שיטות מתקדמות בפייתון

Lambda & List comprehension

map

- נגדיר רשימה כלשהו ונגדיר פונקציה שמכפילה ב-2

```
Items = [2,7,3]
def double(arg):
        return arg*2
```

- הדרך הקלאסית להכפיל מספרים רבים ברשימת items

```
for item in items:
        double(item)
```

- (פישטתי קצת את הקוד. מה חסר פה?)
- אבל, אנחנו מאמינים שפייתון חשב על דרך קצרה יותר...

```
y = map(double, items)
print(list(y))
```

- מה יקרה אם תעשו בלי list? מה יקרה אם תנסו להדפיס בתוך ה map ?

lambda *arguments : expression*

- הדרך הקלאסית להגדיר פונקציה

```
def double(x):
        return x*2
print(double(3))
```

- אבל... אם היה וכו' ובאמת:

```
double = lambda x: x*2
print(double(3))
lambda a,b,c : a*b + c
```

שורה אחת בלבד.
אפשר לעבוד עם כל הטיפוסים

יצירת משפחת פונקציות על ידי למדה

```
def desc(colour):
        return lambda obj : print("The " + obj + " is " + colour)
green = desc("green")
green("table")
blue = desc("blue")
blue("chair")
```

- כתבו פונקציות double ו-triple לכפול ולשלש

map(lambda x : f(x), x)

- עכשיו נשלב את שני הקיצורים שלמדנו: גם ללופ גם לפונקציה

```
y = map(lambda x: x*2, items)
print(list(y))
```

- התחלנו את המצגת ב-4 שורות וסיימנו בשורה אחת.
- בפועל כותבים: y = list(map(lambda x: x*2, items))

מורכבות map + lambda

- רעיון יותר מורכב

```
def multiply(x):
    return (x*x)
def add(x):
    return (x+x)

funcs = [multiply, add]
for i in range(5):
    value = list(map(lambda x: x(i), funcs))
    print(value)
```

filter & reduce

- באופן דומה אנחנו יכולים לקצר

```
y = []
for item in items:
        if item < 5:
                y.append(item)
```

- על ידי פונקציה דומה ל-map ושמה filter

```
y = list(filter(lambda x: x < 5, items))
```

- עד עכשיו הוצאנו רשימה מרשימה וזה מה שיש בפייתון. כדי להוציא נתון אחד מחישוב על רשימה צריך

```
product = 1
list = [1, 2, 3, 4]
for num in list:
   product = product * num
```

- אבל יש ספריה בשם functool עם מודול בשם reduce

```
product = reduce(lambda x,y: x*y, [1,2,3,4])
```

zip

- בעיה: שתי רשימות

```
x = [1,2,3,4]
y = [“a”,”b”,”c”,”d”]
```

- אנחנו רוצים לקבל

```
[(“a”,1), (“b”, 2), (“c”, 3), (“d”, 4)]
```

- בואו נכתוב פונקציה לזה...
- פונקציה בילט אין של פייתון מקצרת:

```
zipped = zip(x,y)
```

- יותר נחמד ((print(dict(zipped
- בואו נפתור את הבעיה השכיחה של מיון רשימה אחת על ידי חברתה, בלי זיפ ועם זיפ (יש כמה פתרונות)

List comprehension
f(x) for x in list

- סינטקסט של פייתון כדרך אחרת לקצר לופ. נחזור ללופ המקורי, אמרנו שאנחנו מחפשים דרך נאה לקצר אותו

```
for item in items:
    y.append(double(item))
```

שקול לביטוי הבא

```
y = [double(x) for x in items]
```

שקול ללמדה הבאה

```
y = list(map(lambda x: double(x), items))
```

List comprehension condition

y = [f(x) for x in list if condition]

y = [double(x) for x in items if x%2]

- זה תנאי אם להפעיל בכלל את הפונקציה. מה קורה אם רוצים להפעיל את הפונקציה אבל עם תנאי

y = [f(x) if condition else g(x) for x in list]

y = [double(x) if x%2 else x for x in items]

nested list comprehension

y = [x for many in all for x in many]

for example

pairs = [(“he”,”she”), (“schnitzel”, “potatoes”), (“white”,”black”)]

y = [x for pair in pairs for x in pair]

- אפשר כך להשטיח מטריצה. בואו נבנה מטריצה

matrix = [[i for i in range(5)] for _ in range(6)]

matrix = [[i*j for i in range(5)] for j in range(6)]

ונשטיח אותה

Dictionary comprehension

dict_variable = {key:value for (key,value) in dictonary.items()}

מילון של תחנות דלק לפי מיקום למילון לפי מרחק

dollar = {“backpack”: 100, “case”: 200, “bag”: 20}

shekel = {product:price*3.4 for (product, price) in dollar}

TypeError: cannot unpack non-iterable int object

shekel = {product:price*3.4 for (product, price) in dollar**.items**()}