

Analysis Chart

Given Data	Required Results
Hours Pay Rate	Gross Pay
Processing Required	Solution Alternatives
$GrossPay = Hours * PayRate$	1. Define the hours worked and pay rate as constants. *2. Define the hours worked and pay rate as input values.

Given data : ข้อมูลที่โปรแกรมต้องได้รับ ค่า Input ต่าง ๆ ทั้งจากระบบ และจาก คีย์บอร์ด

Required Results : การกำหนดผลลัพธ์ที่ต้องการได้จากโปรแกรม

Processing Required : รายการของการประมวลผล สิ่งที่โปรแกรมต้องการใช้ในการประมวลผล เช่น สูตรคำนวณ สมการต่างๆ เงื่อนไข และการประมวลผลอย่างอื่น เช่น การเรียงลำดับ การค้นหา การแก้ไขข้อมูล

Solution Alternatives : วิธีการขั้นตอนในการแก้ปัญหา *อธิบายวิธีการในการแก้ปัญหา

IPO Chart

Input	Processing	Module Reference	Output
Hours Worked Pay Rate	1. Enter Hours Worked 2. Enter Pay Rate 3. Calculate Pay 4. Print Pay 5. End	<i>Read</i> <i>Calc</i> <i>Print</i> <i>PayRollControl</i>	Gross pay










การนำเข้าข้อมูล (Input) : ข้อมูลที่โปรแกรมต้องได้รับ ที่จากผู้ใช้งาน และจากระบบ







การประมวลผล(Process) : ขั้นตอนวิธีการทำงานในโปรแกรม ตั้งแต่ต้นจนจบ

Module Reference : เป็นการเขียนอ้างอิงโมดูลที่เกี่ยวข้อง เช่น Read Calc Print *โมดูลคือการทำงานต่างๆ ของโปรแกรม

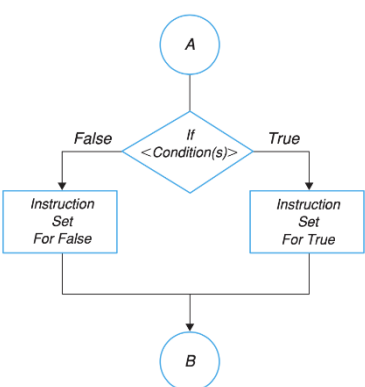
การแสดงผล (Output) : เป็นการกำหนดผลลัพธ์ที่ต้องการให้แสดงผลออกมา

FlowChart

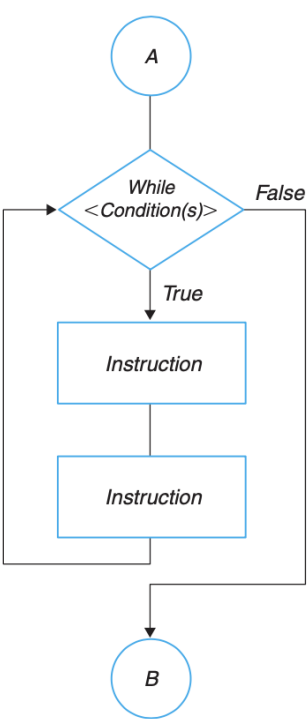
สัญลักษณ์	ชื่อสัญลักษณ์	คำอธิบาย
	Terminator	จุดเริ่มต้นและจุดสิ้นสุดของโปรแกรม
	Decision	การตรวจสอบเงื่อนไข การตัดสินใจ
	Process	คำสั่งในการประมวลผล หรือการกำหนดตัวแปร
	Flow Line	เส้นทางการทำงานของโปรแกรม และการไหลของข้อมูล
	Input/Output	การอ่านข้อมูลจากแหล่งข้อมูลสำรอง หรือการแสดงผลข้อมูลผลลัพธ์จากการประมวลผล
	Manual Input	การรับข้อมูลจากทางคีย์บอร์ด
	Document	การแสดงผลข้อมูลทางเครื่องพิมพ์
	Display	การแสดงผลข้อมูลออกทางจอภาพ
	Preparation	การกำหนดค่าต่างๆ ส่วนหน้าในการทำงานที่ซ้ำๆกัน

	Predefined Process	โปรแกรมย่อย หรือมอดูล ที่ใช้ในการทำงานของโปรแกรม
	Connect	การรวมจุด หรือการเชื่อมต่อจุด
	Off page Connector	การเชื่อมโยงไปยังหน้ากระดาษถัดไปเนื่องจากผังงานยาว
	Sort	การเรียงข้อมูล
	Magnetic Disk	การนำข้อมูล หรือบันทึกข้อมูลจาก Hard disk
	Comment	ใช้ในการแสดงคำอธิบายโปรแกรม หรือหมายเหตุ

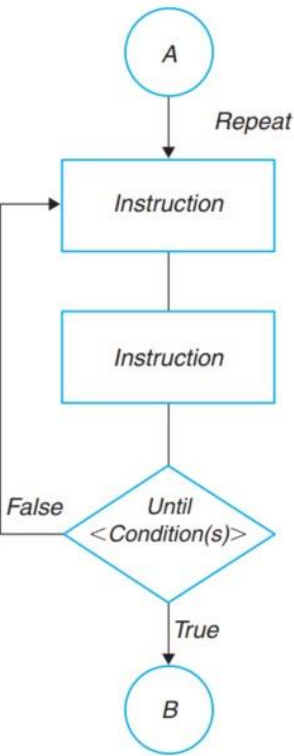
แบบมีเงื่อนไข



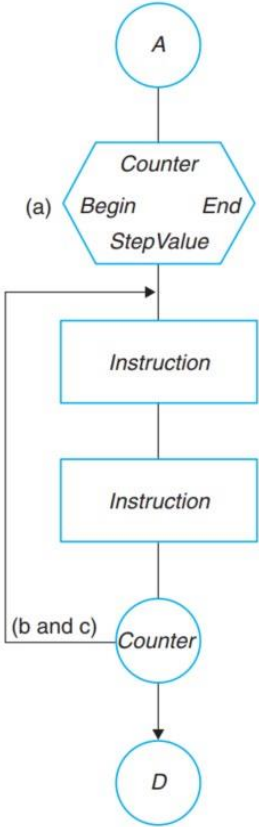
Loop While



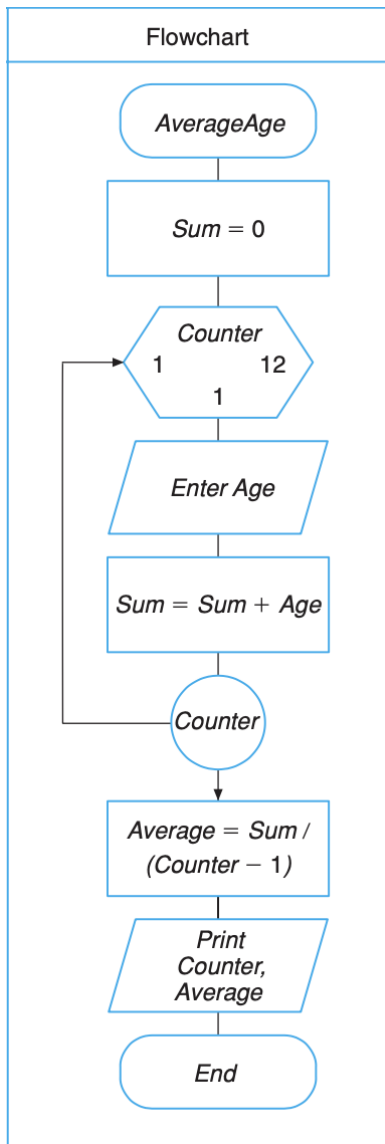
Do While



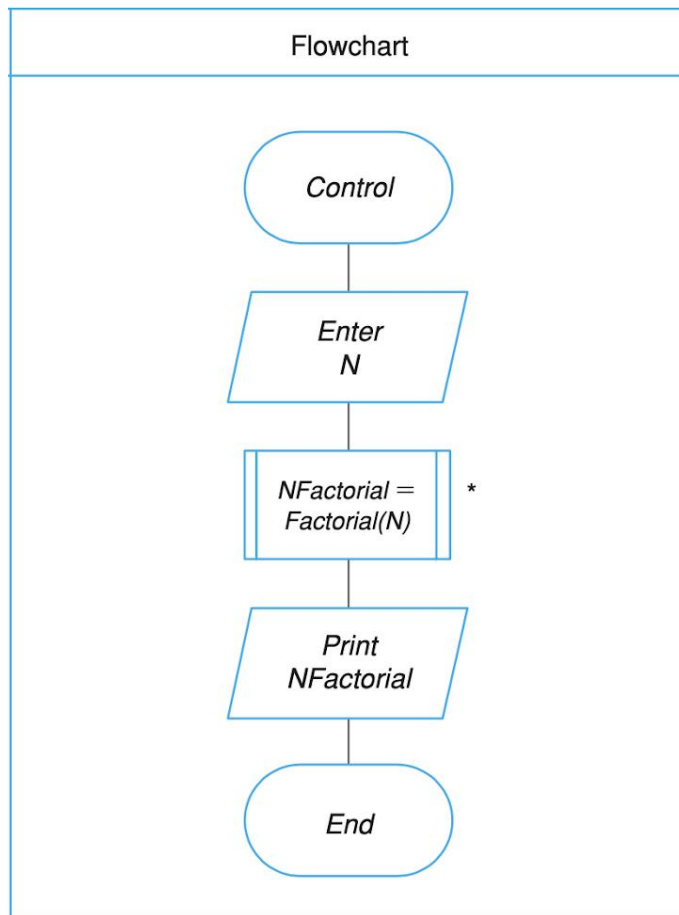
For Loop



ตัวอย่าง For Loop



ตัวอย่างใช้ ฟังก์ชัน



```
numbers = [21, 34, 51, 23, 37, 44, 60, 11, 94, 99]
```

```
result = []
```

```
result2 = {}
```

```
for i in range(0,len(numbers)):
```

```
    group = []
```

```
    for j in range(0,len(numbers)):
```

```
        if numbers[j]%10 == i:
```

```
            group.append(numbers[j])
```

```
    result.append(group)
```

```
    result2[i] = group
```

```
group_most = [[]]
```

```
result3 = {}
```

```
for i in result2:
```

```
    if len(result2[i]) > len(group_most[0]):
```

```
        group_most.clear()
```

```
        group_most.append(result2[i])
```

```
        result3.clear()
```

```
        result3[i] = result2[i]
```

```
    elif len(result2[i]) == len(group_most[0]):
```

```
        group_most.append(result2[i])
```

```
        result3[i] = result2[i]
```

```
print(result2)
```

```
print(result3)
```

ไม่เกี่ยวกับสไลด์เอาไปเฉยๆ

Complexity **อะไรบ้างที่ใช้ในการตรวจสอบว่า โค้ดนี้ดีแค่ไหน ให้ตอบตามสไลด์นี้

Complexity Theory (Big O Notation)

เป็นการหาความซับซ้อนของระยะเวลา (time execution)
เราจะมีค่าหนึ่งถึงทรัพยากรที่ใช้ในการแก้ปัญหา



Sort การเรียงข้อมูลแบบต่างๆ

1. Bubble Sort

วิธีเปรียบเทียบข้อมูลใกล้เคียงกันทีละคู่ และสลับตำแหน่งถ้าจำเป็น ซ้ำไปเรื่อย ๆ จนกว่าข้อมูลทั้งหมดจะเรียงลำดับ

```
python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

# ตัวอย่างการใช้งาน
data = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(data)
print("Bubble Sort:", data) # Output: [11, 12, 22, 25, 34, 64, 90]
```

2. Selection Sort

เลือกค่าที่เล็กที่สุดจากส่วนที่ยังไม่เรียงและสลับกับตำแหน่งแรกในส่วนที่ยังไม่เรียง

```
python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

# ตัวอย่างการใช้งาน
data = [64, 25, 12, 22, 11]
selection_sort(data)
print("Selection Sort:", data) # Output: [11, 12, 22, 25, 64]
```

3. Insertion Sort

แทรกแต่ละค่าจากส่วนที่ยังไม่ได้เรียงไปยังตำแหน่งที่เหมาะสมในส่วนที่เรียงแล้ว

```
python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# ตัวอย่างการใช้งาน
data = [12, 11, 13, 5, 6]
insertion_sort(data)
print("Insertion Sort:", data) # Output: [5, 6, 11, 12, 13]
```

5. Quick Sort

เลือกค่าหนึ่งเป็น pivot แล้วแบ่งข้อมูลตามค่าที่เล็กกว่าหรือใหญ่กว่า pivot

```
python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# ตัวอย่างการใช้งาน
data = [10, 7, 8, 9, 1, 5]
sorted_data = quick_sort(data)
print("Quick Sort:", sorted_data) # Output: [1, 5, 7, 8, 9, 10]
```

4. Merge Sort

ใช้การแบ่งและรวม (Divide and Conquer)

```
python 🔗 คัดลอก 🔗 แก้ไข

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]

        merge_sort(left)
        merge_sort(right)

        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1

        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1

        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1

# ตัวอย่างการใช้งาน
data = [12, 11, 13, 5, 6, 7]
merge_sort(data)
print("Merge Sort:", data) # Output: [5, 6, 7, 11, 12, 13]
```



6. Heap Sort

ใช้โครงสร้างข้อมูล heap

```
python 🔗 คัดลอก 🔗 แก้ไข

import heapq

def heap_sort(arr):
    heap = []
    for value in arr:
        heapq.heappush(heap, value)
    sorted_arr = [heapq.heappop(heap) for _ in range(len(heap))]
    return sorted_arr

# ตัวอย่างการใช้งาน
data = [4, 10, 3, 5, 1]
sorted_data = heap_sort(data)
print("Heap Sort:", sorted_data) # Output: [1, 3, 4, 5, 10]
```

7. Python's Built-in Sort

ใช้เมธอด `sort()` หรือ `sorted()` ของ Python ที่ใช้ Timsort ($O(n \log n)$)

```
python 🔗 คัดลอก 🔗 แก้ไข

data = [64, 25, 12, 22, 11]
data.sort() # จัดเรียงแบบ in-place
print("Built-in Sort:", data) # Output: [11, 12, 22, 25, 64]

# หรือใช้ sorted() ซึ่งไม่แก้ไขลิสต์ต้นฉบับ
data = [64, 25, 12, 22, 11]
sorted_data = sorted(data)
print("Built-in Sorted:", sorted_data) # Output: [11, 12, 22, 25, 64]
```

Queue ตัวอย่างโปรแกรมที่เก็บข้อมูลแบบคิว

ตัวอย่างที่ 1: ระบบคิวร้านอาหาร

จำลองลูกค้ารอเรียกคิวที่ร้านอาหาร

```
python

from collections import deque

restaurant_queue = deque()

# เพิ่มลูกค้าในคิว
def add_to_queue(queue, customer_name):
    queue.append(customer_name)
    print(f"{customer_name} ถูกเพิ่มในคิว")

# เรียกคิว
def call_next_customer(queue):
    if queue:
        customer = queue.popleft()
        print(f"เชิญ {customer} รับบริการ")
    else:
        print("ไม่มีลูกค้าในคิว!")

# ตัวอย่างการใช้งาน
add_to_queue(restaurant_queue, "ลูกค้าคนที่ 1")
add_to_queue(restaurant_queue, "ลูกค้าคนที่ 2")
add_to_queue(restaurant_queue, "ลูกค้าคนที่ 3")

call_next_customer(restaurant_queue)
call_next_customer(restaurant_queue)
call_next_customer(restaurant_queue)
call_next_customer(restaurant_queue)
```

ตัวอย่างที่ 3: การจัดการข้อความในระบบส่งข้อความ

จำลองระบบการจัดการข้อความที่ส่งมาทีละข้อความ

```
python

from collections import deque

message_queue = deque()

def receive_message(queue, message):
    queue.append(message)
    print(f"รับข้อความใหม่: '{message}'")

def process_message(queue):
    if queue:
        message = queue.popleft()
        print(f"กำลังประมวลผลข้อความ: {message}")
    else:
        print("ไม่มีข้อความในคิว!")

# การใช้งาน
receive_message(message_queue, "สวัสดี")
receive_message(message_queue, "ประชุมเริ่มเวลา 9 โมง")
receive_message(message_queue, "อัปเดตงานเสร็จพร้อมยัง")

process_message(message_queue)
process_message(message_queue)
process_message(message_queue)
process_message(message_queue)
```

ตัวอย่างที่ 2: ระบบงานพิมพ์เอกสาร

จัดการเอกสารที่ต้องรอการพิมพ์ในลำดับ FIFO

```
python

from collections import deque

print_queue = deque()

def add_document(queue, document_name):
    queue.append(document_name)
    print(f"เพิ่มเอกสาร '{document_name}' เข้าคิวการพิมพ์")

def print_document(queue):
    if queue:
        document = queue.popleft()
        print(f"กำลังพิมพ์เอกสาร: {document}")
    else:
        print("ไม่มีเอกสารในคิว!")

# การใช้งาน
add_document(print_queue, "เอกสารที่ 1")
add_document(print_queue, "เอกสารที่ 2")
add_document(print_queue, "เอกสารที่ 3")

print_document(print_queue)
print_document(print_queue)
print_document(print_queue)
print_document(print_queue)
```

ตัวอย่างที่ 4: คิวในระบบเรียกเลขลำดับธนาคาร

จำลองระบบการจัดการเรียกคิวธนาคารที่มีหลายประเภท

```
python

from collections import deque

queue_general = deque() # คิวทั่วไป
queue_priority = deque() # คิวพิเศษ

def add_to_queue(queue, customer_name):
    queue.append(customer_name)
    print(f"{customer_name} ถูกเพิ่มในคิว")

def serve_customer(priority_queue, general_queue):
    if priority_queue:
        customer = priority_queue.popleft()
        print(f"ให้บริการลูกค้าพิเศษ: {customer}")
    elif general_queue:
        customer = general_queue.popleft()
        print(f"ให้บริการลูกค้าทั่วไป: {customer}")
    else:
        print("ไม่มีลูกค้าในคิว!")

# การใช้งาน
add_to_queue(queue_general, "ลูกค้าทั่วไป 1")
add_to_queue(queue_priority, "ลูกค้าพิเศษ 1")
add_to_queue(queue_general, "ลูกค้าทั่วไป 2")
add_to_queue(queue_priority, "ลูกค้าพิเศษ 2")

serve_customer(queue_priority, queue_general)
serve_customer(queue_priority, queue_general)
serve_customer(queue_priority, queue_general)
serve_customer(queue_priority, queue_general)
serve_customer(queue_priority, queue_general)
```

Stack ตัวอย่างโปรแกรมที่เก็บข้อมูลแบบ stack

ตัวอย่างที่ 1: ตรวจสอบวงเล็บสมดุล (Balanced Parentheses)

โปรแกรมใช้ Stack เพื่อตรวจสอบว่าการเปิดและปิดวงเล็บในสตริงมีความสมดุลหรือไม่

```
python
def is_balanced_parentheses(expression):
    stack = []
    for char in expression:
        if char in "({[":
            stack.append(char)
        elif char in ")}]":
            if not stack:
                return False
            top = stack.pop()
            if (top == '(' and char != ')') or \
               (top == '[' and char != ']') or \
               (top == '{' and char != '}'):
                return False
    return not stack

# ทดสอบการทำงาน
print(is_balanced_parentheses("({[]})")) # Output: True
print(is_balanced_parentheses("([)]")) # Output: False
```

ตัวอย่างที่ 3: การเปลี่ยนเลขฐาน (Decimal to Binary Conversion)

ใช้ Stack ในการแปลงเลขฐาน 10 ไปเป็นเลขฐาน 2

```
python
def decimal_to_binary(num):
    stack = []
    while num > 0:
        stack.append(num % 2)
        num //= 2

    binary = ""
    while stack:
        binary += str(stack.pop())
    return binary

# ทดสอบการทำงาน
print(decimal_to_binary(10)) # Output: 1010
print(decimal_to_binary(42)) # Output: 101010
```

ตัวอย่างที่ 2: การย้อนกลับข้อความ (Reverse a String)

ใช้ Stack ในการเก็บอักขระแต่ละตัวของข้อความ และนำออกมาทีละตัวเพื่อย้อนกลับข้อความ

```
python
def reverse_string(string):
    stack = []
    for char in string:
        stack.append(char)

    reversed_string = ""
    while stack:
        reversed_string += stack.pop()
    return reversed_string

# ทดสอบการทำงาน
print(reverse_string("Stack Example")) # Output: elpmoxE kcatS
```

ตัวอย่างที่ 4: การเลิกทำ (Undo Functionality)

โปรแกรมจำลองการทำงานแบบ "Undo" โดยใช้ Stack เก็บค่าของการกระทำก่อนหน้า

```
python
class TextEditor:
    def __init__(self):
        self.stack = []
        self.current_text = ""

    def type(self, text):
        self.stack.append(self.current_text)
        self.current_text += text

    def undo(self):
        if self.stack:
            self.current_text = self.stack.pop()
        else:
            print("ไม่มีการกระทำให้เลิกทำ!")

    def show(self):
        print(f"Current Text: '{self.current_text}'")

# ทดสอบการทำงาน
editor = TextEditor()
editor.type("Hello")
editor.show() # Output: Hello
editor.type(" World")
editor.show() # Output: Hello World
editor.undo()
editor.show() # Output: Hello
editor.undo()
editor.show() # Output: (ว่างเปล่า)
```

List ตัวอย่างโปรแกรมที่เก็บข้อมูลแบบ stack

1. จัดการลิสต์ของตัวเลข

โปรแกรมสำหรับเพิ่ม ลบ และค้นหาในลิสต์

```
python
# สร้างลิสต์ของตัวเลข
numbers = [10, 20, 30, 40, 50]

# เพิ่มค่าในลิสต์
numbers.append(60)
print("หลังจากเพิ่ม:", numbers)

# ลบค่าที่ระบุ
numbers.remove(30)
print("หลังจากลบค่า 30:", numbers)

# ค้นหาค่าที่ระบุ
if 40 in numbers:
    print("พบค่า 40 ในลิสต์ที่ตำแหน่ง:", numbers.index(40))
else:
    print("ไม่มีค่า 40 ในลิสต์")
```

หลังจากเพิ่ม: [10, 20, 30, 40, 50, 60]
หลังจากลบค่า 30: [10, 20, 40, 50, 60]
พบค่า 40 ในลิสต์ที่ตำแหน่ง: 2

4. คำนวณสถิติจากลิสต์

ใช้ลิสต์เก็บคะแนนนักเรียนและคำนวณค่าเฉลี่ย

```
python
# คะแนนของนักเรียน
scores = [75, 85, 90, 95, 88]

# คำนวณค่าเฉลี่ย
average = sum(scores) / len(scores)
print("คะแนนเฉลี่ย:", average)

# หาค่าสูงสุดและต่ำสุด
highest = max(scores)
lowest = min(scores)
print("คะแนนสูงสุด:", highest)
print("คะแนนต่ำสุด:", lowest)

# แสดงคะแนนที่มากกว่าค่าเฉลี่ย
above_average = [score for score in scores if score > average]
print("คะแนนที่มากกว่าค่าเฉลี่ย:", above_average)
```

คะแนนเฉลี่ย: 86.6
คะแนนสูงสุด: 95
คะแนนต่ำสุด: 75
คะแนนที่มากกว่าค่าเฉลี่ย: [90, 95, 88]

2. สร้างลิสต์ของชื่อและเรียงลำดับ

จัดเก็บชื่อในลิสต์แล้วจัดเรียงข้อมูลตามตัวอักษร

```
python
# สร้างลิสต์ของชื่อ
names = ["John", "Alice", "Zara", "Bob"]

# เรียงลำดับชื่อ
names.sort()
print("เรียงลำดับชื่อ:", names)

# เพิ่มชื่อใหม่และเรียงลำดับใหม่
names.append("Eve")
names.sort()
print("หลังจากเพิ่มและเรียงใหม่:", names)
```

เรียงลำดับชื่อ: ['Alice', 'Bob', 'John', 'Zara']
หลังเพิ่มและเรียงใหม่: ['Alice', 'Bob', 'Eve', 'John', 'Zara']

3. ลิสต์สองมิติ (Matrix)

จัดการลิสต์ในรูปแบบของเมทริกซ์ (Matrix)

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

# เข้าถึงค่าภายในเมทริกซ์
print("ค่าที่ตำแหน่ง [1][2]:", matrix[1][2]) # ค่าใน row 1 column 2

# แสดงเมทริกซ์
print("แสดงเมทริกซ์:")
for row in matrix:
    print(row)

# แก้ไขค่าในเมทริกซ์
matrix[0][1] = 99
print("หลังแก้ไขเมทริกซ์:")
for row in matrix:
    print(row)
```

ค่าที่ตำแหน่ง [1][2]: 6
แสดงเมทริกซ์:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
หลังแก้ไขเมทริกซ์:
[1, 99, 3]
[4, 5, 6]
[7, 8, 9]

Reverse String

- Some of the common ways to reverse a string are:
 - A. Using **Slicing** to create a reverse copy of the string.
 - B. Using **for loop** and appending characters in reverse order
 - C. Using **while loop** to iterate string characters in reverse order and append them
 - D. Using **string join()** function with reversed() iterator
 - E. Creating a **list** from the string and then calling its **reverse()** function
 - F. Using Recursion

10

Reverse String(Slicing)

sequence[start:stop:step]

```
def reverse_slicing(s):  
    return s[::-1]  
  
input_str = 'INE-KMUTNB'  
  
if __name__ == "__main__":  
    print('Reverse String using slicing =', reverse_slicing(input_str))
```

- if `__name__ == "__main__":` is used to execute some code only if the file was run directly, and not imported.

11

Reverse String (for loop)

```
def reverse_for_loop(s):  
    s1 = ''  
    for c in s:  
        s1 = c + s1 # appending chars in reverse order  
    return s1  
  
input_str = 'INE-KMUTNB'  
if __name__ == "__main__":  
    print('Reverse String using for loop =', reverse_for_loop(input_str))
```

12

Reverse String (while loop)

```
def reverse_while_loop(s):  
    s1 = ''  
    length = len(s) - 1  
    while length >= 0:  
        s1 = s1 + s[length]  
        length = length - 1  
    return s1  
  
input_str = 'INE-KMUTNB'  
if __name__ == "__main__":  
    print('Reverse String using while loop =', reverse_while_loop(input_str))
```

13

Reverse String(string join)

```
def reverse_str_join(s):  
    s1 = ''.join(reversed(s))  
    return s1  
  
input_str = 'INE-KMUTNB'  
if __name__ == "__main__":  
    print('Reverse String using sting join =', reverse_str_join(input_str))
```

14

Reverse String(list)

```
def reverse_list(s):  
    temp_list = list(s)  
    temp_list.reverse()  
    return ''.join(temp_list)  
  
input_str = 'INE-KMUTNB'  
if __name__ == "__main__":  
    print('Reverse String using list =', reverse_list(input_str))
```

15

Reverse String(recursive)

```
def reverse_recursion(s):  
    if len(s) == 0:  
        return s  
    else:  
        return reverse_recursion(s[1:]) + s[0]  
  
input_str = 'INE-KMUTNB'  
if __name__ == "__main__":  
    print('Reverse String using recursive =', reverse_recursion(input_str))
```

16

פקไว้อุ่นใจนิดนึ่ง

result = dict(zip(data,amount))

highest_data = max(dict_data, key=dict_data.get)

highest_value = dict_data[highest_data]