01076103, 01076104

Programming Fundamental

Programming Project

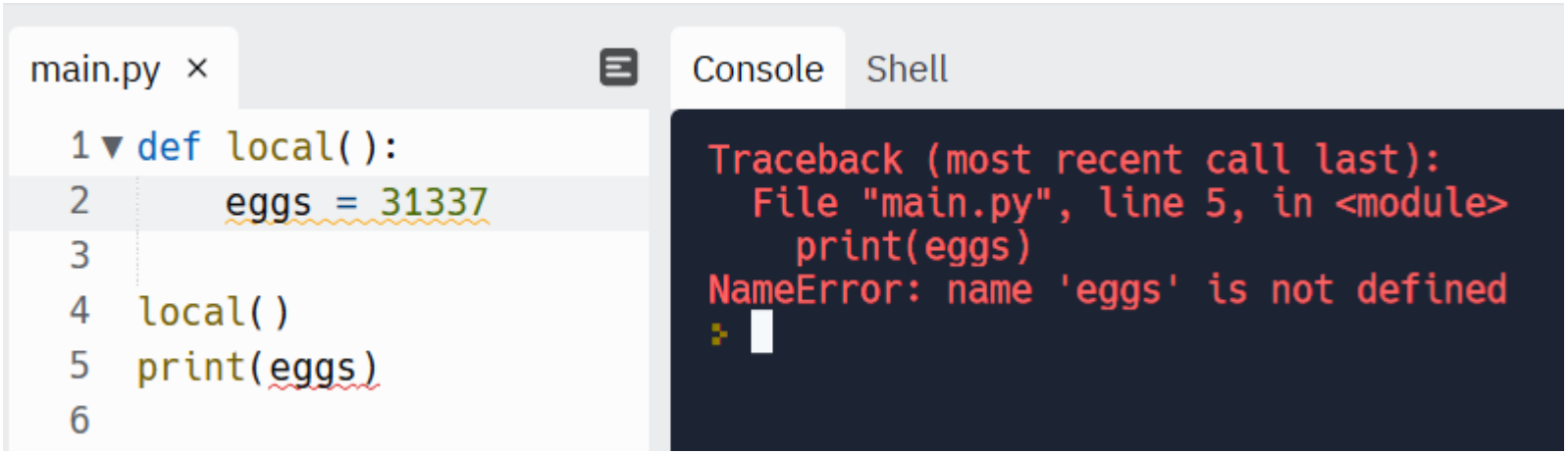
Variable Scope, Recursion, List Comprehension

Variable Scope

- ในการใช้ฟังก์ชัน จะมีเรื่องหนึ่งที่ต้องพิจารณา คือ ขอบเขตที่ใช้งานได้ของแต่ละตัวแปร
- ตัวแปร หรือ argument ของฟังก์ชันจะเรียกว่า local scope ส่วนตัวแปรที่กำหนดไว้นอกฟังก์ชัน จะเรียกว่า global scope
- ให้มองว่า scope ก็เหมือนกับกล่อง ตัวแปรที่อยู่ในกล่อง จะเกิดขึ้นมาได้ก็ต้องมีกล่องเสียก่อน จากนั้นจึงสร้างตัวแปร ดังนั้นตัวแปรที่อยู่ในฟังก์ชัน ก็จะสร้างขึ้นตอนที่ฟังก์ชันถูกเรียก และเมื่อ return ก็เหมือนกับกล่องถูกทำลาย ตัวแปรในกล่องก็จะถูกทำลายไปด้วย
- คำว่า local แปลว่าใกล้ๆ ดังนั้นโปรแกรมที่อยู่ภายนอกฟังก์ชัน จะไม่สามารถเข้าถึงข้อมูลที่อยู่ในฟังก์ชัน แต่โปรแกรมในฟังก์ชัน สามารถอ้างถึงตัวแปรที่อยู่นอกตัวได้

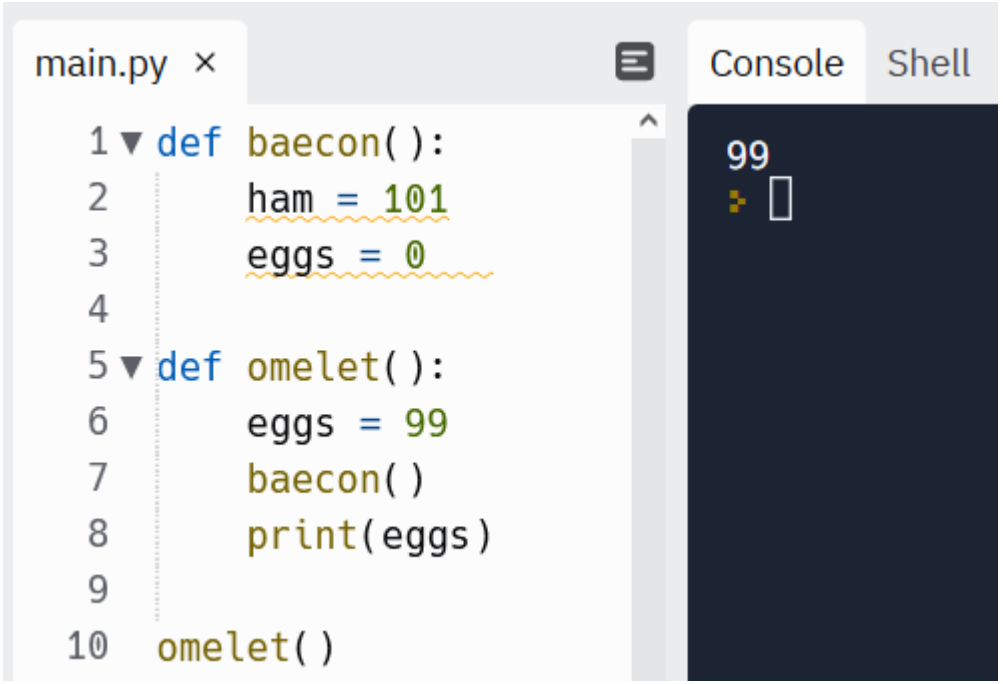
Variable Scope

- โปรแกรมที่อยู่ภายนอกฟังก์ชัน จะไม่สามารถเข้าถึงข้อมูลที่อยู่ในฟังก์ชัน ตามตัวอย่าง



Variable Scope

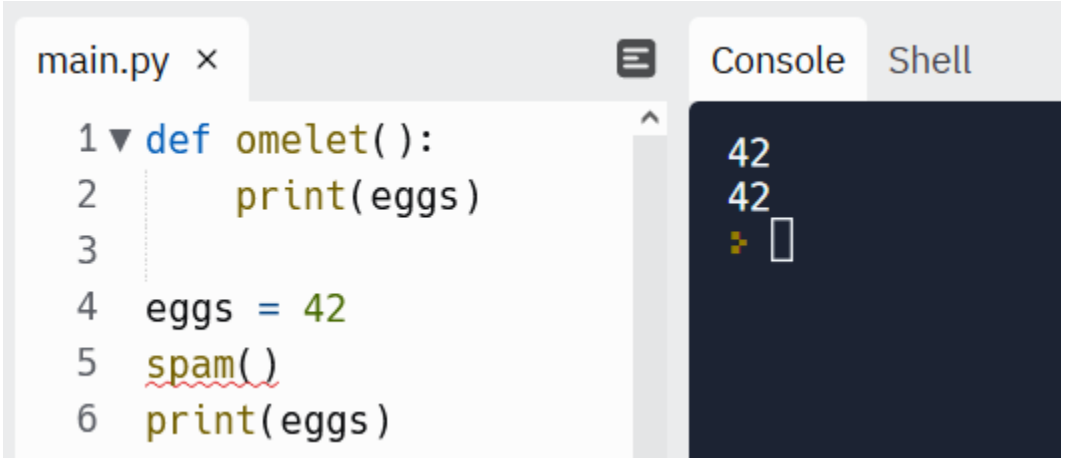
- โปรแกรมที่อยู่คนละ variable scope ถือเป็นคนละตัวแปรกัน แม้จะมีชื่อเดียวกัน
- ตัวแปร eggs กำหนดใน omelet ต่อมา omelet เรียก beacon ทำให้มีการสร้าง eggs ขึ้นมาอีกตัวหนึ่ง กำหนดค่า = 0
- แต่เมื่อกลับมา จะพบว่าตัวแปร eggs มีค่าเท่าเดิม
- เพราะตัวแปร eggs ทั้ง 2 ตัว ถือเป็นคนละตัวกัน เพราะ local คือ เฉพาะขอบเขตนั้น



https://autbor.com/otherlocalscopes/.

Variable Scope

- ตัวแปรที่กำหนดไว้ใน global scope สามารถอ้างถึงโดย local ได้
- จะเห็นว่าตัวแปร eggs มีการกำหนดไว้ที่ global scope
- แต่ในฟังก์ชัน omelet ซึ่งไม่มีการกำหนดตัวแปรนี้ไว้ ก็สามารถอ้างถึงได้เช่นกัน



Variable Scope

- กรณีที่ local scope และ global scope มีชื่อเดียวกัน ถือเป็นตัวแปรคนละตัวกัน

main.py

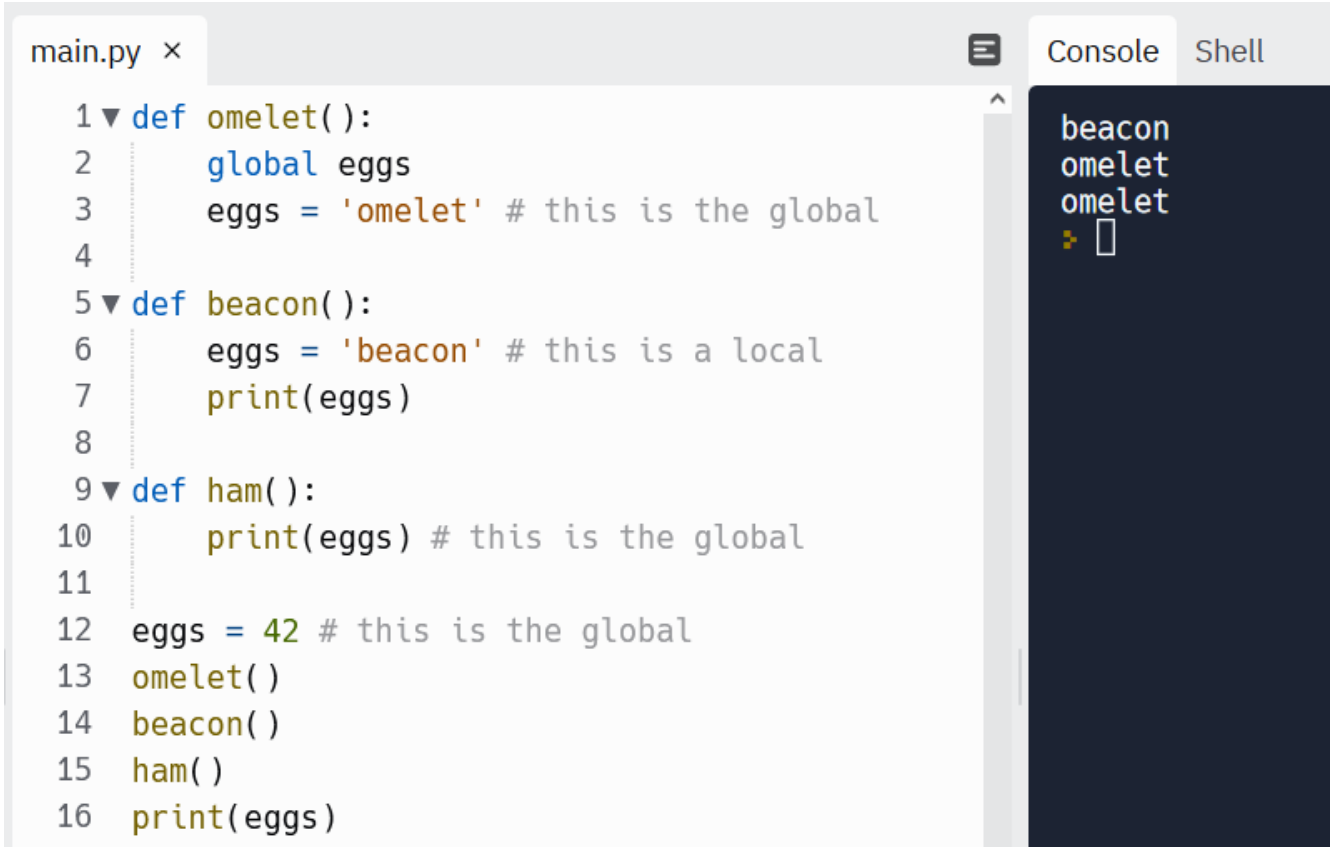
```
1  def omelet():
2      eggs = 'omelet local'
3      print(eggs)
4
5  def beacon():
6      eggs = 'beacon local'
7      print(eggs)
8      omelet()
9      print(eggs)
10
11 eggs = 'global'
12 beacon()
13 print(eggs)
```

Console Shell

```
beacon local
omelet local
beacon local
global
```

Variable Scope

- แต่เราสามารถกำหนดให้ตัวแปรแบบ local scope ให้เห็นในระดับ global ได้โดยคำสั่ง global



Variable Scope

- สรุปหลักการ
- LEGB
  - Local: ถ้าตัวแปร x อยู่ภายใน function โปรแกรม python จะใช้ตัวแปรใน function (local)
  - Enclosing: ถ้าตัวแปร x ไม่อยู่ใน local scope แต่พบใน function ที่อยู่ใน function ด้านนอก, โปรแกรม python จะใช้ในตัวแปรใน enclosing function's scope.
  - Global: ถ้าตัวแปร x ไม่อยู่ทั้งใน local scope และ enclosing function's scope โปรแกรม python จะค้นหาใน global เป็นลำดับต่อไป
  - Built-in: ถ้าไม่พบตัวแปร x ในที่ใดๆ โปรแกรม python จะพยายามหาใน built-in scope

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py

```
1  #| Example 1: Single Definition
2
3  x = 'global'
4  def f():
5      def g():
6          print(x)
7      g()
8
9  f()
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py

```
#| Example 2: Double Definition

x = 'global'
def f():
    x = 'enclosing'

    def g():
        print(x)

    g()

f()
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py

```
1  #| Example 3: Triple Definition
2
3  x = 'global'
4  def f():
5      x = 'enclosing'
6      def g():
7          x = 'local'
8          print(x)
9      g()
10
11 f()
12
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  #| Example 5: Local และ Enclosing Name Space
2
3  def f():
4      print('Start f()')
5      def g():
6          print('Start g()')
7          print('End g()')
8          return
9      g()
10     print('End f()')
11     return
12
13 f()
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  #| Example : หากแก้ไขตัวแปรแบบ Global ใน function จะเกิดอะไรขึ้น
2  var = 100  # A global variable
3  def increment():
4      var = var + 1  # Try to update a global variable
5
6  increment()
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  #| Example : กรณีกำหนดตัวแปร Local แล้วอ้างถึง
2  var = 100  # A global variable
3  def func():
4      print(var)  # Reference the global variable, var
5      var = 200   # Define a new local variable using the same name, var
6
7  func()
8
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  #| Example 10: การอ้างถึงตัวแปร Global ภายใน function
2  counter = 0  # A global name
3  def update_counter():
4      counter = counter + 1  # Fail trying to update counter
5
6  update_counter()
7
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  counter = 0  # A global name
2
3  def update_counter():
4      global counter  # Declare counter as global
5      counter = counter + 1  # Successfully update the counter
6
7  update_counter()
8  print (counter)
9  update_counter()
10 print (counter)
11 update_counter()
12 print (counter)
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  global_counter = 0  # A global name
2
3  def update_counter(counter):
4      return counter + 1  # Rely on a local name
5
6  global_counter = update_counter(global_counter)
7  print (global_counter)
8  global_counter = update_counter(global_counter)
9  print (global_counter)
10 global_counter = update_counter(global_counter)
11 print (global_counter)
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  #| Example 12: การสร้างตัวแปร Global จากภายใน function
2  def create_lazy_name():
3      global lazy  # Create a global name, lazy
4      lazy = 100
5      return lazy
6
7  create_lazy_name()
8  print(lazy)  # The name is now available in the global scope
9
```

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py

```
1 def func():
2     var = 100  # A nonlocal variable
3     def nested():
4         nonlocal var  # Declare var as nonlocal
5         var += 100
6     nested()
7     print(var)
8 func()
```

Function

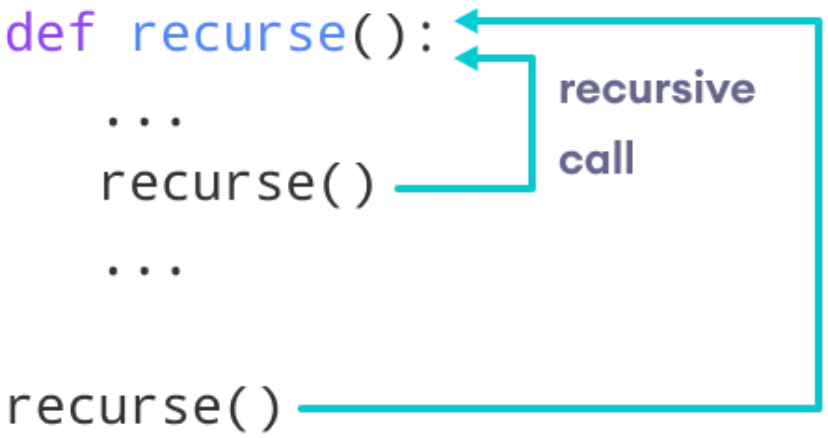
- **Ex. 6.1** ให้เขียนฟังก์ชันชื่อ test_ pangram เพื่อตรวจสอบว่า String ที่รับเข้ามาเป็น pangram หรือไม่ โดย pangram คือ string ที่ประกอบด้วยตัวอักษรทุกตัวในภาษาอังกฤษ (ต้องมีอักษรแต่ละตัวปรากฏอย่างน้อย 1 ครั้ง) เช่น

  The quick brown fox jumps over the lazy dog

- รับข้อมูลเป็น string และ return เป็น True หรือ False

Recursion

- เป็นวิธีการทางโปรแกรมเพื่อใช้ในการแก้ปัญหาบางแบบ โดยการเรียกใช้ฟังก์ชันเดียวกับฟังก์ชันที่เรียก อาจเรียกว่า “การเรียกตัวเอง” หรือ การเรียกซ้ำ
- รูปแบบการใช้งาน Recursion จะเป็นไปตามรูป คือ ภายในฟังก์ชันเองมีการเรียกชื่อฟังก์ชันตัวเอง



Recursion

- ตัวอย่างที่นิยมใช้ในการอธิบายการทำงานของ Recursion คือ โปรแกรมที่คำนวณค่า Factorial
- องค์ประกอบที่สำคัญของ recursive function คือ 1) ต้องมีจุดที่วนกลับ 2) ต้องมีการเรียกตนเอง

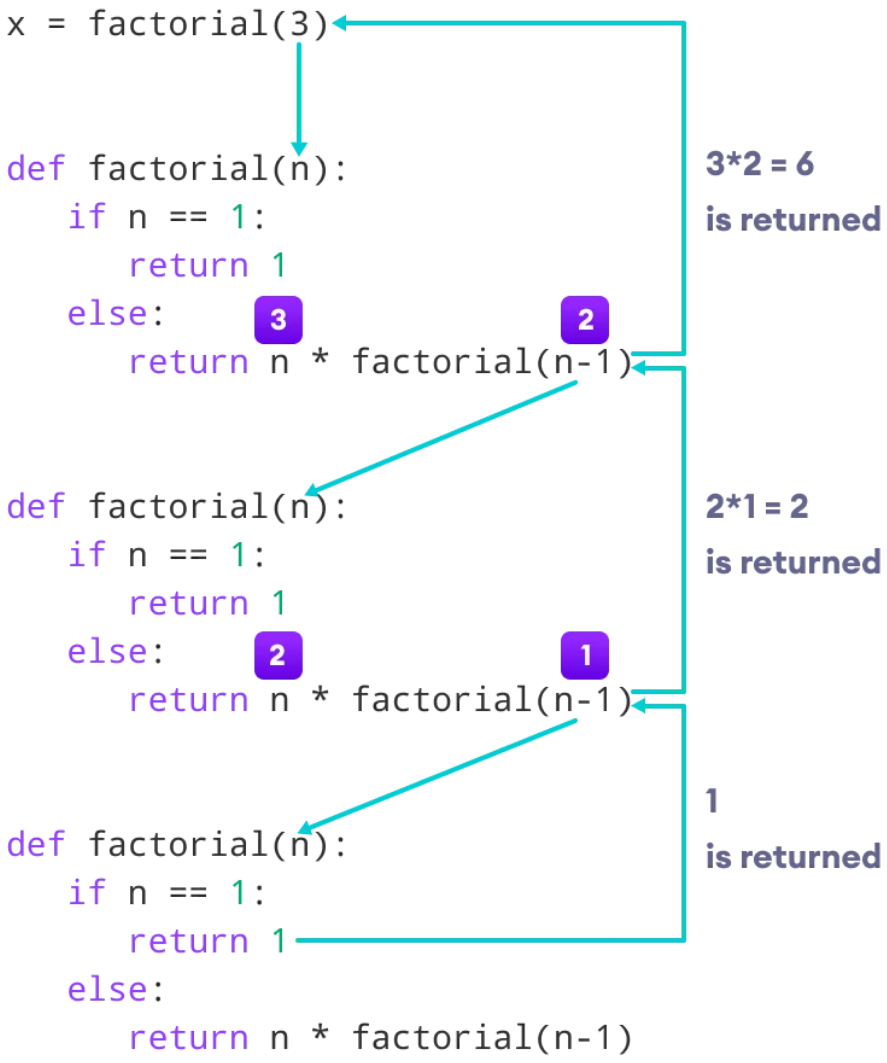
```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

Recursion

- factorial(3)
  - เรียกใช้ครั้งแรกด้วย 3
- 3 * factorial(2)
  - เรียกใช้ครั้งที่ 2 ด้วย 2
- 3 * 2 * factorial(1)
  - เรียกใช้ครั้งที่ 3 ด้วย 1
- 3 * 2 * 1
  - กลับจากเรียกครั้งที่ 3 ด้วยค่า 1
- 3 * 2
  - กลับจากการเรียกครั้งที่ 2 ด้วยค่า 2
- 6
  - กลับจากการเรียกครั้งที่1 ด้วยค่า 6



Recursion

- ตัวอย่างนี้เป็นการเขียน function power ซึ่งเป็นฟังก์ชันยกกำลัง เช่น power(4,7) คือ 4 คูณกัน 7 ครั้ง
- จะเห็นว่า recursive คือ การวนลูปแบบหนึ่ง โดยมีเงื่อนไขของการย้อนกลับ และ การสะสมตัวงานไปเรื่อยๆ

```
def power(num, topwr):
    if topwr == 0:
        return 1
    else:
        return num * power(num, topwr - 1)


print(f'4 to the power of 7 is {power(4, 7)}')
print(f'2 to the power of 8 is {power(2, 8)}')
```

Recursion

- อีกตัวอย่างหนึ่ง เป็นอนุกรม Fibonacci เปรียบเทียบระหว่างเขียนด้วย for
- เป็นอนุกรมที่อนุกรมถัดไป เกิดจากตัวเลข 2 ตัวข้างหน้าบวกกัน เราจึงนำ list เข้ามาช่วยเก็บค่าของอนุกรมตัวก่อนหน้า (ถ้าไม่ใช้ list ต้องใช้ตัวแปรมาเก็บอีก 2 ตัว)

```python
def fibonacci(n):
    sequence = []
    if n == 1:
        sequence = [0]
    else:
        sequence = [0,1]
        for i in range(1, n-1):
            sequence.append(sequence[i-1] + sequence[i])
    return sequence

print(fibonacci(10))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Recursion

- แต่เมื่อเขียนในแบบ recursive จะเห็นว่าโปรแกรมจะคล้ายกัน เพียงแต่เปลี่ยนจากการวน loop เป็นการเรียกตัวเองซ้ำเท่านั้น

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

- อย่างไรก็ตาม function นี้จะส่งค่า Fibonacci ลำดับที่ n กลับมาเท่านั้น หากจะพิมพ์ทั้งอนุกรม หรือ เก็บใน list จะต้องเขียนโปรแกรมเพิ่มเติม

Recursion

- โปรแกรมแสดงอนุกรม Fibonacci จำนวน 10 ตัว

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)


number = 10

print('Fibonacci sequence:')
for i in range(number):
    print(fibonacci(i),end=', ')
```

Recursion

- **ตัวอย่าง** โจทย์ที่เหมาะสมกับวิธี Recursion
- กระต่ายตัวหนึ่ง ต้องการกระโดดขึ้นบันได โดยกระต่ายสามารถกระโดดได้ครั้งละ 1 หรือ 2 ขั้นเท่านั้น ให้รับจำนวนขั้นบันได แล้วบอกว่ามีวิธีในการกระโดดกี่วิธี
- สมมติเหตุการณ์
  - กรณี บันได 1 ขั้น ได้ 1 วิธี คือ กระโดด 1 ขั้น
  - กรณี บันได 2 ขั้น ได้ 2 วิธี คือ กระโดด 1 ขั้น 2 ครั้ง และ กระโดดทีเดียว 2 ขั้น
  - กรณี บันได 3 ขั้น ได้ 3 วิธี คือ 1) กระโดด 1 ขั้น 3 ครั้ง 2) กระโดด 1 ขั้น แล้วจึงกระโดด 2 ขั้น 3) กระโดด 2 ขั้น แล้วจึงกระโดด 1 ขั้น
  - กรณี บันได 4 ขั้น จะกระโดดได้ 5 วิธี 1) ทีละขั้น 2) 1, 1, 2 ขั้น 3) 1, 2, 1 ขั้น 4) 2, 1, 1 ขั้น 5) 2, 2 ขั้น

Recursion

- สามารถเขียนเป็นตารางได้ดังนี้

จำนวนขั้น	จำนวนวิธีกระโดด
1	1
2	2
3	3
4	5
5	8
6	13
7	21

Recursion

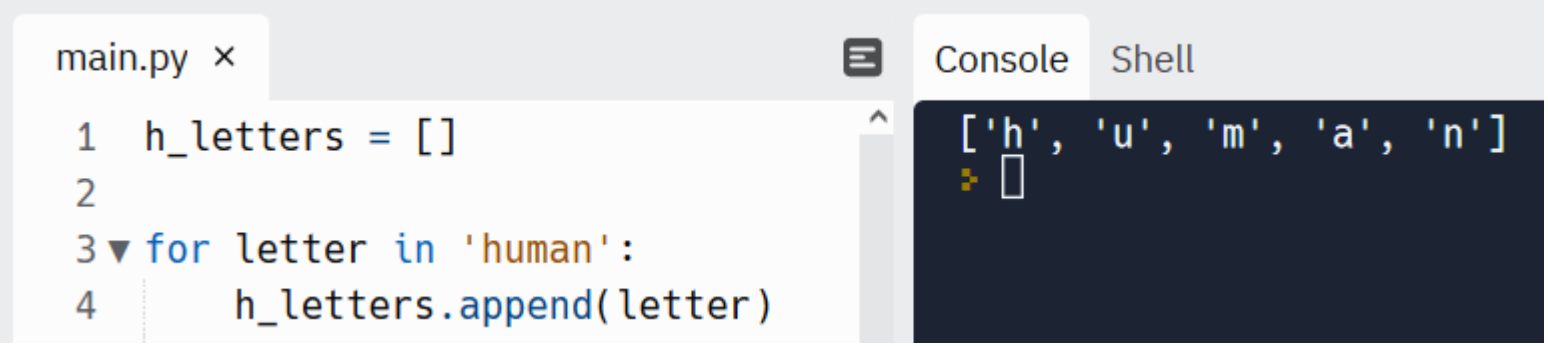
- จากตารางจะพบว่า จำนวนวิธีการกระโดดที่จำนวนขั้นที่ n จะเท่ากับผลรวมของจำนวนวิธีการกระโดดที่จำนวนขั้นที่ n-1 กับ จำนวนวิธีการกระโดดที่จำนวนขั้นที่ n-2
- jump(n) = jump(n-1)+jump(n-2) โดย n ไม่เท่ากับ 1 และ 2
- ซึ่งตรงนี้ก็จะเห็นว่า โจทย์นี้จริงๆ คือ การหา Fibonacci ลำดับที่ n นั่นเอง
- จะเห็นว่าในการมองโจทย์นั้น บางครั้งเราจะต้องจำลองการทำงานให้เห็นวิธีทำงานเสียก่อน จึงจะเขียนโปรแกรมได้

Recursion

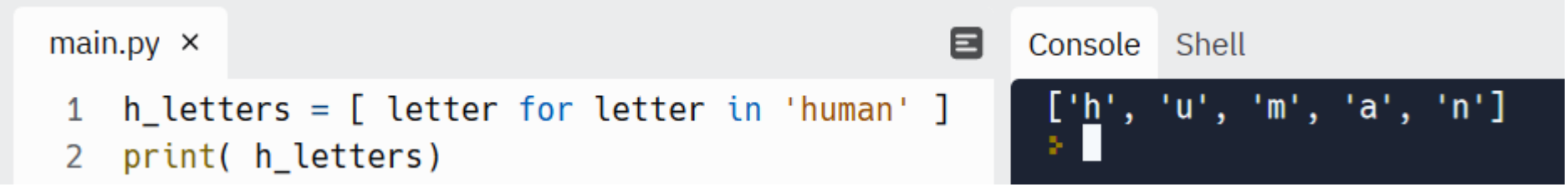
- **Exercise 6.2** จงเขียน โปรแกรมที่รวมผลลัพธ์ ของตัวเลขที่อยู่ใน List โดยกำหนดให้ฟังก์ชัน ชื่อ find_sum(n_list,n) โดย n_list เป็น List ของตัวเลขที่มีความยาว n โดยให้เขียนในแบบ Recursion

List comprehension

- เป็นวิธีการขั้นสูง ที่ทำให้เขียนโปรแกรมเพื่อสร้าง List ได้สั้นลงในบางกรณี ดูตัวอย่างโปรแกรม



- หากเขียนในแบบ List Comprehension จะเขียนได้เป็น



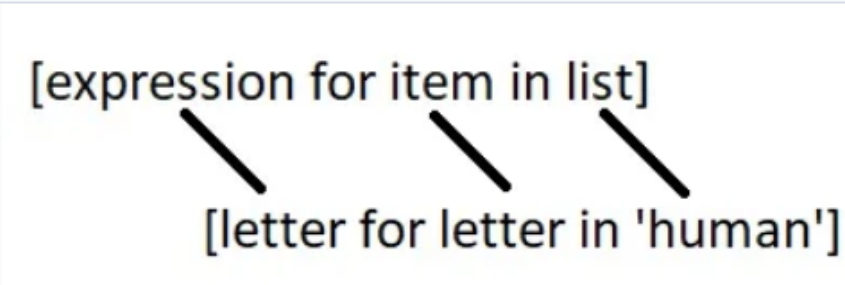
- จะเห็นว่าผลการทำงานเหมือนเดิม แต่โปรแกรมสั้นลง

List comprehension

- รูปแบบของ List Comprehension จะเริ่มด้วย expression แล้วตามด้วย for loop โดย expression จะกระทำกับแต่ละ element ใน list แล้วคืนค่ากลับมา เนื่องจากทั้ง list comprehension อยู่ใน list ดังนั้นค่าที่คืนกลับมาก็จะอยู่ในอีก list หนึ่ง

Syntax of List Comprehension

```
[expression for item in list]
```



List comprehension

```
for (set of values to iterate):
  if (conditional filtering):
    output_expression()
```

```
[ output_expression() for(set of values to iterate) if(conditional filtering) ]
```

List comprehension

- การใช้งาน List comprehension มีด้วยกัน รูปแบบแรก คือ Map โดยเป็นการกระทำกับทุกสมาชิกใน List ในรูปแบบใดรูปแบบหนึ่ง โดยหลังจากทำงาน จะมีจำนวนสมาชิกเท่าเดิม
- **ตัวอย่าง** เป็นการนำ List เดิมมายกกำลังสอง

main.py ×

```
1  square = [num**2 for num in range(1,10)]
2  print(square)
```

Console Shell

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- **ตัวอย่าง** เป็นการนำ List เดิมมาเปลี่ยนเป็นตัวใหญ่

main.py ×

```
1  fruits = ["apple", "banana", "cherry",
   "kiwi", "mango"]
2  newlist = [x.upper() for x in fruits]
3  print(newlist)
```

Console Shell

```
['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
```

List comprehension

- Map สามารถเปลี่ยนแปลงเฉพาะ สมาชิกบางตัวตามเงื่อนไขที่กำหนดได้
- กรณีนี้สามารถใช้ if ในการค้นหาสมาชิกที่จะดำเนินการตามเงื่อนไขได้
- จากตัวอย่าง เงื่อนไขที่กำหนด คือ เมื่อ x = 3 ให้เปลี่ยนเป็นคำว่า three แต่ตัวอื่นไม่ต้องเปลี่ยน

main.py ×

```
1  newlist = ["three" if x == 3 else x \
2             for x in range(1,10)]
3  print(newlist)
```

Console Shell

```
[1, 2, 'three', 4, 5, 6, 7, 8, 9]
```

List comprehension

- รูปแบบที่ 2 เรียกว่า Filter โดยเป็นการเลือกสมาชิกบางตัวตามเงื่อนไขที่กำหนด ดังนั้นผลลัพธ์จะมีจำนวนสมาชิกลดลง
- **ตัวอย่าง** เป็นการสร้าง List ของเลขคู่ จาก List ของเลขตั้งแต่ 1-20

main.py ×

```
1  number_list = [ x for x in range(20) \
2                  if x % 2 == 0]
3  print(number_list)
```

Console Shell

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- **ตัวอย่าง**นี้ จะมีหลายเงื่อนไขก็ได้ จะเห็นว่าระหว่าง if จะเหมือนกับมี and

main.py ×

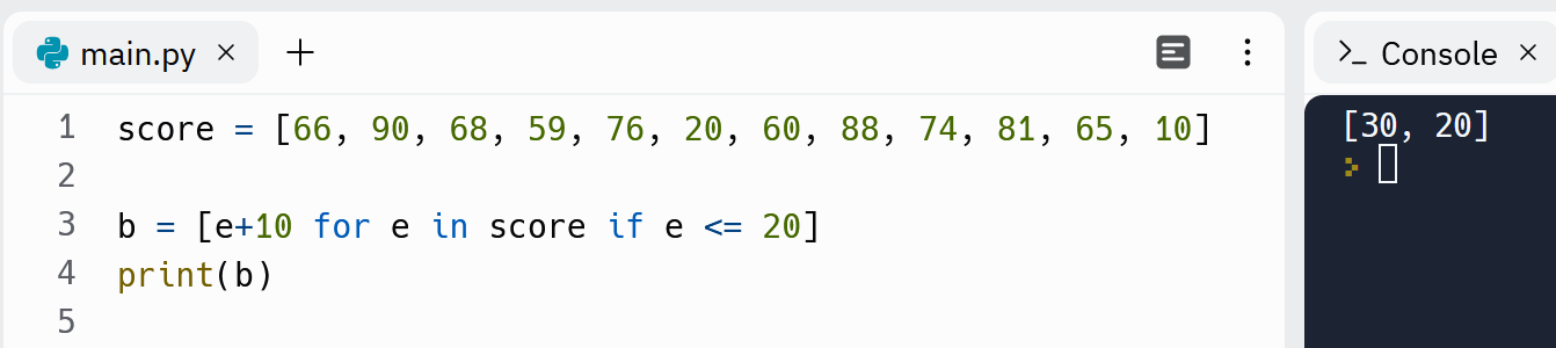
```
1  num_list = [y for y in range(100) \
2              if y % 2 == 0 if y % 5 == 0]
3  print(num_list)
```

Console Shell

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

List comprehension

- สำหรับรูปแบบที่ 3 คือ ใช้ร่วมกันทั้ง Map และ Filter
- ตัวอย่าง สมมติมี List ของคะแนน ซึ่งหากต้องการจะหาเฉพาะคนที่ได้คะแนนน้อยกว่า 20 และเพิ่มให้อีก 10 คะแนน



List comprehension

- จะเห็นว่ากรณีที่โปรแกรมไม่ซับซ้อนมากเกินไป สามารถใช้ List Comprehension ได้ แต่ข้อเสีย คือ โปรแกรมอ่านยากขึ้น ดังนั้นถ้าไม่ชำนาญควรเขียนแบบเดิม หรือใช้แต่แบบง่ายๆ

main.py ×

```
1  transposed = []
2  matrix = [[1, 2, 3, 4], [4, 5, 6, 8]]
3
4  for i in range(len(matrix[0])):
5      transposed_row = []
6
7      for row in matrix:
8          transposed_row.append(row[i])
9      transposed.append(transposed_row)
10
11 print(transposed)
```

Console Shell

```
[[1, 4], [2, 5], [3, 6], [4, 8]]
```

main.py ×

```
1  matrix = [[1, 2, 3, 4], [4, 5, 6, 8]]
2  transpose = [[row[i] for row in matrix]\
3               for i in range(4)]
4  print (transpose)
```

Console Shell

```
[[1, 4], [2, 5], [3, 6], [4, 8]]
```

List comprehension

- การใช้ List Comprehension ที่น่าสนใจ

```
x = [int(e) for e in input().split()]
```

อ่าน String ด้วย input() และแยกออกเป็น List ของ String ด้วย Split() จากนั้น
นำ String มาแปลงเป็นจำนวนเต็มและเก็บใน List ประโยชน์ คือ รับข้อมูลทีละหลายค่า

```
t = ','.join([str(e) for e in x])
```

สร้างลำดับของผลลัพธ์ที่คั่นด้วยเครื่องหมาย , โดยแปลงตัวเลขใน List เป็น String
แล้วนำไป join กันอีกที

```
c = sum( [1 for e in x if e%2==0])
```

นับว่า List มีจำนวนคู่กี่ตัว โดยสร้าง List ที่เพิ่มเลข 1 ทุกครั้งที่พบจำนวนคู่ใน List แล้ว Sum

```
b = [ (1 if x[i] >= 0 else -1) for i in range(len(x))]
```

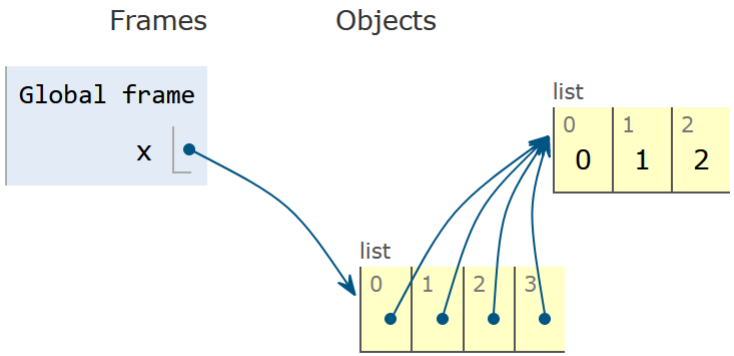
สร้าง List b จาก List x โดยถ้า x[i] >=0 ให้เป็น 1 มิฉะนั้นให้เป็น -1

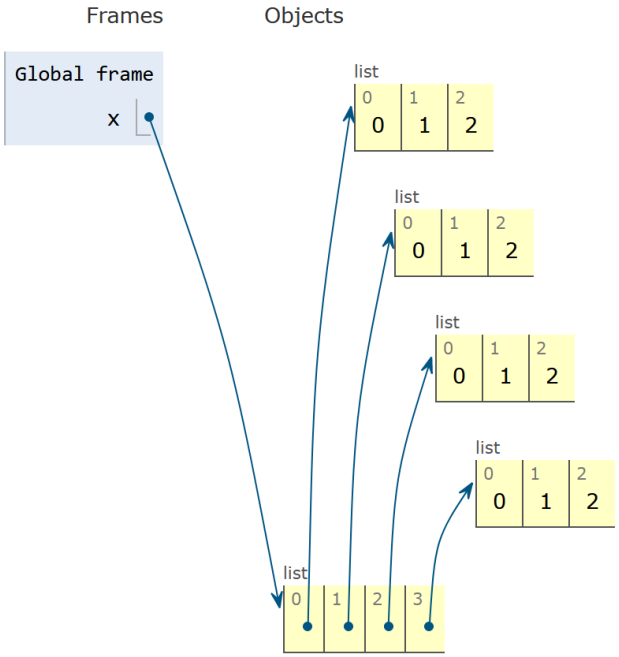
List comprehension

- คำสั่งในบรรทัดใด ต่อไปนี้ที่ต่างออกไป

```
x = [ [e for e in range(3)] for k in range(4) ]
x = [ list(range(3)) for k in range(4) ]
x = [ [0,1,2] for k in range(4) ]
x = [ [0,1,2] ] *4
print(x)
```

- แม้ว่าผลลัพธ์จะเหมือนกันก็ตามแต่การทำงานต่างกัน
- บรรทัด 1-3 จะได้รูปขวา แต่บรรทัดที่ 4 จะเป็นรูปล่าง





List comprehension

- **ตัวอย่าง** ต้องการหาตัวเลขอยู่ระหว่าง 1-100 ที่หารด้วยตัวเลขตั้งแต่ 2-9 ลงตัว

```
1  nums = range(1,100)
2  answer = [num for num in nums if True in
3            [True for divisor in range(2,10)
4             if num % divisor == 0]]
5  print(answer)
```

List comprehension

- **Exercise 6.3** กำหนดให้ List x เก็บ String และตัวแปร c เก็บตัวอักษร
- ให้สร้าง List d ที่เก็บจำนวนครั้งที่ตัวอักษรใน c ปรากฏในแต่ละ String ของ List x โดยใช้ List comprehension
  - เช่น x = [‘abba’, ‘babana’, ‘ann’]; c = ‘a’
  - จะได้ d = [2, 3, 1]
- ให้เขียนในฟังก์ชัน count_char_in_string(x,c) แล้ว return เป็น List คำตอบ

List comprehension

- **Exercise 6.4** กำหนดให้ List x เป็น List ของจำนวนเต็ม
- ให้เขียนโปรแกรมเพื่อลบจำนวนเต็มทุกตัวใน x ที่มีค่าเป็นลบ โดยใช้ List comprehension
  - เช่น x = [[1, -3, 2], [-8, 5], [-1, -4, -3]]
  - ได้คำตอบเป็น [[1, 2], [5], []]
- ให้เขียนในฟังก์ชัน delete_minus(x) แล้ว return เป็น List คำตอบ
- Hint : ถ้าคิดไม่ออกให้ทดลองเขียนแบบปกติก่อน แล้วค่อยลดรูป

List comprehension

- **Exercise 6.5** ให้เขียนโปรแกรมเพื่อรับข้อมูล 1 บรรทัด ที่ประกอบด้วยจำนวนเต็มหลายจำนวน (คั่นด้วยช่องว่าง)
- ให้ส่งคืนว่ามีจำนวนที่เป็นลบกี่จำนวน โดยใช้ List comprehension
- ให้เขียนในฟังก์ชัน count_minus(str) แล้ว return เป็นคำตอบ

List comprehension

- **Exercise 6.6** ให้เขียนโปรแกรมเพื่อรับ string 1 ตัว
- ให้ส่งคืนเฉพาะตัวอักษรที่เป็นภาษาอังกฤษ โดยใช้ List comprehension
- ให้เขียนในฟังก์ชัน only_english(string1) แล้ว return เป็นคำตอบเป็น string

List comprehension

- **Exercise 6.7** กำหนดให้ list x และ y เป็น list ของจำนวนเต็ม โดยมีขนาดเท่ากัน
- ให้ return list ที่เป็นผลบวกของ list x และ y โดยใช้ list comprehension
- เขียนให้ function ชื่อ add2list(lst1,lst2)

For your attention