

INE-PYTHON

LECTURE - 06 LISTS AND TUPLES

What You Will Learn

- Introduction to lists
- List Methods and Useful Built-in Functions
- Copying and Processing Lists
- Two-dimensional Lists
- Tuples

LISTS



Introduction

SEQUENCE TYPES

ประเภทลำดับพื้นฐาน ได้แก่ ลิสต์ (lists) และ ทูเพิล (tuples) ซึ่งทั้งลิสต์และทูเพิลเป็นประเภทของลำดับที่สามารถเก็บข้อมูลได้หลายประเภท ความแตกต่างระหว่างลิสต์กับทูเพิลมีดังนี้:

- **ลิสต์ (List)** เป็นแบบ เปลี่ยนแปลงได้ (mutable) หมายความว่า โปรแกรมสามารถเปลี่ยนแปลงเนื้อหาภายในลิสต์ได้
- **ทูเพิล (Tuple)** เป็นแบบ เปลี่ยนแปลงไม่ได้ (immutable) หมายความว่า เมื่อถูกสร้างขึ้นแล้ว เนื้อหาภายในทูเพิลจะไม่สามารถเปลี่ยนแปลงได้อีก

LISTS

ลิสต์เป็นโครงสร้างข้อมูลที่ใช้งานบ่อยที่สุดอย่างหนึ่งในภาษา Python เมน้ำสำหรับหั้งผู้เริ่มต้นและนักพัฒนาที่มีประสบการณ์ โดยลิสต์เป็นโครงสร้างข้อมูลที่มีลำดับ (ordered) กล่าวคือ สามารถอ่านได้ตามลำดับที่แน่นอน และจะไม่เปลี่ยนแปลงเว้นแต่จะมีการแก้ไขอย่างเจาะจง ซึ่งคุณลักษณะนี้ช่วยให้สามารถเข้าถึงสมาชิกได้อย่างสม่ำเสมอ นอกจากนี้ลิสต์ยังสามารถเปลี่ยนแปลงได้ (mutable) หมายความว่าเนื้อหาภายในสามารถเปลี่ยนได้หลังจากการสร้าง ซึ่งมีประโยชน์อย่างมากในการอัปเดตข้อมูล การจัดเรียง หรือการสร้างโครงสร้างข้อมูลแบบไดนามิกระหว่างการทำงานของโปรแกรม

คุณสมบัติสำคัญอีกประการของลิสต์คือสามารถเก็บข้อมูลได้หลายประเภทภายในลิสต์เดียว เช่น จำนวนเต็ม ตัวเลขศนิยม สตริง หรือแม้แต่ลิสต์อื่น ๆ ทำให้มีความยืดหยุ่นสูง ตัวอย่างเช่น ลิสต์เดียวสามารถเก็บทั้งจำนวนเต็ม สตริง และลิสต์อยู่ได้พร้อมกัน ซึ่งช่วยให้สามารถจัดระเบียบข้อมูลที่ซับซ้อนได้ เช่น การสร้างเมทริกซ์หรือต้นไม้ข้อมูล (tree)

นอกจากนี้ ลิสต์ใน Python ยังมีเมธอดในตัวที่หลากหลาย ช่วยให้การเพิ่ม ลบ หรือค้นหาข้อมูลเป็นเรื่องง่าย ส่งผลให้ลิสต์เป็นเครื่องมือสำคัญและทรงพลังในงานเขียนโปรแกรมที่หลากหลาย ตั้งแต่การเก็บข้อมูลอย่างง่ายไปจนถึงการสร้างอัลกอริธึมที่ซับซ้อน

LISTS

Key Concepts of Lists:

- Ordered, mutable collection
- Can contain any data type

```
1 fruits = ["apple", "banana", "cherry"]
2 numbers = [1, 2, 3, 4, 5]
3 mixed = ["apple", 1, 2.5, True]
```

Listing 6.1: List Example

CREATING AN ACCESSING LISTS

แนวคิดสำคัญของการสร้างและเข้าถึงลิสต์:

- ใช้งานเล็บเหลี่ยม []
- ตัวอย่าง: fruits = ["apple", "banana", "cherry"]
- การเข้าถึงด้วยดัชนี (ทั้งบวกและลบ)
- ตัวอย่าง: fruits[0] (สมาชิกตัวแรก), fruits[-1] (สมาชิกตัวสุดท้าย)

```
1 # Creating and accessing lists
2 prime_numbers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
3 print(f"Prime numbers: {prime_numbers}")
4 fifth_prime = prime_numbers[4]
5 print(f"Fifth prime number: {fifth_prime}")
```

Listing 6.2: Creating and Accessing Lists

NEGATIVE INDEXING

```
1 colors = ["red", "blue", "green", "yellow", "purple"]
2 second_to_last_color = colors[-2]
3 print(f"Second to last color: {second_to_last_color}")
```

Listing 6.3: Example: Negative Indexing

MODIFYING LISTS

แนวคิดสำคัญของการสร้างและเข้าถึงลิสต์:

- การเปลี่ยนค่าด้วยดัชนี
- ตัวอย่าง: `fruits[1] = "blueberry"`

```
1 # Modifying lists
2 shapes = ["circle", "square", "triangle", "rectangle", "hexagon"]
3 shapes[1] = "ellipse"
4 shapes[3] = "pentagon"
5 print(f"Modified shapes: {shapes}")
```

Listing 6.4: Example:Modifying lists

LISTS



List Methods

LIST METHODS

เมธอดทั่วไปของลิสต์:

- `append()`: เพิ่มค่าหนึ่งค่าไปที่ท้ายลิสต์
- `extend()`: เพิ่มสมาชิกทั้งหมดจากลิสต์อื่นไปที่ท้ายลิสต์ปัจจุบัน
- `insert()`: แทรกค่าที่ตำแหน่งที่ระบุ
- `remove()`: ลบค่าที่พบครั้งแรกของค่าที่ระบุ
- `pop()`: ลบและคืนค่าจากตำแหน่งที่ระบุ
- `index()`: คืนค่าตำแหน่งแรกที่พบค่าที่ระบุ
- `count()`: นับจำนวนครั้งที่ค่าปรากฏในลิสต์
- `clear()`: ลบค่าทั้งหมดในลิสต์
- `sort()`: จัดเรียงค่าจากน้อยไปมาก
- `reverse()`: สลับลำดับของสมาชิกในลิสต์

LIST METHODS (APPEND)

เมธอด `append()` ใช้เพิ่มค่าหนึ่งค่าไปที่ท้ายของลิสต์ ซึ่งช่วยให้สามารถเพิ่มสมาชิกใหม่ได้แบบไนามิก หมาย
สำหรับการสร้างลิสต์ที่จะเปลี่ยนแปลง หรือจากอินพุตของผู้ใช้ โดยรับประกันว่าสมาชิกใหม่จะถูกเพิ่มต่อท้ายเสมอ

```
1 # Append method
2 fruits = ["apple", "banana", "cherry"]
3 more_fruits = ["mango", "pineapple"]
4 for fruit in more_fruits:
5     fruits.append(fruit)
6 print(f"Fruits after append: {fruits}")
7 # Output: Fruits after append: ['apple', 'banana', 'cherry', 'mango', 'pineapple']
```

Listing 6.5: Example: List Append

LIST METHODS(INSERT)

เมธอด `insert()` ใช้เพิ่มสมาชิกใหม่ลงในตำแหน่งที่ระบุภายในลิสต์ ให้คุณสามารถควบคุมตำแหน่งการเพิ่มค่าได้อย่างแม่นยำ โดยสมาชิกที่อยู่เดิมจะถูกเลื่อนไปทางขวาเพื่อเปิดที่ว่างสำหรับสมาชิกใหม่

```
1 # Insert method
2 berries = ["raspberry", "blackberry"]
3 berries.insert(1, "strawberry")
4 berries.insert(2, "blueberry")
5 print(f"Berries after insert: {berries}")
6 # Output: Berries after insert: ['raspberry', 'strawberry', 'blueberry',
    'blackberry']
```

Listing 6.6: Example: List Insert

LIST METHODS(REMOVE)

เมธอด `remove()` ใช้ลบค่าที่พบครั้งแรกของค่าที่ระบุจากลิสต์ โดยไม่ต้องทราบตำแหน่งที่แน่นอน ถ้าค่าที่ระบุปรากฏหลายครั้ง จะลบแค่ครั้งแรกเท่านั้น และสมาชิกถัดไปจะถูกเลื่อนเข้ามานแทนที่

```
1 # Remove method
2 fruits_with_duplicates = ["apple", "banana", "apple", "cherry", "apple", "kiwi"]
3 while "apple" in fruits_with_duplicates:
4     fruits_with_duplicates.remove("apple")
5 print(f"Fruits after remove: {fruits_with_duplicates}")
6 # Output: Fruits after remove: ['banana', 'cherry', 'kiwi']
```

Listing 6.7: Example: List Remove

LIST METHODS(POP)

เมธอด `pop()` ใช้ลบและคืนค่าของสมาชิกที่ตำแหน่งที่ระบุจากลิสต์ ทำให้สามารถลบและใช้งานค่าดังกล่าวได้ในคำสั่งเดียว หากไม่ระบุตำแหน่ง จะลบและคืนค่าสมาชิกสุดท้ายในลิสต์ วิธีนี้มีประโยชน์โดยเฉพาะในการนีที่ทำงานแบบสแต็ก (stack)

```
1 # Pop method
2 grades = [85, 90, 78, 92, 88]
3 third_grade = grades.pop(2)
4 grades.append(third_grade)
5 print(f"Grades after pop: {grades}")
6 # Output: Grades after pop: [85, 90, 92, 88, 78]
```

Listing 6.8: Example: List Pop

LIST METHODS(INDEX)

เมธอด `index()` ใช้ค้นหาตำแหน่งแรกที่พบค่าที่ระบุในลิสต์ หากไม่พบค่าจะเกิด `ValueError` สามารถระบุตำแหน่งเริ่มต้นเพื่อค้นหาค่าซ้ำถัดไปในลิสต์ได้ หมายสำหรับค้นหาข้อมูลเฉพาะภายในลิสต์อย่างมีประสิทธิภาพ

```
1
2 # Using index() to find the first occurrence of "dog"
3 animals = ["cat", "dog", "rabbit", "hamster", "dog", "parrot"]
4 first_dog_index = animals.index("dog")
5 print(f"The first occurrence of 'dog' is at index: {first_dog_index}")
# Output: The first occurrence of 'dog' is at index: 1
6
7
8 # Using index() to find the second occurrence of "dog"
9 second_dog_index = animals.index("dog", first_dog_index + 1)
10 print(f"The second occurrence of 'dog' is at index: {second_dog_index}")
# Output: The second occurrence of 'dog' is at index: 4
```

Listing 6.9: Example: List Index

LIST METHODS(CLEAR)

เมธอด clear() ใช้ลบสมาชิกทั้งหมดในลิสต์ ทำให้ลิสต์ว่างเปล่าโดยไม่ต้องสร้างลิสต์ใหม่ หมายสำหรับการรีเซตค่าภายในลิสต์โดยยังคงใช้โครงสร้างเดิม

```
1 # Clear method
2 nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
3 for sublist in nested_list:
4     sublist.clear()
5 print(f"Nested list after clear: {nested_list}")
6 # Output: Nested list after clear: [[], [], []]
```

Listing 6.10: Example: List Clear

LIST METHODS(SORT)

เมธอด sort() ใช้จัดเรียงสมาชิกในลิสต์จากน้อยไปมากโดยปริยาย สามารถปรับให้เรียงจากมากไปน้อยหรือกำหนดเกณฑ์การจัดเรียงอื่น ๆ ได้ การเรียงลิสต์ช่วยให้การจัดการ เปรียบเทียบ และวิเคราะห์ข้อมูลมีประสิทธิภาพมากขึ้น

```
1 numbers = [4, 2, 3, 1, 5]
2 numbers.sort()
3 print(numbers) # Output: [1, 2, 3, 4, 5]
```

Listing 6.11: Example: List Sort

LIST METHODS(VERSE)

เมธอด reverse() ใช้สลับลำดับของสมาชิกในลิสต์ โดยสมาชิกตัวแรกจะถูกเป็นตัวสุดท้าย และตัวสุดท้ายจะถูกเป็นตัวแรก เมธอดนี้จะเปลี่ยนแปลงลิสต์เดิมทันที เหมาะสำหรับการประมวลผลข้อมูลในลำดับย้อนกลับ

```
1 numbers = [1, 2, 3, 4, 5]
2 numbers.reverse()
3 print(numbers) # Output: [5, 4, 3, 2, 1]
```

Listing 6.12: Example: List Reverse

LIST_METHODS.PY

```
1 heroes = ['Ironman', 'Thor', 'Hulk', 'Superman', 'Spiderman']
2 h2 = ['Dr. Strange', 'Cpt. America', 'Black Panther', 'Ant Man']
3
4 heroes.insert(0, h2[0])
5 print(heroes.index('Thor'))
6 heroes.insert(heroes.index('Thor'), h2[1])
7 print(heroes)
8 heroes.remove('Superman')
9 heroes.append('Ant Man')
10 print(heroes)
11 heroes.sort()
12 print(heroes)
13 heroes.reverse()
14 print(heroes)
15 newheroes = heroes
16 newheroes[0] = 'Wonder Women'
17 print(heroes)
18 copyheroes = [] + heroes
19 print(copyheroes)
20 copyheroes[0] = 'Hanuman'
21 print(heroes)
22 print(copyheroes)
```

2

```
['Dr. Strange', 'Ironman', 'Cpt. America', 'Thor', 'Hulk', 'Superman', 'Spiderman']
['Dr. Strange', 'Ironman', 'Cpt. America', 'Thor', 'Hulk', 'Spiderman', 'Ant Man']
['Ant Man', 'Cpt. America', 'Dr. Strange', 'Hulk', 'Ironman', 'Spiderman', 'Thor']
['Thor', 'Spiderman', 'Ironman', 'Hulk', 'Dr. Strange', 'Cpt. America', 'Ant Man']
['Wonder Women', 'Spiderman', 'Ironman', 'Hulk', 'Dr. Strange', 'Cpt. America', 'Ant Man']
['Wonder Women', 'Spiderman', 'Ironman', 'Hulk', 'Dr. Strange', 'Cpt. America', 'Ant Man']
['Wonder Women', 'Spiderman', 'Ironman', 'Hulk', 'Dr. Strange', 'Cpt. America', 'Ant Man']
['Hanuman', 'Spiderman', 'Ironman', 'Hulk', 'Dr. Strange', 'Cpt. America', 'Ant Man']
```

LISTS



Slicing

SLICING LIST

การเข้าถึงบางส่วนของลิสต์ในภาษา Python สามารถทำได้โดยใช้ไวยากรณ์การตัดช่วง `string[start: stop: step]` ซึ่งช่วยให้คุณสามารถดึงข้อมูลช่วงหนึ่งจากลิสต์หรือสตริงตามดัชนีและค่าก้าวที่กำหนด มาดูรายละเอียดของแต่ละพารามิเตอร์ :

คำอธิบาย:

- `start`: ดัชนีเริ่มต้นของช่วงที่ต้องการตัด หากไม่ระบุ จะเริ่มจากตำแหน่งแรกของลิสต์
- `stop`: ดัชนีสิ้นสุดของช่วง (ไม่รวมตำแหน่งที่ระบุ) หากไม่ระบุ จะสิ้นสุดที่ตำแหน่งสุดท้ายของลิสต์
- `stride`: ค่าก้าว (จำนวนตำแหน่งที่จะข้ามไปในแต่ละขั้น) หากไม่ระบุ ค่าปริยายคือ 1 ซึ่งหมายถึงค่าทุกตำแหน่งที่ต่อกัน

```
1 data = list(range(100))
2 sliced_data = data[10:51:5]
3 print(f"Sliced data: {sliced_data}")
```

Listing 6.13: Example: List Slicing

SLICING LIST

```
1 # Slicing a list
2 numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3
4 # Slicing from index 2 to 5
5 print(numbers[2:6]) # Output: [2, 3, 4, 5]
6
7 # Slicing with step
8 print(numbers[1:8:2]) # Output: [1, 3, 5, 7]
9
10 # Slicing from start to a position
11 print(numbers[:4]) # Output: [0, 1, 2, 3]
12
13 # Slicing from a position to the end
14 print(numbers[6:]) # Output: [6, 7, 8, 9]
```

NEGATIVE SLICING LIST

คุณสามารถใช้ดัชนีลบเพื่อตัดช่วงข้อมูลจากด้านท้ายของลิสต์ ซึ่งช่วยให้สามารถเข้าถึงสมาชิกจากท้ายรายการได้อย่างสะดวก ตัวอย่างเช่น `list[-3:]` จะดึงสมาชิกสามตัวสุดท้ายของลิสต์ วิธีนี้ช่วยให้สามารถจัดการกับลิสต์ได้อย่างยืดหยุ่นและเป็นธรรมชาติ

```
1 numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2
3 # Negative slicing
4 print(numbers[-5:-1]) # Output: [5, 6, 7, 8]
5
6 # Slicing with negative step
7 print(numbers[::-1]) # Output: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Listing 6.15: Example: Negative Slicing

STRING SLICING

Begin End
↑ ↑
ss[Start:Stop:Stride]
↓
How many character to move forward

```
print(ss[0:12:2])  
SammyShark!
```

ss = "Sammy Shark!"	
print(ss[4])	y
print(ss[6:11])	Shark
print(ss[:5])	Sammy
print(ss[7:])	hark!
print(ss[-4:-1])	ark
print(ss[6:11])	Shark
print(ss[6:11:1])	Shark
print(ss[0:12:2])	SmySak
print(ss[0:12:4])	Sya
print(ss[::-4])	Sya
print(ss[::-1])	!krahS ymmaS
print(ss[::-2])	!rh ma

LISTOPERATOR.PY

```
1 even_numbers = [2, 4, 6, 8, 10]
2 heroes = ['Ironman', 'Thor', 'Hulk', 'Spiderman']
3 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4
5 print(numbers[-5:])
6 numbers[8] = 99
7 print(numbers)
8
9 pluslist = heroes + even_numbers
10 print(pluslist)
11 print(len(numbers))
```

LISTS



Built-in Functions For Working With Lists

BUILT-IN FUNCTION THAT WORK WITH LISTS

เมธอดทั่วไปของลิสต์:

- `append()`: เพิ่มค่าหนึ่งค่าไปที่ท้ายลิสต์
- `extend()`: เพิ่มสมาชิกทั้งหมดจากลิสต์อื่นไปที่ท้ายลิสต์ปัจจุบัน
- `insert()`: แทรกค่าที่ตำแหน่งที่ระบุ
- `remove()`: ลบค่าที่พบครั้งแรกของค่าที่ระบุ
- `pop()`: ลบและคืนค่าจากตำแหน่งที่ระบุ
- `index()`: คืนค่าตำแหน่งแรกที่พบค่าที่ระบุ
- `count()`: นับจำนวนครั้งที่ค่าปรากฏในลิสต์
- `clear()`: ลบค่าทั้งหมดในลิสต์
- `sort()`: จัดเรียงค่าจากน้อยไปมาก
- `reverse()`: สลับลำดับของสมาชิกในลิสต์

BUILT-IN FUNCTION THAT WORK WITH LISTS

```
1 # Example list
2 numbers = [4, 2, 9, 1, 5, 6]
3
4 # 1. len(): Get the length of the list
5 length = len(numbers)
6 print(f"Length of the list: {length}") # Output: Length of the
    list: 6
7
8 # 2. sum(): Calculate the sum of all elements in the list
9 total_sum = sum(numbers)
10 print(f"Sum of all elements: {total_sum}") # Output: Sum of all
     elements: 27
11
12 # 3. max(): Find the maximum value in the list
13 max_value = max(numbers)
14 print(f"Maximum value: {max_value}") # Output: Maximum value: 9
15
16 # 4. min(): Find the minimum value in the list
17 min_value = min(numbers)
18 print(f"Minimum value: {min_value}") # Output: Minimum value: 1
19
20 # 5. sorted(): Return a sorted version of the list
21 sorted_numbers = sorted(numbers)
22 print(f"Sorted list: {sorted_numbers}") # Output: Sorted list:
    [1, 2, 4, 5, 6, 9]
```

BUILT-IN FUNCTION THAT WORK WITH LISTS

```
24 # 6. any(): Check if any element in the list is True
25 bool_list = [False, True, False]
26 any_true = any(bool_list)
27 print(f"Is any element True? {any_true}") # Output: Is any
     element True? True
28
29 # 7. all(): Check if all elements in the list are True
30 all_true = all(bool_list)
31 print(f"Are all elements True? {all_true}") # Output: Are all
     elements True? False
32
33 # 8. list(): Convert an iterable to a list (if not already a list
     )
34 string = "hello"
35 char_list = list(string)
36 print(f"List of characters: {char_list}") # Output: List of
     characters: ['h', 'e', 'l', 'l', 'o']
37
38 # 9. reversed(): Return a reverse iterator of the list
39 reversed_numbers = list(reversed(numbers))
40 print(f"Reversed list: {reversed_numbers}") # Output: Reversed
     list: [6, 5, 1, 9, 2, 4]
41
42 # 10. enumerate(): Return an iterator of tuples containing index
     and value
43 enumerated_numbers = list(enumerate(numbers))
44 print(f"Enumerated list: {enumerated_numbers}")
45 # Output: Enumerated list: [(0, 4), (1, 2), (2, 9), (3, 1), (4,
     5), (5, 6)]
```

LISTS



Modifying List

LISTS OPERATOR

- **Repetition operator:** makes multiple copies of a list and joins them together
 - The * symbol is a repetition operator when applied to a sequence and an integer
 - Sequence is left operand, number is right
 - General format: `list * n`
- You can iterate over a list using a for loop
 - Format: `for x in list:`

BARISTA_PAY.PY

```
1 # This program calculates the gross pay for
2 # each of Megan's baristas.
3
4 # NUM_EMPLOYEES is used as a constant for the
5 # size of the list.
6 NUM_EMPLOYEES = 6
7
8 def main():
9     # Create a list to hold employee hours.
10    hours = [0] * NUM_EMPLOYEES
11
12    # Get each employee's hours worked.
13    for index in range(NUM_EMPLOYEES):
14        print('Enter the hours worked by employee ', \
15              index + 1, ': ', sep='', end='')
16        hours[index] = float(input())
17
18    # Get the hourly pay rate.
19    pay_rate = float(input('Enter the hourly pay rate: '))
20
21    # Display each employee's gross pay.
22    for index in range(NUM_EMPLOYEES):
23        gross_pay = hours[index] * pay_rate
24        print('Gross pay for employee ', index + 1, ': $', \
25              format(gross_pay, ',.2f'), sep='')
26
27    # Call the main function.
28 main()
```

HEROES.PY EXERCISE-I

```
heroes = ['Ironman', 'Thor', 'Hulk', 'Spiderman']
```

1. Display Heroes
 2. Add Heroes
 3. Insert Heroes
 4. Remove Heroes
 5. Display Sorted Heroes (Ascending / Descending)
- ** Write in functional way

LISTS



Two-Dimensional Lists

TWO-DIMENSIONAL LISTS

ลิสต์สองมิติ หรือที่เรียกว่า 2D lists หรือเมทริกซ์ (matrices) คือการสร้างลิสต์ช้อนอยู่ภายในลิสต์อีกทีหนึ่ง ซึ่งใช้แทนโครงสร้างแบบตารางที่มีแถวและคอลัมน์ เหมือนกับข้อมูลในตารางหรือสเปรดชีต ในภาษา Python ลิสต์สองมิติคือ ลิสต์ของลิสต์ โดยที่แต่ละลิสต์ย่อยจะแทนหนึ่งแถวของเมทริกซ์

การสร้างลิสต์สองมิติ

เราสามารถสร้างลิสต์สองมิติได้โดยการซ้อนลิสต์หลายลิสต์ไว้ในลิสต์หลัก ตัวอย่างเช่น:

ตัวอย่าง:

```
1 matrix = [
2     [1, 2, 3],
3     [4, 5, 6],
4     [7, 8, 9]
5 ]
```

Listing 6.22: Creating a Two-Dimensional List

TWO-DIMENSIONAL LISTS

	Column 0	Column 1
Row 0	'Joe'	'Kim'
Row 1	'Sam'	'Sue'
Row 2	'Kelly'	'Chris'

```
students = [ ['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris'] ]
```

TWO-DIMENSIONAL LISTS

	Column 0	Column 1	Column 2
Row 0	scores[0][0]	scores[0][1]	scores[0][2]
Row 1	scores[1][0]	scores[1][1]	scores[1][2]
Row 2	scores[2][0]	scores[2][1]	scores[2][2]

```
scores = [[0, 0, 0],  
          [0, 0, 0],  
          [0, 0, 0]]
```

TWO-DIMENSIONAL LISTS

```
1 matrix[0][1] = 10
2 print(matrix)
3 # Output:
4 # [
5 #     [1, 10, 3],
6 #     [4, 5, 6],
7 #     [7, 8, 9]
8 # ]
```

Listing 6.24: Modifying Elements in a 2D List

```
1 for row in matrix:
2     for element in row:
3         print(element, end=" ")
4     print()
```

Listing 6.25: Iterating Through a Two-Dimensional List

TWO-DIMENSIONAL LISTS

การประยุกต์ใช้งาน

ลิสต์สองมิติถูกใช้งานอย่างแพร่หลายในหลายบริบท เช่น:

- **เมตริกซ์:** สำหรับการคำนวณทางคณิตศาสตร์ เช่น การบวกเมทริกซ์ การคูณเมทริกซ์ เป็นต้น
- **กริด (Grid):** สำหรับใช้ในเกมกระดาน เช่น กระดานหมากรุก หรือเกม XO (tic-tac-toe)
- **ตารางข้อมูล:** สำหรับจัดเก็บข้อมูลในรูปแบบແຕວและຄอລິ້ນ໌ เช่น ฐานข้อมูลแบบตาราง
- **ภาพ:** สำหรับเก็บค่าพิกเซลของภาพในการประมวลผลภาพ ซึ่งแต่ละสมาชิกແທນค่าพิกเซล

TWO-DIMENSIONAL LISTS

```
1 # This program assigns random numbers to
2 # a two-dimensional list.
3 import random
4
5 # Constants for rows and columns
6 ROWS = 3
7 COLS = 4
8
9 def main():
10     # Create a two-dimensional list.
11     values = [[0, 0, 0, 0],
12                [0, 0, 0, 0],
13                [0, 0, 0, 0]]
14
15     # Fill the list with random numbers.
16     for r in range(ROWS):
17         for c in range(COLS):
18             values[r][c] = random.randint(1, 100)
19
20     # Display the random numbers.
21     print(values)
22
23 # Call the main function.
24 main()
```

TUPLES



Introduction

TUPLES

ทูเพิล () คือโครงสร้างข้อมูลที่มีลำดับและ ไม่สามารถเปลี่ยนแปลงได้ (immutable) ในภาษา Python หมายความว่า เมื่อสร้างทูเพิลขึ้นมาแล้ว จะไม่สามารถแก้ไข เพิ่ม หรือลบสมาชิกภายในได้ ทูเพิลคล้ายกับลิสต์ตรงที่สามารถเก็บหลายค่าภายในตัวมันได้ แต่ต่างกันตรงที่ทูเพิลไม่สามารถเปลี่ยนแปลงได้ ซึ่งความไม่เปลี่ยนแปลงนี้ทำให้ทูเพิลเป็นทางเลือกที่น่าใช้ถือในการจัดกลุ่มข้อมูลที่ควรคงที่ตลอดการทำงานของโปรแกรม เช่น การเก็บค่าพิกัด ชุดข้อมูลที่ไม่ควรเปลี่ยนแปลง หรือระเบียนข้อมูลavar

TUPLES

แนวคิดสำคัญของ Tuple:

- **โครงสร้างที่มีลำดับ:** ทูเพิลจะคงลำดับของสมาชิกไว้
- **ใช้งานแบบ ()**
- **ไม่เปลี่ยนแปลง:** ไม่สามารถเปลี่ยนค่าภายในทูเพิลหลังจากที่สร้างแล้ว
- **ข้อมูลหลากหลายประเภท:** ทูเพิลสามารถเก็บค่าหลายประเภทลงในตัวเดียว
- **แฮชได้ (Hashable):** ทูเพิลสามารถใช้เป็นคีย์ในดิกชันนารีได้
- **การแพ็คและแยกแพ็คทูเพิล:** สามารถจัดกลุ่มหลายค่าลงในทูเพิล และแยกออกมาเก็บในตัวแปรแต่ละตัวได้

CREATE TUPLES

เราสามารถสร้างทูเพิลได้โดยวางค่าที่คั่นด้วยเครื่องหมายจุลภาค (comma) ไว้ในวงเล็บ () เช่น `my_tuple = (1, 2, 3)` จะสร้างทูเพิลที่มี 3 ค่า

การสร้างทูเพิล

คุณสามารถสร้างทูเพิลได้โดยใส่ค่าภายในวงเล็บ () และคั่นแต่ละค่าด้วยเครื่องหมายจุลภาค

```
1 # Creating a tuple
2 my_tuple = (1, 2, 3, "apple", "banana")
3 print(my_tuple)
```

Listing 6.26: Creating a tuple

TUPLES

- Tuples do not support the methods:
 - **append**
 - **remove**
 - **insert**
 - **reverse**
 - **sort**

TUPLES

- Advantages for using tuples over lists:
 - Processing tuples is faster than processing lists
 - Tuples are safe
 - Some operations in Python require use of tuples
- **list()** function: converts tuple to list
- **tuple()** function: converts list to tuple

SINGLE ELEMENT TUPLES

NOTE: If you want to create a tuple with just one element, you must write a trailing comma after the element's value, as shown here:

```
my_tuple = (1,)      # Creates a tuple with one element.
```

If you omit the comma, you will not create a tuple. For example, the following statement simply assigns the integer value 1 to the `value` variable:

```
value = (1)          # Creates an integer.
```

```
1 # Single element tuple
2 single_element_tuple = (5,)
3 print(type(single_element_tuple)) # Output: <class 'tuple'>
```

Listing 6.30: Single element tuple

TUPLES PACK, UNPACK

Tuple Packing and Unpacking

Tuple Packing is when you assign multiple values to a single variable separated by commas, automatically creating a tuple. **Tuple Unpacking** allows you to assign the values from a tuple to multiple variables.

```
1 # Tuple Packing
2 my_tuple = 1, 2, 3 # Parentheses are optional
3 print(my_tuple) # Output: (1, 2, 3)
4
5 # Tuple Unpacking
6 a, b, c = my_tuple
7 print(a) # Output: 1
8 print(b) # Output: 2
9 print(c) # Output: 3
```

Listing 6.29: Tuple packing and unpacking

EXERCISE-II

Update Inventory:

- Write a function `update_inventory(inventory, item_name, quantity_sold)` to reduce the quantity of a specified item after a sale.

Calculate Total Value:

- Write a function `calculate_total_value(inventory)` to calculate the total value of all items in stock.

Find Most Expensive Item:

- Write a function `find_most_expensive(inventory)` to find and return the name of the most expensive item.

Add or Update Item:

- Write a function `add_item(inventory, item_name, quantity, price)` to add a new item or update an existing one.

```
inventory = [
    ["Apple", 50, 0.75],
    ["Banana", 100, 0.50],
    ["Orange", 75, 0.80]
]
```

Actions:

- Update inventory after selling 20 bananas.
- Calculate the total value of the inventory.
- Find the most expensive item.
- Add "Eggs" with 30 units at \$0.25, then update it to 50 units at \$0.30.

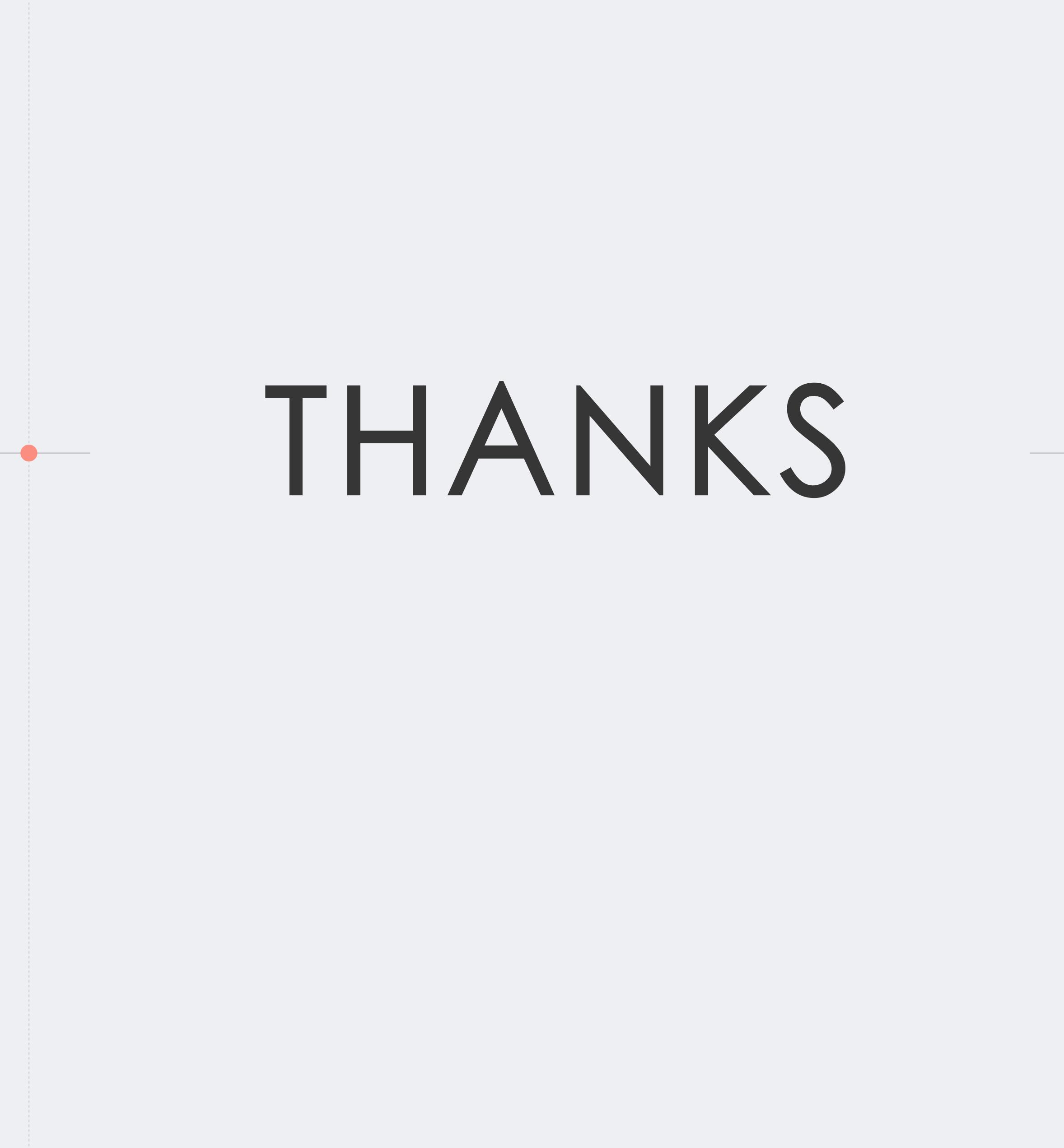
SUMMARY

Lists, including:

- Repetition and concatenation operators
- Indexing
- Techniques for processing lists
- Slicing and copying lists
- List methods and built-in functions for lists
- Two-dimensional lists

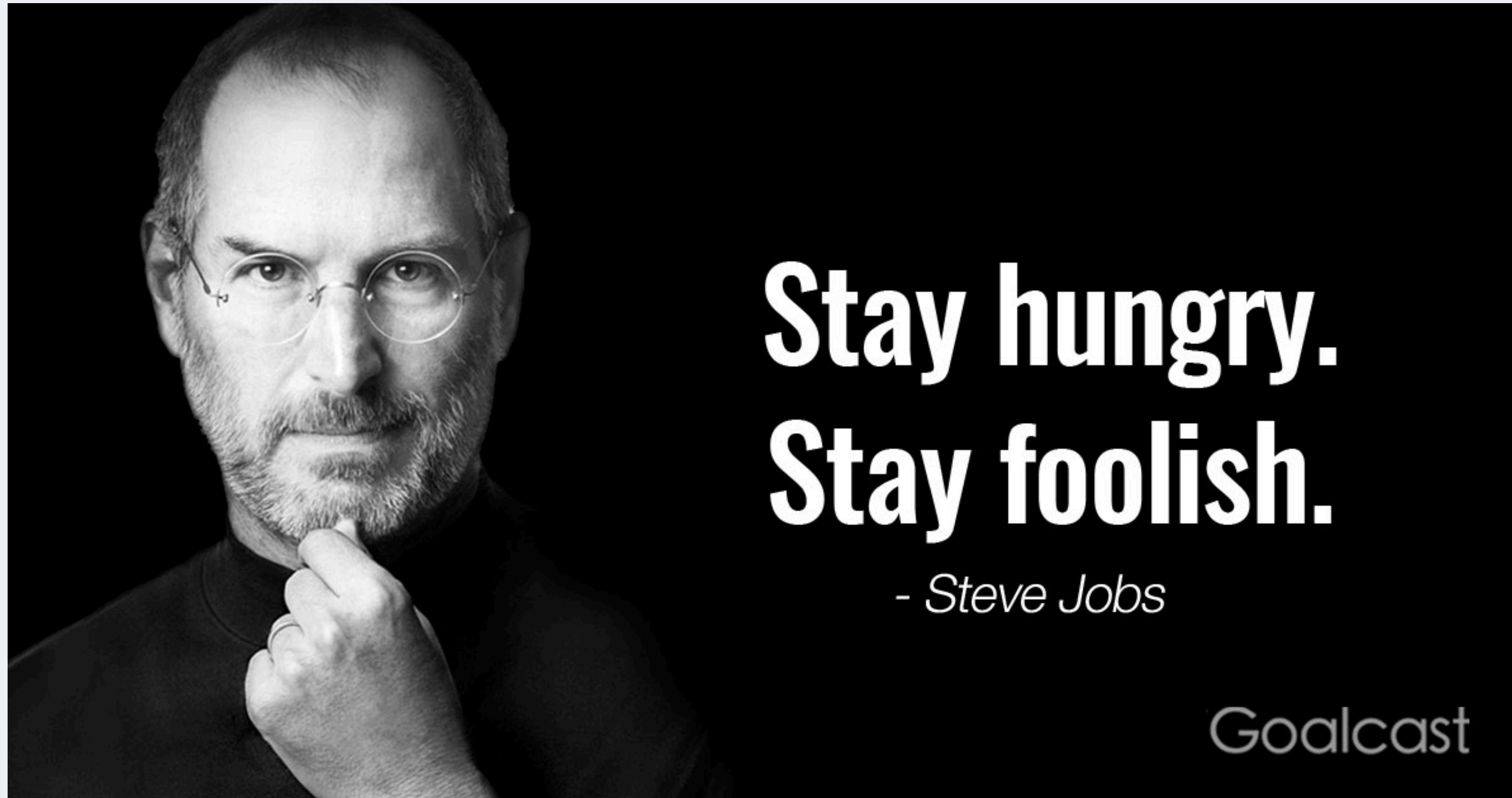
Tuples, including:

- Immutability
- Difference from and advantages over lists
- Plotting charts and graphs with the **matplotlib** Package



THANKS

WORD OF THE WISE



**Stay hungry.
Stay foolish.**

- Steve Jobs

Goalcast