
הפרש - שחקנים ששיחקו רק בסרט באטמן ולא שיחקו באמריקן
(אלמנטים שקיימים רק בקבוצה השניה ולא בראשונה)

```
batman = {'Christian Bale', 'Michael Keaton', 'Ben Affleck'}
american = {'Christian Bale', 'Ben Affleck', 'Tom Hanks'}

# difference
difference = batman.difference(american) # {'Michael Keaton'}
difference = batman - american # {'Michael Keaton'}
```

קבוצה - Set
מבנה נתונים שמונע כפילויות:
פעולות:

שחקנים ששיחקו בסרט אמריקן

שחקנים ששיחקו בסרט באטמן



הפרש סימטרי- שחקנים ששיחקו רק בסרט באטמן או רק בסרט אמריקן
אבל לא בשניהם

```
batman = {'Christian Bale', 'Michael Keaton', 'Ben Affleck'}
american = {'Christian Bale', 'Ben Affleck', 'Tom Hanks'}

# symmetric difference
symmetric_difference = batman.symmetric_difference(american) # {'Michael Keaton', 'Tom Hanks'}
symmetric_difference = batman ^ american # {'Michael Keaton', 'Tom Hanks'}
```

קבוצה - Set
המרה של קבוצה לרשימה
המרה של רשימה לקבוצה

נוח לנו לעבוד עם רשימות:

בפרוייקט מסוים נקבל רשימה כפרמטר:

נרצה להסיר כפילויות ולחזור לעבוד עם רשימה.

המרה של set לרשימה:

```
# convert set to list
actors_list = list(batman ) # ['Christian Bale', 'Michael Keaton', 'Ben Affleck']
actors_list.append('Ben Affleck')


given_list = [2, 2, 4, 4, 3, 4, 7, 8, 10]
# convert list to set
as_set = set(given_list) # {2, 3, 4, 7, 8, 10}


# remove duplicates from list:
lst = list(set(given_list)) # [2, 3, 4, 7, 8, 10]
```

המרה של רשימה לset:
(מסיר כפילויות)

הסרה של כפילויות מרשימה:
הופכים רשימה לסט -> ואז שוב לרשימה
(בדרך הוסרו הכפילויות)

מילונים:

יצירת מילון חדש:

```
# new dictionary:
dict1 = {}
dict2 = dict()
dict_3 = {
    "name": "moe" ,
    "age": 30
}
dict_4 = dict(name="moe", age=30)
```

גישה לאיברים לפי מפתח:

```
batman_dict = {
    "Dark Knight": "Christian Bale",
    "Batman": "Michael Keaton",
    "Batman v Superman": "Ben Affleck"
}

# get value by key:
print(batman_dict["Dark Knight"]) # Christian Bale

#print(batman_dict["Darkest Knight"]) # Raise KeyError

# get value by key with get method: avoid KeyError
print(batman_dict.get("Darkest Knight")) # None
```

מילונים:

עדכון והסרה של איברים:

```
counter_dictionary = {
    "a": 3,
    "b": 1,
    "n": 2
}

#update a value:
counter_dictionary["a"] = 2

# add a value:
counter_dictionary["j"] = 1

# remove a value:
del counter_dictionary["j"]
ac = counter_dictionary.pop("a")
counter_dictionary.popitem() # remove last item
counter_dictionary.clear() # remove all items
```

מילונים:

לולאות/איטרציות

```
harry_potter_books = {
    "Philosopher's Stone": 1997,
    "Chamber of Secrets": 1998,
    "Prisoner of Azkaban": 1999,
    "Goblet of Fire": 2000
}

for key in harry_potter_books.keys():
    print(key)

for value in harry_potter_books.values():
    print(value) # 1997, 1998, 1999, 2000

for key, value in harry_potter_books.items():
    print(key, value)
```

מפתחות

ערכים

מפתחות וערכים

מילונים:

```
person_dict = {
    "name": "moe",
    "age": 30
}

# check if key exists:
if "name" in person_dict:
    print(person_dict["name"])

if "birthday" not in person_dict:
    print("birthday not exists")
```

Exceptions

```python
# Basic try/except block
try:
    x = 1 / 0 # This will raise a ZeroDivisionError
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Exceptions

אפשר להשתמש בבלוק של except מספר פעמים אם יש טיפול שונה בכל שגיאה:

```
# Catching multiple exceptions
try:
    num = int(input("Enter a number: ")) #value error if not a number
    result = 10 / num # ZeroDivisionError if num is 0
except ValueError:
    print("You must enter a number.")
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Exceptions

תפיסת מספר שגיאות באותו except:

```
# Catching multiple exceptions with a single block
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except (ValueError, ZeroDivisionError):
    print("Invalid input.")
```

שימושי במקרה שאנו מטפלים
ב2 השגיאות באופן זהה

Exceptions

שימוש באובייקט של השגיאה המקורי

```
try:
    num = int(input("Enter a number: "))
except ValueError as e:
    print("That's not a number!")
    print(e) # prints the error message
```

טוב למטרות הבנה של השגיאה
פירוט נוסף לגבי השגיאה

לא נציג את זה למשתמש הקצה

Exceptions

תפיסה של כל השגיאות **Exception**

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except Exception as e:
    print("Something went wrong.")
    print(e)
```

מומלץ להבין את השגיאות האפשריות
ולטפל בהן - הוצאת הודעה מתאימה ללקוח קצה

תפיסה של כל השגיאות Exception

טוב להוצאת שגיאות לא ידועות
שגיאות לא צפויות ללוג

Exceptions

Exception תפיסה של כל השגיאות

```
# Catching all exceptions:
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ValueError:
    print("Value error - You must enter a number.")
except ZeroDivisionError:
    print("ZeroDivisionError - You can't divide by zero.")
except Exception as e:
    print("Something went wrong.")
    print(e)
```

Exception תפיסה של כל השגיאות
אם יש שגיאות שלא צפינו מראש

מומלץ להבין את השגיאות האפשריות
ולטפל בהן - הוצאת הודעה מתאימה ללקוח קצה

תפיסה של כל השגיאות Exception

טוב להוצאת שגיאות לא ידועות
שגיאות לא צפויות ללוג

Exceptions

בלוק finally:
ירוץ בכל מקרה - גם אם הtry הצליח וגם אם הtry נכשל:

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ValueError:
    print("Value error - You must enter a number.")
except ZeroDivisionError:
    print("ZeroDivisionError - You can't divide by zero.")
except Exception as e:
    print("Something went wrong.")
    print(e)
finally:
    print("This will always execute.")
```

Exceptions

זריקת שגיאה - raise

```
def draw_square(length):
    if length <= 0:
        raise ValueError("Length must be greater than 0.")
    for _ in range(length):
        print("*" * length)

draw_square(-5)
```

Exceptions

תרגיל:

נרצה לכתוב פונקציה שיודעת להתמודד עם קלט לא תקין:

אם הקלט לא תקין - נבקש מהמשתמש להזין שוב.

פתרון עם רקורסיה:

```
def get_number(message = "Enter a number: "):
    try:
        num = int(input(message))
        return num
    except ValueError:
        print("Invalid input.")
        return get_number(message)


print(get_number())
```

פתרון יעיל יותר עם לולאות בעמוד הבא:

Exceptions

תרגיל:

נרצה לכתוב פונקציה שיודעת להתמודד עם קלט לא תקין:

אם הקלט לא תקין - נבקש מהמשתמש להזין שוב.

פתרון עם רקורסיה:

```
def get_number(message = "Enter a number: "):
    while True:
        try:
            num = int(input(message))
            return num
        except ValueError:
            print("Invalid input")

print(get_number())
```

תרגילים:

1

כתבו פונקציה שמקבלת קלט של מספר עשרוני:
ומתמודדת עם שגיאות.

2

"אנא הזן מספר בין 1 ל4 בלבד"

כתבו פונקציה שמקבלת קלט של מספר שלם:
הפונקציה תקבל 3 פרמטרים:
1) הודעה לבקשת קלט
2) מספר מינימום
3) מספר מקסימום

ומתמודדת עם שגיאות.

```
def get_number(message = "Enter a number: ", start = 1, end = 4):
    while True:
        try:
            num = int(input(message))
            #check if the number is within the range
            return num
        except ValueError:
            print("Invalid input")
```

רמזים לתרגיל 2

רק אם הקלט תקין
נחזיר ערך
אחרת נדפיס הזן שוב
והלולאה תדאג להריץ שוב

שיעורי בית:

עבור הקבוצות הבאות:

```
country_A = {"Python", "Machine Learning", "High-Speed Rail", "Startups", "Cloud Computing"}
country_B = {"Python", "Cybersecurity", "High-Speed Rail", "Renewable Energy", "Blockchain"}
```

יש להדפיס:

1) את החיתוך בין 2 הקבוצות
2) את האיחוד בין 2 הקבוצות
3) את ההפרש בין 2 הקבוצות
4) את ההפרש הסימטרי בין 2 הקבוצות

שיעורי בית:

עבור המילון הבא:

```
students = {
    'Elsa': [90, 85, 95],
    'Anna': [89, 79, 84],
    'Olaf': [100, 100, 100]
}
```

יש להדפיס:

1) את כל השמות של הסטודנטים (keys)
2) את כל הערכים של הציונים (values)
3) את השמות ואת הערכים ביחד (items)
4) לכל סטודנט יש להדפיס את ממוצע הציונים
5) הוסיפו את הסטודנט hans ורשימת ציונים [60, 60, 60]
6) הוסיפו את הסטודנט כריסטוף ומחקו את הסטודנט hans

כמה אותיות יש במילה:

```
word = "supercalifragilisticexpialidocious"

כמה פעמים מופיעה האות e?
כמה אותיות מרכיבות את המילה?
כמה אותיות מרכיבות את המילה ללא חזרות?
```

שיעורי בית:

עבור הרשימה הבאה:

```
numbers = [1, 2, 3, 2, 4, 1, 5, 6, 6, 7]
```

יש להדפיס:

1) את כל האיברים הייחודיים

2)*** את כל האיברים שמופיעים פעם אחת בלבד (לולאות)

3)*** עבור כל מספר יש לספור כמה פעמים הוא הופיע (לולאות)

התרגילים 2 ו3 מאתגרים ומצריכים חשיבה

ש״ב: רשימת קריאה מומלצת

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set