

# PPL: Lab 7

## Conventional Interfaces

# What is a Conventional Interfaces?

- ❖ Conventional Interface הינו מבנה נתוני המשותף להרבה מרכיבים בספריה של שפת התכנות, כאשר ניתן לנצל מרכיבים אלו עבור עיבוד נתונים מגוון.
- ❖ נתבונן ב 2 בעיות שנראות בתחילת לא קשורות האחת לשניה:
  1. מצא את סכום אַל האיברים הזוגיים בסדרת פיבונאצ'י.
  2. יצר "ראשי תיבות" של משפט באנגלית, המורכב ארוך ורך ממילים שמתחלות באות גדולה.
- ❖ הדמיון בין בעיות אלה נובע מכך שניתן לפרק אותן לפעולות פשוטות אשר מקבלות רצף כלשהו כקלט, ומחזירות רצף אחר כקלט.

# נתבון בבעיה הראשונה

מצא את סכום כל האיברים הזוגיים בסדרת פיבונאצ'י.

- ❖ ניתן לפרק את הפעולות הנדרשות באופן הבא:

enumerate	map	filter	accumulate
-----	---	-----	-----
naturals (n)	fib	iseven	sum

- ❖ ניקח את כל המספרים הטבעיים עד  $n$
- ❖ נחשב עבור כל אחד את האיבר המתאים בסדרת פיבונאצ'י
- ❖ ניקח רק את האיברים הזוגיים בסדרה
- ❖ נחשב את סכומם

# פתרון בעיה ראשונה

❖ נגידר פונקציה לחישוב האיבר ה-k בסדרת פיבונacci

```
>>> def fib(k):
    """Compute the kth Fibonacci number."""
    prev, curr = 1, 0 # curr is the first Fibonacci number.
    for _ in range(k - 1):
        prev, curr = curr, prev + curr
    return curr
```

# פתרון בעיה ראשונה

❖ וכן נגידר פונקציה לבדיקה האם מספר הוא זוגי:

```
>>> def iseven(n):  
    return n % 2 == 0
```

❖ נשתמש בפונקציה `map` אשר מפעילה פונקציה על רצף נתון ומחזירה את התוצאות, ובפונקציה `filter`, אשר מסננת ערכים העומדים בתנאי מסוים רצף נתון. דוגמא לשימוש:

```
>>> nums = (5, 6, -7, -8, 9)  
>>> tuple(filter(iseven, nums))  
(-8, 6)
```

```
>>> sum(map(abs, nums))
```

# פתרון בעיה ראשונה

- ❖ כתעת ניתן למשם את הפונקציה even\_fib אשר מחשבת את הסכום הרצוי.

```
>>> def sum_even_fibs(n):  
    """Sum the even members of the first n Fibonacci numbers."""  
    return sum(filter(iseven, map(fib, range(1, n+1))))  
  
>>> sum_even_fibs(20)  
3382
```

## פתרון בעיה שנייה

"יצר "ראשי תיבות" של משפט אנגלי, המורכב אך ורק ממילים מהתחלות באות גדולה.

- ❖ גם כאן, ניתן לפרק את הפעולות הנדרשות באופן דומה:

enumerate	filter	map	accumulate
-----	-----	-----	-----
words	iscap	first	tuple

- ❖ ניקח את כל המילים במשפט הנתון
- ❖ נבחר מתוכן רק את המילים שמתחלות באות גדולה
- ❖ ניקח רק את האות הראשונה
- ❖ לחבר אותן `tuple`

## פתרון בעיה שנייה

- ❖ תחילה נזכיר פונקציה `(split()` אשר מקבלת מחרוזת ותו, ומחזירה רשימה של תת-מחרוזות המופרדות לפי התו הנתון:

```
>>> tuple('Spaces between words'.split())
('Spaces', 'between', 'words')
```

- ❖ נגידר 2 פונקציות – אחת לחילוץ אות ראשונה במחרוזת ואחת לבדיקה האם מילה מתחילה באות גדולה

```
>>> def first(s):
    return s[0]
>>> def iscap(s):
    return len(s) > 0 and s[0].isupper()
```

# פתרון בעיה שנייה

❖ CUT ניתן למש פונקציה אשר יוצרת ראשי תיבות:

```
>>> def acronym(name):
        """Return a tuple of the letters that form the acronym for name."""
        return tuple(map(first, filter(iscap, name.split())))
>>> acronym('Sami shamoon Collage of Engineering Department of Software Engineering')
('S', 'C', 'E', 'D', 'S', 'E')
```

# Generator expressions

❖ שפת פיתון תומכת בגישה נוספת לעיבוד רצפים, הנקראת Generator expressions.

❖ גישה זו מספקת פונקציונליות דומה לגישה הקודמת אך דורשת הגדרה של פחות פונקציות

❖ ביטויים אלה משלבים את עקרונות המיפוי והפילטור באופן הבא:

<map expression> for <name> in <sequence expression> if <filter expression>

ע"מ להעיר את הביטוי השפה מעריצה את ה`sequence expression` אשרמחזיר ערך iterable (שניתן לבצע עליו איטרציה). לאחר מכן, לפי הסדר כל אלמנט נקשר ל-<name> (שם של משתנה), פונקציית הפילטור מוערכת עליו, ואם מוחזרת אמת פונקציית המיפוי מחושבת. הערך הסופי של הביטוי גם הוא iterable וניתן להעבירה לפונקציות כגון `tuple`, `sum`, `max`, `min`.

```
>>> def sum_even_fibs(n):
```

```
    return sum(fib(k) for k in range(1, n+1) if fib(k) % 2 == 0)
```

```
>>> def acronym(name):
```

```
    return tuple(w[0] for w in name.split() if iscap(w))
```

# Reduce.

- ❖ בדוגמה הקודמת השתמשנו בפונקציות ספציפיות (tuple או sum) כדי לזכור את התוצאות. שפות תכונות פונקציונליות (כמו פיתון) כוללות בתוכן פונקציות צבירה מסדר גבולה כלליות הנកראות בשנות רבים. שפת פיתון כוללת את הפונקציה `reduce` תחת הספרייה `functools`, אשר סוכמת הפעולות של פונקציה מקבלת 2 ארגומנטים על אלמנטים של רצף משמאל לימין, עד לצמצומו לערך בודד.  
הקוד הבא מחשב 5 עצרת:

```
>>> from operator import mul  
>>> from functools import reduce  
>>> reduce(mul, (1, 2, 3, 4, 5))
```

# Reduce.

- ❖ בעזרת שימוש בגיישה כללית לצבירה זו, ניתן לחשב את מכפלת כל האיברים הזוגיים בסדרת פיבונacci ולא רק את הסכום:

```
>>> def product_even_fibs(n):
    """Return the product of the first n even Fibonacci numbers, except 0."""
    return reduce(mul, filter(iseven, map(fib, range(2, n+1))))
```

```
>>> product_even_fibs(20)
```

```
123476336640
```

או להגדיר פונקציית צבירה כללית:

```
>>> def reduce_even_fibs(n,f):
    return reduce(f, filter(iseven, map(fib, range(2, n+1))))
```

```
>>> reduce_even_fibs(20,lambda x,y:x*y) #mul
```

```
123476336640
```

```
>>> reduce_even_fibs(20,lambda x,y:x+y) #sum
```

```
3382
```