Functions

Functions

- Function definition
  - parameter
  - return
  - local variable
  - global variable

Function

ฟังก์ชัน (function) เป็นโปรแกรมย่อยที่สร้างขึ้นเพื่อ

- ให้ทำงานเฉพาะอย่าง
- สามารถนำมาใช้ได้ในภายหลัง
- ทำให้เกิดความสะดวกในการจัดทำโปรแกรมขนาดใหญ่
- และช่วยในการตรวจสอบความถูกต้องของโปรแกรมได้ง่ายขึ้น

ในภาษาไพธอนมีฟังก์ชันให้ใช้งาน (Built-in function) อยู่แล้วมากมาย บทนี้ เราจะมาเรียนการสร้างฟังก์ชันเอง

Function definition (การนิยามฟังก์ชัน)

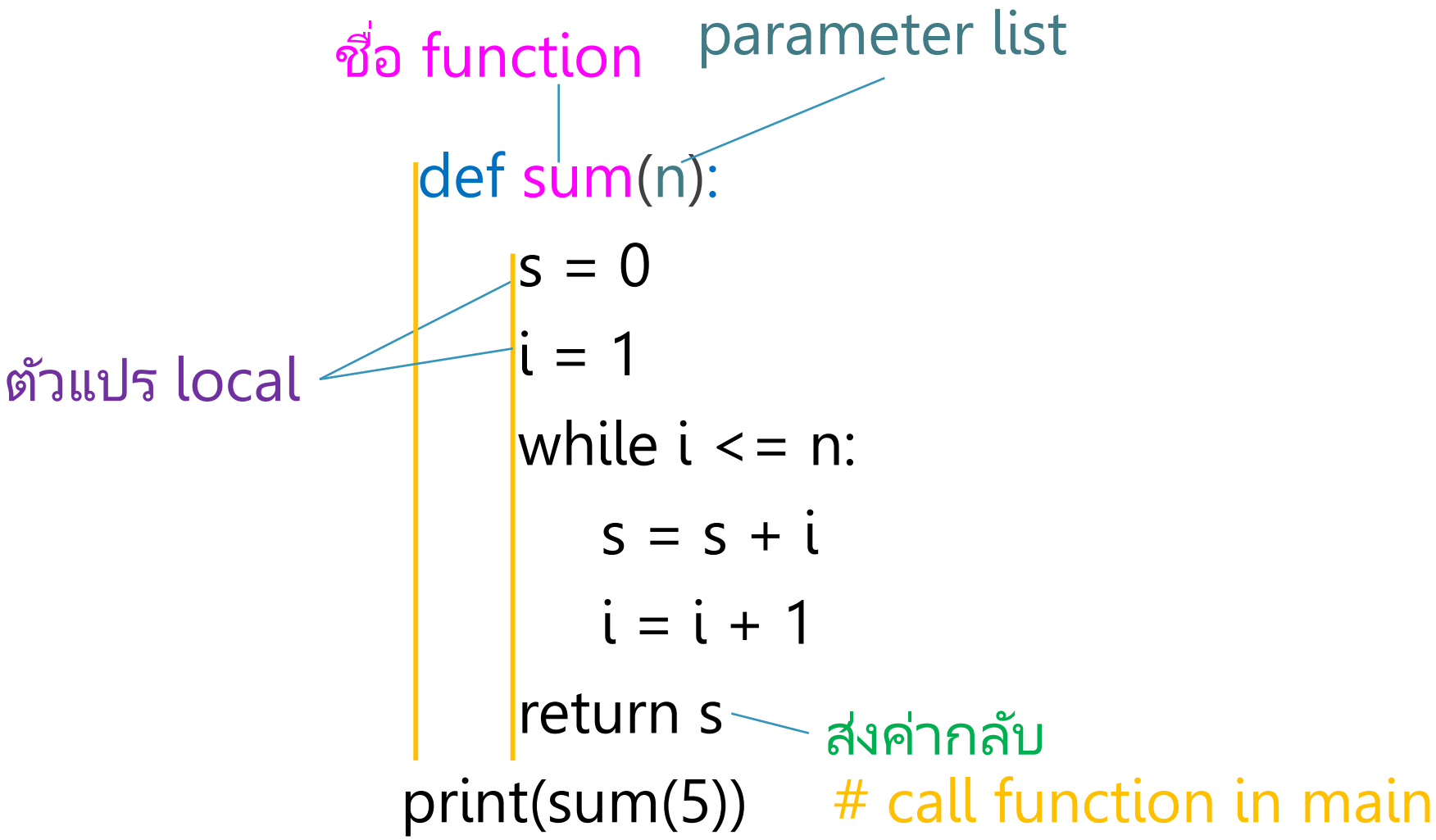
- โครงสร้างของฟังก์ชัน

```
def function_name (parameter list):
    function_body
```

ย่อหน้า (indent)

- def คือ keyword ที่บอกว่า เป็นจุดเริ่มต้นการนิยามฟังก์ชัน
- ชื่อฟังก์ชัน มีกฎการตั้งชื่อเหมือนกับการตั้งชื่อตัวแปร
- ภายในวงเล็บ คือ ชื่อตัวแปรที่รับค่าที่ส่งเข้ามาทำงานภายในฟังก์ชัน ถ้าไม่มีก็ปล่อยว่างไว้ ถ้ามีมากกว่า 1 ตัวแปรให้คั่นด้วยเครื่องหมาย comma (,)
- จบบรรทัด def ด้วยเครื่องหมาย colon (:)
- function_body คือชุดคำสั่งโปรแกรมที่จะทำงานเพื่อให้ได้ผลลัพธ์ตามที่ต้องการ เขียนคำสั่งภายในฟังก์ชันโดยต้องเยื้องย่อหน้า

ตัวอย่าง : function เพื่อหาผลรวมของ 1 ถึง n



Parameter list

- รายการของชื่อตัวแปรที่รับค่าจากคำสั่งเรียกใช้ฟังก์ชันที่ส่งค่าข้อมูลเข้ามาทำงานภายในฟังก์ชัน ถ้าไม่มีก็ปล่อยว่างไว้ ถ้ามีมากกว่า 1 ตัวแปรให้คั่นด้วยเครื่องหมาย comma (,)
- ถ้าตัวแปร parameter มีชนิดเป็น int, float, string, boolean จะมีการสร้างเนื้อที่ใหม่ในหน่วยความจำเพื่อเก็บค่าของตัวแปร ดังนั้น ถ้าตั้งชื่อตัวแปร parameter ซ้ำกับชื่อตัวแปรในโปรแกรมหลัก โปรแกรมจะมองเป็นคนละตัวแปรกัน เมื่อทำงานจนจบฟังก์ชันแล้ว จะคืนเนื้อที่ในหน่วยความจำที่จองไว้เก็บค่าตัวแปร parameter เพื่อให้โปรแกรมอื่นนำเนื้อที่นั้นไปใช้งานต่อไป (ในบทนี้เรียนแบบนี้ก่อน)
- ถ้าตัวแปร parameter มีชนิดเป็น list, dict จะมีการเชื่อมตัวแปรพารามิเตอร์นี้ไปผูกกับค่าตัวแปรที่ผู้เรียกใช้ฟังก์ชันส่งเข้ามา ถ้ามีการแก้ค่าในฟังก์ชัน ก็จะแก้ค่าตัวแปรนั้นในส่วนของผู้เรียกใช้ฟังก์ชันด้วย

Local variable

- ตัวแปรที่กำหนดค่าในฟังก์ชัน คอมพิวเตอร์จะสร้างเนื้อที่ในหน่วยความจำเพื่อเก็บค่าของตัวแปรให้ โดยจะมีขอบเขตการมองเห็นตัวแปรแบบ local นั่นคือ ถูกเรียกใช้ได้จากคำสั่งที่อยู่ภายในฟังก์ชันเท่านั้น
- เมื่อทำงานจนจบฟังก์ชันแล้ว จะมีการคืนเนื้อที่ในหน่วยความจำที่จองไว้เก็บค่าตัวแปรแบบ local variables เหล่านี้ เพื่อให้โปรแกรมอื่นนำเนื้อที่นั้นไปใช้งานต่อไป
- ดังนั้น ถ้าตั้งชื่อตัวแปร local ซ้ำกับชื่อตัวแปรในโปรแกรมหลัก โปรแกรมจะมองเป็นคนละตัวแปรกัน

return statement

- คำสั่ง return คือคำสั่งที่สั่งให้หยุดการทำงานของฟังก์ชันแล้วกลับออกไปยังจุดที่เรียกใช้ฟังก์ชัน พร้อมทั้งส่งค่าคืน (ถ้ามี)
- return None คือ หยุดการทำงานของฟังก์ชันแล้วกลับออกไปโดยไม่ส่งค่าคืน
- return None = return
- ถ้าไม่เจอคำสั่ง return จะ return จาก function เมื่อจบคำสั่งสุดท้ายของ function โดยไม่ส่งค่าคืน
- สามารถมีคำสั่ง return มากกว่า 1 คำสั่งในฟังก์ชัน
- ส่งค่าคืนได้ค่าเดียว ถ้าต้องการคืนหลายค่า ใช้ list, tuple เช่น

```
return -1                          return a #เช่น a = 10
return b #เช่น b = "end"           return x #เช่น x = [1, 2, 3]
```

ตัวอย่าง : function เพื่อหาผลรวมของ 1 ถึง n

การรันโปรแกรม

```
def sum(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
#in main program
print(sum(5))
```

เรียกใช้ function ในโปรแกรมหลัก

```
#ไม่ใช้ฟังก์ชัน
s = 0
i = 1
while i <= 5:
   s = s+i
   i = i+1
print(s)
```

ตัวอย่าง : function เพื่อหา n!

```
def find_fac(n):
    fac = 1
    for x in range(1, n+1):
        fac = fac * x
    return fac
#in main program
n = int(input("Enter n : "))
print(find_fac(n))        # เรียกใช้ function
```

```
#ไม่ใช้ฟังก์ชัน
n = int(input('Enter n : '))
fac = 1
for x in range(1, n+1):
  fac = fac * x
print(fac)
```

```
#ในโปรแกรมหลักอาจเขียนคำสั่งเป็นแบบนี้
n = int(input("Enter n : "))
fac = find_fac(n) # เรียกใช้ function
print(fac)
```

ลองรันโปรแกรม สังเกต ตัวแปร n, fac

ตัวอย่าง : เขียน function เพื่อหาระยะทางของจุด 2 จุด

```
def dist(x1, y1, x2, y2):
  return ((float(x1) - float(x2))**2 + (float(y1) - float(y2))**2)**0.5
#in main program
point1 = input('Enter first point (x y): ')
point2 = input('Enter second point (x y): ')
x1, y1 = point1.split()
x2, y2 = point2.split()
distance = dist(x1, y1, x2, y2)
print('Distance of', point1, 'and', point2, 'is', distance)
```

การรันโปรแกรม

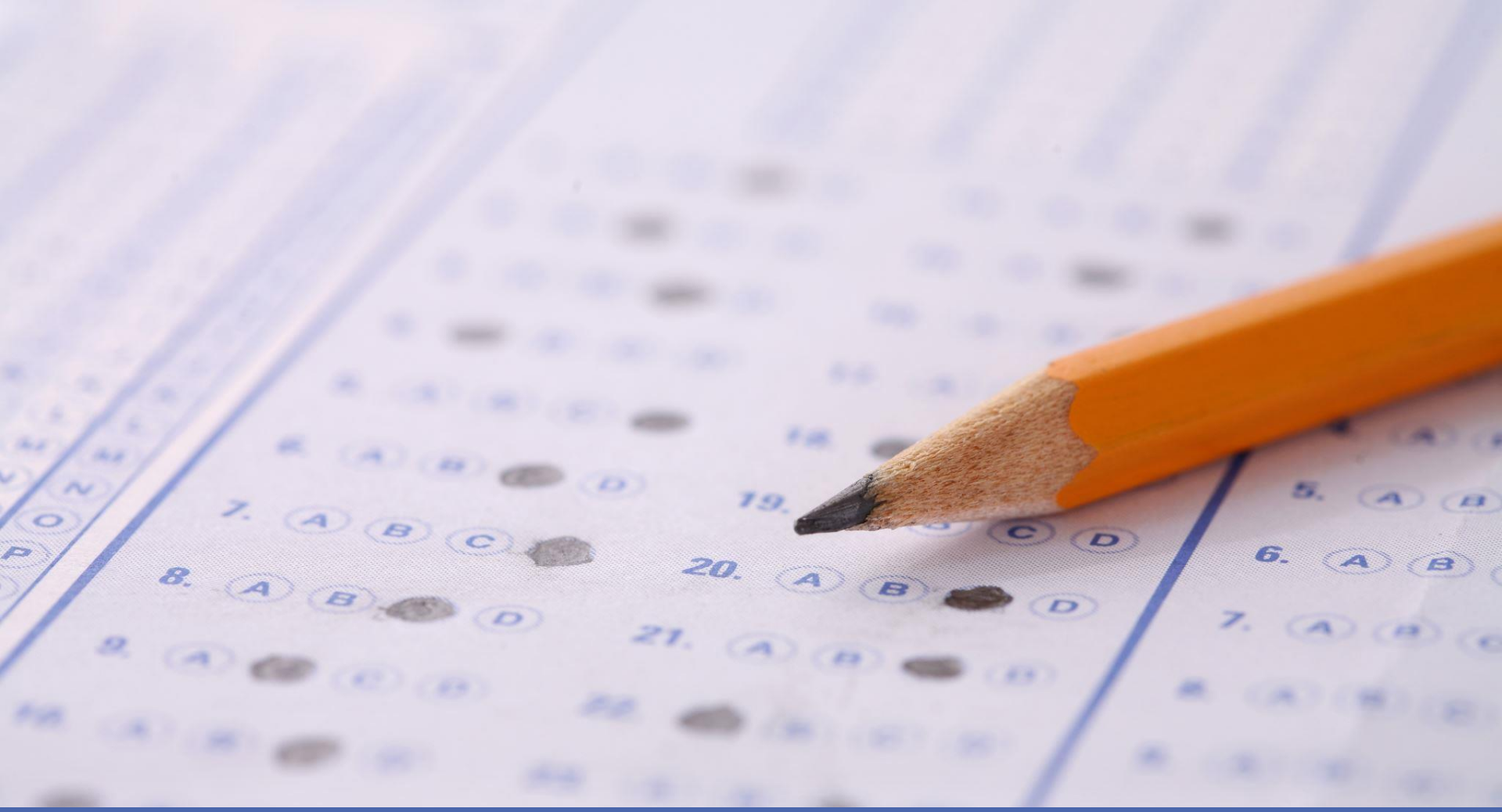
```
Enter first point (x y): 1 3<enter>
Enter second point (x y): 2 5<enter>
Distance of 1 3 and 2 5 is 2.23606797
```

ตัวอย่าง :เขียน function เพื่อคืนค่าสตริงของชื่อและนามสกุล ซึ่งขึ้นอยู่กับค่าตัวแปร reverse ว่าสั่งให้สลับที่หรือไม่ ถ้าไม่ใส่ชื่อหรือนามสกุลจะฟ้อง Invalid

```
def writeName(fn, ln, reverse):
```

```
การรันโปรแกรม
Enter your firstname : <enter>
Enter your lastname : Sawasdee<enter>
The name is Invalid
การรันโปรแกรม
Enter your firstname : <enter>
Enter your lastname : <enter>
The name is Invalid
การรันโปรแกรม
Enter your firstname : Sathu<enter>
Enter your lastname : Sawasdee<enter>
The name is Sawasdee Sathu
```

```
#in main program
firstname = input("Enter your firstname : ")
lastname = input("Enter your lastname : ")
print("The name is", writeName(firstname, lastname, True))
```



Quiz

Quiz

1. จงเขียนฟังก์ชัน sum_digits ที่รับพารามิเตอร์เป็นจำนวนเต็ม 1 จำนวน แล้วคำนวณหาผลรวมของตัวเลขแต่ละหลักในเลขจำนวนนั้นส่งค่าคืนกลับออกไปยังผู้เรียกใช้

2. จงเขียนฟังก์ชัน rectangle ที่รับพารามิเตอร์เป็นจำนวนเต็ม 2 จำนวน (m, n) แล้วพิมพ์รูปกล่องขนาด m×n แสดงเป็นผลลัพธ์ ตัวอย่างเช่น

    rectangle(2,4) จะได้ผลลัพธ์ทางจอภาพเป็น

    ****

    ****

3. จงเขียนฟังก์ชัน check_date ที่รับพารามิเตอร์เป็นจำนวนเต็ม 3 จำนวน (m, d, y) โดยปีเป็นปีคริสตศักราช และตรวจสอบว่าเป็นวันที่ที่ถูกต้องหรือไม่ ส่งค่าคืนกลับเป็นชนิด boolean

วันอธิกวารจะอยู่ในปี ค.ศ. ที่หารด้วย 4 ลงตัวแต่หารด้วย 100 ไม่ลงตัว หรือปีนั้นเป็นปี ค.ศ. ที่หารด้วย 400 ลงตัว

Global variables

- คือตัวแปรที่อยู่นอกขอบเขตของฟังก์ชัน
- ฟังก์ชันสามารถใช้งานตัวแปร global ได้
- ถ้าในฟังก์ชันต้องการเอาค่าของตัวแปร global มาใช้งาน สามารถอ้างชื่อตัวแปร global ได้

```
def myfunc():
  print("x :", x)
  a = x + 7
  print("a :", a)
x = 5
myfunc()
print("x :", x)
```

ผลลัพธ์ที่ได้คือ

```
x : 5
a : 12
x : 5
```

ข้อควรระวัง แบบนี้ไม่ได้ใช้ตัวแปร global

- ถ้าในฟังก์ชันมีการสร้างตัวแปรชื่อเดียวกันกับตัวแปร global ที่อยู่นอกฟังก์ชัน จะถือเป็นการสร้างตัวแปร local ขึ้นใหม่

```
def myfunc():
  x = 10
  a = x + 7
  print(a, x)
x = 5
myfunc()
print("x :", x)
```

ผลลัพธ์ที่ได้คือ
17 10
x : 5

Global variables

- ถ้าในฟังก์ชันต้องการแก้ไขหรือกำหนดค่าของตัวแปร global คือให้มีการแก้ค่าภายในฟังก์ชันและส่งผลกับตัวแปร global ด้วย
- จะต้องประกาศให้ชัดเจน ว่าจะใช้ตัวแปร global นั้น

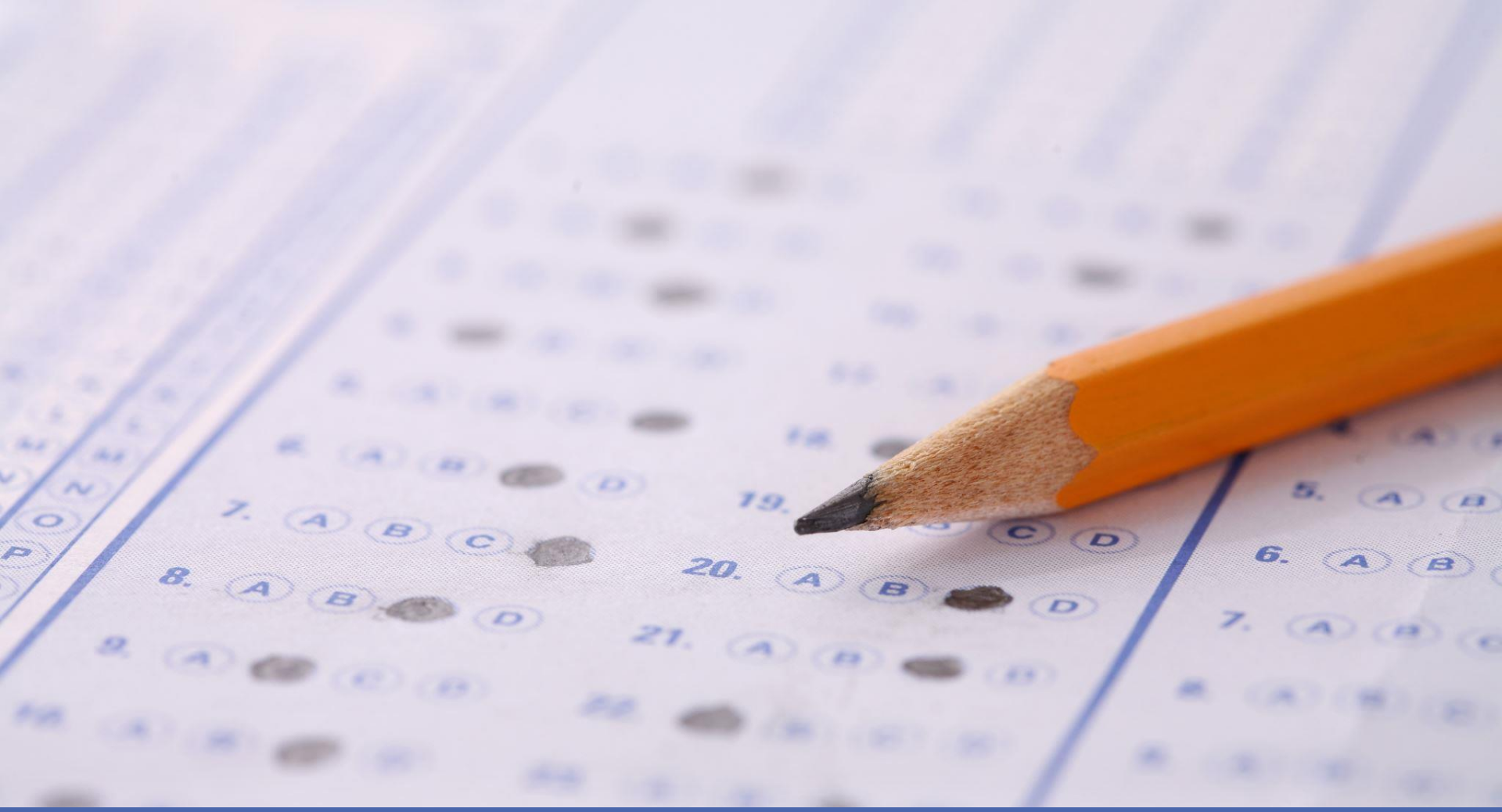
```
def myfunc():
  global x
  x = 10
  a = x + 7
  print(a, x)
x = 5
myfunc()
print("x :", x)
```

ผลลัพธ์ที่ได้คือ
17 10
x : 10

ตัวอย่าง : จงแสดงผลลัพธ์ที่ได้

def f(a, b):
  a = 10
  b = 0
  return a+b
 	
x = 5
y = 7
f(x,y)
print(x, y)
	def f(a, b):
	  a = 10
	  b = 0
	  return a+b
	
	x = 5
	y = 7
	a = f(x,y)
	print(a, x, y)
		def f(a, b):
		  a = 10
		  b = 0
		  return a+b
		
		a = 5
		b = 7
		c = f(a,b)
		print(a, b, c)

5 7	10 5 7	5 7 10



Quiz

Quiz จงแสดงผลลัพธ์ที่ได้

```
def f(a, b):
    a = 10
    b = 0
    x = 0
    y = 1
    print(x, y)
    return a-b

x = 5
y = 7
z = f(x, y)
print(x, y, z)
```

```
def f(a, b):
    global x, y
    a = 10
    b = 0
    x = 0
    y = 1
    print(x, y)
    return a-b

x = 5
y = 7
z = f(x, y)
print(x, y, z)
```

แบบฝึกหัด

1. จงเขียนฟังก์ชัน รับพารามิเตอร์เป็นจำนวนจริง 3 จำนวน แล้วส่งค่ากลับเป็นจำนวนน้อยสุด และให้เขียนโปรแกรมในส่วนของโปรแกรมหลักเพื่อรับค่าจำนวนจริง 3 จำนวนแล้วเรียกใช้ฟังก์ชันและเอาผลที่ได้จากฟังก์ชันมาแสดงเป็นผลลัพธ์

2. จงเขียนฟังก์ชัน รับพารามิเตอร์เป็นจำนวนเต็ม 2 จำนวน เพื่อคำนวณค่า $C(m,n) = \frac{m!}{n!(m-n)!}$ โดยใช้ฟังก์ชันคำนวณ factorial ที่เขียนไว้แล้ว แล้วส่งค่าที่คำนวณได้กลับออกจากฟังก์ชัน และให้เขียนโปรแกรมในส่วนของโปรแกรมหลักเพื่อรับค่าจำนวนเต็ม 2 จำนวน แล้วเรียกใช้ฟังก์ชันและเอาผลที่ได้จากฟังก์ชันมาแสดงเป็นผลลัพธ์

3. จงเขียนฟังก์ชัน รับพารามิเตอร์เป็นจำนวนเต็ม 1 จำนวน เพื่อตรวจสอบว่าเป็นจำนวนเฉพาะหรือไม่ โดยให้ส่งค่าคืนเป็นชนิด Boolean และให้เขียนโปรแกรมในส่วนของโปรแกรมหลักโดยให้มีการวนรับค่าตัวเลขที่ต้องการตรวจสอบ หยุดวนรับค่าเมื่อจำนวนที่รับเข้ามามีค่า <2