



מבני נתונים ואלגוריתמים

מבוא ל-Python
פולינה לוצקר

מנהלות

- מרצה הקורס: פרופסור יורם לוזון
- מתרגלת: פולינה לוצקר
- אימייל: polinalutbiu@gmail.com
- שעות קבלה: יום שני 13:00–15:00 , בתיאום מראש.
- אתר הקורס: [math-wiki](#)
- דרישות קדם: לינארית 1, אינפי 1, **ידע בתכנות**.

תרגילי בית

- מדי שבוע יתפרסם תרגיל בית חדש.
- שבוע אחד תרגיל תכנותי ושבוע אחד תרגיל תאורטי.
- תרגילי תכנות הם להגשה –חובה!
 - ההגשה דרך `submit`
 - <https://submit.cs.biu.ac.il/cgi-bin/welcome.cgi>
 - יש לכתוב בשפת Python 3
 - פידבק ישלח לכם למייל אחרי ההגשה – הפידבק אינו הציון שלכם!
 - אין בדיקה ידנית
 - תהיה בדיקת העתקה
- התרגילים התאורטיים אינם להגשה אלא לאימון עצמי בלבד.

שרת האוניברסיטה

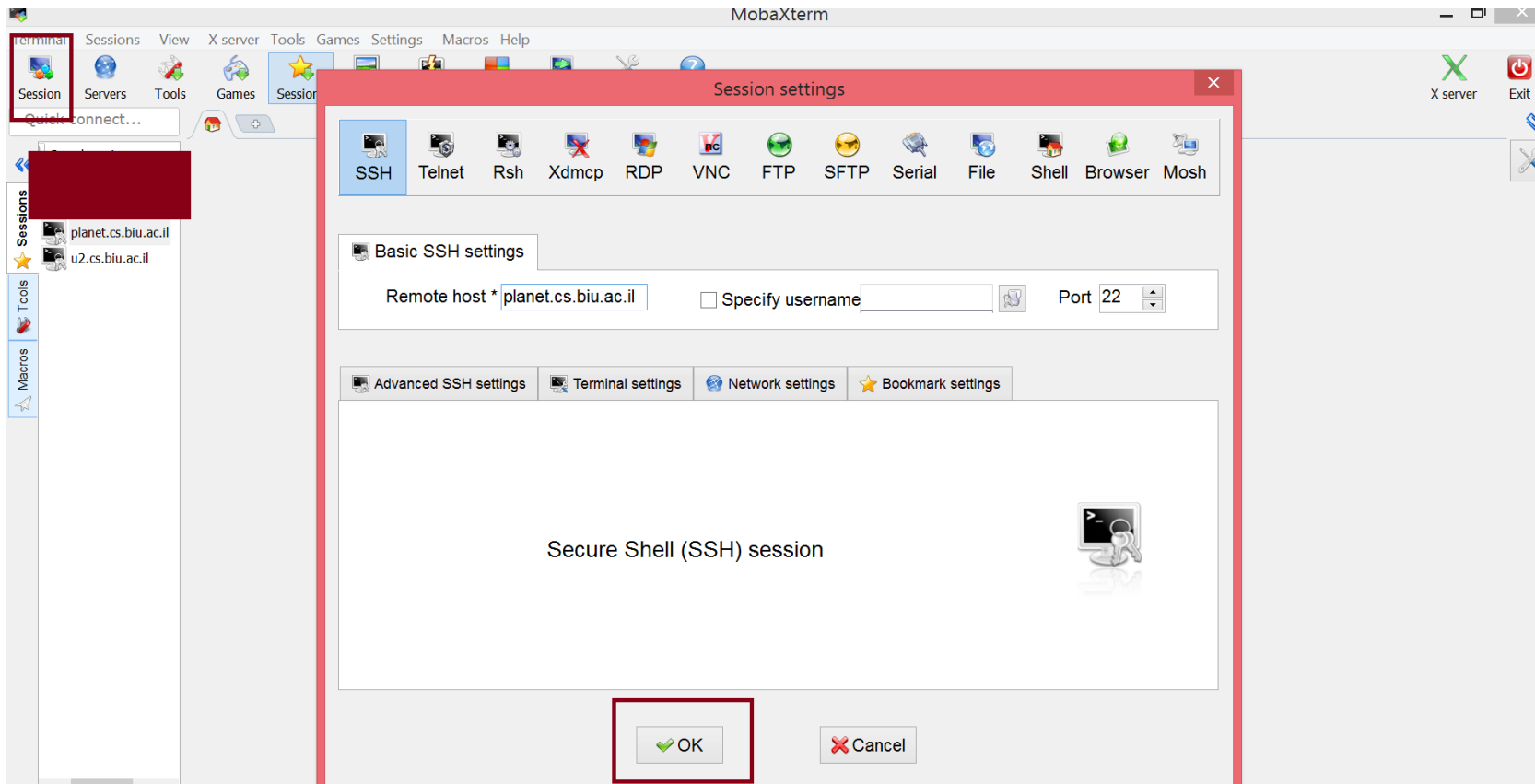
- השיעורי הבית ייבדקו על שרתי האוניברסיטה, ולכן עליכם להריץ את השיעורי בית על השרת ולוודא שמתקבל הפלט הרצוי.

- יש להוריד MobaXterm

<https://mobaxterm.mobatek.net/download-home-edition.html>

- פקודת הרצה: python3


שרת האוניברסיטה



- כרגע השיעורי בית יבדקו על השרת planet של האוניברסיטה, יכול להיות אריץ על שרת 2ט- אעדכון!
- לאחר מכן תצטרכו להכניס שם וסיסמה שקיבלתם מהחובשים

סביבת עבודה

- עליכם להוריד interpreter
- [כאן](#)
- תזכרו שאתם עובדים עם פייתון 3.
- סביבת עבודה **מומלצת**: pycharm
- ניתן להוריד [כאן](#)
- הגירסא החינמית מספיקה לקורס.

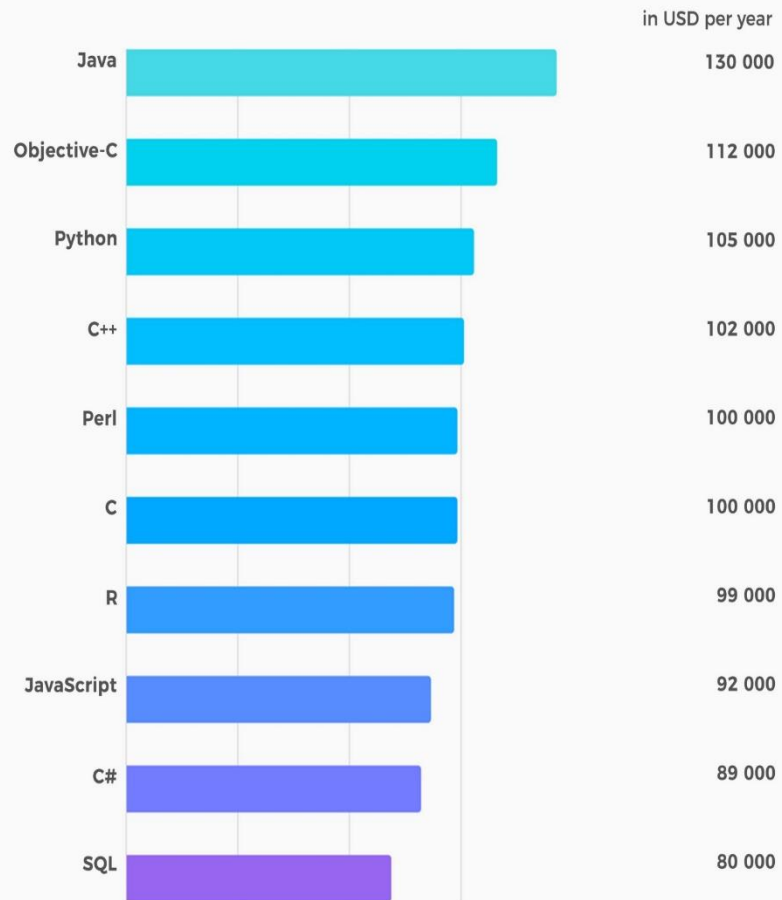


מבוא ל-PYTHON

קצת על PYTHON

- Object Oriented
- Interpreted
- תחביר קל ללימוד ולקריאה
- שפה ניידת – ניתן להריץ על מגוון מערכות הפעלה.
- ניהול זיכרון אוטומטי – Garbage collection
- שפת קוד פתוח
- מכילה בתוכה אוסף מכובד של ספריות סטנדרטיות-ניתן להסתמך על קיומן בכל מימוש של פייתון.

TOP 10
CHALLENGEROCKET.COM RANKING
OF PROJECTED EARNINGS IN 2017 BY A PROGRAMMING LANGUAGE



CHALLENGEROCKET.COM

קצת על PYTHON

מאפיינים תחביריים

- המבנה של בלוקי קוד בתוכנית נקבע על ידי הזחות! עצם הזחת הבלוק מגדירה אותו כבלוק תכנותי שונה.
- תנאים בפקודות תנאי ולולאות נכתבים ללא סוגריים, ומזוהים על ידי המהדר בעזרת מילים שמורות ותו ';' המופיע לאחריהם.
- פקודות מופרדות לרוב על ידי מעבר שורה. שימוש בנקודה-פסיק להפרדה הוא אופציונלי.

He : You are ';' to my code

She : Sorry, I Code in Python!!

HELLO WORLD!

- Python 2:

```
print "Hello World!"
```

```
print ("Hello World!")
```

- Python 3:

```
print ("Hello World!")
```

BUILT-IN TYPES

```
'''  
Numeric Types  
'''  
  
2017    # int  
3.14    # float  
2+7j    # complex  
  
# Sequence types:  
"Polina"    # string  
(1, 2, "abc")    # tuple  
[1, 2, "abc"]    # list  
{'Year': 2017, 'Month': 'October'}    # dict  
  
True, False    # Bool
```

משתנים

- משתנה הוא שם המתייחס לערך
- יצירת משתנה – מציינים שם משתנה ואת הערך השייך לו. אין צורך להצהיר על סוג הטיפוס או גודלו!

```
>>> name = "Polina"
>>> type (name)
<class 'str'>
>>> age = 25
>>> type (age)
<class 'int'>
>>> flag = True
>>> type(flag)
<class 'bool'>
```

- כל המשתנים בפייתון הם אובייקטים!

משתנים – המשך

```
>>> i=2017
>>> type(i)
<class 'int'>
>>> i="2017"
>>> type(i)
<class 'str'>
```

שימו לב להבדל בין 2017
ל"2017"!

- ניתן לשנות את טיפוס המשתנה בכל עת.
- שמות משתנים:
 - רק אותיות באנגלית (קטנות או גדולות) או ספרות.
 - לא יכול להתחיל בספרה.
 - נהוג להשתמש באותיות קטנות בלבד עם קו מפריד.
 - גם בפייתון יש מילים שמורות (28 במספר).

```
and      continue  else      for      import   not      raise
assert   def        except    from     in       or
         return
break    del        exec      global   is       pass     try
class    elif      finally   if       lambda  print    while
```

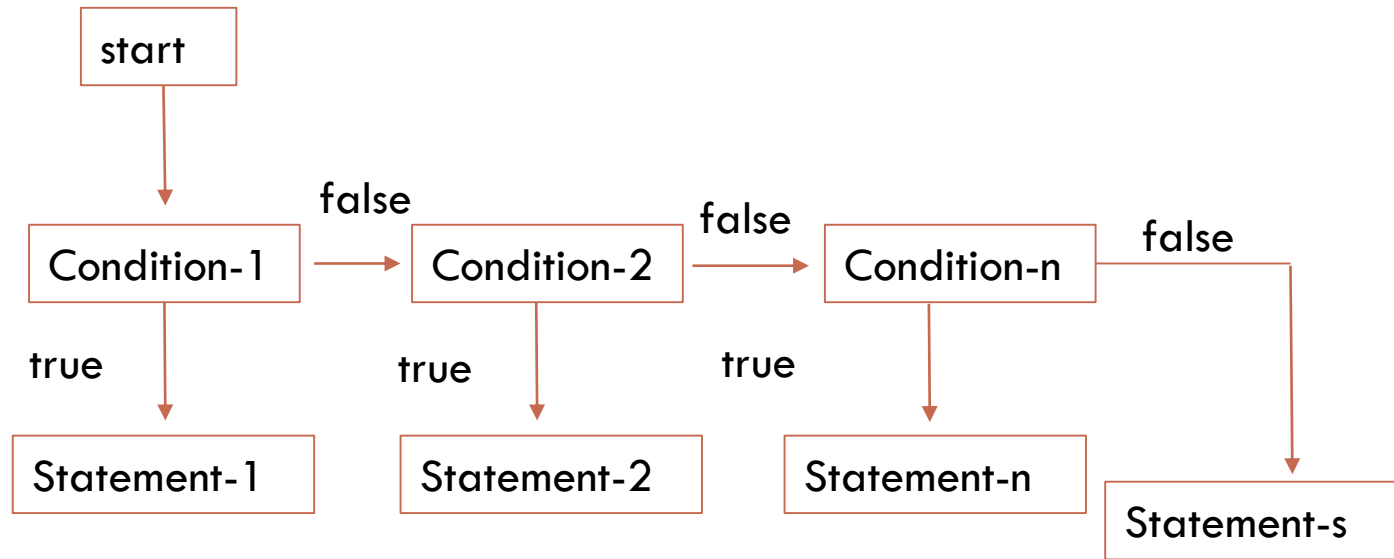
אופרטורים

- אופרטורים הם סמלים מיוחדים, שמסמלים חישובים פשוטים כדוגמת חיבור וכפל.
- הערכים בהם משתמשים האופרטורים לחישוב נקראים **אופרנדים**, וצירוף של אופרטור ואופרנדים נקרא **ביטוי**.
- מרבית האופרטורים בפייתון מבצעים את הפעולה הצפויה מהמקבילה המתמטית אותם הם מסמלים

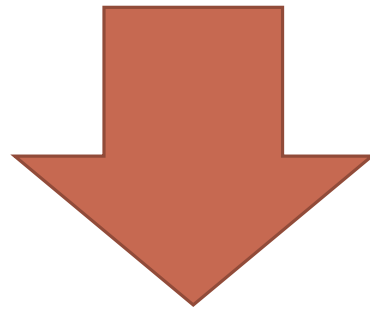
• [לעוד מידע](#)

ביטויי תנאי

דוגמה בפייתון:

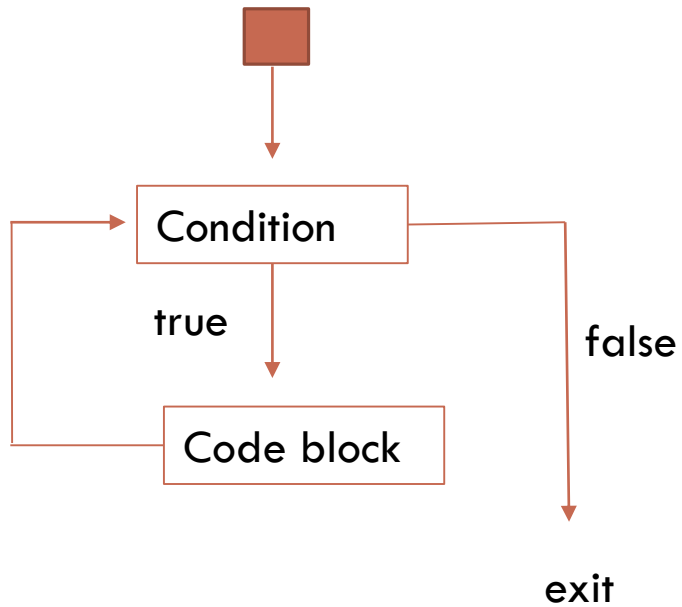


```
# Input from the user
x = int(input())
if x == 0:
    print("X is 0")
elif x < 0:
    print("X is negative")
else:
    print("X is positive")
```



לולאות WHILE

דוגמה:



```
# Input from the user
x = int(input())
while x > 0:
    print(x)
    x -= 1
```

Input:5

Output:

5

4

3

2

1

פונקציית RANGE

- `range` מייצרת טיפוס שהוא אובייקט "מחולל ערכים" (generator).
- `range(start,end,step)` – רץ מערך ההתחלה (`start`) עד הערך הסופי `end` (לא כולל) במספר צעדים (`step`) לא חובה לציין מספר צעדים. כברירת מחדל יתקדם רק צעד אחד. אפשר גם לרוץ אחורנית עם `-step`. אם נשמיט את הערך `start` נתחיל מ-0. **הפונקציה מקבלת רק מספרים שלמים.**

• דוגמה: `range(10,1,-2)`

- האובייקט הזה לא שומר את כל הערכים מראש אלא מייצר אותם באופן דינאמי כאשר יש צורך וכך אין צורך לשמור שום ערך בזיכרון.

• אין לפייתון אפשרות להציג את כל הערכים בבת אחת כמו שהוא מציג את הרשימה.

• דוגמה:

```
>>> print(range(1,10))
range(1, 10)
>>> print(list(range(1,10)))
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> print(list(range(10,1,-2)))
[10, 8, 6, 4, 2]
```

- המחליפה של הפונקציה `xrange` בפייתון 2. (אין `xrange` בפייתון 3, ואין את המימוש של `range` של פייתון 2)

לולאת FOR

דוגמה:

הלולאה רצה על הערכים שהאובייקט range מחולל. ניתן לרוץ על ערכים של אובייקטים אחרים גם, לדוגמה:

```
food= ["pizza", "sushi", "apple"]
```

```
for i in food:  
    print(i)
```

Output:

```
pizza  
sushi  
apple
```

```
sum=0  
for i in range(10):  
    sum+=i  
print(sum)
```

Output:45

פונקציות

```
def functionname( parameters ):  
    "function_docstring"  
    function_suite  
    return [expression]
```

- הגדרת פונקציה מתבצעת בדרך הבאה:
- שימו לב להזחה!

- בפייתון גם הפונקציות הן אובייקטים!

```
>>> print(type(print))
```

```
<class 'builtin_function_or_method'>
```

פונקציית ה-INPUT

- הפונקציה משמשת לקריאת קלט מהמשתמש.
- הפונקציה מחזירה `string` – יש לעשות המרות בהתאם.
- הקלט נגמר ב `newline`
- דוגמה:

```
>>> food = input("what is your favorite food? ")
```

```
what is your favorite food? Pizza and sushi
```

```
>>> print(food)
```

```
Pizza and sushi
```



תרגיל

- עליכם לכתוב תוכנית המקבלת מטריצה מהמשתמש.
- הקלט : מספר השורות מטריצה שאיבריה הם מספרים שלמים.

פתרון

```
def get_matrix():  
    # Get the row size and convert it to int  
    row_size = int(input("Please enter the column size: "))  
  
    # An empty matrix  
    matrix=[]  
  
    for i in range(row_size):  
        # Get a new row  
        new_row = input()  
  
        # Turn it to a array  
        # The method split() returns a list of all the words in the string,  
        # using str as the separator  
        new_row_array = new_row.split(' ')  
        new_row_array = [int(x) for x in new_row_array]  
  
        # Add the row to the matrix  
        # The method append() appends a passed obj into the existing list.  
        matrix.append(new_row_array)  
  
    return matrix
```

הערה: הקוד מניח שהקלט תקין



מחלקות

מחלקות – איך יוצרים?

```
class ClassName:  
    'Optional class documentation string'  
    class_suite
```

- המבנה הבסיסי של המחלקה בפייתון:

- פעולות שמבוצעות על האובייקטים נקראות "מתודות"
- נוכל להוסיף מתודות לפי הצורך.

מחלקות-איך יוצרים?

- בעת יצירת המחלקה נשאל – מה הפעולות שהוא צריך לעשות (מתודות)? איזה מאפיינים יהיו למחלקה?
- מה היחסים של המחלקה עם מחלקות אחרות?

מחלקות – דוגמה

```
class Cat:
    # Like a constructor
    def __init__ (self, name, colour, age, food):
        self.name = name
        self.colour = colour
        self.age = age
        self.food = food

    def meow(self):
        return "meow!"

    def happy_birthday(self):
        self.age += 1

    def feed(self):
        return self.food
```

מחלקות-המשך הדוגמה

```
class Cat:
    # Like a constructor
    def __init__(self, name, colour, age, food):
        self.name = name
        self.colour = colour
        self.age = age
        self.food = food

    def meow(self):
        return "meow!"

    def happy_birthday(self):
        self.age += 1

    def feed(self):
        return self.food
```

```
# Create a new cat
my_cat = Cat("Fifa", "gray", 8, "fish")

# Use the "happy birthday" method
my_cat.happy_birthday()

print(my_cat.meow())
```

Output:meow!



מחלקות – המשך דוגמה

```
class crazy_cat_lady:
```

```
    num_of_cats = 0
```

```
    def __init__(self, cats=[], favorite=None):
```

```
        self.cats = cats
```

```
        self.num_of_cats = len(cats)
```

```
        self.favorite_cat = favorite
```

```
    # Add a new cat- append to the list
```

```
    def add_new_cat(self, new_cat):
```

```
        self.cats.append(new_cat)
```

```
        self.num_of_cats += 1
```

```
    # Create a new crazy cat lady
```

```
    crazy_cat_lady_new = crazy_cat_lady()
```

```
    print(crazy_cat_lady_new.favorite_cat)
```

Output: None

```
    # Add a cat
```

```
    crazy_cat_lady_new.add_new_cat(my_cat)
```

```
    print(crazy_cat_lady_new.cats)
```

Output:[<__main__.Cat object at 0x02B6DAB0>]

```
    print(crazy_cat_lady_new.num_of_cats)
```

Output:1



מחלקות-המשך

- מיהו ה-`self` הוא האובייקט עצמו (כמו `this`)
- `Self` הוא בעצם ההבדל בין מתודה לפונקציה (כך רואים שהפונקציה מקושרת לאובייקט)
- לא מועבר באופן מפורש (מועבר מאחורי הקלעים)
- תמיד מופיע ראשון כארגומנט במתודה!!! (גם אם תשנו את השם של הארגומנט מ `self`)
- כאשר ניגשים לשדות באובייקט מתוך מתודות הפנימיות של האובייקט, חייבים לקרוא דרך ה-`self`, גם למשתנים.
- מתודה `_init_`
- לא בדיוק `constructor`, יותר מדויק להתייחס למתודה בתור מאתחל. השינויים העיקריים הם בעיקר בנושא של ירושה, שלא חלק מהקורס.
- נקרא כל פעם שיוצרים אובייקט חדש.
- ניתן להגדיר ערכי `default`.
- אין צורך לדאוג לשחרור זיכרון!
- על מנת להדפיס את המחלקה, מומלץ לממש מתודה `toString()`

כימוס מידע

- אחד העקרונות של תיכנות מונחה עצמים הוא כימוס מידע, ולכן ישנה התייחסות גם בפייתון ל `public` | `private`.
- רק אם מה ש `public` במחלקה נגיש מחוץ למחלקה.
- בפייתון מה שקובע את הסטטוס הנ"ל הוא השם של הפונקציה/מתודה/`attribute`. כל דבר הוא `public` אלא אם כן שמו מתחיל בשני קווים תחתונים `__`.
- מקובל להתייחס לכל שם עם `_` קו תחתון אחד כאל `private`, אם כי השפה אינה מחייבת. זאת מוסכמה, על מנת שמי שמשתמש בקוד יתייחס לזה כ `private`.

מודולים

- בפייתון ישנה דרך למקם הגדרות בקובץ אחד ולהשתמש בקטע קוד אחרים. קובץ כזה נקרא **מודול**. יכול להיות קובץ שאנחנו כתבנו או קובץ שכתבו בשבילנו.
- הגדרות שנמצאות במודול ניתנות ל**יבוא** לתוך מודולים אחרים או ל**מודול הראשי**
- למה צריך מודולים?
 - מקל על תחזוקת התוכנית.
 - שימוש חוזר בקוד.
 - עוזר בחלוקת עבודה – אם מספר אנשים עובדים על פרויקט לא אפשרי לעבוד על קובץ יחיד. אך אל דאגה – בקורס הזה על כל קוד יעבוד רק אדם אחד.
 - אם הקוד מרוכז בקובץ יחיד, אז כל שינוי בקובץ היה מצריך את הידור הקוד מחדש

מודלים – IMPORT

• לצורך ההדגמה נייבא תור (הראשון שנכנס הוא הראשון שיוצא, FIFO)

```
import queue
```

```
q=queue.Queue()
```

• דרך א':

```
from queue import Queue
```

```
q = Queue()
```

```
import queue as fifo
```

```
q = fifo.Queue()
```

• דרך ב':

- כאשר ניגשים למאפיינים ומתודות ולא רוצים להשתמש בשם המודול בכל פעם.
- כאשר רוצים לייבא רק מספר מצומצם של מתודות.

• אם המודול מכיל פונקציות בעלות אותו שם כמו של פונקציות במודול שאנו כותבים, אפשר להשתמש רק ב `import module`.

• ניתן לשנות שם למודול:

קבצים

- קובץ לפי וויקפדיה: **קובץ** הוא משאב מערכת לוגי המאפשר אחסון מידע, אשר זמין לתוכניות מחשב לקריאה או כתיבה. לרוב קובץ יישמר לטווח ארוך באמצעי לאחסון נתונים כגון דיסק קשיח, או דיסק און קי. בשל הגיוון הרב של המידע הקיים לרוב במחשבים, השימושים השונים והתוכנות השונות העושות שימוש במידע, קיימים סוגי קבצים רבים המותאמים לכל צורך.
- אפשר לחשוב על כל קובץ כעל קובץ שורות עם טקסט.
- `<=` כל שורת טקסט היא בעצם מחרוזת שמסתיימת בסימן שורה חדשה ("`\n`")
- לפני שנוכל לפתוח לקרוא קובץ עלינו להגיד באיזה קבצים נרצה להשתמש ולאיזו מטרה.
- נעשה זאת בעזרת פונקציית `open()` המחזירה אובייקט (`handle`) בעזרתו נוכל לגשת לקובץ או לעדכן את הקובץ.

קבצים-המשך

```
# For reading from this path  
file=open(path, "r")
```

```
# For writing to file in this path  
file=open(path, "w")
```

```
# For reading from this path  
file=open(path, "r")  
# This is all the lines in the file  
lines = file.readlines()
```

```
# Go over them  
for i in range(len(lines)):  
    print (lines[i])
```

- לדוגמה:

- קובץ שלא קיים אפשר לפתוח רק לשמירה,

- אז פייתון יצור קובץ חדש עבורנו.

- לאחר שפתחנו קובץ תקין לקריאה, ניתן לקרוא ממנו. דוגמה:

קבצים - המשך

- ניתן לכתוב בתוך קובץ (שנפתח לכתיבה) בעזרת מתודה `write`
- ניתן לכתוב רק טיפוסים מסוג `string`
- חשוב לזכור לסגור כל קובץ **פעם אחת בלבד** לאחר השימוש עם המתודה `close()`

PYTHON MUTABLE VS IMMUTABLE

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	