w08-Lec2

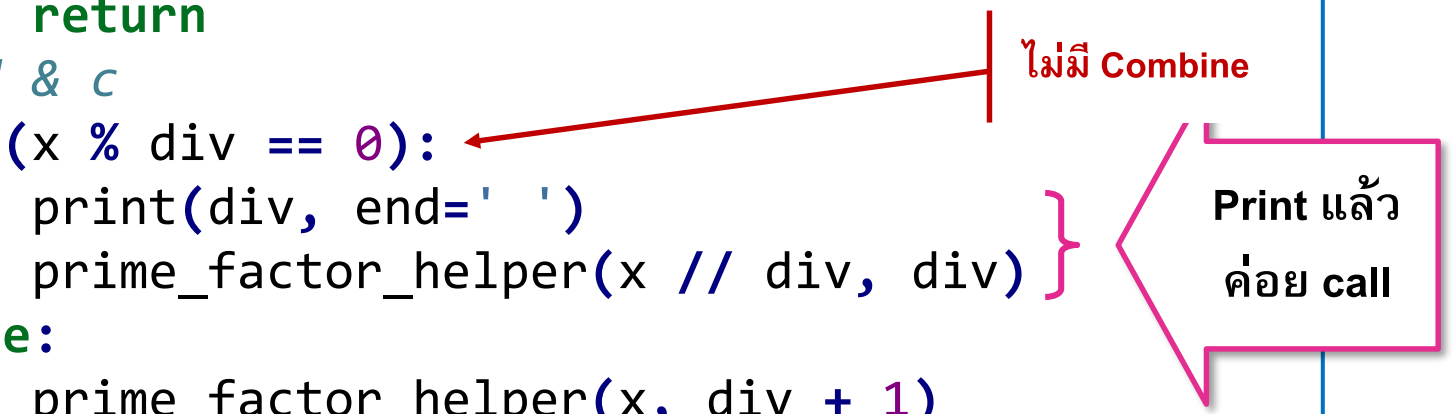
Recursion Part II

for 204111

Kittipitch Kuptavanich

Example 4: Prime Factor [key]

```
02 def prime_factor(x):
03     prime_factor_helper(x, 2)
04
05
06 def prime_factor_helper(x, div):
07     # base case
08     if div > x ** 0.5:
09         print(x)
10         return
11     # d & c
12     if (x % div == 0):
13         print(div, end=' ')
14         prime_factor_helper(x // div, div)
15     else:
16         prime_factor_helper(x, div + 1)
```



Tail vs Head Recursions

- เราเรียก recursion ที่มี recursive call อยู่ส่วนหลังของ function ว่า tail recursion
  - Tail recursion มีลักษณะคล้าย loop
    - ทำงานส่วนของตัวเองก่อน แล้วส่งให้เพื่อนทำ
      - myworkLoad ก่อน แล้วค่อย call recurse(smallerArguments)
    - ไม่จำเป็นต้องรอผล return จากเพื่อนค่อยตัดสินใจ
      - ไม่ต้องรอ answerFromSubproblem เพื่อมา combine

Tail vs Head Recursions [2]

- ถ้า recursion อยู่ส่วนต้นของ function เราเรียก recursion แบบนี้ว่า head recursion
  - แบ่งงานให้เพื่อนก่อน แล้วต้องรอผลเพื่อมา combine
    - ต้องนำผล จาก `recurse(smallerArguments)` มา combine ถึง return ได้

Tail vs Head Recursions [3]

Tail Recursion

```
def tail(n):

    if (n == 1):
        return

    print(n):

    tail(n-1)
```

Head Recursion

```
def head(n):

    if (n == 1)
        return

    head(n-1)

    print(n)
```

ผลลัพธ์ของการ traverse list โดยใช้วิธี head vs tail?

Tail vs Head Recursions [4]

```
02 def dgt_print_h1(n):
03     if n == 0:
04         return
05
06
07
08     dgt_print_h1(n//10)
09     print(n%10, end=" ")
```

```
02 def dgt_print_t1(n, d=None):
03     if dgts is None:
04         dgts = int(log10(n))
05
06     if dgts < 0:
07         return
08
09     div = 10**dgts
10     print(n//div, end=" ")
11     dgt_print_t1(n%div, dgts-1)
```

```
>>> from math import log10
>>> num = 345
>>> digit_print_h1(num)
>>> print("\n-------")
>>> digit_print_t1(num)
```

```
3 4 5
-------
3 4 5
```

Note: `math.log10` in CPython has a problem in the numbers outside the range [-999999999999997, 999999999999997],

Tail vs Head Recursions [5]

```
02 def dgt_print_h2(n, dgts=None):
03     if dgts is None:
04         dgts = int(log10(n))
05
06     if dgts < 0:
07         return
08
09     div = 10**dgts
10     dgt_print_h2(n%div, dgts-1)
11     print(n//div, end=" ")
```

```
02 def dgt_print_t2(n):
03     if n == 0:
04         return
05
06
07
08     print(n%10, end=" ")
09     dgt_print_t2(n//10)
10
```

```
>>> from math import log10
>>> num = 345
>>> digit_print_h2(num)
>>> print("\n-------")
>>> digit_print_t2(num)
```

```
3 4 5
-------
5 4 3
```

Note: `math.log10` in CPython has a problem in the numbers outside the range [-999999999999997, 999999999999997],

Example 5: String Traversing

- ฟังก์ชันด้านล่างนับจำนวนครั้งที่อักขระ *key* ปรากฏใน String *word*

```
02 def count_letter(word, key):
03     if word == '':
04         return 0
05
06     count_tail = count_letter(word[1:], key)
07
08     if word[0] == key:
09         return count_tail + 1
10     return count_tail
11
12 print(count_letter('banana', 'a'))
```

Example 6: in_both()

- ฟังก์ชันด้านล่างแสดงผลอักขระที่ซ้ำใน String *word1* และ *word2*

```
02 def in_both(word1, word2):
03     if word1 == '':
04         return
05
06     if word1[0] in word2:
07         print(word1[0])
08     in_both(word1[1:], word2)
```

- เมื่อเปรียบเทียบ String 'apple' และ 'orange'

```
>>> in_both('apples', 'oranges')
a
e
s
```

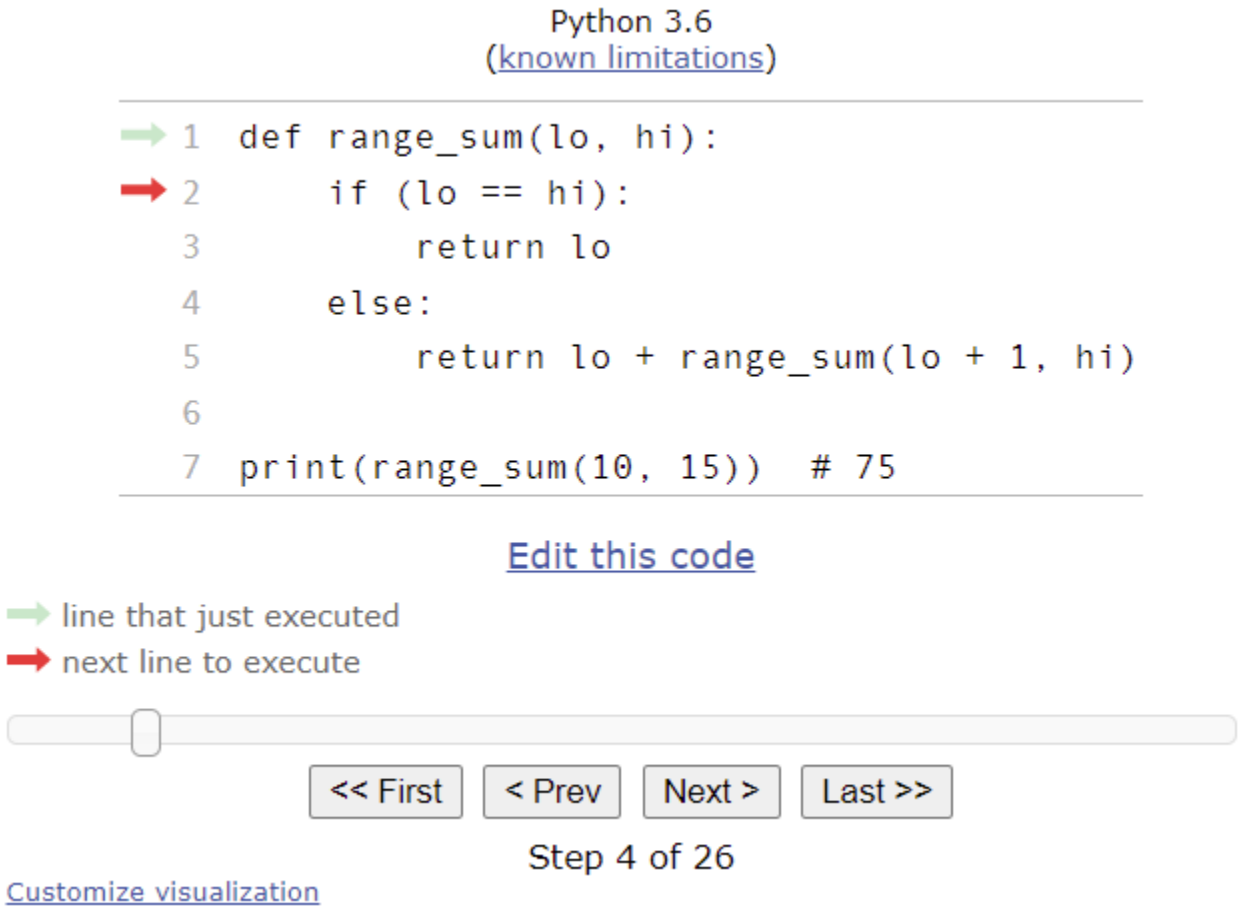
http://www.kosbie.net/cmu/fall-12/15-112/handouts/notes-recursion/notes-recursion.html

Practice 0: Range Sum

```
03 def range_sum(lo, hi):
04     if lo == hi:
05         return ____________________________
06     else:
07         return ____________________________
08
09 print(range_sum(10, 15))  # 75
```

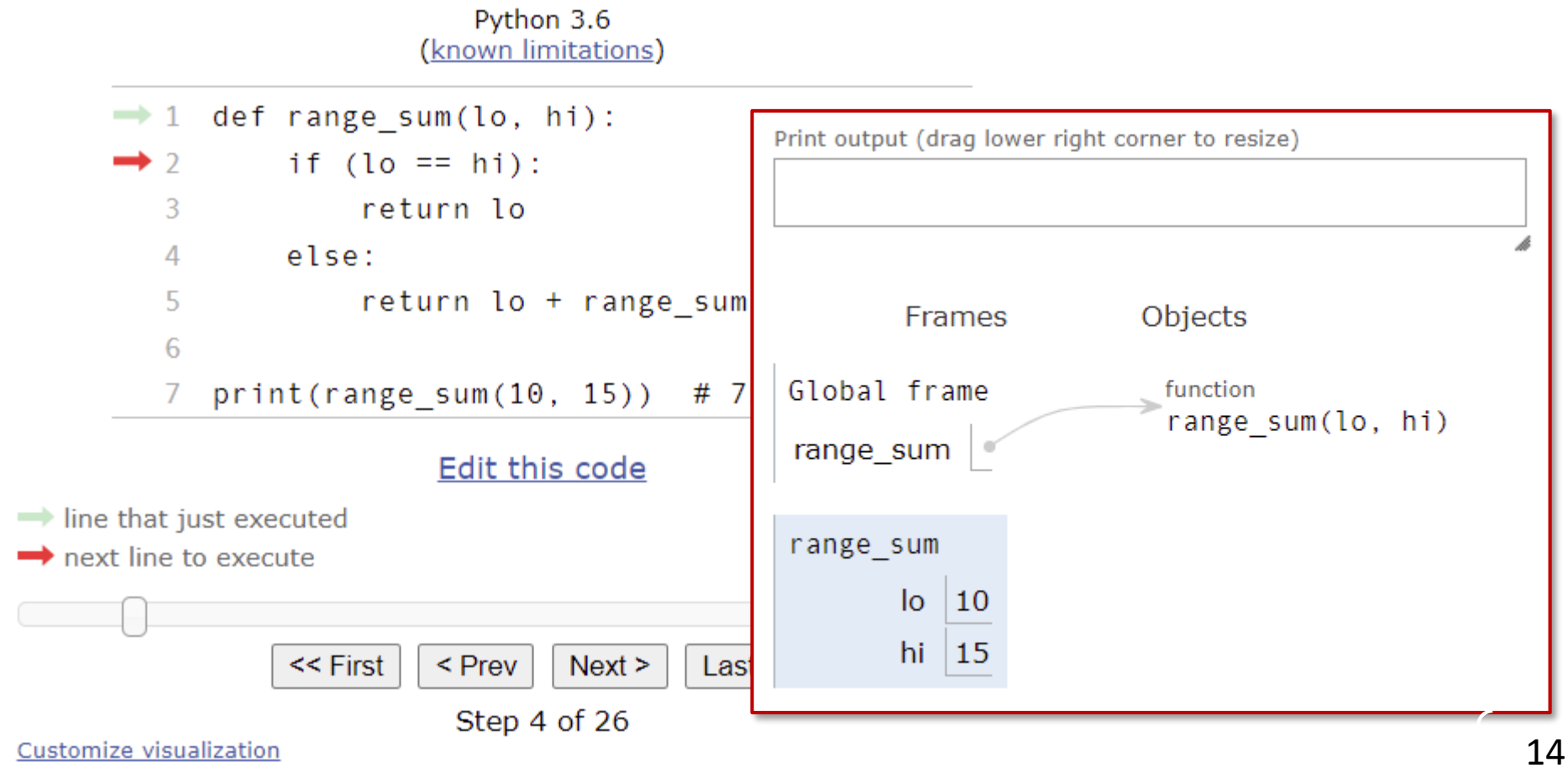
Visualize code in Python

- https://pythontutor.com/render.html **Demo:** 



Visualize code in Python [2]

- https://pythontutor.com/render.html **Demo:**



http://www.kosbie.net/cmu/fall-12/15-112/handouts/notes-recursion/notes-recursion.html

Practice 1: Digit Count

```
03 def digit_count(n):
04     if n == 0:
05         return n
06     else:
07         return 1 + digit_count(x//10)
08
09 print(digit_count(1015))  # 4
```

http://www.kosbie.net/cmu/fall-12/15-112/handouts/notes-recursion/notes-recursion.html

Practice 2: Power

```
02 def power(base, exp):
03     # assume exp is non-negative integer
04     if exp == 0:
05         return 1
06     else:
07         return base * power(base, exp-1)
08
09 print(power(2, 5))  # 32
```

ตัวท้ายวนลูป

ตามสูตร power → base ** exp

http://www.kosbie.net/cmu/fall-12/15-112/handouts/notes-recursion/notes-recursion.html

Practice 3: List Sum

```
09 def list_sum(list_a):
10     if len(list_a) == 0:
11         return 0
12     else:
13         return list_a[0] + list_sum(list_a[1:])
14
15 print(list_sum([2, 3, 5, 7, 11]))  # 28
```

http://www.kosbie.net/cmu/fall-12/15-112/handouts/notes-recursion/notes-recursion.html

Practice 4: Interleave

```
02 def interleave(list1, list2):
03     # assuming the length are the same
04     if not list1:
05         return
06     else:
07         return
08
```

[] list2

[list1[0] + list2[0] + interleave(list1[1:], list2[1:])

```
>>> print(interleave([1, 2, 3], ['a', 'b', 'c']))
[1, 'a', 2, 'b', 3, 'c']
```

Practice 5: Vowel Count

```
02 def vowel_count(word):
03     if not word:
04         return 0
05     elif word[0].lower() in 'aeiou':
06         return 1 + vowel_count(word[1:])
07     else:
08         return vowel_count(word[1:])
09
10 print(vowel_count('happy birthday')) # 3
```

- เราสามารถเข้าถึงอักขระแต่ละตัวใน **String** ได้ด้วยเครื่องหมาย **Bracket** `[]` เช่น `a[1]` (อักขระตัวแรกมี **Index** เท่ากับ **0**)

```
>>> fruit = 'banana'
>>> print(fruit[0]) # b
```

Practice 6: Sequence

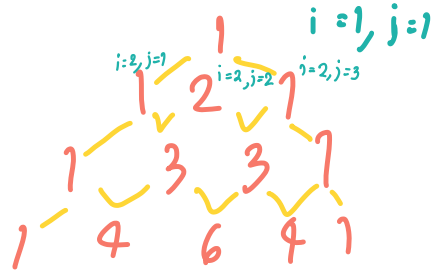
- พจน์ที่ k ของ Sequence a มี Definition ดังนี้

$$a_k = \begin{cases} 2, & k = 1 \\ a_{k-1} + 2k, & k > 1 \end{cases}$$

- ให้เขียนฟังก์ชัน Recursive `term_k(`k`)` เพื่อคำนวณค่าพจน์ a_k

```
02 def term_k(k):
03     if k == 1:
04         return [2]
05     else
06         return [2*k] + term_k(k-1)
```

Practice 7: Pascal



- ในแถว (row) ใด ๆ พจน์ที่ 0 จะมีค่าเท่ากับ 1
- ในแถวที่ `i` และพจน์ที่ `j` ใด ๆ พจน์ที่ `j` จะมีค่าเป็น 1 หาก `j` เท่ากับ `i` (สังเกตแนวทแยงมุม)
- ในกรณีอื่น ๆ ในแถวที่ `i` พจน์ ที่ `j` ใด ๆ จะมีค่าเท่ากับ ผลบวกของพจน์ที่ `j` และพจน์ที่ `j` - 1 ในแถวก่อนหน้า (แถวที่ `i` - 1)

$f(i,j) =$ [ ]

- กรณีอื่น ๆ ให้มีค่าเป็น 0

j

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
[0]	1	0	0	0	0	0	0	0
[1]	1	1	0	0	0	0	0	0
[2]	1	2	1	0	0	0	0	0
[3]	1	3	3	1	0	0	0	0
[4]	1	4	6	4	1	0	0	0
[5]	1	5	10	10	5	1	0	0
[6]	1	6	15	20	15	6	1	0
[7]	1	7	21	35	35	21	7	1

i

Practice 7: Pascal [2]

```
02 def pascal(i, j):
03     if j == 0 or j == i:
04         return 1
05     else:
06         return i[j]+i[j-1] + pascal(i-1, j-1)
07
08
09 print(pascal(7, 3))  # 35
```

- ให้เขียนฟังก์ชัน **Recursive pascal(i, j)** เพื่อหาค่าของพจน์ที่ j ในแถวที่ i

Practice 8: Digit Sum

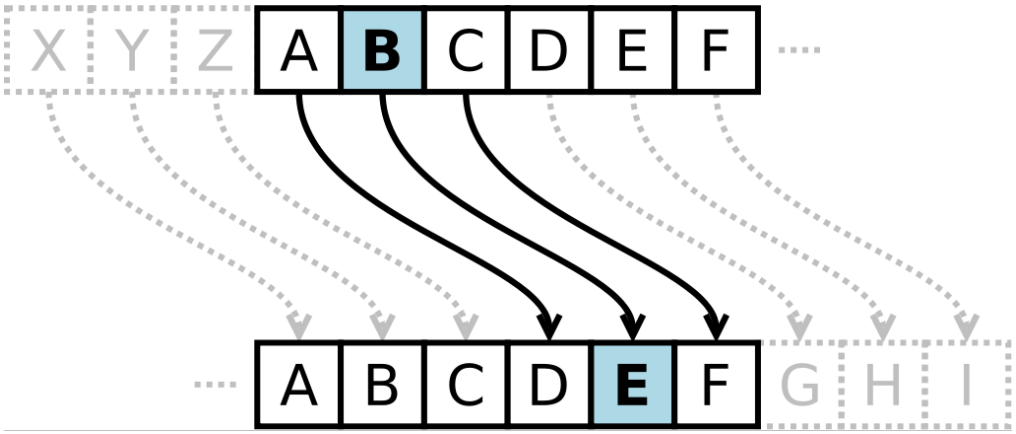
HW02_1

```
08 def digit_sum(n):
09     if _____:
10         return _____
11     else:
12         return _____
13
14 print(digit_sum(1027))    # 10
```

https://th.wikipedia.org/wiki/รหัสซีซาร์

Practice 9: Caesar Cipher

- **Caesar cipher หรือ Caesar shift เป็นเทคนิคการเข้ารหัสที่ง่ายและแพร่หลายที่สุด โดยใช้หลักการแทนที่ตัวอักษร ซึ่งในแต่ละตัวอักษรที่อยู่ในข้อความจะถูกแทนที่ด้วยตัวอักษรที่อยู่ลำดับถัดไปตามจำนวนตัวอักษรที่แน่นอน**



Practice 9: Caesar Cipher [2]

```
02 import string
03 def caesar_cipher(word, shift):
04     if word == "":
05         print()
06         return
07
08     alpha = string.ascii_lowercase
09     key = alpha[shift:] + alpha[0:shift]
10
11     ____________________________________________
12     ____________________________________________
13     caesar_cipher(word[1:], shift)
14
15 caesar_cipher("happy", 3)       # kdssb
```

Practice 10: Caesar Decoding

```
02 def caesar_cipher(word, shift, encode=True):
03     if word == "":
04         print()
05         return
06     if not encode:
07         ____________________________________________
08
09     alpha = string.ascii_lowercase
10     key = alpha[shift:] + alpha[0:shift]
11
12     ____________________________________________
13     ____________________________________________
14     caesar_cipher(word[1:], shift)
15
16 caesar_cipher("kdssb", 3, False) # happy
17
```

Reference

- **http://www.kosbie.net/cmu/fall-12/15-112/handouts/notes-recursion/notes-recursion.html**