

# Python

TomerBu

# **נושאים:**

**מיליוןים**

**טיפול בשגיאות**

**מלחמות וירושה**

**ריבוי ארגומנטים לפונקציה עם kwargs**

**עבודה עם ספריות חיצונית**

# תוכנית לקורס:

JS

python

Node

Django

בשיעור הבא:

Express

Django Rest Framework

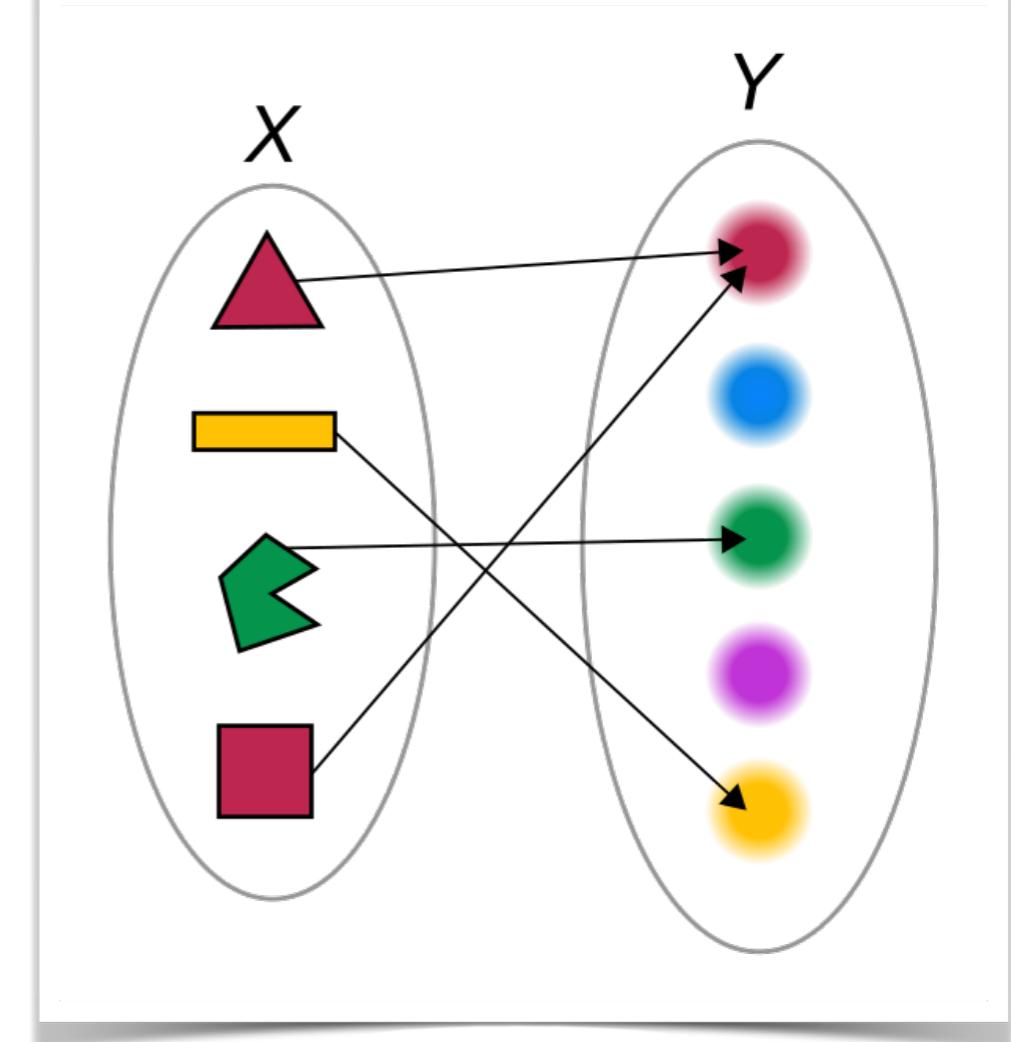
# מילונים

```
# new dictionary  
dict1 = {}  
dict2 = dict()  
  
# must use "" for keys  
dict3 = {  
    "name": "dave",  
    "age": 20  
}  
  
dict4 = dict(name="dave", age=20)
```

מילון חדש וריק:

מילון חדש עם ערכים:

מילון: מבנה נתונים  
זוגות של key, value  
אין חשיבות לסדר  
והמפתחות ייחודיים



# מילונים

מילון: מבנה נתונים  
זוגות של key, value  
אין חשיבות לסדר  
והמפתחות ייחודיים

```
# must use "" for keys
dict3 = {
    "name":"dave",
    "age":20
}
```

גישה לערכים במילון:

```
# read values from dict:
print(dict3["name"]) #dave
print(dict3.get("age"))#20
```

גישה למפתח שלא קיים במילון:

```
# access non-existing value:
print(dict3.get("id"))#None
print(dict3["id"])#raises KeyError
```

לדוגמא שם אמצעי - לא תמיד קיים במילון  
מילון יכול להגיע מממד-נתונים ללא שם אמצעי

# מילונים

מילון: מבנה נתונים  
זוגות של key, value  
אין חשיבות לסדר  
והמפתחות ייחודיים

```
# must use "" for keys
dict3 = {
    "name":"dave",
    "age":20
}
```

גישה לערכים במילון:

```
# read values from dict:
print(dict3["name"]) #dave
print(dict3.get("age"))#20
```

גישה למפתח שלא קיים במילון:

```
# access non-existing value:
print(dict3.get("id"))#None
print(dict3["id"])#raises KeyError
```

לדוגמא שם אמצעי - לא תמיד קיים במילון  
מילון יכול להגיע מממד-נתונים ללא שם אמצעי

# מילונים

```
# update and delete:  
counter_dictionary = {  
    "a": 3,  
    "b": 1,  
    "c": 2  
}  
  
# update a value:  
counter_dictionary["b"] = 2  
  
# add a value:  
counter_dictionary["d"] = 1  
  
# remove a value:  
del counter_dictionary["d"]  
a = counter_dictionary.pop("a")  
  
# remove a value from the end:  
last_item = counter_dictionary.popitem()  
# remove all items:  
counter_dictionary.clear()
```

פעולות עדכון ומחיקה:

# מילונים

ספרו כמה פעמים מופיעה כל אות בטקסט הבא:

```
text = """  
Mary had a little lamb, little lamb, little lamb.  
Mary had a little lamb, its fleece was white as snow.  
And everywhere that Mary went. Mary went.  
"""
```

פתרו בעזרת מילונים ולא בעזרת Counter מובנה

```
for letter in 'abcdefg':  
    counter_dict[letter] = 0  
  
    ↓  
  
counter_dictionary = {  
    "a": 0,  
    "b": 0,  
    "c": 0  
}  
}
```

```
for letter in text:  
    if letter.is_alpha  
        dict[letter] += 1
```

# מילונים

בנייה של מילון שמורכב מאותיות באנגלית כמפתח  
והערך של כל אות הוא אפס:

```
from string import ascii_lowercase as az  
  
counter_dict = {}  
  
million_new_rick:  
    for letter in az:  
        counter_dict[letter] = 0  
        abc  
    print(counter_dict)
```

# מילונים

```
from string import ascii_lowercase as az
counter_dict = {}

for letter in az:
    counter_dict[letter] = 0

text = """
    Mary had a little lamb, little lamb, little lamb.
    Mary had a little lamb, its fleece was white as snow.
    And everywhere that Mary went. Mary went.
"""

text_copy = text.lower().strip()

for letter in text_copy:
    if letter in az:
        counter_dict[letter] += 1

print(counter_dict)
```

בנייה של מילון שמורכב מאותיות באנגלית כמפתח  
והערך של כל אות הוא אפס:

## מילונים: מפתחות, ערכים, :items

```
harry_potter_books = {  
    "Philosopher's Stone": 1997,  
    "Chamber of Secrets": 1998,  
    "Prisoner of Azkaban": 1999,  
    "Goblet of Fire": 2000,  
    "Order of the Phoenix": 2003,  
    "Half-Blood Prince": 2005,  
    "Deathly Hallows": 2007  
}  
  
# dictionary keys:  
for k in harry_potter_books.keys():      מפתחות  
    print(k)  
  
# dictionary values:  
for v in harry_potter_books.values():      ערכים  
    print(v) # 1997  
  
# dictionary values:                      מפתחות+ערכים  
for name, year in harry_potter_books.items():  
    print(name, ":", year)  
  
# dictionary values:  
for name, year in harry_potter_books.items():  
    print(name, ":", year, sep="-", end="\n")
```

אפשר להשתמש בfsring - מעולה!  
שימוש מורחב בפונקציה :print

## מילונים: בדיקה האם מפתח קיים במלון:

```
person1 = {  
    "name": "Harry",  
    "age": 11,  
}  
  
# check if key exists in the dict:  
  
if "name" in person1:  
    print(person1["name"])  
  
if "birthday" not in person1:  
    print("birthday not set")
```

מילונים שימושיים מאוד גם לJSON שmagick מ��ע

# טיפול בשגיאות

## Exception

מילות מפתח expect raise

The screenshot shows a code editor window with two tabs: "3\_exceptions.py" and "3\_exceptions.py > ...". The code in the editor is:

```
1 # נאיבי (חסר טיפול בשגיאות):  
2 x = int(input("enter a number: "))
```

The line "x = int(input("enter a number: "))" is highlighted in red, indicating a syntax error. The status bar at the bottom of the editor shows "File 3\_exceptions.py 1 Error".

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    NUGET    SPELL CHECKER    PLAYWRIGHT

⑥ iTomerss-iMac-14732:Lec4 iTomers\$ /usr/local/bin/python3 /Volumes/E4TB/W270225MR/Python/Lec4/3\_exceptions.py  
enter a number: 3t

Traceback (most recent call last):

  File "/Volumes/E4TB/W270225MR/Python/Lec4/3\_exceptions.py", line 2, in <module>  
    x = int(input("enter a number: "))

**ValueError: invalid literal for int() with base 10: '3t'**

iTomerss-iMac-14732:Lec4 iTomers\$

הטיפוס של השגיאה

שם השגיאה

תיאור של השגיאה

**ValueError**

# טיפול בשגיאות

## Exception

מילות מפתח expect raise try

```
try:  
    x = int(input("enter a number: "))  
    # do stuff  
except ValueError:  
    print("numbers only")
```

# טיפול בשגיאות

## Exception

```
try:  
    x = int(input("enter a number: "))  
    result = 10 / x  
    print(f"Result: {result}")  
except ZeroDivisionError:  
    print("Division by zero is not allowed.")  
except ValueError:  
    print("Numbers only")
```

הבחנה בין מספר סוגי שגיאות:

```
try:  
    x = int(input("enter a number: "))  
    result = 10 / x  
    print(f"Result: {result}")  
except (ZeroDivisionError, ValueError):  
    print("Invalid input.")
```

ביצוע פעולה זהה ב 2 מקרים שונים:

# טיפול בשגיאות

## Exception

```
try:  
    x = int(input("enter a number: "))  
    result = 10 / x  
    print(f"Result: {result}")  
except Exception:  
    print("Invalid input.")
```

לא משנה מה הייתה השגיאה - נוציה הודעה כללית

# טיפול בשגיאות Exception

```
try:  
    x = int(input("enter a number: "))  
    result = 10 / x  
    print(f"Result: {result}")  
except ValueError:  
    print("Please enter a valid integer.")  
except ZeroDivisionError:  
    print("Division by zero is not allowed.")  
except Exception:  
    print("An unexpected error occurred.")
```

גישה מושלבת:

מייפוי של כל סוגי השגיאות שאנו מכירים:

ובנוסף בлок של All :CatchAll  
כדי לתפוס שגיאות לא צפויות  
במקרה זהה נדפיס לקובץ לוג

כך להבא - נוכל לחזק את הקוד  
עם עוד בלוקים של שגיאות ידועות

# טיפול בשגיאות Exception

```
try:  
    x = int(input("enter a number: "))  
    result = 10 / x  
    print(f"Result: {result}")  
except ValueError:  
    print("Please enter a valid integer.")  
except ZeroDivisionError:  
    print("Division by zero is not allowed.")  
except Exception:  
    print("An unexpected error occurred.")
```

גישה מושלבת:

מייפוי של כל סוגי השגיאות שאנו מכירים:

ובנוסף בлок של All :CatchAll  
כדי לתפוס שגיאות לא צפויות  
במקרה זהה נדפיס לקובץ לוג

כך להבא - נוכל לחזק את הקוד  
עם עוד בלוקים של שגיאות ידועות

# טיפול בשגיאות

## Exception

```
x = int(input("enter a number: "))
result = 10 / x
print(f"Result: {result}")

print("Good bye")
```

הוזן ערך ttt

האם יוצג ?goodbye

שגיאה עוצרת את התוכנית

```
try:
    x = int(input("enter a number: "))
    result = 10 / x
    print(f"Result: {result}")
except Exception as e:
    print("message", e)
    print("Line number", e.__traceback__.tb_lineno)
    print("File info", e.__traceback__.tb_frame)

print("Good bye")
```

שימוש באובייקט של השגיאה:

אם תפסנו את השגיאה התוכנית ממשיכה

## זריקה של שגיאה: raise

```
def draw_square(size):
    for _ in range(size):
        print(size * "█")
draw_square(4)
```



```
def draw_square(size):
    if not isinstance(size, int):
        raise TypeError("Size must be an integer")
    if size <= 0:
        raise ValueError("Size must be a positive integer")
```

```
# do the logic:
for _ in range(size):
    print(size * "█")
```

```
# test edge cases:
draw_square(11.1)
draw_square("ff")
draw_square(0)
draw_square(-3)
```

באחריות של המתכנת שכותב פונקציה  
לספק פידבק לגבי הפרמטרים

כדי شيء משתמש בפונקציה קיבל מידע  
איך להשתמש בה יותר נכון

# מחלקות וירושה

מחלקה - תבנית לייצור אובייקטים:

**תכונות:** first name, last name , email, birthdate

**פעולות:** החניה סיסמא, שלח מייל, שם מלא, האם יש יום הולדת

Car Object	Car Properties	Car Methods
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white	car.start() car.drive() car.brake() car.stop()

```
class Car:  
    # properties:  
    # constructor function  
    def __init__(self, name, model, weight, color):  
        self.name = name  
        self.model = model  
        self.weight = weight  
        self.color = color
```

מוסכמתות שמות: שם מחלקה באות גודלה בלבד:

**מетодה** = שיטה בתוך מחלקה

לא קוראים לזה פונקציה (-)

# חלוקת וירושה

חלוקת תכונות ופעולות:

```
class Car:  
    def __init__(self, name, model, weight, color):  
        self.name = name  
        self.model = model  
        self.weight = weight  
        self.color = color  
  
    def start(self):  
        print(f"{self.name} {self.model} is starting...")  
  
    def html(self):  
        return f"""  
        <div class='car'>  
            <h2>{self.name}</h2>  
            <p>Model: {self.model}</p>  
            <p>Weight: {self.weight} kg</p>  
            <p>Color: {self.color}</p>  
        </div>  
        """  
  
c = Car("Fiat", "uno", 850, "white")  
c.start()
```

# מחלקות וירושה

מחלקה תכונות ופעולות:

```
class Car:  
    def __init__(self, name, model, weight, color):  
        self.name = name  
        self.model = model  
        self.weight = weight  
        self.color = color  
  
    def __str__(self):  
        return f"Car 🚗: name: {self.name}"  
  
    def start(self):  
        print(f"{self.name} {self.model} is starting...")  
  
c = Car("Fiat", "uno", 850, "white")  
c.start()  
print(c)
```

בנייה:

תכונות:

מה יראו בprint:

מחלקה - מבנה / תבנית לעובדה:

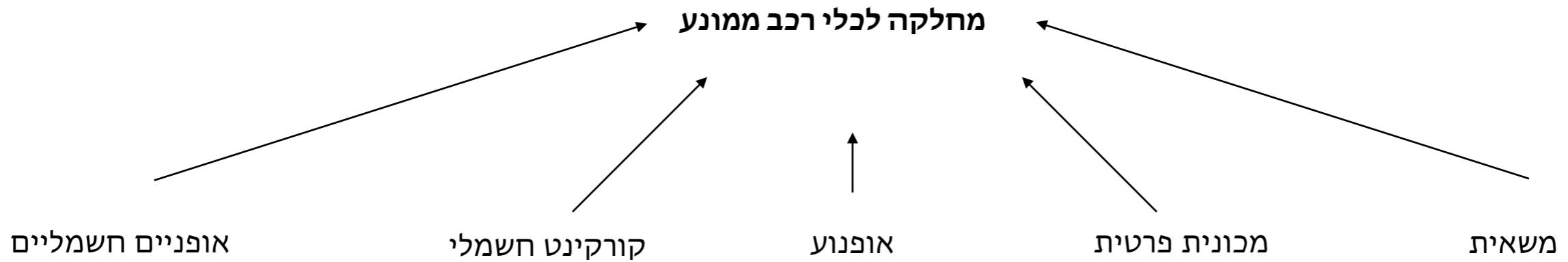
בנייה

תכונות

הMETHOD \_\_str\_\_ חלק מהמבנה של מחלקה

METHODS - פעולה עם האובייקט  
(יכולות לגשת לתכונות)

# ירושה

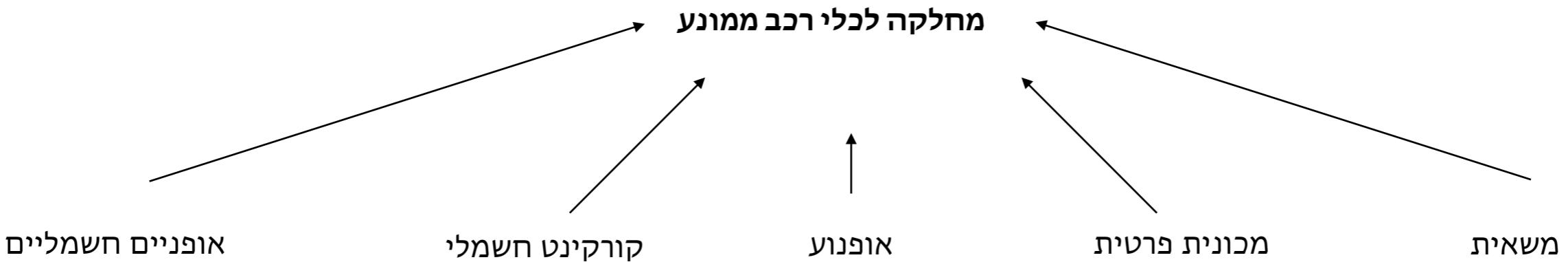


חסכון בקוד

אם נכתב במחלקה של "רכב ממונע"  
וכל המחלקות ירשו ממנו

יהיה פחות קוד בכל אחת מהמחלקות שירשנות  
(כתובים פעם אחת - משתמשים הרבה פעמים)

# ירושה



```
class Animal:  
    def __init__(self, name):  
        self.name = name  
  
    # str for friendly representation:  
    def __str__(self):  
        return f"Hey, I'm {self.name}"  
  
    # str for debugging - serious representation  
    def __repr__(self):  
        return f"<Animal name='{self.name}'/>"  
  
b = Animal("b")  
print(b)
```

# ירושה

```
class Animal:  
    def __init__(self, name):  
        self.name = name  
  
    # str for friendly representation:  
    def __str__(self):  
        return f"Hey, I'm {self.name}"  
  
    # str for debugging – serious representation  
    def __repr__(self):  
        return f"<Animal name='{self.name}'/>"  
  
    ↗ ירושה:  
class Cat(Animal):  
    def make_sound(self):  
        print(self.name, 'Miaooooou')  
  
f = Cat('Fluffy')  
f.make_sound()
```

# ירושה

```
class Animal:  
    def __init__(self, name):  
        self.name = name  
  
    # str for friendly representation:  
    def __str__(self):  
        return f"Hey, I'm {self.name}"  
  
    # str for debugging – serious representation  
    def __repr__(self):  
        return f"<Animal name='{self.name}'/>"
```

לכל בעל חיים יש "שם"

למחלקה כלב יש תכונה נוספת:  
"גזע הכלב"

ירושה:

כшиוצרים בניאי לDog שירוש מ-Animal  
הבנייה צריך לקבל את 2 התכונות - גם name וגם breed

name בעבר  
Animal:

```
class Dog(Animal):  
    def __init__(self, name, breed):  
        super().__init__(name)  
        self.breed = breed  
    def make_sound(self):  
        print(self.name, 'Woof Woof')
```

ירושה:

הוספה תכונה:

```
class Dog(Animal):  
    def __init__(self, name, breed):  
        super().__init__(name)  
        self.breed = breed  
    def make_sound(self):  
        print(self.name, 'Woof Woof')
```

# ירושה

```
class A:  
    pass
```

```
class B(A):  
    pass
```

המחלקה B יורשת מהמחלקה A:  
لمחלקה B יש את כל המתודות והתכונות של המחלקה A

# ירושא מרובה

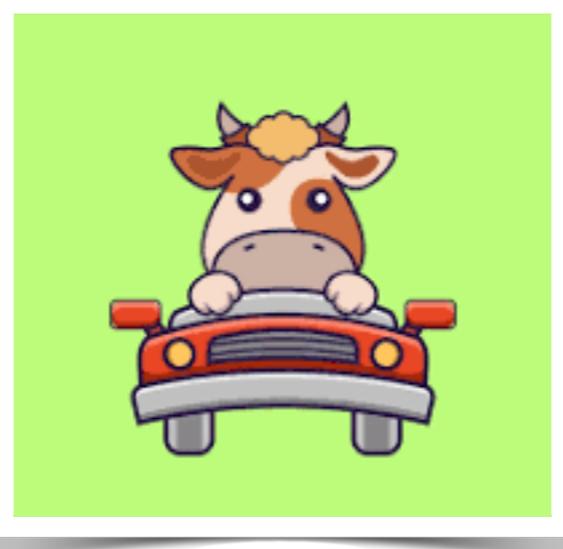
```
class A:  
    pass
```

```
class B:  
    pass
```

```
class C(A, B):  
    pass
```

פיתון מאפשר ירושא מרובה:

המחלקה C יורשת מהמחלקה A והמחלקה B:



המחלקה Cow(Animal, Driver)

# ריבוי ארגומנטים לפונקציה עם kwargs

```
def draw_text(**kwargs):
    """
    kwargs: color, size, text, font, bold
    """

    color = kwargs.get("color")
    size = kwargs.get("size")
    font = kwargs.get("font")
    text = kwargs.get("text")
    bold = kwargs.get("bold")

    print(f"Drawing text: {text} with {kwargs}")
    print(color, size, font, text, bold)

draw_text(text="Hola", font="Arial")
```

יכולת להיות פונקציה שמקבלת מספר רב של פרמטרים:

פיתון מאפשר לנו להגיד את כל הארגומנטים כמיילון

אפשר גם להגיד טיפוס ושמות לארגומנטים  
כדי לספק השלמה אוטומטית

# ריבוי ארגומנטים לפונקציה עם טיפוסים

```
from typing import TypedDict, Unpack

class DrawArgs(TypedDict):
    color: str
    size: int
    text: str
    font: str
    bold: bool
    italic: bool
    underline: bool

def draw_text(**kwargs: Unpack[DrawArgs]):
    """
    kwargs: color, size, text, font, bold, italic, underline
    """
    print(kwargs["color"])
    print(kwargs["size"])
    print(kwargs["text"])

draw_text(color="red", size=12, text="Hello")
```

# עבודה עם ספריות חיצונית:

pygames

משחקים

turtle

צייר גרפי

tkinter

תוכנות לדסktop

opencv

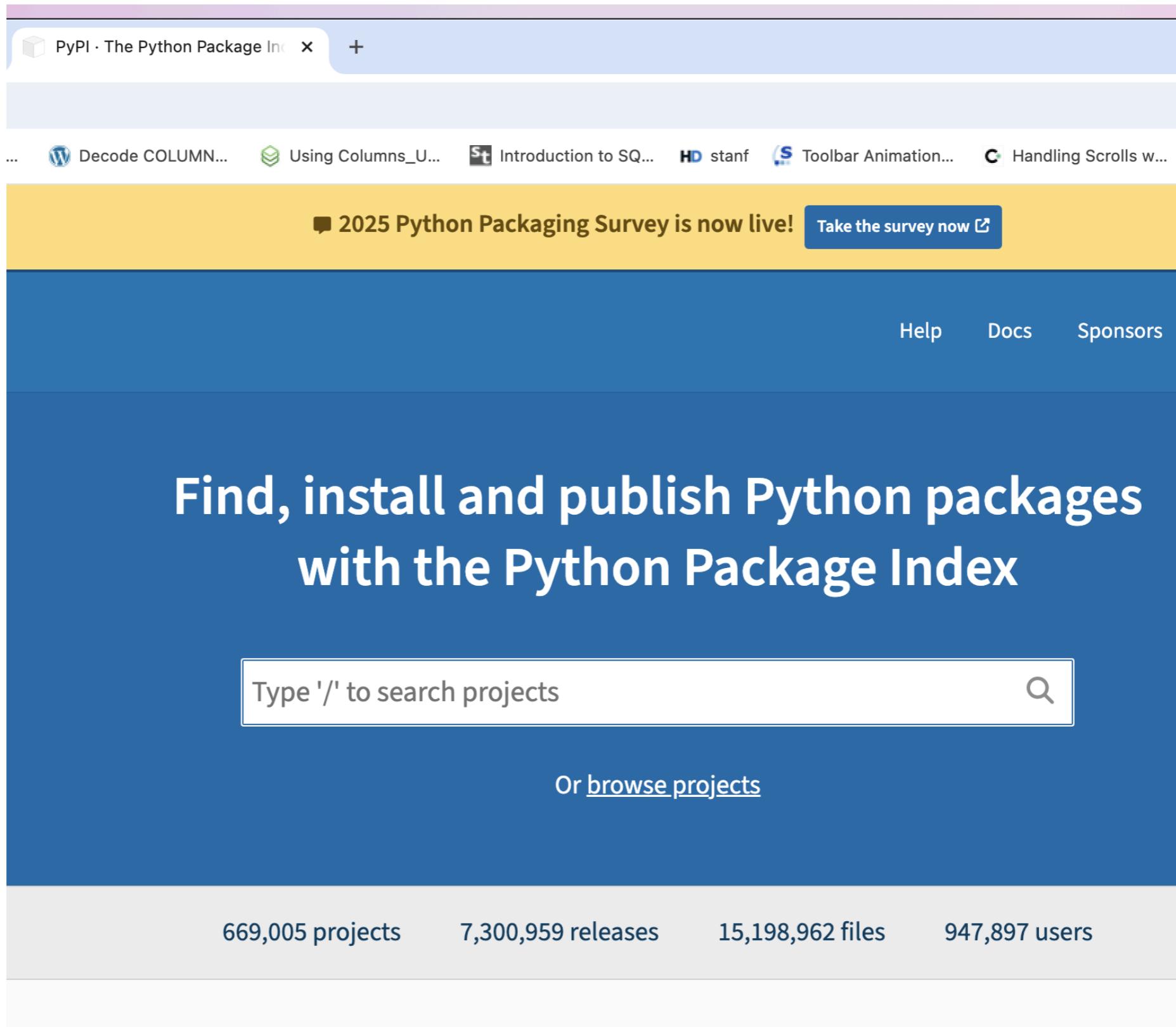
ניתוח גרפי מתקדם

...

אינספור ספריות AI

אפשר לבנות צד שרת עם פיתון

# עבודה עם ספריות חיצונית:



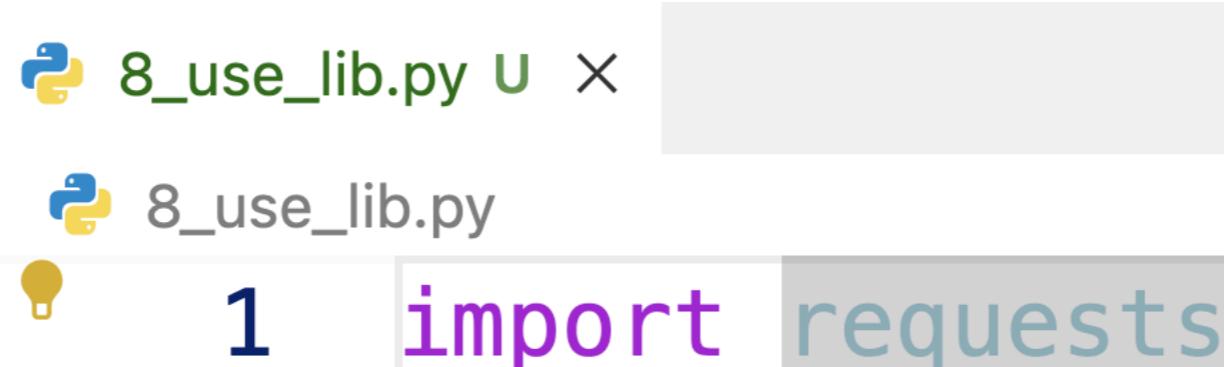
התקנה של ספריה:

התקנה של גלובלית במחשב:

1

<https://pypi.org/project/requests/>

pip install requests



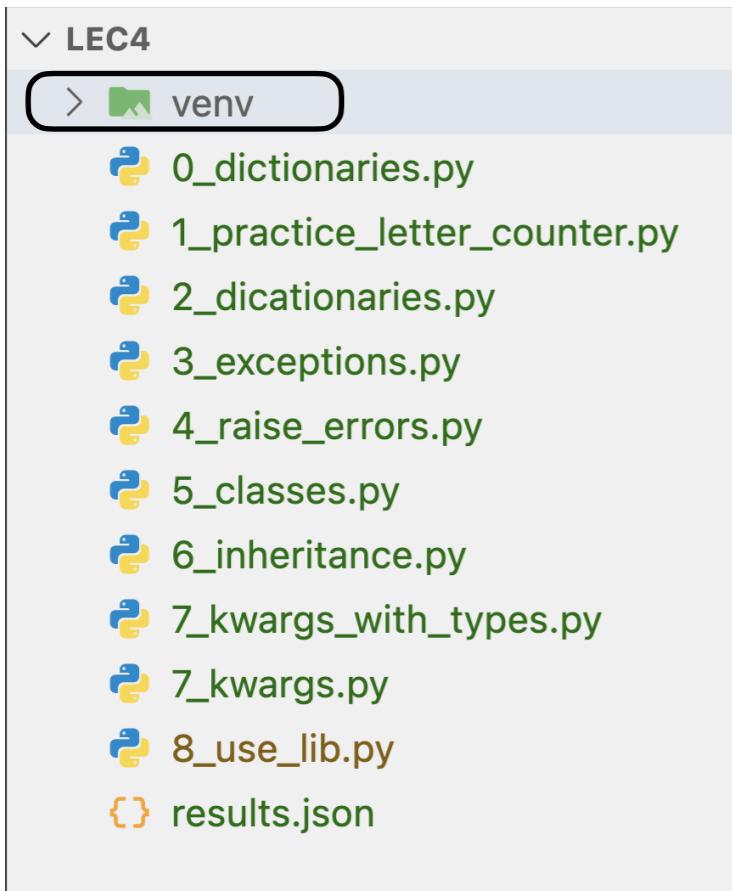
The image shows a screenshot of a code editor with two tabs. The top tab is labeled '8\_use\_lib.py' and has a small Python icon next to it. The bottom tab is also labeled '8\_use\_lib.py' and has a small Python icon next to it. Below the tabs, there is a line of code: '1 import requests'. The word 'import' is highlighted in purple, and 'requests' is highlighted in blue. A yellow lightbulb icon is positioned to the left of the line number '1'.

```
8_use_lib.py U ×  
8_use_lib.py  
1 import requests
```

# התקנה של ספריה:

התקנה מקומית בתחום פרויקט:

```
python -m venv venv
.\venv\Scripts\activate
```



```
python -m venv .venv
• ./.venv/bin/activate
```

```
python -m venv .venv
source ./.venv/bin/activate
```

# התקנה של ספריה:

**pip install requests**

```
import requests

url = 'https://google.com'

response = requests.get(url)

print(response.status_code)
print(response.text)
```

# עבודה עם הספרייה של JSON - פענוח של requests

**pip install requests**

<https://opentdb.com/api.php?amount=10&category=27&difficulty=easy&type=multiple>

```
import requests

url = 'https://opentdb.com/api.php?
amount=10&category=27&difficulty=easy&type=multiple'

response = requests.get(url)

if response.status_code != 200:
    print("Server error")
    print(response.status_code)
    print(response.text)
    exit(0)

json = response.json()

questions = json['results']

for q in questions:
    print("QUESTION", q['question'])
    print("ANSWER", q['correct_answer'])
    print("ANSWER", q['incorrect_answers'])
```

# שיעור 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?

# • מושך טריויה על חיים

שימוש ב-API עם פירטן OpenTDB API

## המטרה:

יצירת משחק טריויה שמביא 10 שאלות על חיים מהאינטרנט  
ונוון לשחקן לענות עליהם

## הספריות שצרכות

```
import requests  
import html
```

## ה-API שנשתמש בו

### OpenTDB - Open Trivia Database

<https://opentdb.com/api.php?amount=10&category=27&type=multiple>

- מביא 10 שאלות רב-ברירה על חיים
- כל שאלה עם 4 תשובה אפשריות
- נתונים בפורמט JSON
- בחינם ללא צורך ברישום

# קבלת ועיבוד נתונים

שלבים 1-2

1

## קבלת השאלות מהאינטרנט

- השתמש ב- `requests.get()` כדי לקבל את הנתונים
- הmr ל-JSON עם `()json.`

```
response = requests.get("https://opentdb.com/api.php?  
amount=10&category=27&type=multiple")  
data = response.json
```

2

## עיבוד השאלות

לכל שאלה יש:

- השאלה עצמה - `question`
- התשובה הנכונה - `correct_answer`
- רשימה של תשובות שגויות (3 תשובות) - `incorrect_answers`

**חשוב מאוד:** השתמש ב- `html.unescape()` כדי לתקן תווים מיוחדים בטקסט!  
למשל: "What's" יփוך ל- "What's"

```
clean_question = html.unescape(question_data['question'])  
clean_answer = html.unescape(question_data['correct_answer'])
```

# הציג וקלו משותמץ

שלבים 3-4

3

## הציג השאלות

- לכל שאלה, צור רשימה של 4 תשובות (1 נכונה + 3 שגויות)
- ערביב את הרשימה עם **(random.shuffle)**
- הציג את השאלה ואת התשובות עם מספרים (1-4)

```
import random

# ייצירת רשימה תשובות
all_answers = [correct_answer] + incorrect_answers
random.shuffle(all_answers)

# הצגת השאלה
for j, answer in enumerate(all_answers):
    print(f"{j+1}. {answer}")
```

4

## קלט מהמשתמש

- בקש מהמשתמש להקש מספר (1-4)
- השתמש ב- **(input)** וודא שהמספר נכון
- וודא שהמספר בטוח 1-4

### דוגמה להציג:

שאלה 1: איזה חיה הכי מהירה בעולם?

1. ברדלס
2. גמר
3. אריה
4. זברה

הקש את מספר הבחירה שלך (1-4):

```
"הקש את מספר הבחירה שלך (1-4)"choice = input
(0-3) # המריה לאינדקס choice = int(choice) - 1
chosen_answer = all_answers[choice]
```

# ✓ בדיקות ומבנה כללי

שלבים 5-6 + מבנה הקוד

5

## בדיקה התשובה ✓

- השווה את התשובה שהמשתמש בחר לתשובה הנכונה
- הdfs אם זה נכון או לא
- אם לא נכון, הצג את התשובה הנכונה

6

## ספרת נקודות 🏆

- התחיל עם **score = 0**
- בכל תשובה נכונה, הוסף 1
- בסוף הצג את הציון הסופי

## מבנה הקוד הכללי

```
# 1. ייבוא ספריות
import requests, html, random

# 2. קיבלת הנתונים מהאיינטגרנט
(...)(response = requests.get
      ()(data = response.json

# 3. לולאה for שעוברת על 10 השאלות:
score = 0
:for i, question_data in enumerate(data['results'])
      # - עיבוד השאלה הנוכחית
      # - הצגת השאלה ו-
```