Programming Fundamentals I Lab.

3. ฟังก์ชัน

ชื่อ_____ รหัสนิสิต_____

ในปฏิบัติการนี้ คุณจะได้รู้จักการเขียนฟังก์ชันที่มีการรับค่าและคืนค่าในรูปแบบต่าง ๆ รวมถึงชนิดข้อมูลพิเศษ `None` และการเขียนคำอธิบาย (comment) ในโปรแกรม

3.1 ฟังก์ชันเบื้องต้น

สร้างไฟล์ `.py` ใหม่ และพิมพ์โค้ดโปรแกรมต่อไปนี้

```
def free():
    print("I am free!!!")
```

หลังจากนั้นให้เซฟและทดลองรันโปรแกรม มีผลอะไรเกิดขึ้นหรือไม่

ให้คุณลองเพิ่มโค้ดเป็นดังนี้

```
def free():
    print("I am free!!!")
print(type(free))
print(free)
```

รันโปรแกรมและบันทึกผลลัพธ์ที่ได้

จากการทดลอง คุณคิดว่าตัวแปร free มีชนิดข้อมูลเป็นประเภทใด

ให้คุณแก้ไขโค้ดเป็นดังนี้

```
def free():
    print("I am free!!!")
print(type(free))
free()
```

รันโปรแกรมและบันทึกผลลัพธ์ที่ได้

ให้คุณแก้ไขโค้ดเป็นดังนี้

```
def free():
    i = 0
    while i < 3:
        print("I am free!!!")
    i += 1
print(type(free))
free()
```

รันโปรแกรมและบันทึกผลลัพธ์ที่ได้

จากการทดลอง คุณคิดว่าคำสั่ง `free()` มีการทำงานอย่างไร

คุณคิดว่าคีย์เวิร์ด `def` มีหน้าที่อย่างไร

ทดลองเปลี่ยนชื่อฟังก์ชันจาก `free` เป็นรูปแบบต่าง ๆ บันทึกผลว่าการตั้งชื่อฟังก์ชันที่ใช้ได้ต้องเป็นเช่นไร

3.2 การรับค่าและคืนค่า

ทดลองแก้ไขโปรแกรมเป็นดังนี้

```
def square(x):
    print(x**2)
square()
```

รันโปรแกรมและบันทึกผลลัพธ์ที่ได้

เปลี่ยนบรรทัดที่เรียก `square()` เป็น `square(5)` รันโปรแกรมและบันทึกผลลัพธ์ที่ได้

ทดลองเปลี่ยนบรรทัด `square(5)` เป็นค่าดังต่อไปนี้ บันทึกผลที่เกิดขึ้นในแต่ละกรณี

`square(-30)`	
`square(24.69)`	
`square(6-2)`	
`square(4+5.37)`	
`square("FREE")`	

แก้ไขโค้ดโปรแกรมเป็นดังนี้

```
def square(x):
    return x**2
square(5)
```

รันโปรแกรมและบันทึกผลลัพธ์ที่ได้

แก้ไขบรรทัด `square(5)` ให้เป็น `print(square(5))` รันและบันทึกผลที่เกิดขึ้น

แก้ไขโค้ดโปรแกรมเป็นดังนี้

```
def square(x):
    return x**2
s = square(5)
print(s)
```

รันโปรแกรมและบันทึกผลลัพธ์ที่ได้

ทดลองเปลี่ยนบรรทัด `s = square(5)` เป็นค่าดังต่อไปนี้ บันทึกผลที่เกิดขึ้นในแต่ละกรณี

`s = 3 + square(-30)`	
`s = square(24.69)/7`	
`s = square(6-2) + square(-3)`	
`s = square(square(4)+5.37)`	
`s = "FREE" * square(-2)`	

คุณคิดว่าคำสั่ง `return` มีหน้าที่อย่างไร

แก้ไขโค้ดโปรแกรมเป็นดังนี้

```
def range_sum(a, b):
    s = 0
```

```
    i = a
    while i <= b:
        s += i
        i += 1
    return s
```

เมื่อรันโปรแกรมจะไม่มีการแสดงค่าใด ๆ เนื่องจากในโค้ดยังไม่มีคำสั่งสำหรับแสดงค่า ให้ลองพิมพ์ `range_sum(10, 50)` ใน Python Shell แล้วบันทึกค่าที่ได้

ให้ทดลองเรียกฟังก์ชัน `range_sum()` ด้วยตัวเลขคู่ต่าง ๆ สังเกตและอธิบายว่าฟังก์ชันนี้คืนค่าเป็นอะไร

ถ้าเราเรียกฟังก์ชัน `range_sum()` โดยใส่ค่า `a` มากกว่า `b` จะได้ผลอย่างไร

จงสร้างฟังก์ชันที่ชื่อ `age_check()` ที่รับพารามิเตอร์สองตัวได้แก่ `name` เป็นสตริงที่แทนชื่อคน และ `age` เป็นจำนวนที่แทนอายุของคนคนนั้น และให้ฟังก์ชันนี้แสดงข้อความออกหน้าจอว่า `Hey,` ตามด้วยชื่อคนที่รับมา และต่อด้วย `, you are old.` ถ้าตัวแปร `age` ที่รับมามีค่าตั้งแต่ 40 ขึ้นไป ไม่เช่นนั้นให้ต่อด้วย `, you are young.`

เขียนฟังก์ชันลงในช่องด้านล่างนี้

3.3 None และ default parameter

ในภาษา Python มีชนิดข้อมูลพิเศษชื่อว่า `NoneType` ซึ่งมีค่าที่เป็นไปได้เพียงค่าเดียวคือ `None` ไว้ใช้แทนความ *ไม่มีอะไร*

ทดลองสร้างฟังก์ชันต่อไปนี้

```
def square(x):
    if type(x) == int or type(x) == float:
        return x**2
    else:
        return None
```

ทดลองรันและเรียกใช้ฟังก์ชัน `square` ด้วยคำสั่งต่อไปนี้ บันทึกผลที่ได้

`print(square(-30))`	
`print(square(24.69))`	
`print(square("FREE"))`	
`print(square(4>5))`	
`print(square(None))`	
`print(square("FREE"+3))`	

จงปรับปรุงฟังก์ชัน `range_sum()` ให้คืนค่าเป็น `None` หากค่า `a` ที่รับมามีค่ามากกว่า `b` เขียนโค้ดทั้งหมดของฟังก์ชันหลังการปรับปรุงในช่องด้านล่างนี้

ทดลองสร้างฟังก์ชันต่อไปนี้

```
def range_sum(to, start_at = 1, step = 1):
    s = 0
```

```
    i = start_at
    while i <= to:
        s += i
        i += step
    return s
```

ทดลองรันและเรียกใช้ฟังก์ชัน `range_sum()` ด้วยคำสั่งต่อไปนี้ บันทึกผลที่ได้

`range_sum(10, 6, 2)`	
`range_sum(10, 6)`	
`range_sum(10)`	
`range_sum(10, step = 2)`	

3.4 comment

ในภาษา Python เราสามารถแทรกคำอธิบายหรือข้อความที่จะไม่ถูกนำไปประมวลผลได้ เพื่อบันทึกรายละเอียดหรือคำอธิบายในโค้ดแต่ละส่วนให้สามารถอ่านเข้าใจได้ง่ายในภายหลัง

ทดลองแก้ไขโค้ดตามตัวอย่างดังนี้

```
def is_prime(n):
    # Return True if n is a prime number
    # otherwise, return False

    if type(n) != int:
        return False # A prime must be an integer.
    elif n <= 1:
        return False # A prime must be more than 1.
    else:
        # For n more than 1,
        # check that if there is an int i(1<i<n) such that n%i==0.
        # If there is such i, n is not a prime.

        i = 2
        while i<n:
            if n%i == 0:
                return False
            i += 1
        return True
```

```
def generate_primes(to, start_at = 1):
    # Print all prime numbers in the range [start_at, to]

    i = start_at
    while i <= to:
        if is_prime(i):
            print(i)
        i += 1
```

รันโปรแกรมและทดลองเรียกฟังก์ชัน `is_prime()` และ `generate_primes()` ด้วยพารามิเตอร์หลาย ๆ แบบ อธิบายว่าฟังก์ชัน `is_prime()` ทำหน้าที่อะไร

ฟังก์ชัน `generate_primes()` ทำหน้าที่อะไร

จงอธิบายวิธีการแทรก comment ในโค้ดโปรแกรมภาษา Python

3.5 ขอบเขตของตัวแปร

ทดลองแก้ไขโค้ดตามตัวอย่างดังนี้

```
def f1(a):
    b = 10
    print("In f1, a = "+str(a)+", b = "+str(b))

a = 1
b = 2
print("In global, a = "+str(a)+", b = "+str(b))
f1(a+b)
print("In global, a = "+str(a)+", b = "+str(b))
```

ทดลองรันและบันทึกผลที่ได้

จากการทดลอง ตัวแปร `a` ที่อยู่ในฟังก์ชัน `f1()` และตัวแปร `a` ที่อยู่นอกฟังก์ชัน `f1()` เป็นตัวแปรเดียวกันหรือไม่

ตัวแปร `b` ที่อยู่ในฟังก์ชัน `f1()` และตัวแปร `b` ที่อยู่นอกฟังก์ชัน `f1()` เป็นตัวแปรเดียวกันหรือไม่

แก้ไขโค้ดตามตัวอย่างดังนี้

```
def f1(a):
    print("In f1, a = "+str(a)+", b = "+str(b))
f1(3)
```

ทดลองรันและบันทึกผลที่ได้

แก้ไขโค้ดตามตัวอย่างดังนี้

```
def f1(a):
    print("In f1, a = "+str(a)+", b = "+str(b))
a = 1
b = 2
f1(a+b)
```

ทดลองรันและบันทึกผลที่ได้

จากการทดลอง ตัวแปร `a` ที่อยู่ในฟังก์ชัน `f1()` และตัวแปร `a` ที่อยู่นอกฟังก์ชัน `f1()` เป็นตัวแปรเดียวกันหรือไม่

ตัวแปร `b` ที่อยู่ในฟังก์ชัน `f1()` และตัวแปร `b` ที่อยู่นอกฟังก์ชัน `f1()` เป็นตัวแปรเดียวกันหรือไม่

ทำไมจึงไม่ขึ้นข้อผิดพลาดว่าไม่รู้จักตัวแปร `b` เหมือนการทดลองที่แล้ว ทั้ง ๆ ที่ในฟังก์ชัน `f1()` ก็ไม่ได้มีการสร้างตัวแปร `b` เช่นกัน

3.6 ฟังก์ชันเวียนเกิด

แก้ไขโค้ดตามตัวอย่างดังนี้

```
def r1(n):
    if n>0:
        print(n)
        r1(n-1)
```

ทดลองรัน และเรียกคำสั่ง `r1(5)` ใน Python Shell บันทึกผลที่ได้

ทดลองแก้ไขบรรทัด `r1(n-1)` ให้เป็น `r1(n+1)` และเรียกคำสั่ง `r1(5)` ใน Python Shell ใหม่ บันทึกผลที่ได้

ทำไมโปรแกรมจึงทำงานไม่สิ้นสุด

แก้ไขโค้ดตามตัวอย่างดังนี้

```
def p(a, b):
    if b == 0:
        return 1
    else:
        return a * p(a, b-1)
```

ทดลองรันและเรียกใช้โดยใส่ค่า `a` และ `b` เป็นจำนวนเต็มต่าง ๆ สังเกตและอธิบายว่าฟังก์ชัน `p()` ทำหน้าที่อะไร

แก้ไขโค้ดตามตัวอย่างดังนี้

```
def fib(n):
    print("fib(",n,")")
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

ทดลองรันและเรียกคำสั่ง `fib(3)` บันทึกผลที่ได้

อธิบายขั้นตอนการทำงานเมื่อเรียกคำสั่ง `fib(3)` โดยละเอียด ว่าทำไมถึงได้ผลเช่นนี้

3.7 โจทย์ปัญหา

1. *จำนวนสมบูรณ์* (perfect number) คือจำนวนเต็มบวกที่มีค่าเท่ากับผลบวกของตัวหารทั้งหมด (ยกเว้นตัวมันเอง) ตัวอย่างเช่น $6 = 1+2+3$, $28 = 1+2+4+7+14$ เป็นต้น จงเขียนฟังก์ชันชื่อ `is_perfect_number()` ที่รับพารามิเตอร์หนึ่งตัว และคืนค่าเป็น `True` ถ้าค่าที่รับมาเป็นจำนวนสมบูรณ์ ไม่เช่นนั้นให้คืนค่าเป็น `False`

   ดูตัวอย่างการเรียกใช้งานดังนี้

```
>>> is_perfect_number(6)
True
>>> is_perfect_number(15)
False
>>> is_perfect_number(28)
True
>>> is_perfect_number(28.5)
False
>>> is_perfect_number("28")
False
```

2. ถ้า n เป็นจำนวนเต็มบวก *แฟกทอเรียล* n ($n!$) จะมีค่าเท่ากับผลคูณ $1 \times 2 \times \ldots \times n$ สำหรับ $n = 0$ เรานิยามให้ $0! = 1$ สำหรับค่านอกเหนือจากนี้จะไม่มีนิยามของแฟกทอเรียล จงเขียนฟังก์ชันชื่อ `factorial()` ที่รับพารามิเตอร์หนึ่งตัว n และคืนค่าเป็น $n!$ หากพารามิเตอร์ที่รับมาไม่มีนิยามสำหรับแฟกทอเรียล ให้คืนค่าเป็น `None`

   ดูตัวอย่างการเรียกใช้งานดังนี้

```
>>> print(factorial(-4))
None
>>> print(factorial(4))
24
```

```
>>> print(factorial(0))
1
```

3. ห.ร.ม. (gcd) ของจำนวนเต็มบวก a และ b คือจำนวนเต็มที่มีค่ามากที่สุดที่หารทั้ง a และ b ลงตัว ตัวอย่างเช่น ห.ร.ม. ของ 8 กับ 12 คือ 4, ห.ร.ม. ของ 5 กับ 25 คือ 5 จงเขียนฟังก์ชันชื่อ `gcd()` ที่รับพารามิเตอร์เป็นจำนวนเต็มสองตัว และคืนค่าเป็น ห.ร.ม. ของจำนวนเต็มทั้งคู่ (ไม่ต้องสนใจกรณีพารามิเตอร์เป็นค่าอื่น ๆ)

   ดูตัวอย่างการเรียกใช้งานดังนี้

```
>>> gcd(8, 12)
4
>>> gcd(25, 5)
5
>>> gcd(400, 400)
400
```

4. จงเขียนฟังก์ชันชื่อ `range_product()` ที่รับพารามิเตอร์สองตัวได้แก่ `to` และ `start_at` โดยให้ฟังก์ชันคืนค่าเป็นผลคูณของจำนวนเต็มตั้งแต่ `start_at` จนถึง `to` หากผู้ใช้เรียกฟังก์ชันโดยใส่ค่า `to` อย่างเดียว ให้ถือว่า `start_at` มีค่าเป็น 1 และหากผู้ใช้เรียกฟังก์ชันโดยใส่ค่า `to` น้อยกว่า `start_at` ให้ฟังก์ชันคืนค่าเป็น None ข้อกำหนดเพื่อความท้าทายสำหรับข้อนี้คือ **ให้คุณเขียนฟังก์ชันนี้โดยห้ามใช้คำสั่งวนรอบเช่น `while` หรือ `for` ทั้งสิ้น** คุณสามารถสร้างฟังก์ชันนี้ได้โดยเขียนเป็นฟังก์ชันเวียนเกิด

   ดูตัวอย่างการเรียกใช้งานดังนี้

```
>>> range_product(10)
3628800
>>> range_product(10, 7)
5040
>>> print(range_product(10, 20))
None
```