คอมพิวเตอร์โปรแกรมมิ่ง (โครงการพี่ติวน้อง)

เพื่อส่งเสริมการมีส่วนร่วมของรุ่นพี่ในการถ่ายทอดความรู้แก้รุ่นน้อง เพื่อเสริมความเข้าใจในรายวิชา

*ยงยุทธ ชวนขุนทด, เมธัส ทองจันทร์, ศุภกร จิรศิริวรกุล, มชัญชยา ประยูรมณีรัตน์

ประจำวันที่ 31 กรกฎาคม พ.ศ. 2568

ภาควิชาเทคโนโลยีสารสนเทศ

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ วิทยาเขตปราจีนบุรี

สารบัญ

1. รายการและเมธอดสำหรับชนิดข้อมูล (Lists and Data Type Methods) 2

1.1. การสร้างและเข้าถึงรายการ (Creating and Accessing Lists) 2

1.2. เมธอดสำหรับการจัดการรายการ (List Methods) 3

1.3. เมธอดสำหรับชนิดข้อมูลอื่นๆ (Methods for Other Data Types) 7

1.4. บททดสอบ 9

2. ฟังก์ชัน (Functions) 12

2.1. การสร้างและเรียกใช้ฟังก์ชัน (Creating and Calling Functions) 12

2.2. พารามิเตอร์และอาร์กิวเมนต์ (Parameters and Arguments) 15

2.3. ขอบเขตของตัวแปร (Variable Scope) 16

2.4. ฟังก์ชัน Built-in ที่สำคัญ (Important Built-in Functions) 17

2.5. บททดสอบ 19

3. การวนซ้ำด้วย While Loop 22

3.1. โครงสร้างพื้นฐานของ While Loop 22

3.2. การควบคุม While Loop ด้วย break และ continue 25

3.3. While Loop กับการประมวลผลข้อมูล 26

3.4. While Loop กับ Nested Loops 27

3.5. การป้องกัน Infinite Loop 28

3.6. บททดสอบ 30

1. รายการและเมธอดสำหรับชนิดข้อมูล (Lists and Data Type Methods)

รายการ (List) เป็นโครงสร้างข้อมูลพื้นฐานใน Python ที่ใช้เก็บข้อมูลหลายค่าไว้ในตัวแปรเดียว คิดเหมือนกับลิสต์รายการสิ่งของที่เราใช้ในชีวิตประจำวัน เช่น รายการซื้อของ รายชื่อเพื่อน หรือรายการคะแนนสอบ ข้อดีของ List คือสามารถเปลี่ยนแปลงข้อมูลได้หลังจากสร้างแล้ว (Mutable) และสามารถเก็บข้อมูลหลายประเภทไว้ด้วยกันได้ เช่น ตัวเลข ข้อความ และค่าความจริง การเรียนรู้การใช้งาน List และเมธอดต่างๆ จะช่วยให้นักศึกษาสามารถจัดการข้อมูลได้อย่างมีประสิทธิภาพ

1.1. การสร้างและเข้าถึงรายการ (Creating and Accessing Lists)

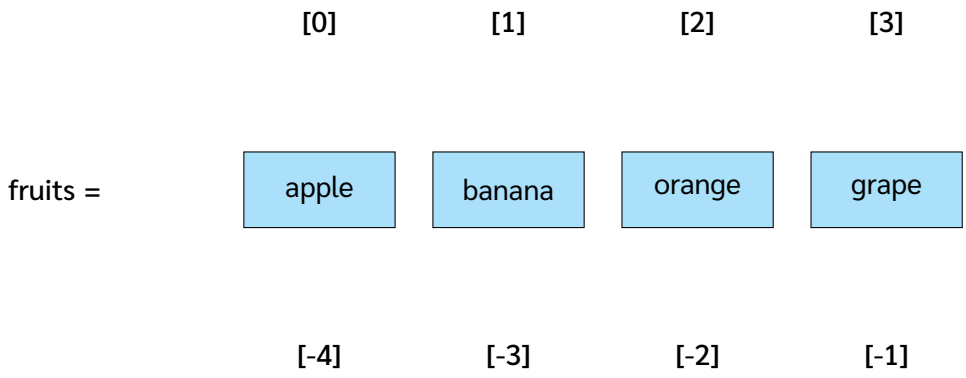
การสร้าง List ใน Python ง่ายมาก เพียงใช้เครื่องหมายวงเล็บเหลี่ยม [ ] และคั่นข้อมูลแต่ละตัวด้วยเครื่องหมายจุลภาค การเข้าถึงข้อมูลใน List ใช้หมายเลขดัชนี (Index) ซึ่งเริ่มนับจาก 0 สำหรับตำแหน่งแรก สิ่งที่ควรจำคือ Python รองรับการนับดัชนีแบบย้อนกลับด้วย โดยใช้เลขลบ เช่น -1 สำหรับตำแหน่งสุดท้าย นอกจากนี้ยังสามารถเลือกข้อมูลหลายตำแหน่งพร้อมกันได้ด้วยเทคนิค Slicing

(ตัวอย่าง) การสร้างและเข้าถึงรายการใน Python

```python
# Creating lists
fruits = ["apple", "banana", "orange", "grape"]
numbers = [1, 2, 3, 4, 5]
mixed = ["hello", 42, 3.14, True]

# Accessing list elements
print("First fruit:", fruits[0])
print("Last fruit:", fruits[-1])
print("Second number:", numbers[1])

# Slicing access
print("First 3 fruits:", fruits[0:3])
print("Last 2 fruits:", fruits[-2:])
```

ผลลัพธ์

```
First fruit: apple
Last fruit: grape
Second number: 2
First 3 fruits: ['apple', 'banana', 'orange']
Last 2 fruits: ['orange', 'grape']
```



รูปที่ 1: การแสดงดัชนีของรายการ (Positive และ Negative Index)

1.2. เมธอดสำหรับการจัดการรายการ (List Methods)

เมธอด (Methods) คือฟังก์ชันพิเศษที่ผูกติดกับ List และช่วยให้เราจัดการข้อมูลได้สะดวกขึ้น คิดเหมือนกับปุ่มกดต่างๆ บนรีโมทคอนโทรล แต่ละปุ่มมีหน้าที่เฉพาะ เช่น เพิ่มข้อมูล ลบข้อมูล หรือจัดเรียงข้อมูล การเรียนรู้เมธอดเหล่านี้จะช่วยให้นักศึกษาเขียนโปรแกรมได้อย่างมีประสิทธิภาพและลดความซับซ้อนของโค้ด

เมธอดสำคัญของรายการ (List Methods)

เมธอดที่ใช้บ่อยในการจัดการรายการ:

- **append(item):** เพิ่มสมาชิกใหม่ที่ท้ายรายการ
- **insert(index, item):** เพิ่มสมาชิกใหม่ที่ตำแหน่งที่กำหนด
- **remove(item):** ลบสมาชิกที่ระบุออกจากรายการ
- **pop(index):** ลบและคืนค่าสมาชิกที่ตำแหน่งที่กำหนด
- **index(item):** หาตำแหน่งของสมาชิกที่ระบุ
- **count(item):** นับจำนวนสมาชิกที่ระบุ
- **sort():** เรียงลำดับรายการ
- **reverse():** กลับลำดับรายการ
- **clear():** ลบสมาชิกทั้งหมดในรายการ

(ตัวอย่าง) การใช้เมธอด append() และ insert() ใน Python

```python
# Create fruit list
fruits = ["apple", "banana"]
print("Original fruits:", fruits)

# Add fruit to the end of the list
fruits.append("orange")
print("After append:", fruits)

# Insert fruit at position 1
fruits.insert(1, "mango")
print("After insert:", fruits)

# Add multiple fruits at once
more_fruits = ["grape", "kiwi"]
fruits.extend(more_fruits)
print("After extend:", fruits)
```

ผลลัพธ์

Original fruits: ['apple', 'banana']

After append: ['apple', 'banana', 'orange']

After insert: ['apple', 'mango', 'banana', 'orange']

After extend: ['apple', 'mango', 'banana', 'orange', 'grape', 'kiwi']

(ตัวอย่าง) การใช้เมธอด remove() และ pop() ใน Python

```python
# Create number list
numbers = [1, 2, 3, 4, 5, 3, 6]
print("Original numbers:", numbers)

# Remove number 3 (first occurrence)
numbers.remove(3)
print("After remove(3):", numbers)

# Remove element at position 2 and store the removed value
removed_item = numbers.pop(2)
print("After pop(2):", numbers)
print("Removed item:", removed_item)

# Remove last element
last_item = numbers.pop()
print("After pop():", numbers)
print("Last item:", last_item)
```

ผลลัพธ์

Original numbers: [1, 2, 3, 4, 5, 3, 6]
After remove(3): [1, 2, 4, 5, 3, 6]
After pop(2): [1, 2, 5, 3, 6]
Removed item: 4
After pop(): [1, 2, 5, 3]
Last item: 6

(ตัวอย่าง) การใช้เมธอด sort() และ reverse() ใน Python

```python
# Create number list
numbers = [64, 34, 25, 12, 22, 11, 90]
print("Original numbers:", numbers)

# Sort in ascending order
numbers.sort()
print("After sort():", numbers)

# Sort in descending order
numbers.sort(reverse=True)
print("After sort(reverse=True):", numbers)

# Reverse the list
numbers.reverse()
print("After reverse():", numbers)

# Sort words
words = ["python", "java", "c++", "javascript"]
words.sort()
print("Sorted words:", words)
```

ผลลัพธ์

Original numbers: [64, 34, 25, 12, 22, 11, 90]

After sort(): [11, 12, 22, 25, 34, 64, 90]

After sort(reverse=True): [90, 64, 34, 25, 22, 12, 11]

After reverse(): [11, 12, 22, 25, 34, 64, 90]

Sorted words: [’c++’, ’java’, ’javascript’, ’python’]

1.3. เมธอดสำหรับชนิดข้อมูลอื่นๆ (Methods for Other Data Types)

เช่นเดียวกับ List สตริง (String) ก็มีเมธอดของตัวเองที่ช่วยจัดการข้อความได้อย่างมีประสิทธิภาพ เมธอดเหล่านี้มีประโยชน์มากในการประมวลผลข้อความ เช่น การทำความสะอาดข้อมูล การค้นหาคำ หรือการแปลงรูปแบบข้อความ ซึ่งเป็นทักษะพื้นฐานที่นักศึกษาจะใช้บ่อยในการเขียนโปรแกรม

เมธอดสำคัญของสตริง (String Methods)

เมธอดที่ใช้บ่อยในการจัดการสตริง:

- **upper()**: แปลงเป็นตัวพิมพ์ใหญ่
- **lower()**: แปลงเป็นตัวพิมพ์เล็ก
- **strip()**: ลบช่องว่างหน้าและหลัง
- **split(separator)**: แยกสตริงตามตัวคั่น
- **join(list)**: รวมรายการเป็นสตริง
- **replace(old, new)**: แทนที่ข้อความ
- **find(substring)**: หาตำแหน่งของข้อความ
- **startswith(prefix)**: ตรวจสอบว่าเริ่มต้นด้วยข้อความที่ระบุ
- **endswith(suffix)**: ตรวจสอบว่าลงท้ายด้วยข้อความที่ระบุ

(ตัวอย่าง) การใช้เมธอดของสตริง ใน Python

```python
# Using basic string methods
text = "  Hello Python World  "
print("Original:", repr(text))
print("Upper:", text.upper())
print("Lower:", text.lower())
print("Strip:", text.strip())

# Splitting and joining strings
sentence = "Python is awesome"
words = sentence.split()
print("Split:", words)
print("Join:", "-".join(words))

# Replacing and searching
message = "I love Java programming"
new_message = message.replace("Java", "Python")
print("Replace:", new_message)
print("Find Python:", new_message.find("Python"))
print("Starts with 'I':", message.startswith("I"))
```

ผลลัพธ์

Original: ’ Hello Python World ’
Upper: HELLO PYTHON WORLD
Lower: hello python world
Strip: Hello Python World
Split: [’Python’, ’is’, ’awesome’]
Join: Python-is-awesome
Replace: I love Python programming
Find Python: 7
Starts with ’I’: True

1.4. บททดสอบ

โจทย์ที่ 1: โปรแกรมจัดการรายการอาหาร

จงเขียนโปรแกรมจัดการรายการอาหารโปรดง่ายๆ โดยใช้ List methods

ความต้องการ:

- สร้างรายการอาหารเริ่มต้น: [”pizza”, ”burger”, ”salad”]
- เพิ่มอาหาร ”pasta” ลงในรายการ
- ลบอาหาร ”salad” ออกจากรายการ
- แสดงรายการอาหารทั้งหมดเรียงตามตัวอักษร
- นับจำนวนอาหารทั้งหมด

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
Initial foods: ['pizza', 'burger', 'salad']
After adding pasta: ['pizza', 'burger', 'salad', 'pasta']
After removing salad: ['pizza', 'burger', 'pasta']
Sorted foods: ['burger', 'pasta', 'pizza']
Total foods: 3
```

พร้อมสร้าง Flowchart แสดงการทำงานของโปรแกรม

โจทย์ที่ 2: โปรแกรมประมวลผลชื่อ

จงเขียนโปรแกรมประมวลผลชื่อง่ายๆ โดยใช้ String methods

ความต้องการ:

- รับชื่อ: "john doe"
- แปลงเป็นตัวพิมพ์ใหญ่
- นับจำนวนตัวอักษร (ไม่นับช่องว่าง)
- แยกชื่อและนามสกุล
- สร้างชื่อย่อ (ตัวอักษรแรกของแต่ละคำ)

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
Original name: john doe
Uppercase name: JOHN DOE
Number of characters: 7
First name: john
Last name: doe
Initials: J.D.
```

พร้อมสร้าง Flowchart แสดงการทำงานของโปรแกรม

โจทย์ที่ 3: โปรแกรมคำนวณคะแนนง่าย

จงเขียนโปรแกรมคำนวณคะแนนง่ายๆ โดยใช้ List methods

ความต้องการ:

- มีคะแนนสอบ: [85, 90, 78, 92, 88]
- หาคะแนนสูงสุดและต่ำสุด
- คำนวณค่าเฉลี่ย
- นับจำนวนคะแนนที่มากกว่า 85
- เรียงคะแนนจากมากไปน้อย

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
Scores: [85, 90, 78, 92, 88]
Highest score: 92
Lowest score: 78
Average score: 86.6
Scores above 85: 3
Sorted scores: [92, 90, 88, 85, 78]
```

พร้อมสร้าง Flowchart แสดงการทำงานของโปรแกรม

2. ฟังก์ชัน (Functions)

ฟังก์ชัน (Functions) เป็นชุดคำสั่งที่ถูกจัดกลุ่มไว้ด้วยกันเพื่อทำงานเฉพาะอย่าง ฟังก์ชันช่วยให้โค้ดมีความเป็นระเบียบ สามารถนำกลับมาใช้ได้ (Reusable) และง่ายต่อการบำรุงรักษา การเขียนฟังก์ชันที่ดีจะช่วยลดการเขียนโค้ดซ้ำๆ และทำให้โปรแกรมมีประสิทธิภาพมากขึ้น

2.1. การสร้างและเรียกใช้ฟังก์ชัน (Creating and Calling Functions)

การสร้างฟังก์ชันใน Python ใช้คำสั่ง **def** ตามด้วยชื่อฟังก์ชันและพารามิเตอร์ในวงเล็บ การเรียกใช้ฟังก์ชันทำได้โดยเขียนชื่อฟังก์ชันตามด้วยวงเล็บและใส่อาร์กิวเมนต์ (ถ้ามี)

โครงสร้างของฟังก์ชัน

โครงสร้างพื้นฐานของฟังก์ชันใน Python:

```
def function_name(parameters):
    """Function description (Docstring)"""
    # Function body
    return value  # Optional
```

ส่วนประกอบสำคัญ:

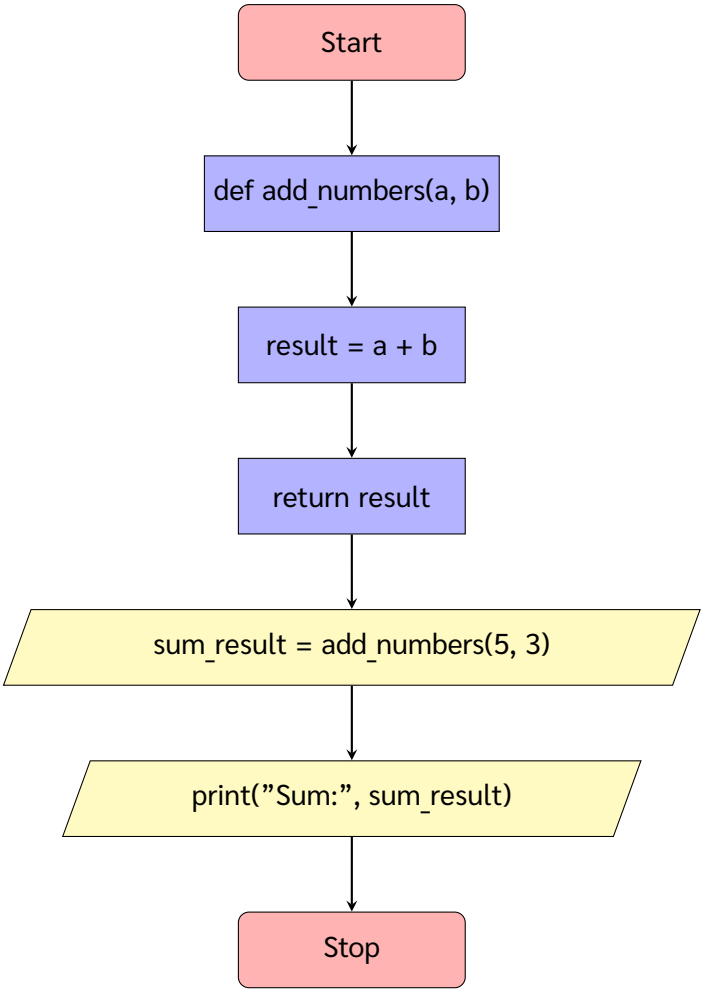
- **def**: คำสั่งสำหรับประกาศฟังก์ชัน
- **function_name**: ชื่อของฟังก์ชัน
- **parameters**: พารามิเตอร์ที่ฟังก์ชันรับเข้ามา
- **return**: คำสั่งส่งค่ากลับ (ไม่จำเป็นต้องมี)

(ตัวอย่าง) ฟังก์ชันพื้นฐานใน Python

```python
# Simple function without parameters
def say_hello():
    print("Hello, World!")

# Function with parameters
def greet(name):
    print(f"Hello, {name}!")

# Function that returns a value
def add_numbers(a, b):
    result = a + b
    return result

# Calling functions
say_hello()
greet("Alice")
sum_result = add_numbers(5, 3)
print("Sum:", sum_result)
```

ผลลัพธ์

```
Hello, World!
Hello, Alice!
Sum: 8
```



รูปที่ 2: Flowchart การสร้างและเรียกใช้ฟังก์ชัน

2.2. พารามิเตอร์และอาร์กิวเมนต์ (Parameters and Arguments)

พารามิเตอร์ (Parameters) คือตัวแปรที่กำหนดในฟังก์ชัน ส่วนอาร์กิวเมนต์ (Arguments) คือค่าที่ส่งให้ฟังก์ชันเมื่อเรียกใช้ Python มีวิธีการส่งอาร์กิวเมนต์หลายแบบ

ประเภทของพารามิเตอร์

ประเภทของพารามิเตอร์ใน Python:

- **Positional Arguments:** อาร์กิวเมนต์ตามตำแหน่ง
- **Keyword Arguments:** อาร์กิวเมนต์ตามชื่อ
- **Default Parameters:** พารามิเตอร์ที่มีค่าเริ่มต้น
- ***args:** อาร์กิวเมนต์จำนวนไม่จำกัด
- ****kwargs:** อาร์กิวเมนต์แบบ keyword จำนวนไม่จำกัด

(ตัวอย่าง) ประเภทของพารามิเตอร์ใน Python

```python
# Function with default parameters
def introduce(name, age=25, city="Bangkok"):
    print(f"Name: {name}, Age: {age}, City: {city}")

# Different ways to call the function
introduce("Alice")  # Use default values
introduce("Bob", 30)  # Specify age
introduce("Charlie", city="Chiang Mai")  # Use keyword argument
introduce("Diana", 28, "Phuket")  # Specify all values

# Function that accepts unlimited arguments
def sum_all(*numbers):
    total = 0
    for num in numbers:
        total += num
    return total

result1 = sum_all(1, 2, 3)
result2 = sum_all(1, 2, 3, 4, 5)
print("Sum 1:", result1)
print("Sum 2:", result2)
```

ผลลัพธ์

```
Name: Alice, Age: 25, City: Bangkok
Name: Bob, Age: 30, City: Bangkok
Name: Charlie, Age: 25, City: Chiang Mai
Name: Diana, Age: 28, City: Phuket
Sum 1: 6
Sum 2: 15
```

2.3. ขอบเขตของตัวแปร (Variable Scope)

ขอบเขตของตัวแปร (Variable Scope) กำหนดว่าตัวแปรสามารถเข้าถึงได้จากส่วนไหนของโปรแกรม ใน Python มีขอบเขตหลัก 2 ประเภท คือ ขอบเขตทั่วไป (Global Scope) และขอบเขตท้องถิ่น (Local Scope)

ขอบเขตของตัวแปร

ประเภทของขอบเขตตัวแปรใน Python:

- **Global Scope:** ตัวแปรที่สามารถเข้าถึงได้จากทุกส่วนของโปรแกรม
- **Local Scope:** ตัวแปรที่สามารถเข้าถึงได้เฉพาะภายในฟังก์ชัน
- **global keyword:** ใช้เพื่อแก้ไขตัวแปร global ภายในฟังก์ชัน
- **nonlocal keyword:** ใช้เพื่อแก้ไขตัวแปรในขอบเขตนอกสุด

(ตัวอย่าง) ขอบเขตของตัวแปรใน Python

```python
# Global variable
count = 0

def test_scope():
    # Local variable
    local_var = "Hello"
    print(f"Inside function: {count}")
    print(f"Local variable: {local_var}")

def modify_global():
    global count
    count = 10
    print(f"Modified global: {count}")

# Call functions
test_scope()
print(f"Outside function: {count}")

modify_global()
print(f"After modification: {count}")
```

ผลลัพธ์

Inside function: 0
Local variable: Hello
Outside function: 0
Modified global: 10
After modification: 10

2.4. ฟังก์ชัน Built-in ที่สำคัญ (Important Built-in Functions)

Python มีฟังก์ชัน Built-in จำนวนมากที่พร้อมใช้งานทันทีโดยไม่ต้อง import เพิ่มเติม ฟังก์ชันเหล่านี้ช่วยให้นักศึกษาสามารถเขียนโปรแกรมได้อย่างมีประสิทธิภาพมากขึ้น เช่น การแปลงข้อมูล การกรองข้อมูล หรือการคำนวณค่าสถิติพื้นฐาน ซึ่งเป็นทักษะที่จำเป็นในการพัฒนาโปรแกรม

(ตัวอย่าง) ฟังก์ชัน Built-in ที่สำคัญใน Python

```python
# Functions for data type conversion
numbers_str = ["1", "2", "3", "4", "5"]
numbers_int = list(map(int, numbers_str))
print("Converted to int:", numbers_int)

# filter() function - filtering data
def is_even(n):
    return n % 2 == 0

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(is_even, numbers))
print("Even numbers:", even_numbers)

# max(), min(), sum() functions
scores = [85, 92, 78, 96, 87]
print("Max score:", max(scores))
print("Min score:", min(scores))
print("Total score:", sum(scores))
print("Average score:", sum(scores) / len(scores))

# sorted() function
names = ["Alice", "Charlie", "Bob", "Diana"]
sorted_names = sorted(names)
print("Sorted names:", sorted_names)

# zip() function
students = ["Alice", "Bob", "Charlie"]
scores = [85, 92, 78]
student_scores = list(zip(students, scores))
print("Student scores:", student_scores)
```

ผลลัพธ์

Converted to int: [1, 2, 3, 4, 5]
Even numbers: [2, 4, 6, 8, 10]
Max score: 96
Min score: 78
Total score: 438
Average score: 87.6
Sorted names: [’Alice’, ’Bob’, ’Charlie’, ’Diana’]
Student scores: [(’Alice’, 85), (’Bob’, 92), (’Charlie’, 78)]

2.5. บททดสอบ

โจทย์ที่ 1: โปรแกรมคำนวณพื้นฐาน

จงเขียนโปรแกรมคำนวณง่ายๆ โดยใช้ Functions
ความต้องการ:

- สร้างฟังก์ชัน add(a, b) สำหรับการบวก
- สร้างฟังก์ชัน multiply(a, b) สำหรับการคูณ
- สร้างฟังก์ชัน square(x) สำหรับการยกกำลังสอง
- สร้างฟังก์ชัน calculate_area(length, width) สำหรับคำนวณพื้นที่สี่เหลี่ยม
- เรียกใช้ฟังก์ชันทั้งหมดและแสดงผล

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
Addition: 5 + 3 = 8
Multiplication: 4 * 6 = 24
Square of 7 = 49
Rectangle area (5 x 3) = 15 square units
```

พร้อมสร้าง Flowchart แสดงการทำงานของโปรแกรม

โจทย์ที่ 2: โปรแกรมจัดการคะแนนง่าย

จงเขียนโปรแกรมจัดการคะแนนโดยใช้ Functions

ความต้องการ:

- สร้างฟังก์ชัน calculate_average(scores) สำหรับคำนวณค่าเฉลี่ย
- สร้างฟังก์ชัน find_highest(scores) สำหรับหาคะแนนสูงสุด
- สร้างฟังก์ชัน count_passed(scores, passing_score) สำหรับนับจำนวนที่ผ่าน
- สร้างฟังก์ชัน get_grade(score) สำหรับแปลงคะแนนเป็นเกรด A, B, C, D, F
- ใช้คะแนนตัวอย่าง [85, 92, 78, 96, 73] และแสดงผลสถิติ

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
Scores: [85, 92, 78, 96, 73]
Average score: 84.8
Highest score: 96
Students passed (>=60): 5
Grade for score 85: B
Grade for score 92: A
Grade for score 78: C
```

พร้อมสร้าง Flowchart แสดงการทำงานของโปรแกรม

การใช้ Random Module

สำหรับการสุ่มตัวเลขใน Python:

- **import random:** นำเข้า module สำหรับการสุ่ม
- **random.randint(a, b):** สุ่มเลขจำนวนเต็มระหว่าง a ถึง b
- **random.choice(list):** สุ่มเลือกสมาชิกจากรายการ
- **random.random():** สุ่มเลขทศนิยมระหว่าง 0.0 ถึง 1.0

โจทย์ที่ 3: เกมทายตัวเลข

จงเขียนเกมทายตัวเลขง่ายๆ โดยใช้ Functions

ความต้องการ:

- สร้างฟังก์ชัน generate_number() สำหรับสุ่มตัวเลข 1-50
- สร้างฟังก์ชัน check_guess(secret, guess) สำหรับเปรียบเทียบคำตอบ
- สร้างฟังก์ชัน get_hint(secret, guess) สำหรับให้คำแนะนำ
- ให้ผู้เล่นทาย 3 ครั้ง
- แสดงผลลัพธ์การเล่น

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
Guess the number (1-50): 25
Too high! Try again.
Guess the number (1-50): 15
Too low! Try again.
Guess the number (1-50): 20
Correct! You win in 3 attempts.
```

3. การวนซ้ำด้วย While Loop

การวนซ้ำด้วย While Loop เป็นโครงสร้างการควบคุมที่สำคัญในการเขียนโปรแกรม ช่วยให้โปรแกรมสามารถทำงานซ้ำๆ ตราบใดที่เงื่อนไขที่กำหนดยังเป็นจริง (True) While Loop เหมาะสำหรับสถานการณ์ที่เราไม่ทราบจำนวนรอบการวนซ้ำล่วงหน้า เช่น การรับข้อมูลจากผู้ใช้จนกว่าจะถูกต้อง หรือการประมวลผลข้อมูลจนครบทุกรายการ ซึ่งเป็นทักษะที่นักศึกษาจะพบและใช้บ่อยในการแก้ปัญหาจริง

3.1. โครงสร้างพื้นฐานของ While Loop

While Loop ประกอบด้วยส่วนสำคัญ 3 ส่วน คือ เงื่อนไขการวนซ้ำ บล็อกคำสั่งที่จะทำงานซ้ำ และการอัพเดตตัวแปร การเข้าใจโครงสร้างนี้อย่างถูกต้องจะช่วยให้นักศึกษาเขียนโปรแกรมได้อย่างปลอดภัยและมีประสิทธิภาพ โดยเฉพาะการป้องกันปัญหา Infinite Loop ที่มักเกิดขึ้นกับผู้เริ่มเรียนโปรแกรมมิ่ง

โครงสร้างของ While Loop

โครงสร้างพื้นฐานของ While Loop ใน Python:

```
while condition:
    # Statements to be repeated
    # Must change variables related to condition
    pass
```

ส่วนประกอบสำคัญ:

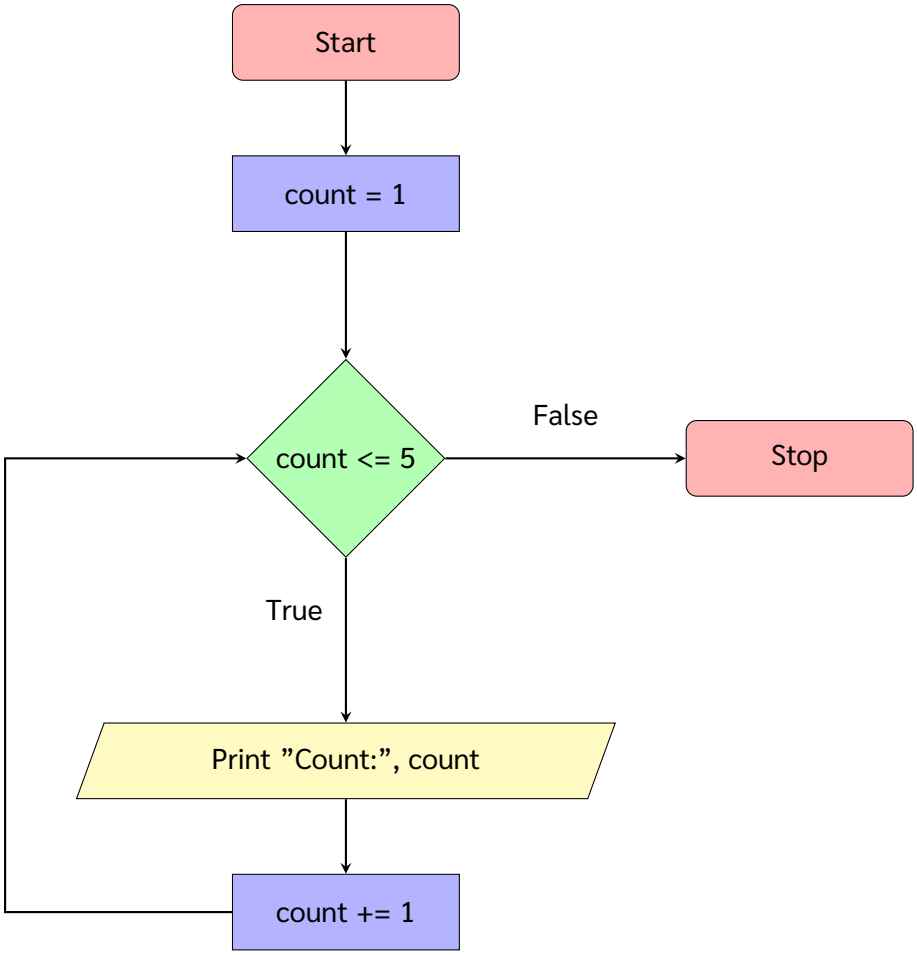
- **while:** คำสั่งสำหรับเริ่มต้นลูป
- **condition:** เงื่อนไขที่ต้องเป็นจริงเพื่อให้ลูปทำงานต่อ
- **loop body:** ชุดคำสั่งที่จะทำงานซ้ำ
- **update:** การเปลี่ยนแปลงตัวแปรเพื่อควบคุมลูป

(ตัวอย่าง) While Loop พื้นฐานใน Python

```python
# Example 1: Count from 1 to 5
count = 1
while count <= 5:
    print("Count:", count)
    count += 1

print("Loop finished!")

# Example 2: Sum numbers from 1 to 5
total = 0
number = 1
while number <= 5:
    total += number
    number += 1

print("Total sum:", total)
```

ผลลัพธ์

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
Loop finished!
Total sum: 15
```



รูปที่ 3: Flowchart ของ While Loop พื้นฐาน

3.2. การควบคุม While Loop ด้วย break และ continue

นอกจากเงื่อนไขหลักแล้ว Python ยังมีคำสั่งพิเศษสำหรับควบคุมการทำงานของลูปได้อย่างละเอียด ได้แก่ **break** และ **continue** การเข้าใจและใช้คำสั่งเหล่านี้อย่างถูกต้องจะช่วยให้นักศึกษาสามารถเขียนโปรแกรมที่มีความยืดหยุ่นและตอบสนองต่อสถานการณ์ต่างๆ ได้ดีขึ้น

คำสั่งควบคุมลูป

คำสั่งสำหรับควบคุมการทำงานของลูป:

- **break:** ออกจากลูปทันที (ไม่ว่าเงื่อนไขจะเป็นจริงหรือเท็จ)
- **continue:** ข้ามการทำงานที่เหลือในรอบปัจจุบัน และไปรอบถัดไป
- **else:** ทำงานเมื่อลูปจบแบบปกติ (ไม่ได้ใช้ break)
- **pass:** ไม่ทำอะไร (ใช้เป็น placeholder)

(ตัวอย่าง) การใช้ break และ continue ใน While Loop

```python
# Example of using break
count = 1
while count <= 10:
    if count == 5:
        break  # Stop loop when count = 5
    print(f"Count: {count}")
    count += 1

# Example of using continue
number = 1
while number <= 5:
    if number == 3:
        number += 1
        continue  # Skip number 3
    print(f"Number: {number}")
    number += 1
```

ผลลัพธ์

```
Count: 1
...

Number: 1
...
```

3.3. While Loop กับการประมวลผลข้อมูล

While Loop มีบทบาทสำคัญในการประมวลผลข้อมูลเมื่อเราไม่ทราบขนาดหรือจำนวนข้อมูลที่แน่นอนล่วงหน้า เช่น การอ่านไฟล์ข้อมูล การรับ input จากผู้ใช้แบบต่อเนื่อง หรือการค้นหาข้อมูลที่ตรงตามเงื่อนไข ทักษะนี้มีความสำคัญมากในการพัฒนาแอปพลิเคชันที่ทำงานกับข้อมูลจริง

(ตัวอย่าง) While Loop ในการประมวลผลข้อมูล

```python
# Example 1: Find average
numbers = [10, 20, 30, 40, 50]
total = 0
index = 0

while index < len(numbers):
    total += numbers[index]
    index += 1

average = total / len(numbers)
print(f"Average: {average}")

# Example 2: Count even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
even_count = 0
index = 0

while index < len(numbers):
    if numbers[index] % 2 == 0:
        even_count += 1
    index += 1

print(f"Even numbers count: {even_count}")
```

ผลลัพธ์

```
Average: 30.0
Even numbers count: 4
```

3.4. While Loop กับ Nested Loops

การซ้อน While Loop เป็นเทคนิคขั้นสูงที่ใช้สำหรับการประมวลผลข้อมูลที่มีโครงสร้างหลายมิติ เช่น การทำงานกับ matrix การสร้างตารางข้อมูล หรือการจำลองสถานการณ์ที่ซับซ้อน แม้จะเป็นเทคนิคที่ต้องใช้ความระมัดระวังในการเขียนเงื่อนไข แต่เป็นทักษะที่สำคัญสำหรับนักศึกษาระดับปริญญาตรี

(ตัวอย่าง) Nested While Loops ใน Python

```python
# Example 1: Create rectangle
row = 1
while row <= 3:
    col = 1
    while col <= 4:
        print("*", end=" ")
        col += 1
    print()  # New line
    row += 1

# Example 2: 2x2 multiplication table
row = 1
while row <= 2:
    col = 1
    while col <= 2:
        result = row * col
        print(f"{row}x{col}={result}", end="  ")
        col += 1
    print()
    row += 1
```

ผลลัพธ์

```
* * * *
* * * *
* * * *

1x1=1  1x2=2
2x1=2  2x2=4
```

3.5. การป้องกัน Infinite Loop

Infinite Loop หรือการวนซ้ำแบบไม่สิ้นสุดเป็นปัญหาที่พบบ่อยและสำคัญในการเขียนโปรแกรม เกิดจากเงื่อนไขที่ไม่เปลี่ยนแปลงหรือไม่มีทางเป็นเท็จได้ การเข้าใจและป้องกันปัญหานี้เป็นทักษะพื้นฐานที่นักศึกษาต้องเรียนรู้ เพื่อให้สามารถเขียนโปรแกรมที่ทำงานได้อย่างมั่นคงและปลอดภัย

เทคนิคป้องกัน Infinite Loop

วิธีการป้องกัน Infinite Loop:

- **ตรวจสอบเงื่อนไข:** ให้แน่ใจว่าเงื่อนไขสามารถเป็นเท็จได้
- **อัพเดทตัวแปร:** มีการเปลี่ยนแปลงตัวแปรในเงื่อนไขภายในลูป
- **ใช้ Counter:** กำหนดจำนวนรอบสูงสุดเป็นเงื่อนไขสำรอง
- **Debug อย่างระมัดระวัง:** ทดสอบลูปด้วยข้อมูลง่ายๆ ก่อน

(ตัวอย่าง) การป้องกัน Infinite Loop

```python
# Wrong example - Infinite Loop
# count = 1
# while count <= 5:
#     print(count)
#     # Forgot to increment count, causing infinite loop

# Correct example - Update variables
count = 1
while count <= 5:
    print(f"Count: {count}")
    count += 1  # Update variable to end the loop

# Example using safety counter
attempts = 0
max_attempts = 3
user_input = ""

while user_input != "exit" and attempts < max_attempts:
    user_input = input(f"Enter 'exit' (attempt {attempts + 1}/{max_attempts}): ")
    print(f"You entered: {user_input}")
    attempts += 1

if attempts >= max_attempts:
    print("Maximum attempts reached.")
```

ผลลัพธ์

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5

Enter 'exit' (attempt 1/3): hello
You entered: hello
Enter 'exit' (attempt 2/3): world
You entered: world
Enter 'exit' (attempt 3/3): exit
You entered: exit
```

3.6. บททดสอบ

โจทย์ที่ 1: โปรแกรมจัดการรายการสินค้าง่าย

จงเขียนโปรแกรมจัดการรายการสินค้าโดยใช้ While Loop

ความต้องการ:

- สร้างเมนูแสดงตัวเลือก: 1-Add, 2-Remove, 3-Show, 4-Exit
- ใช้ While Loop เพื่อให้โปรแกรมทำงานต่อเนื่องจนกว่าผู้ใช้เลือก Exit
- เริ่มต้นด้วยรายการสินค้า: [”apple”, ”banana”]
- เพิ่มสินค้าใหม่เมื่อเลือก Add
- ลบสินค้าเมื่อเลือก Remove
- แสดงรายการสินค้าทั้งหมดเมื่อเลือก Show

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
=== Shopping List Manager ===
1. Add item
2. Remove item
3. Show all items
4. Exit
Enter choice (1-4): 3
Current items: ['apple', 'banana']

Enter choice (1-4): 1
Enter item to add: orange
Added: orange

Enter choice (1-4): 3
Current items: ['apple', 'banana', 'orange']

Enter choice (1-4): 4
Goodbye!
```

โจทย์ที่ 2: เกมทายตัวเลขง่าย

จงเขียนเกมทายตัวเลขโดยใช้ While Loop

ความต้องการ:

- สุ่มตัวเลข 1-20 สำหรับให้ผู้เล่นทาย
- ใช้ While Loop ให้ผู้เล่นทายได้สูงสุด 5 ครั้ง
- ให้คำแนะนำ "Too high!" หรือ "Too low!" หลังแต่ละครั้ง
- แสดงจำนวนครั้งที่เหลือหลังแต่ละครั้ง
- ถ้าทายถูกให้แสดงผลชนะและจบเกม
- ถ้าทายครบ 5 ครั้งแล้วยังไม่ถูกให้แสดงคำตอบ

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
=== Number Guessing Game ===
I'm thinking of a number between 1-20
You have 5 attempts to guess it!

Attempt 1/5 - Enter your guess: 15
Too high! Try again.

Attempt 2/5 - Enter your guess: 8
Too low! Try again.

Attempt 3/5 - Enter your guess: 12
Congratulations! You guessed it in 3 attempts!
The number was 12
```

โจทย์ที่ 3: โปรแกรมคำนวณคะแนนสอบ

จงเขียนโปรแกรมคำนวณคะแนนสอบโดยใช้ While Loop

ความต้องการ:

- ใช้ While Loop รับคะแนนสอบจากผู้ใช้ (ป้อน -1 เพื่อจบ)
- ตรวจสอบคะแนนต้องอยู่ระหว่าง 0-100 เท่านั้น
- คำนวณค่าเฉลี่ย คะแนนสูงสุด และต่ำสุด
- นับจำนวนคะแนนที่ผ่าน (≥60) และไม่ผ่าน (<60)
- แสดงผลสรุปเมื่อผู้ใช้ป้อน -1
- ถ้าไม่มีคะแนนให้แสดงข้อความเตือน

ตัวอย่าง Output ที่ต้องการ:

ผลลัพธ์ที่คาดหวัง

```
=== Score Calculator ===
Enter scores (0-100), enter -1 to finish:

Enter score: 85
Valid score added: 85

Enter score: 92
Valid score added: 92

Enter score: 150
Invalid! Score must be between 0-100

Enter score: 78
Valid score added: 78

Enter score: -1

=== Summary ===
Total scores: 3
Scores: [85, 92, 78]
Average: 85.0
Highest: 92
Lowest: 78
Passed (>=60): 3
Failed (<60): 0
```