단국대학교
SW중심대학

창의적 사고와 코딩

Lecture 5-2. 함수

함수 호출의 인수가 함수의 매개변수로 전달되는 방법

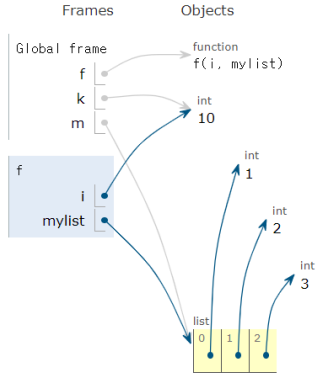
참조값에 의한 인수 전달

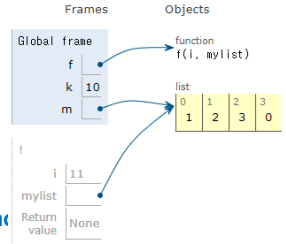
- 일반적인 프로그래밍에서 함수를 호출할 때, 변수를 전달하는 2가지 방법
  - **값에 의한 호출(call-by-value)**
    - 변수의 값만 전달
    - 인수의 값이 해당 매개변수에 복사되어 함수가 반환 될 때 호출자의 변수가 변경되지 않는다.
  - **참조에 의한 호출(Call-by-reference)**
    - 인수에 대한 참조를 매개변수에 제공되어 결과적으로 함수는 인수를 수정할 수 있다. 즉 호출자의 변수가 변경된다.
    - Call by Reference를 사용하면 인수를 복사 할 필요가 없으므로 계산 시간과 메모리 공간을 모두 절약
    - 함수 호출에서 변수가 실수로 변경 될 수 있으므로 각별한 주의 필요
- 파이썬은 어떨까?
  - 객체에 의한 호출(Call-by-Object)이라고하는 메커니즘을 사용
    - Call by Object Reference 또는 Call by Sharing 이라고도 불림
  - 정수, 문자열 또는 튜플과 같은 변경 불가능한 인수를 함수에 객체 참조를 통해 전달되고 전달은 값에 의한 호출처럼 작동
  - 변경이 가능한 객체를 인수로 함수에 전달하면 객체 참조를 통해 전달되고 함수 내부에서 변경이 가능
    - 함수 내부에서 재 할당 하는 경우 파이썬은 별도의 지역 변수를 만들어 호출자의 변수는 변경되지 않는다.

참조값에 의한 인수 전달

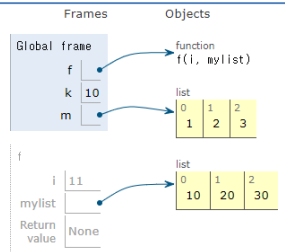
- **파이썬에서는 함수 호출시 입력된 인수(argument) 객체의 레퍼런스를 생성하여, 레퍼런스 값을 복사하여 매개변수에 전달 ➔ 전달되는 인수가 mutable, immutable에 따라 다른 결과가 일어남**
  - 인수가 mutable 객체일때, 함수 내에서 인수의 내용을 변경하면 변경사항이 호출자에게 반영되지만, 함수내에서 인수에 새로운 값을 할당하면 호출자에 아무런 변화가 없음
  - 인수가 immutable 객체일때, 입력파라미터의 값이 함수 내에서 변경될 수 없으며, 함수 내에서 새로운 객체의 레퍼런스를 입력파라미터에 할당되어도 함수 외부(Caller)의 값은 변하지 않는다.

```
def f(i, mylist):
    i = i + 1
    mylist.append(0)

k = 10        # k는 int (immutable)
m = [1,2,3]   # m은 리스트 (mutable)

f(k, m)       # 함수 호출
# 10 [1,2,3,0] 출력
print(k, m)   # 호출자 값 체크
```






```
def f(i, mylist):
    i = i + 1
    mylist = [10,20,30]

k = 10        # k는 int (immutable)
m = [1,2,3]   # m은 리스트 (mutable)

f(k, m)       # 함수 호출
# 10 [1,2,3] 출력
print(k, m)   # 호출자 값 체크
```



참조값에 의한 인수 전달

```
def f(i, mylist):
    print("Argument i=",i," id=",id(i))
    print("Argument mylist=",mylist," id=",id(mylist))

    i = i + 1
    mylist.append(0)
    #mylist = [10,20,30]

    print("i=i+1을 실행하고 난 후 i=",i," id=",id(i))
    print("리스트에 항목을 추가한 후  mylist=",mylist," id=",id(mylist))

k = 10       # k는 int (immutable)
m = [1,2,3]  # m은 리스트 (mutable)

print('호출전 k 의 id = ', id(k))
print('호출전 m의 id = ', id(m))

f(k, m)      # 함수 호출

print('함수 실행 후 k, m :', k, m) #호출자의 값체크
```

```
호출전 k 의 id =  1470036032
호출전 m의 id =  50787144
Argument i= 10  id= 1470036032
Argument mylist= [1, 2, 3]  id= 50787144
i=i+1을 실행하고 난 후 i= 11  id= 1470036048
리스트에 항목을 추가한 후  mylist= [1, 2, 3, 0]  id= 50787144
함수 실행 후 k, m : 10 [1, 2, 3, 0]
```

```
def f(i, mylist):
    print("Argument i=",i," id=",id(i))
    print("Argument mylist=",mylist," id=",id(mylist))

    i = i + 1
    mylist = [10,20,30]

    print("i=i+1을 실행하고 난 후 i=",i," id=",id(i))
    print("mylist = [10, 20, 30] 실행한 후  mylist=",mylist," id=",id(mylist))

k = 10       # k는 int (immutable)
m = [1,2,3]  # m은 리스트 (mutable)

print('호출전 k 의 id = ', id(k))
print('호출전 m의 id = ', id(m))

f(k, m)      # 함수 호출

print('함수 실행 후 k, m :', k, m) #호출자의 값체크
```

```
호출전 k 의 id =  1470036032
호출전 m의 id =  53801704
Argument i= 10  id= 1470036032
Argument mylist= [1, 2, 3]  id= 53801704
i=i+1을 실행하고 난 후 i= 11  id= 1470036048
mylist = [10, 20, 30] 실행한 후  mylist= [10, 20, 30]  id= 53838728
함수 실행 후 k, m : 10 [1, 2, 3]
```

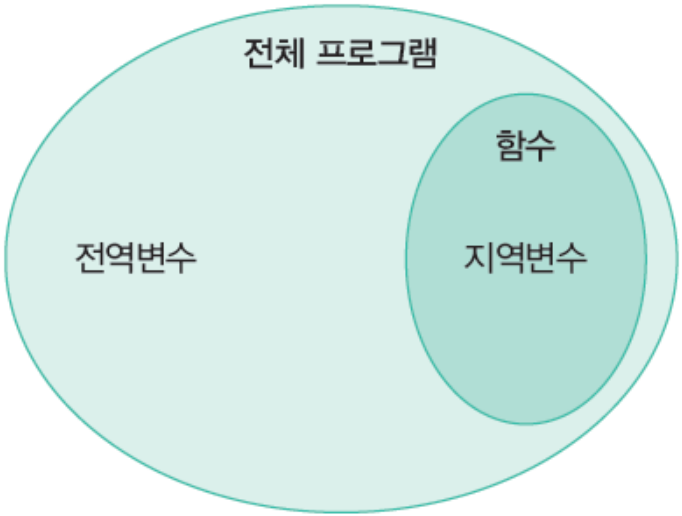
파이썬은 처음에는 call by referenc처럼 동작하지만, 변수에 새로운 객체를 할당하자마자 파이썬은 call by value 로 "전환". 즉, 지역 변수 x가 생성되고 전역 변수 x의 값이 복사된다.

yunyi96@gmail.com

지역변수와 전역변수

변수의 종류

- **지역 변수(local variable): 함수 안에서 선언되는 변수**
  - 함수 내에서만 사용 가능
- **전역 변수(global variable): 함수 외부에서 선언되는 변수**
  - 프로그램 전체에서 사용 가능




```
width = 10
height = 20

def area(w, h):
    result = w * h
    return result

print(area(width, height))
```

함수의 외부에 있기 때문에 전역 변수이다.

함수의 내부에 있기 때문에 지역 변수이다.

지역 변수

```
def sub():
        s = "바나나가 좋음!"
        print(s)

sub()
```

```
바나나가 좋음!
```

전역 변수

```
def sub():
        print(s)

s = "사과가 좋음!"
sub()
```

사과가 좋음!

지역 변수의 범위(Scope)

- 지역 변수는 함수 안에서만 사용이 가능하다.
- 아래의 코드에서 지역 변수를 찾아보자.

```
def calc_tax(price, tax_rate=0.1):
    total = price + (price * tax_rate)
    return total

my_price = float(input("금액을 입력하시요:"))
totalPrice = calc_tax(my_price)
print("일반 세율 적용시 :", total)
```

오류가 없을까?

지역 변수의 범위

- 함수 안에서 사용한 변수는 함수 안에서만 존재하고 함수가 종료되면 변수는 메모리에서 사라지게 된다.

```
def calc_tax(price, tax_rate=0.1):
    total = price + (price * tax_rate)
    return total

my_price = float(input("금액을 입력하시요:"))
totalPrice = calc_tax(my_price)
print("일반 세율 적용시 :", total)
```

```
금액을 입력하시요:1000000
Traceback (most recent call last):
  File "C:/Users/yuni/Desktop/6.py", line 7, in <module>
    print("일반 세율 적용시 :", total)
NameError: name 'total' is not defined
>>>
```

yunyi96@gmail.com

전역 변수의 범위

- 전역 변수는 어디서나 사용할 수 있다.
- 아래의 코드에서 전역 변수를 찾아보자.

```
def calculate_area ():
    result = 3.14 * r**2
    return result

r = float(input("원의 반지름: "))
area = calculate_area()
print(area)
```

함수 안에서 전역 변수 변경하기

```
def calculate_area (radius):
    area = 3.14 * radius**2      # 전역변수 area에 계산값을 저장하려 함
    return

area = 0
r = float(input("원의 반지름: "))
calculate_area(r)
print(area)
```

여기서 새로운 지역 변수 area가 생성된다.

```
원의 반지름: 10
0
>>>
```

왜 0이 나올까?

함수 안에서 변수의 값에 저장하면
새로운 지역변수가 만들어짐!!!

함수 안에서 전역 변수 변경하기

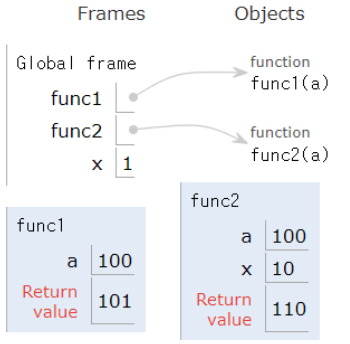
- 함수 안에서 전역변수의 값을 변경하고 싶은 경우는?
- global을 사용하여 전역 변수에 값을 저장한다고 알려야 한다.
  ➔ 가급적 전역변수와 지역변수의 이름은 다른게 사용하는 것이 바람직

```
def calculate_area (radius):
    global area
    area = 3.14 * radius**2
    return

area = 0
r = float(input("원의 반지름: "))
calculate_area(r)
print(area)
```

```
원의 반지름:10
314.0
>>>
```

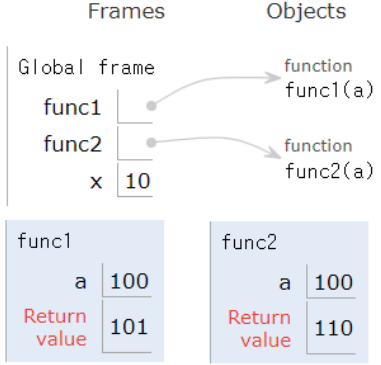
- 함수 내에서 전역 변수 x의 값을 변경하면 그 결과는 어떻게 될까? 전역변수의 값을 함수에서 변경하고자 하는 경우에는 어떻게 할까?

```
def func1(a):
    return a + x

def func2(a):
    x = 10
    return a + x

x = 1
print("func1(100)의 결과 :", func1(100))
print("func2(100)의 결과 :", func2(100))
print("함수 수행후 x의 값 :", x)
```




```
func1(100)의 결과 : 101
func2(100)의 결과 : 110
함수 수행후 x의 값 : 1
>>>
```

```
def func1(a):
    return a + x

def func2(a):
    global x
    x = 10
    return a + x

x = 1
print("func1(100)의 결과 :", func1(100))
print("func2(100)의 결과 :", func2(100))
print("함수 수행후 x의 값 :", x)
```




```
func1(100)의 결과 : 101
func2(100)의 결과 : 110
함수 수행후 x의 값 : 10
>>>
```

Lab: 매개변수 = 지역변수

- 다음 프로그램의 실행결과는 어떻게 될까?

```
# 함수가 정의된다.
def sub( mylist ):
   # 리스트가 함수로 전달된다.
   mylist = [1, 2, 3, 4]   # 새로운 리스트가 매개변수로 할당된다.
   print ("함수 내부에서의 mylist: ", mylist)
   return

# 여기서 sub() 함수를 호출한다.
mylist = [10, 20, 30, 40];
sub( mylist );
print ("함수 외부에서의 mylist: ", mylist)
```

```
함수 내부에서의 mylist:  [1, 2, 3, 4]
함수 외부에서의 mylist:  [10, 20, 30, 40]
```

yunyi96@gmail.com

Lab: 상수

- 파이를 전역 변수로 선언하고 이것을 이용하여서 원의 면적과 원의 둘레를 계산하는 함수를 작성해보자.

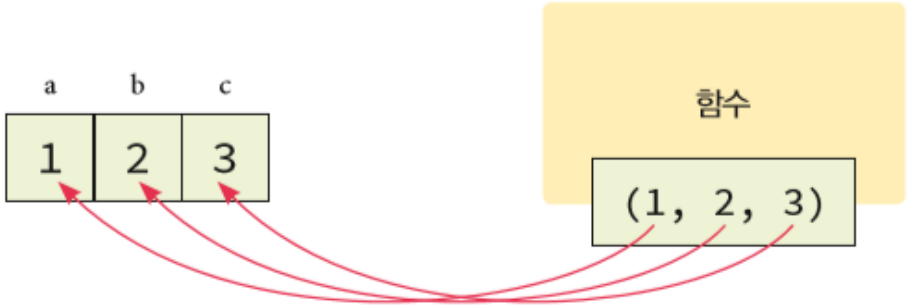
```
원의 반지름을 입력하시오: 10
원의 면적: 314.159265358979
원의 둘레: 62.8318530717958
```

Solution

여러 개의 값 반환하기

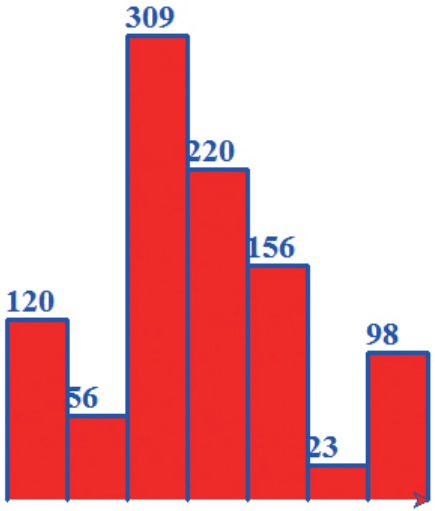
```
def sub():
    return 1, 2, 3

a, b, c = sub()
print(a, b, c)
```

```
1 2 3
```



Lab. 막대 그래프 그리기

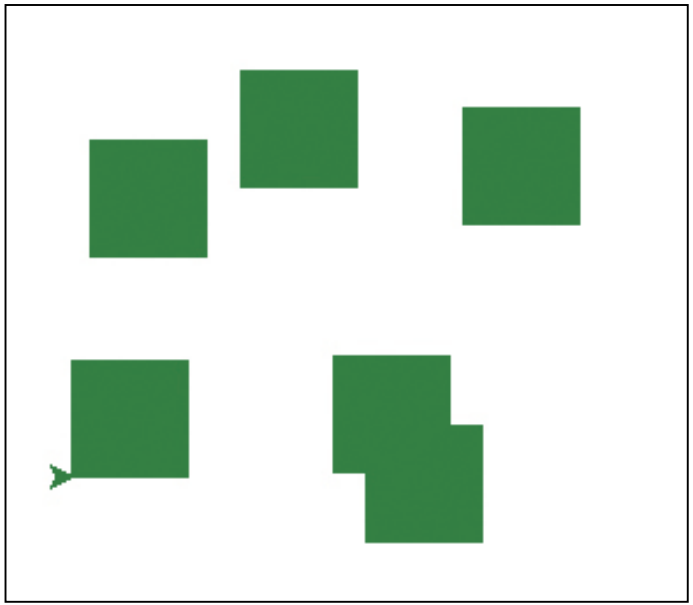
- 파이썬의 터틀 그래픽을 이용해서 막대 그래프를 그려보자.



Solution

예제: 클릭하는 곳에 사각형 그리기

- 사용자가 화면에서 마우스 버튼을 클릭한 경우, 클릭 된 위치에 사각형을 그리는 프로그램을 작성해 보자. 앞에서 작성한 square() 함수도 사용한다.



콜백 함수

- 이벤트가 발생했을 때, 이벤트를 처리하는 함수를 콜백 함수(callback function)라고 부른다.
- 터틀 그래픽에서도 마우스가 클릭 되었을 때 호출되는 콜백 함수를 등록할 수 있다.

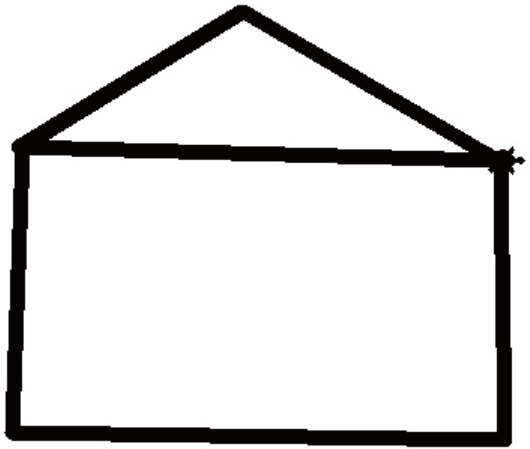
```
def drawit(x, y):    # x와 y는 마우스가 클릭된 위치
        t.penup()
...
...
s = turtle.Screen()      # 그림이 그려지는 캔버스 생성
s.onscreenclick(drawit) # 마우스 클릭 이벤트 처리 함수를 등록
```

Solution

```
import turtle
t = turtle.Turtle()

def square(length):
    for i in range(4):
        t.forward(length)
        t.left(90)

def drawit(x, y):
    t.penup()
    t.goto(x, y)
    t.pendown()
    t.begin_fill()
    t.color("green")
    square(50)
    t.end_fill()

s = turtle.Screen()      # 그림이 그려지는 캔버스 생성
s.onscreenclick(drawit) # 마우스 클릭 이벤트 처리 함수를 등록
```

예제: 마우스로 그림 그리기

- 이번 실습에서는 drawit() 안에 goto()를 넣어서 거북이를 클릭된 위치로 이동시키도록 하자.
- 현재 위치에서 클릭된 위치까지 선이 그려 진다.



Solution

```
import turtle

def draw(x, y):
    t.goto(x, y)

t = turtle.Turtle()
t.shape("turtle")
t.pensize(10)
s = turtle.Screen()       # 그림이 그려지는 캔버스 생성
s.onscreenclick(draw)     # 마우스 클릭 이벤트 처리 함수를 등록
```

Screen 이벤트 함수

turtle.onscreenclick(fun, btn=1, add=None)	화면을 클릭했을 때 명시한 함수를 실행 Fun: 현재 거북이의 좌표인 (x, y)를 인수로 하는 실행되는 함수 btn: 마우스 버튼 1: 왼쪽 마우스 버튼 2: 중간 마우스 버튼 3: 오른쪽 마우스 버튼 Add: True 또는 False True: 새로운 바인딩 추가 False: 전 바인딩 대체 import turtle t = turtle.Turtle() s = t.Screen() s.onscreenclick(t.goto)
turtle.onkey(fun, key)	fun: 인수가 없는 함수 key: 문자열 : 키 (예 : "a") 또는 키 기호 (예 : "space“, “Up”, “Down”, “Right”, “Left”) s = t.Screen() s.onkey(t.penup, "Up")

Screen 이벤트 함수

turtle.listen(xdummy=None, ydummy=None)	• 키 이벤트를 수집하기 위해 TurtleScreen에 포커스를 설정
	- 이 명령어를 실행시켜야 키 입력모드가 실행되어 입력된 키에 반응함
	- 보통 코드의 끝 부분에 위치함.
	def right():
	t.fd(10)
	
	def left():
	t.ld(10)
	
	s = t.Screen()
	s.onkey(right, "Right") # 키보드 이벤트 처리 함수를 등록
	s.onkey(left, "Left")
	s.listen() # 키보드 이벤트를 기다린다.

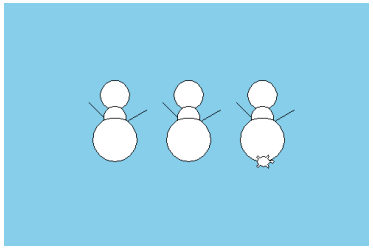
turtle.mainloop( ) turtle.done( )	- 코드가 끝이 나도 turtle 그래픽 창을 계속 열어두라는 명령어 - 윈도우 창이 종료되지 않고 event loop를 통해 계속 창을 유지시켜줌 - 반드시코드 가장 마지막에 와야함.

터틀 그래픽 함수

<table>
<tr>
<td>turtle.setheading(to_angle)
turtle.seth(to_angle)</td>
<td>입력한 각도로 거북이의 방향을 설정함.
to_angle : 거북이의 방향 각도
<table><tr><th>표준 모드</th><th>로고 모드</th></tr><tr><td>0 - 동쪽</td><td>0 - 북쪽</td></tr><tr><td>90 - 북쪽</td><td>90 - 동쪽</td></tr><tr><td>180 - 서쪽</td><td>180 - 남쪽</td></tr><tr><td>270 - 남쪽</td><td>270 - 서쪽</td></tr></table>
<pre>>>> turtle.setheading(90)
>>> turtle.heading()
90.0</pre></td>
</tr>
</table>

눈사람을 그리는 프로그램 작성
(눈사람 그리는 함수 작성)

- 눈사람 그리는 함수: draw_snowman(x,y)
- 터틀 그래픽에서 배경색을 하늘색(skyblue)로 설정

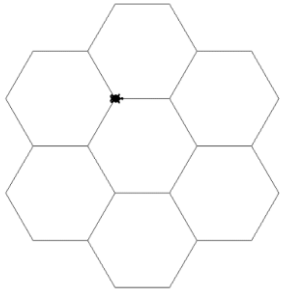


draw_snowman(x,y)

1. 좌표 (x,y) 로 이동
2. 반지름 20인 원을 그림
3. (x, y-25) 이동
4. 거북이의 머리 방향을 135°로 설정
5. 전진 50, 후진 50
6. 거북이의 머리 방향을 30°로 설정
7. 전진 50, 후진 50
8. 거북이의 머리 방향을 0°로 설정
9. 반지름 15인 원 그림
10. (x, y-70) 이동
11. 반지름이 30인 원 그림

벌집 모양을 그려보자

- 6각형을 그리는 함수: draw_hexa()



draw_hexa()

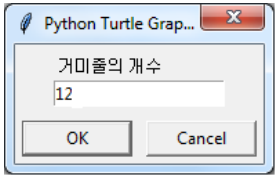
1. 6번 반복
   ① 전진 100
   ② 왼쪽으로 60°만큼 회전

벌집 모양 그리기

1. 6번 반복
   ① 6각형 그림 draw_hexa()
   ② 전진 100
   ③ 오른쪽으로 60°만큼 회전

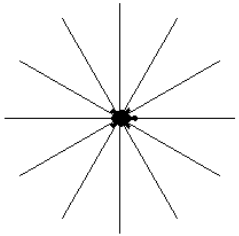
거미줄과 같은 모양을 그려보자

- draw_line(length): 선을 그리고 거북이가 처음 위치로 돌아오도록 하는 함수



draw_line(length)
① 전진 length
② 전진 -length

거미줄 그리기
① 거미줄의 개수를 textinput()을 이용하여 입력 받음
② 거미줄 개수 만큼 반복
  ① 선을 그림
  ② 오른쪽으로 (360/거미줄 개수) ° 회전



마우스를 클릭하면 다각형을 그리는 프로그램을 작성

- 다각형을 그릴 때마다 다각형 종류, 변의 길이, 색이 변경되도록 작성