스마트 모빌리티 프로그래밍

Ch 9. 파이썬 그래픽 프로그래밍
– 터틀 그래픽, tkinter GUI

영남대학교 정보통신공학과

교수 김 영 탁

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- ◆ 파이썬 그래픽 모듈/패키지
- ◆ 터틀 (turtle) 그래픽
  - 터틀 그래픽 개요
  - 터틀 그래픽 기본
  - 터틀 그래픽 이벤트 처리
  - 터틀 그래픽 기반 애니메이션
- ◆ tkinter GUI 프로그래밍
  - tkinter 개요
  - tkinter GUI 프로그래밍
  - tkinter 기반 그래프 그리기
  - tkinter 키보드 및 마우스 이벤트 처리
  - tkinter 기반 애니메이션

파이썬 그래픽 모듈/패키지

파이썬 그래픽 모듈/패키지

◆ 그래픽 사용자 접속 (GUI : Graphic User Interface)

- 컴퓨터 장치 사용에서 다양한 버튼, 텍스트 입력창, 스크롤바, 아이콘 등의 그래픽 입력 기능을 사용하여 프로그램 실행 제어, 실행 결과를 그래프나 이미지로 표현
- 편리하며 간편한 프로그램 사용 및 제어

◆ 파이썬 그래픽 모듈/패키지

- 그래픽 사용자 접속의 기본 기능들을 포함하는 모듈과 패키지 사용
- 파이썬 터틀 그래픽
- tkinter GUI 패키지
- Matplotlib 패키지
- Kivy GUI 패키지
- WxPython 패키지

파이썬 터틀 그래픽 모듈

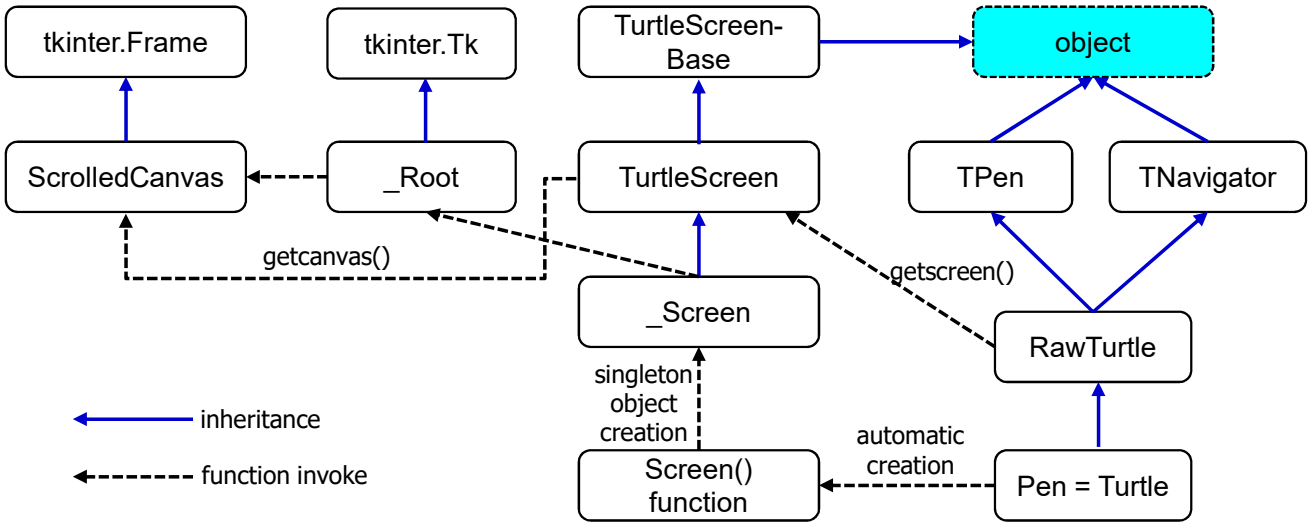
◆ 파이썬 터틀 그래픽

- 컴퓨터 프로그램의 기본 개념을 그래픽 기반으로 쉽게 배울 수 있도록 1966년 Wally Feurzig와 Seymour Papert가 개발하였던 Logo 프로그래밍 언어 기반 터틀 그래픽을 파이썬에서 제공하는 모듈
- 2차원 평면 상에 다양한 도형을 쉽게 그릴 수 있으며, 지정된 선 색상과 채우기 색상 지정 가능
- 지정된 폰트, 크기, 색상의 텍스트를 지정된 위치에 출력 가능
- tkinter 패키지를 사용하여 구현되어 있음
- 터틀 그래픽 설명자료: https://docs.python.org/3.3/library/turtle.html?highlight=turtle

파이썬 터틀 그래픽에서 사용되는 클래스 체계

◆ 터틀 그래픽 모듈에서 사용되는 클래스들의 체계도

- 파이썬 기본 클래스인 object로 부터 상속받은 그래픽 클래스
- Turtle() 클래스는 TPen()과 Tnavigator()를 상속
- TurtleScreen()과 SctrolledCanvas() 클래스를 사용하여 도형을 출력하는 스크린과 캔버스 생성 및 관리



터틀 그래픽 클래스

터틀그래픽 클래스	설명
object	파이썬 프로그램의 기본 객체
TPen	파이썬 프로그램 환경에서 그리기의 속성 (attribute) 들을 설정
TNavigator	파이썬 프로그램 환경에서 그리기 객체의 움직임에 관련된 메소드 제공
RawTurtle	클래스 TPen과 Tnavigator로부터 상속받으며, 터틀 (turtle) 객체를 생성하는 부모 클래스. RawTurtle.getscreen() 함수를 사용하여 파이썬 프로그램에서 class TurtleScreen 객체를 받아올 수 있음
Turtle	파이썬 프로그램 환경에서 터틀 그리기 객체를 생성하기 위한 클래스
TurtleScreenBase	파이썬 기본 객체 (object)를 상속 받으며, 기본적인 그래픽 기능을 제공
TurtleScreen	파이썬 프로그램의 그래픽 기능을 위한 윈도우 스크린의 메소드 들을 제공 (배경 색상 (background color), 배경 이미지 (background image), 윈도우 및 캔버스 크기) 파이썬 프로그램은 TurtleScreen.getcanvas() 함수를 사용하여 class ScrolledCanvas 객체 (default canvas)를 받아올 수 있음
tkinter.Tk	tkinter을 통하여 연결되는 Tk (tool kit)
_Root	Tk 클래스 기본 객체이며, 터틀 스크린에서 그래픽 객체들이 포함될 수 있는 환경 제공
tkinter.Frame	tkinter을 통하여 연결되는 그래픽 프레임
_Screen	Screen() 함수를 사용하여 생성되는 싱글톤 객체
ScrolledCanvas	스크롤 바를 가지는 캔버스

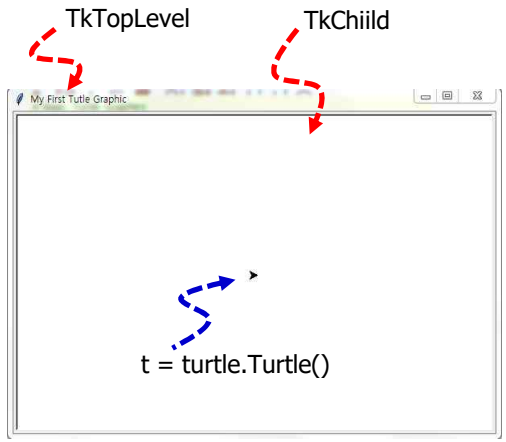
터틀 스크린 관련 함수

구분	터틀 스크린 함수/메소드	설명
윈도우 설정 및 초기화	setup(width, height, startx, starty)	(width, height) 는 TkChild 윈도우의 크기를 설정 (startx, starty)는 모니터 상에서의 메인 윈도우 (TkTopLevel) 시작 위치 좌표를 설정
	title(titleString)	메인 윈도우(TkTopLevel)의 제목 바 (title bar) 설정
	window_width() window_height()	TkChild 윈도우의 가로와 세로 설정
스크린 설정 및 초기화	bgcolor(*args)	TurtleScreen의 배경색 설정
	bgpic(picname = None)	TurtleScreen의 배경 그림 설정
	clearscreen(), clear()	TurtleScreen에 그려진 도형과 터틀 객체들을 삭제하고 초기화
	resetscreen(), reset()	스크린상의 모든 터틀들을 리셋
	screensize(canvwidth = None, canvheight =None, bg=None)	스크롤바가 생성되는 캔버스의 크기를 설정 윈도우의 크기를 변경하지 않음
캔버스	getcanvas()	tkinter를 사용하여 도형을 그릴 TurtleScreen의 캔버스 객체를 반환
좌표계 설정	setworldcoordinates (llx, lly, urx, ury)	사용자 좌표계 설정
모드 설정	mode(mode = None)	모드 설정 (“standard”, “logo”, “world”) standard: 초기 진행 방향이 오른쪽이며, 양수 (θ > 0)의 각도는 반시계 방향을 의미 logo: 초기 진행방향이 윗쪽이며, 양수 (θ > 0)의 각도는 시계 방향을 의미
	colormode(cmode = None)	cmode = 1.0 또는 255 RGB 컬러코드 값을 0 ~ cmode범위 값으로 지정

터틀 그래픽 기본 예제 – 윈도우, 스크린, 캔버스

```
# Basic Turtle Graphics

import turtle
t = turtle.Turtle()  # create turtle.Pen()
print("turtle.mode() : ", turtle.mode())
screen = turtle.getscreen()
width = turtle.window_width()
height = turtle.window_height()
scrn_size = turtle.screensize()
canv_width = screen.canvwidth
canv_height = screen.canvheight

print("default: window width ({}), height ({})".format(width, height))
print("default: screen size : ", scrn_size)
print("default: canvas width ({}), height({})".format(canv_width, canv_height))

turtle.setup(600, 400)
turtle.title("My First Tutle Graphic")
width = turtle.window_width()
height = turtle.window_height()
print("after setup(600, 400), width ({}), height ({})".format(width, height))
```

```
>>>
======= RESTART: C:/YTK-Progs/2018 Book (Python)/ch 12 GUI
turtle.mode() :  standard
default: window width (960), height (810)
default: screen size :  (400, 300)
default: canvas width (400), height(300)
after setup(600, 400), width (600), height (400)
>>>
```



터틀 모양과 크기 설정 (1)

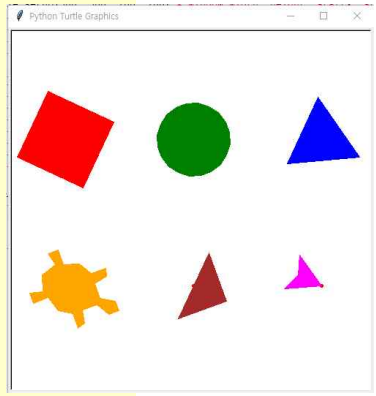
구분	터틀 모양, 크기에 관련된 함수/메소드	설 명
터틀의 모양	shape(name=None)	터틀의 모양을 이름으로 지정된 것으로 변경 (name의 예: 'arrow', 'turtle', 'circle', 'square', 'triangle', 'classic')
	shapesize(stretch_wid=None, stretch_len=None, outline=None)	터틀 크기를 설정. stretch_wid는 터틀의 이동 방향에 수직 방향의 크기를 설정하며, stretch_len은 터틀 이동방향의 크기를 설정. 터틀이 표준모드에서 위쪽 (90°)으로 이동할 때 stretch_wid는 터틀의 가로크기가 되며, stretch_len은 터틀의 세로 크기를 설정. outline은 선의 굵기를 설정.
터틀의 화면표시	hideturtle(), ht()	터틀 숨기기
	showturtle(), st()	터틀 보이기
	isvisible()	터틀이 보여지는 상태이면 True를 반환

터틀 모양과 크기 설정 (2)

구분	터틀 모양, 크기에 관련된 함수/메소드	설 명
터틀 방향 및 모양 변경	resizemode (rmode=None)	터틀 크기 재조정 모드를 설정: 'auto', 'user', 'noresize' auto: pen 굵기에 따라 크기 조절 user: shapesize()에 설정된 확장 비율에 따라 크기 조절 noresize: 터틀의 크기 변경 없음
	shearfactor (shear=None)	shear factor를 설정
	tilt(angle)	터틀을 "angle" 각도만큼 기울임
	tiltangle(angle=None)	지정된 각도로 터틀을 회전
	shapetransform (t11=None, t12=None, t21=None, t22=None)	현재 터틀 모양의 지정된 값으로 변경하여 반환
터틀 객체 정보	get_shapepoly()	현재 터틀의 모양을 나타내는 다각형 좌표를 반환
	clone()	현재 위치의 터틀을 복사
	getturtle(), getpen()	터틀 객체를 반환
	getscreen()	현재 터틀이 그려진 터틀 스크린 객체를 반환
	setundobuffer(size)	undo() 메소드에서 사용되는 최대 undo 회수를 설정

터틀의 기본 설정 모양 및 위치 설정

```
# Turtle Graphics - window, screen

import turtle
import time
turtle.setup(500, 500, 200, 200) # window width, height, startx, starty
turtles = []
turtle_names = ['square', 'circle', 'triangle', 'turtle', 'arrow', 'classic']
turtle_colors = ['red', 'green', 'blue', 'orange', 'brown', 'magenta']
turtle_positions = [(-175, 100), (0, 100), (175, 100),\
     (-175, -100), (0, -100), (175, -100)]

for i in range(len(turtle_names)):
    t = turtle.Turtle()  # create turtle.Pen()
    t.shape(name = turtle_names[i])
    t.color(turtle_colors[i])
    t.shapesize(5)
    t.up(); t.setpos(turtle_positions[i]);  t.down()
    turtles.append(t)

turn_angle = 5 # 5 degree for each rotation
for i in range(360//turn_angle): # repeat the rotation with given angle
    for t in turtles:
        t.up();  t.right(turn_angle); t.down() # turn right by the given angle
        t.dot(5, 'red')
    time.sleep(1)
```



터틀 그리기 및 이동 (1)

구분	터틀 이동 및 그리기 관련 함수/메소드	설 명
위치 설정	setposition(x, y = None)	터틀을 지정된 좌표 위치 (x, y)로 이동 setpos() goto()와 동일함
	setx(x)	터틀의 X축 좌표를 x로 설정
	sety(y)	터틀의 Y축 좌표를 y로 설정
방향 전환	setheading(to_angle)	터틀의 진행 방향을 to_angle로 변경, seth()로도 가능 mode()로 터틀의 모드를 설정함에 따라 방향 기준이 달라짐: 표준모드 (standard mode) 0: 동쪽, 90: 북쪽, 180: 서쪽, 270: 남쪽
	right(angle)	터틀의 진행방향을 현재 방향으로부터 오른쪽 (시계 방향)으로 'angle 만큼 회전. 회전각도 설정은 rt(), degree(), radians(), and mode()에 따라 설정됨
	left(angle)	터틀의 진행방향을 현재 방향으로부터 왼쪽 (반시계 방향)으로 'angle 만큼 회전. 회전각도 설정은 rt(), degree(), radians(), and mode()에 따라 설정됨

터틀 그리기 및 이동 (2)

구분	터틀 이동 및 그리기 관련 함수/메소드	설 명
이동	home()	터틀을 좌표 (0, 0)로 이동, 방향은 초기값으로 설정
	forward(dist),	터틀을 현재 진행방향으로 'dist'만큼 이동시킴. fd()로도 가능
	backward(dist)	터틀을 현재 진행방향으로 'dist' 만큼 후진 시킴, back()과 bk()로도 가능
이동 속도	speed(speed=None)	터틀의 이동속도를 설정 (설정값: 0 ~ 10) "fastest": 0; "fast": 10, "normal" : 6, "slow" : 3, "slowest" : 1
도형 그리기	circle(radius, steps=None)	주어진 반경 (radius) 크기의 원을 그림. 만약 radius >0 이면 반시계 방향으로 그리며, 만약 radius <0이면 시계방향으로 원을 그림. steps이 지정되지 않으면 원을 그리며, steps값은 반지름 radius의 원주 위에 설정되는 꼭지점의 개수를 설정하며, 다각형을 그릴 수 있도록 함.
	dot(size=None, *color)	크기 (size)와 색상(color)의 점을 그림
스탬프	stamp()	터틀의 모양을 복사하고, 표시하며, stamp_id를 반환 clearstamp(stamp_id)로 삭제할 수 있음
	clearstamp(stampid)	stampid의 스탬프 (터틀 객체의 복사)를 삭제
	clearstamp(n=None)	만약 n = None이면 모든 스탬프들을 삭제. 만약 n> 0이면 n개의 스탬프를 삭제. 만약 n <0이면 마지막 n개의 스탬프를 삭제
실행 제어	undo()	마지막으로 수행한 터틀 움직임을 취소

터틀 객체 상태, 펜 생성 및 조절 (1)

구분	터틀 상태, 각도에 관련된 함수/메소드	설 명
펜 생성, 설정	pen(pen=None, **pendict)	딕셔너리 pen으로 설정된 펜을 생성함. pendict는 키워드 인수임
	pensize(width=None)	펜의 굵기를 thickness로 설정. width()로도 가능.
	pencolor(*args)	펜 색상을 설정. 색상 이름의 예: "red", "ff0000" (R, G, B value of 0~1 or 0~255)
펜 표시 조절	pendown()	펜을 내려 쓰기가 가능하게 함. 기본 설정으로 down() 상태임
	penup()	펜을 올려 도형 그리기가 나타나지 않게 함. up()으로도 가능.
	isdown()	만약 펜이 내려져 (down) 있으면 True를 반환
현재 위치 좌표	position()	터틀의 현재 위치, pos()로도 가능
	xcor()	터틀의 현재 위치 X좌표값
	ycor()	터틀의 현재 위치 Y좌표값
방향 및 거리	towards(x, y=None)	터틀의 현재 위치와 좌표 (x, y)간의 각도를 반환
	distance(x, y=None)	터틀의 현재 위치와 좌표 (x, y)간의 거리를 반환 x는 다른 터틀 객체가 지정될 수 있음
	heading()	터틀의 방향을 절대 각도로 반환

터틀 객체 상태, 펜 생성 및 조절 (2)

구분	터틀 상태, 각도에 관련된 함수/메소드	설 명
각도 계산	degrees(fullcircle=360.0)	터틀의 각도 단위를 설정. 기본값은 360임
	radians()	각도의 단위를 라디안으로 설정. degrees(2*math.pi)와 동일
도형 출력 관리	clear()	터틀 그리기를 삭제함. 단 터틀의 위치와 방향은 유지함
	reset()	터틀 그리기를 삭제하며, 터틀을 스크린의 중앙 (초기 위치)로 이동시킴
문자열 출력	write(arg, move=False, align="left", font=("Arial", 8, "normal"))	인수 (arg)로 주어진 문자열을 지정된 맞춤 (align)과 지정된 폰트 (font)로 설정하여 출력

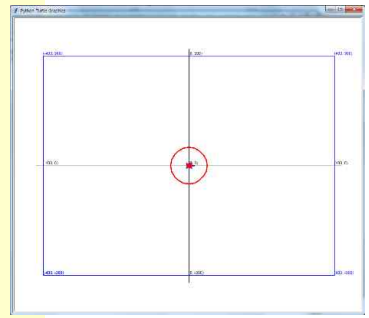
터틀 색상, 채우기, 다각형 및 복합 도형 생성

구분	터틀의 색생과 채우기 관련 함수/메소드	설 명
도형 색상 채우기	color(*args)	터틀 펜의 색상을 설정
	fillcolor(*args)	도형의 채우기 색상을 설정
	filling()	만약 채우기 모드이면 True를 반환, 채우기 모드가 아니면 False를 반환
	begin_fill()	도형의 색상 채우기를 시작
	end_fill()	도형의 색상 채우기를 종료
다각형 생성	begin_poly()	다각형 꼭지점의 기록을 시작. 터틀의 현재위치가 시작 위치임.
	end_poly()	다각형 꼭지점의 기록을 종료. 터틀의 현재위치가 마지막 위치임.
	get_poly()	마지막으로 기록된 다각형을 반환하며, register_ shape(), shape() 및 class Shape()에서 사용됨
복합 도형 생성	Shape. addcomponent()	다른 색상을 가진 다수의 다각형으로 구성된 복합 도형을 위한 class Shape 객체에 도형 객체를 추가
	register_shape()	복합도형 (compound shape)의 등록

터틀 그래픽 기본

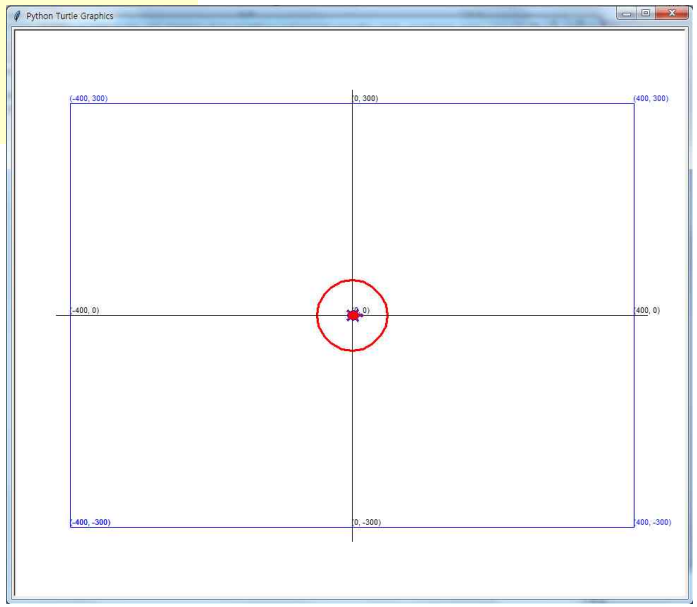
터틀 그래픽에서 좌표 지정 도형 그리기 (1)

```
# Turtle - coordinates, grid (1)
import turtle
pen = turtle.Turtle()
pen.shape('turtle')
pen.color("black", "black") # black line, black fill

c = turtle.Turtle()
c.shape('circle')
c.shapesize(5,5,3) # strech_width, strech_height, outline (thickness)
c.color("red", "white") # red line, white filling
c.setpos(0, 0)

width, height, margin = 800, 600, 20
pen.write((0, 0))
pen.up(); pen.setpos(-(width/2 + margin), 0); pen.down()
pen.setpos((width/2 + margin), 0)
pen.up(); pen.setpos(0, -(height/2 + margin)); pen.down()
pen.setpos(0, (height/2 + margin))

axis_ends = [(-width//2, 0), (width//2, 0), (0, height//2), (0, -height//2)]
for p in range(len(axis_ends)):
    pen.up(); pen.goto(axis_ends[p]); pen.down()
    pen.write(axis_ends[p])

pen.up();  pen.setpos(-width//2, -height//2); pen.down()
pen.color("blue", "red")
```



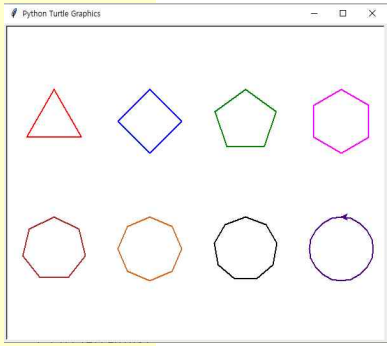
터틀 그래픽에서 좌표 지정 도형 그리기 (2)

```
# Turtle - coordinates, grid (2)

rectangle_ends = [(-width//2,-height//2), (width//2, -height//2),\
             (width//2, height//2), (-width//2, height//2),\
             (-width//2,-height//2)]
for p in range(len(rectangle_ends)):
    pen.goto(rectangle_ends[p])
    pen.write(rectangle_ends[p])

pen.up(); pen.home(); pen.down()
```



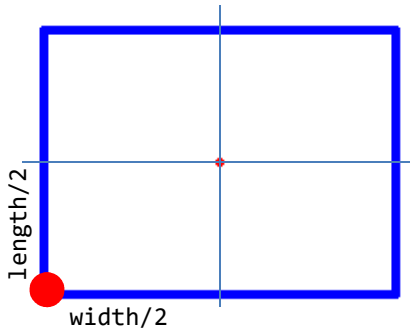
circle()을 사용한 원 및 다각형 그리기

```
# Turtle - Drawing polygons with circle
import turtle
turtle.setup(600, 500)
turtle.mode("standard") # mode standard sets initial heading right
t = turtle.Turtle()
t.pensize(2)
t.up(); t.shape(name='classic'); t.down()
radius = 50

polygons = [(-225, 150, "red", 3), (-75, 150, "blue", 4), (75, 150, "green", 5),\
            (225, 150, "magenta", 6), (-225, -50, "brown", 7),\
            (-75, -50, "chocolate", 8),(75, -50, "black", 9),\
            (225, -50, "indigo", 0)]

for i in range(len(polygons)):
    (pos_x, pos_y, shape_color, num_vertices) = polygons[i]
    t.up(); t.goto(pos_x, pos_y); t.setheading(180); t.down()
    t.color(shape_color)
    if num_vertices > 2:
        t.circle(radius, steps = num_vertices)
    else:
        t.circle(radius)
```



forward(), right(), left()를 사용한 도형 그리기

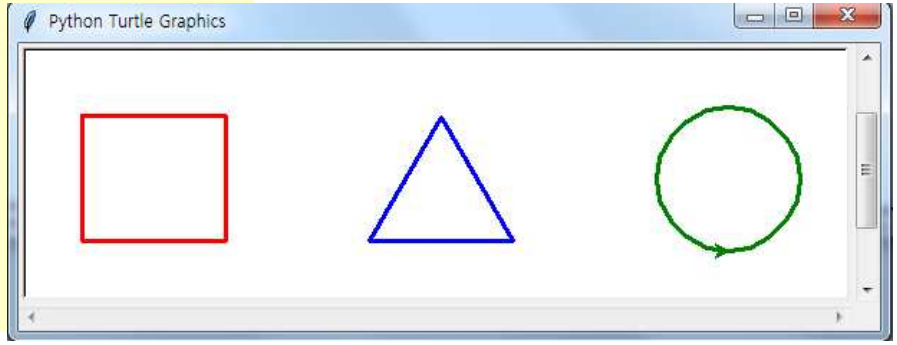
```
#Simple Python Program for Shape Drawing with Text Inputs (1)

import turtle
import math

def draw_rectangle(t, cl="black", width=10, height=10):
    t.up();
    t.goto(t.xcor()-width//2,\
       t.ycor()-height//2); t.down()
    t.color(cl)
    t.forward(width);  t.left(90)
    t.forward(height); t.left(90)
    t.forward(width);  t.left(90)
    t.forward(height); t.left(90)

def draw_triangle(t, cl="black", side=10):
    t.up();
    t.goto(t.xcor()-side//2,\
       t.ycor()-side*math.sin(math.pi/3)/2); t.down()
    t.color(cl)
    t.forward(side); t.left(120)
    t.forward(side); t.left(120)
    t.forward(side); t.left(120)

def draw_circle(t, cl="black", radius=10):
    t.up(); t.goto(t.xcor(), t.ycor()-radius); t.down()
    t.color(cl)
    t.circle(radius)
```



tan(θ) = 2.0 * h / side
h = side * tan(θ) / 2.0

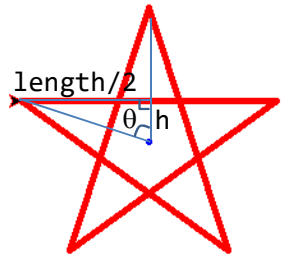
forward(), right(), left()를 사용한 도형 그리기

```
#Simple Python Program for Shape Drawing with Text Inputs (2)

# ----------------------
turtle.setup(600, 200)
tt = turtle.Turtle()
tt.shape('classic')
tt.pensize(3)

tt.up(); tt.goto(-200, 0); tt.down()
draw_rectangle(tt, "red", 100, 87)

tt.up(); tt.goto(0, 0); tt.down()
draw_triangle(tt, "blue", 100)

tt.up(); tt.goto(200, 0); tt.down()
draw_circle(tt, "green", 50)
```



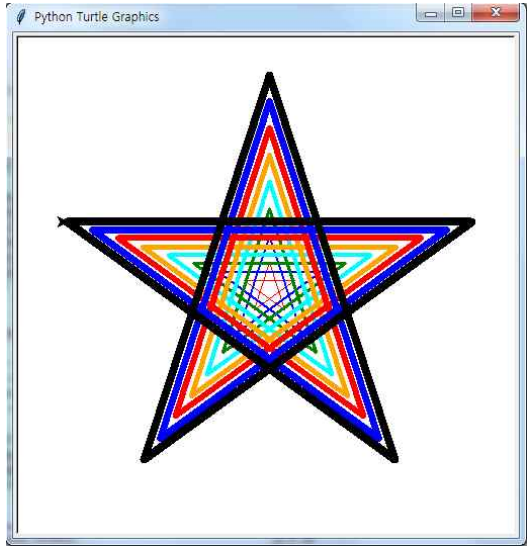
터틀 그래픽을 사용한 별 그리기

```
# Simple Python Turtle Graphic Program to Draw a Star

import turtle
import math
turtle.setup(500, 500) #set width and height of canvas
t = turtle.Turtle()
t.shape('classic')

def draw_star(length = 100, thickness = 3, cl = 'red'):
    t.pencolor(cl)
    t.width(thickness)
    start_x = -length/2
    theta = math.radians(360 / 5)
    start_y = length / (2.0 * math.tan(theta))
    t.penup() #pen up to prohibit drawing
    t.goto(start_x, start_y) # moving to starting position
    t.pendown() #pen down to draw
    for i in range(5):
        t.forward(length)
        t.right((360 * 2)/5)
    t.penup()

# -----------------
length = 50
thickness = 1
colors = ['red', 'blue', 'green', 'cyan',\
          'orange', 'red', 'blue', 'black']
for i in range(len(colors)):
    draw_star(length, thickness, colors[i])
    length += 50
    thickness += 1
```



$\tan(\theta) = length / (2.0 * h)$

$h = length / (2.0 * \tan(\theta))$



지정된 위치, 색상, 폰트의 문자열 출력

◆ write() 함수

write() 함수의 파라메터	속성	설명
font	family	텍스트의 글꼴 (times, arial, tahoma)
	size	텍스트 글꼴의 크기
	weight	텍스트를 진하게 또는 보통 (bold, normal)
	slant	텍스트를 기울게 (roman, italic)
	underline	텍스트의 밑줄 (False, True)
	overstrike	텍스트의 취소선 (False, True)
align	정렬	텍스트의 정렬 (left, center, right)

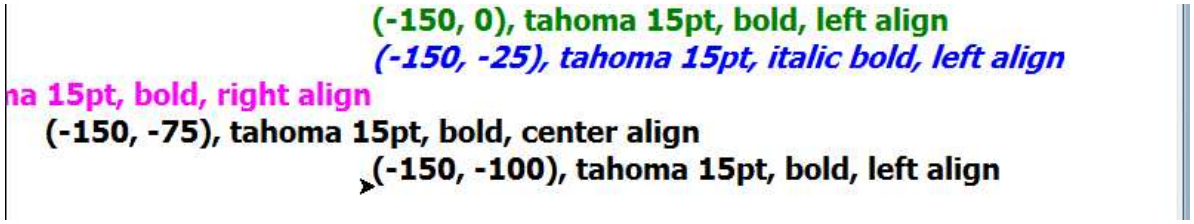
터틀 객체 write() 함수를 사용한 텍스트 출력

◆ write(), font

```
# turtle - write()  (1)

import turtle
turtle.setup(800, 400)

def jumpTo(t, x, y):
    t.up(); t.setpos(x, y); t.down()

t = turtle.Turtle()
jumpTo(t, -150, 150); t.color("red")
t.write("(-150, 150), times 10pt", font=("times", 10))
jumpTo(t, -150, 125); t.color("red")
t.write("(-150, 125), times 10pt", font=("times", 10, "bold"))
jumpTo(t, -150, 100); t.color("blue")
t.write("(-150, 100), arial 12pt", font=("arial", 12, "normal"))
jumpTo(t, -150, 75); t.color("blue")
t.write("(-150, 75), arial 12pt",  font=("arial", 12, "bold"))
jumpTo(t, -150, 50); t.color("green")
t.write("(-150, 50), tahoma 15pt", font=("tahoma", 15, "italic"))
```

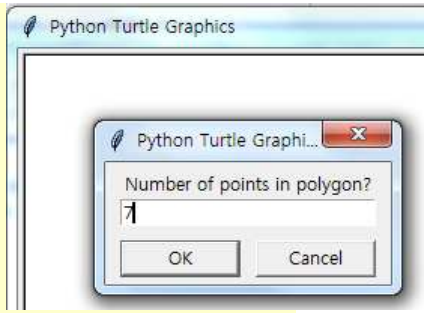
터틀 객체 write() 함수를 사용한 텍스트 출력

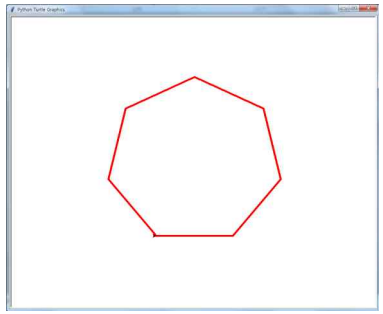
```
# turtle – write()  (2)

jumpTo(t, -150, 25); t.color("red")
t.write("(-150, 25), tahoma 15pt, bold", font=("tahoma", 15, "bold"))
jumpTo(t, -150, 0); t.color("green")
t.write("(-150, 0), tahoma 15pt, bold, left align", align="left", font=("tahoma", 15, "bold"))
jumpTo(t, -150, -25); t.color("blue")
t.write("(-150, -25), tahoma 15pt, italic bold, left align", align="left", font=("tahoma", 15, "italic bold"))
jumpTo(t, -150, -50); t.color("magenta")
t.write("(-150, -50), tahoma 15pt, bold, right align", align="right", font=("tahoma", 15, "bold"))
jumpTo(t, -150, -75); t.color("black")
t.write("(-150, -75), tahoma 15pt, bold, center align", align="center", font=("tahoma", 15, "bold"))
jumpTo(t, -150, -100); t.color("black")
t.write("(-150, -100), tahoma 15pt, bold, left align",  align="left", font=("tahoma", 15, "bold"))
```



입력 기능을 함께 가지는 도형 그리기 – textinput()

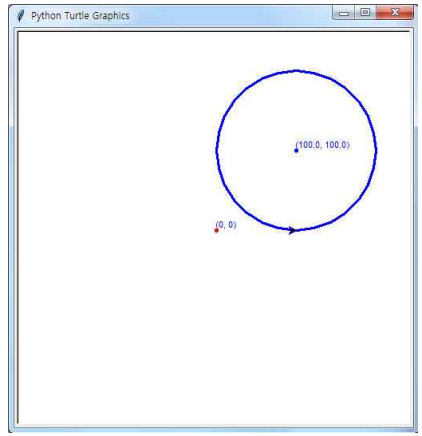
```
# Sample program that draws a polygon with input N
import turtle

# program section
t = turtle.Turtle()
t.shape('classic')
t.width(5)# set pen thickness
t.pencolor("red")
s = turtle.textinput("", "Number of points in polygon?")
n = int(s)

t.up()
t.setpos(-100, -200) # move to bottom for increased space for drawing
t.down()
for i in range(n):
    t.forward(200)
    t.left(360/n)
```

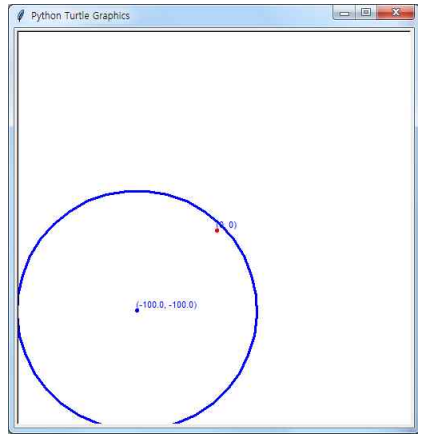




입력 기능을 함께 가지는 도형 그리기 – numinput()

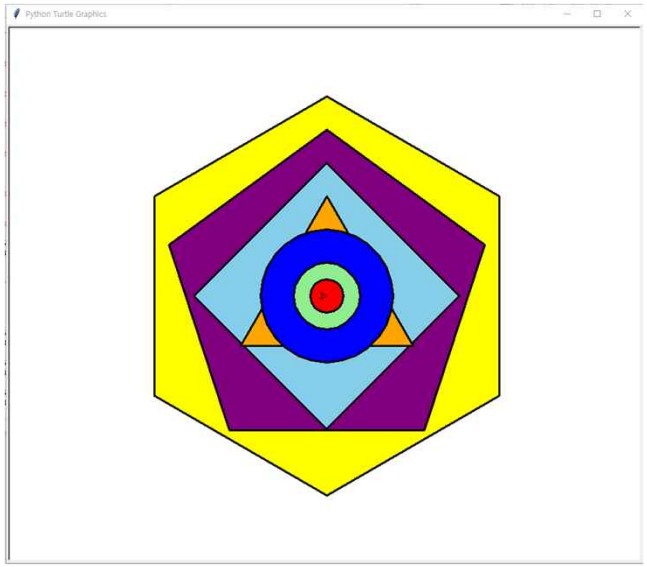
```
# turtle graphic - numberinput()

import turtle

WIN_WIDTH = 500
WIN_HEIGHT = 500
MAX_RADIUS = 200
turtle.setup(WIN_WIDTH, WIN_HEIGHT)
t = turtle.Turtle()
t.shape('classic')
t.width(3)
t.pencolor('blue')

center_x = turtle.numinput("input", "center_x : ",\
     minval=-WIN_WIDTH/2, maxval=WIN_WIDTH/2)
center_y = turtle.numinput("input", "center_y : ",\
     minval=-WIN_HEIGHT/2, maxval=WIN_HEIGHT/2)
radius = turtle.numinput("input", "radius in float : ",\
     minval=0, maxval=WIN_WIDTH/2)

t.dot(5, 'red')
t.write((0, 0))

t.up(); t.goto(center_x, center_y); t.down()
t.dot(5, 'blue')
t.write((center_x, center_y))
t.up(); t.goto(center_x, center_y - radius); t.down()
t.circle(radius)
```





도형의 색상 채우기 – fillcolor()

```
# turtle - bgcolor(), color(), width(), fill()

import turtle
t = turtle.Turtle()
t.shape('classic')
colors = ["red", "light green", "blue", "orange",\
          "sky blue", "purple", "yellow"]
num_vertices = len(colors)
radius = 50
t.speed(1)
t.width(3)
t.dot(5, "red"); t.write(t.position())
for i in range(num_vertices-1, -1, -1):
    circle_steps = i
    t.color('black')
    t.fillcolor(colors[i])
    t.begin_fill()
    if i == 0:
        t.up(); t.setposition(0, radius//2); t.down()
        t.circle(-radius//2)
    elif i == 1 or i == 2:
        t.up(); t.setposition(0, radius * i); t.down()
        t.circle(-radius * i)
    else:
        t.up(); t.setposition(0, radius * i); t.down()
        t.circle(-radius * i, steps = circle_steps)
    t.end_fill()
    t.up(); t.setposition(0, 0); t.down() # return to the origin
```



터틀 그래픽 이벤트 처리

터틀 이벤트 처리 (Event Handling) (1)

구분	터틀 이벤트 처리 함수/메소드	설명
키 입력 관련	listen(xdummy=None, ydummy=None)	TurtleScreen에 집중하여 키보드 이벤트를 받을 수 있도록 대기시킴.
	onkey(func, key) onkeyrelease(func, key)	입력키의 <key-press> 이벤트와 <key- release>이벤트를 함수 func() 에 바인드 시킴. 키는 Tk'의 키 심볼과 동일함 (예: "a", "space", "Up", "Down", "Left", "Right")
	onkeypress(func, key=None)	입력키의 <key-press> 이벤트를 함수 func() 에 바인드시킴
마우스 클릭 관련	onclick(func, btn=1, add=None)	터틀 객체에 마우스가 클릭되는 것을 함수 func(x, y)에 연결시킴. 마우스 버튼 번호는 btn = 1 (left mouse button); btn = 3 (right mouse button)으로 매핑 됨. 만약 add == True이면 새로운 바인딩을 추가. 만약 add == False이면 이전에 바인딩 되었던 내용을 교체시킴
	onscreenclick(func, btn=1, add=None)	터틀을 선택한 후 마우스가 click될 때 발생되는 <mouse-click>이벤트를 함수 func(x, y)에 바인딩시킴
	onrelease(func, btn=1, add=None)	터틀을 선택한 후 마우스가 release되는 이벤트를 함수 func(x, y)에 매핑시킴
	ondrag(func, btn=1, add=None)	마우스 드래그 이벤트를 함수 func(x, y)에 매핑시킴

터틀 이벤트 처리 (Event Handling) (2)

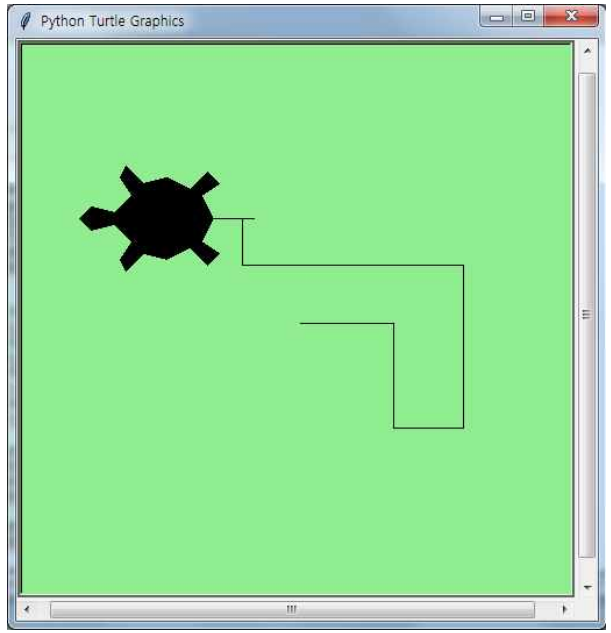
구분	터틀 이벤트 처리 함수/메소드	설명
문자열, 숫자 입력	textinput(title, prompt)	다이얼로그 박스를 사용하여 문자열을 입력받고, 입력된 문자열을 반환함
	numinput(title, prompt, default=None, minval=None, maxval=None)	다이얼로그 박스로 숫자를 입력받고, 입력된 숫자를 반환
loop 제어	ontimer(func, t_ms=0)	t_ms 밀리초 (millisecondsset) 뒤에 함수 func()가 호출되도록 타이머를 설정
	mainloop()	이벤트 반복 구문을 시작하도록 하며, tkinter main-loop() 함수가 실행되도록 함
	done()	이벤트 반복구문을 종료함

터틀 그래픽의 애니메이션

구분	터틀 애니메이션 관련 함수/메소드	설명
터틀 객체	getshapes()	사용 가능한 터틀 형상을 리스트로 반환
	register_shape(name, shape=None)	주어진 이름 (name)으로 터틀 형상을 등록. name의 터틀 형상은 gif파일, 문자열 및 형상의 좌표로 등록됨.
	turtules()	화면 (screen)에 표시되어 있는 터틀의 리스트를 제공
터틀 애니 메이션	delay(delay=None)	애니메이션 제어를 위한 밀리초 단위의 지연을 설정
	tracer(n=None, delay=None)	tracer(False)은 애니메이션을 중지시키며, tracer(True)은 애니메이션을 가동시킴. 만약 n이 양수이면 화면은 매 n-번째 update에서 갱신됨. delay는 update 간격을 설정함
	update()	사전에 설정된 지연시간 뒤에 화면을 갱신 (update) 시킴
	bye()	터틀 그래픽 화면을 지움
	exitonclick()	스크린에서 마우스를 클릭하면 turtle.bye()가 실행되도록 함.

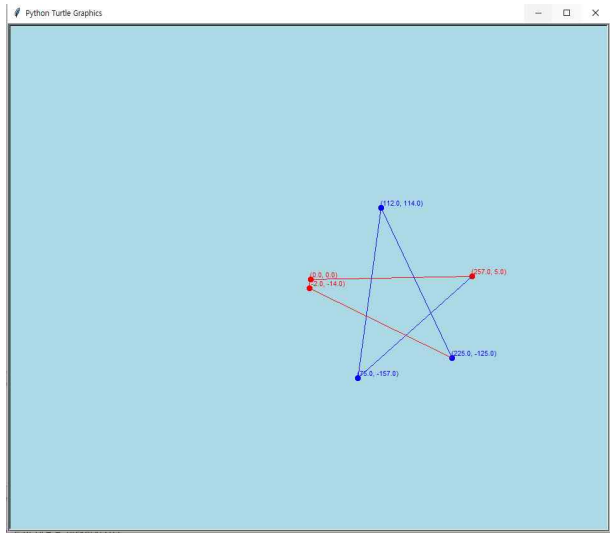
표준 입력장치 (키보드) 이벤트 처리

```
# turtle graphic - keyboard input event handling

import turtle

screen = turtle.Screen()
screen.screensize(500, 500, 'light green')
turtle.setup(500, 500) #set canvas width, canvas height
t = turtle.Turtle()
t.shape('turtle')
t.shapesize(5)


def turn_left_90():
    t.left(90)
def turn_right_90():
    t.right(90)
def forward_10():
    t.forward(10)
def backward_10():
    t.backward(10)

screen.onkey(turn_left_90, "Up")
screen.onkey(turn_right_90, "Down")
screen.onkey(forward_10, "Right")
screen.onkey(backward_10, "Left")

screen.listen()
screen.mainloop()
```



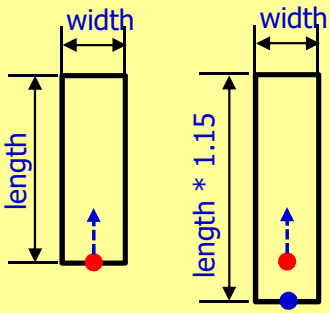
마우스 이벤트 처리 (Mouse Event Processing)

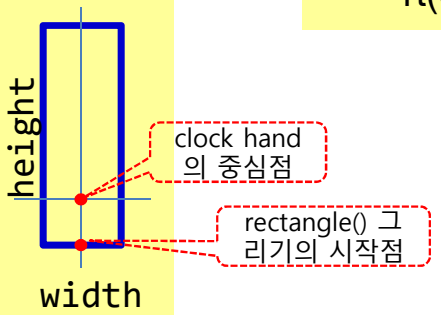
```
# Turtle mouse event processing
import turtle

board = turtle.Screen()
board.screensize(800, 600, 'light blue')

pointer = turtle.Turtle()
pointer.color('red')
pointer.pencolor('red')
pointer.ht()

def teleport_btn1(x,y):
    pointer.pencolor('red')
    pointer.goto(x,y)
    pointer.dot(10, "red")
    pointer.write((pointer.xcor(), pointer.ycor()))

def teleport_btn3(x,y):
    pointer.pencolor('blue')
    pointer.goto(x,y)
    pointer.dot(10, "blue")
    pointer.write((pointer.xcor(), pointer.ycor()))

def quitThis():
    board.bye()

pointer.dot(10, "red")
pointer.write((pointer.xcor(), pointer.ycor()))
board.onclick(teleport_btn1, btn=1)
board.onclick(teleport_btn3, btn=3)
board.onkey(quitThis,'q')

board.listen()
board.mainloop()
```



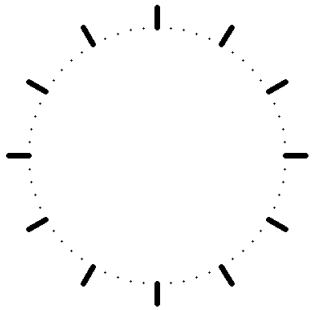
터틀 그래픽 기반 애니메이션 - 아날로그 시계

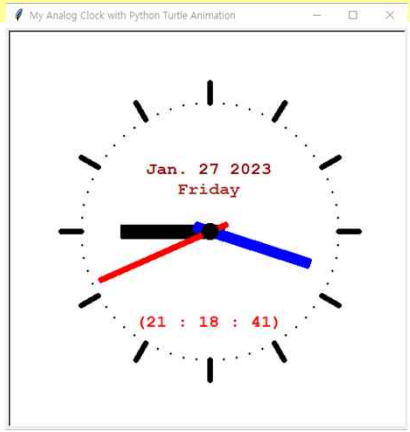
```
# Analog Clock Animation (1)
import turtle
from turtle import *
from datetime import datetime

def rectangle(length, width):
    rt(90);     fd(width/2)
    lt(90);     fd(length)
    lt(90);     fd(width)
    lt(90);     fd(length)
    lt(90);     fd(width/2)

def make_hand_shape(name, length, width):
    reset()
    backward(length*0.15) #
    begin_poly()
    rectangle(length*1.15, width)
    end_poly()
    clock_hand = get_poly()
    register_shape(name, clock_hand)
```






```
# Analog Clock Animation (2)

def clockface(radius):
    reset()
    pensize(7)
    for i in range(60):
        up(); forward(radius); down()
        if i % 5 == 0: # at each 5 min/sec
            fd(25)
            up(); backward(radius+25); down()
        else:
            dot(3) # at each min/sec except 5 min/sec
            up(); backward(radius); down()
        rt(6)
```



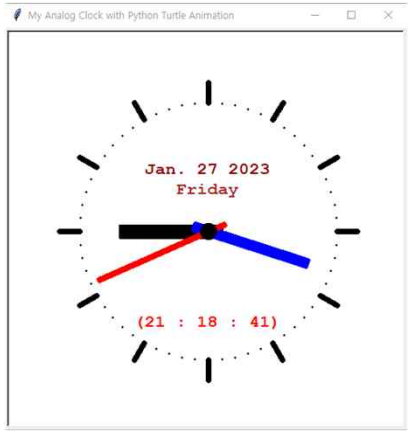


Analog Clock Animation (3)

```
def setup():
    global sec_hand, min_hand, hour_hand, hands_center, pen
    mode("logo")
    make_hand_shape("sec_hand", 150, 5)
    make_hand_shape("min_hand",  130, 10)
    make_hand_shape("hour_hand", 110, 15)
    clockface(160)
    hour_hand = Turtle()
    hour_hand.shape("hour_hand")
    hour_hand.color("black", "black")

    min_hand = Turtle()
    min_hand.shape("min_hand")
    min_hand.color("blue1", "blue1")

    sec_hand = Turtle()
    sec_hand.shape("sec_hand")
    sec_hand.color("red", "red")

    for hand in sec_hand, min_hand, hour_hand:
        hand.resizemode("user")
        hand.shapesize(1, 1, 3)
        hand.speed(0)
    ht()
                              hands_center = Turtle()
                          hands_center.shape("circle")
    pen = Turtle();    pen.ht();     pen.pu() # penup()
```

Analog Clock Animation (4)

```
def getWeekDayName(date_time):
    weekday_name = ["Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday", "Sunday"]
    return weekday_name[date_time.weekday()]

def getDateString(date_time):
    month_name = ["", "Jan.", "Feb.", "Mar.", "Apr.", "May",
      "June", "July", "Aug.", "Sep.", "Oct.", "Nov.", "Dec."]
    yr = date_time.year
    mn_str = month_name[date_time.month]
    dy = date_time.day
    return “{:s} {:d} {:d}“.format(mn_str, dy, yr)
```

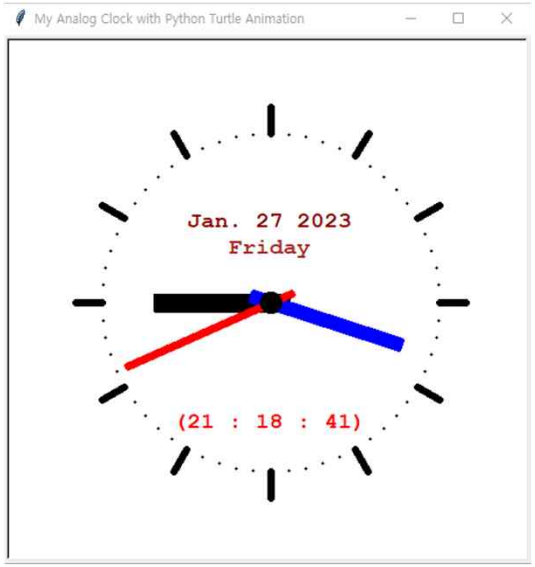


Analog Clock Animation (5)

```
def tick():
    date_time = datetime.today() # read system clock
    sec, minute, hour = date_time.second,\
        date_time.minute,  date_time.hour
    try:
        tracer(False)  # Terminator can occur here
        pen.clear()

        pen.setpos(0, 65)
        pen.pencolor("darkred")
        pen.write(getDateString(date_time), align="center",
            font=("Courier", 14, "bold"))
        pen.setpos(0, 40)
        pen.pencolor("brown")
        pen.write(getWeekDayName(date_time),\
           align = "center", font=("Courier", 14, "bold"))
                            pen.setpos(0, -125)
        hhmmss = "(%2d : %2d : %2d)"%(hour, minute, sec)
                            pen.pencolor("red")
                        pen.write(hhmmss, align="center",
                                font=("Courier", 14, "bold"))
                    sec_hand.setheading(6*sec)  # or here
                        min_hand.setheading(6*minute)
                        hour_hand.setheading(30*hour)
                                tracer(True)
                ontimer(tick, 100) # repeat at every 100 ms
                                except Terminator:
                    pass  # turtle demo user pressed STOP
```

Analog Clock Animation (6)

```
if __name__ == "__main__":
    turtle.setup(500, 500)
    turtle.title("My Analog Clock with Python Turtle Animation")
    tracer(False)
    setup()
    tracer(True)
    tick()
    mainloop()
```



tkinter GUI 프로그래밍 개요

- Window, Widget, Frame, Canvas

Tcl/Tk 및 tkinter

◆ Tcl (Tool Command Language) /Tk (toolkit)

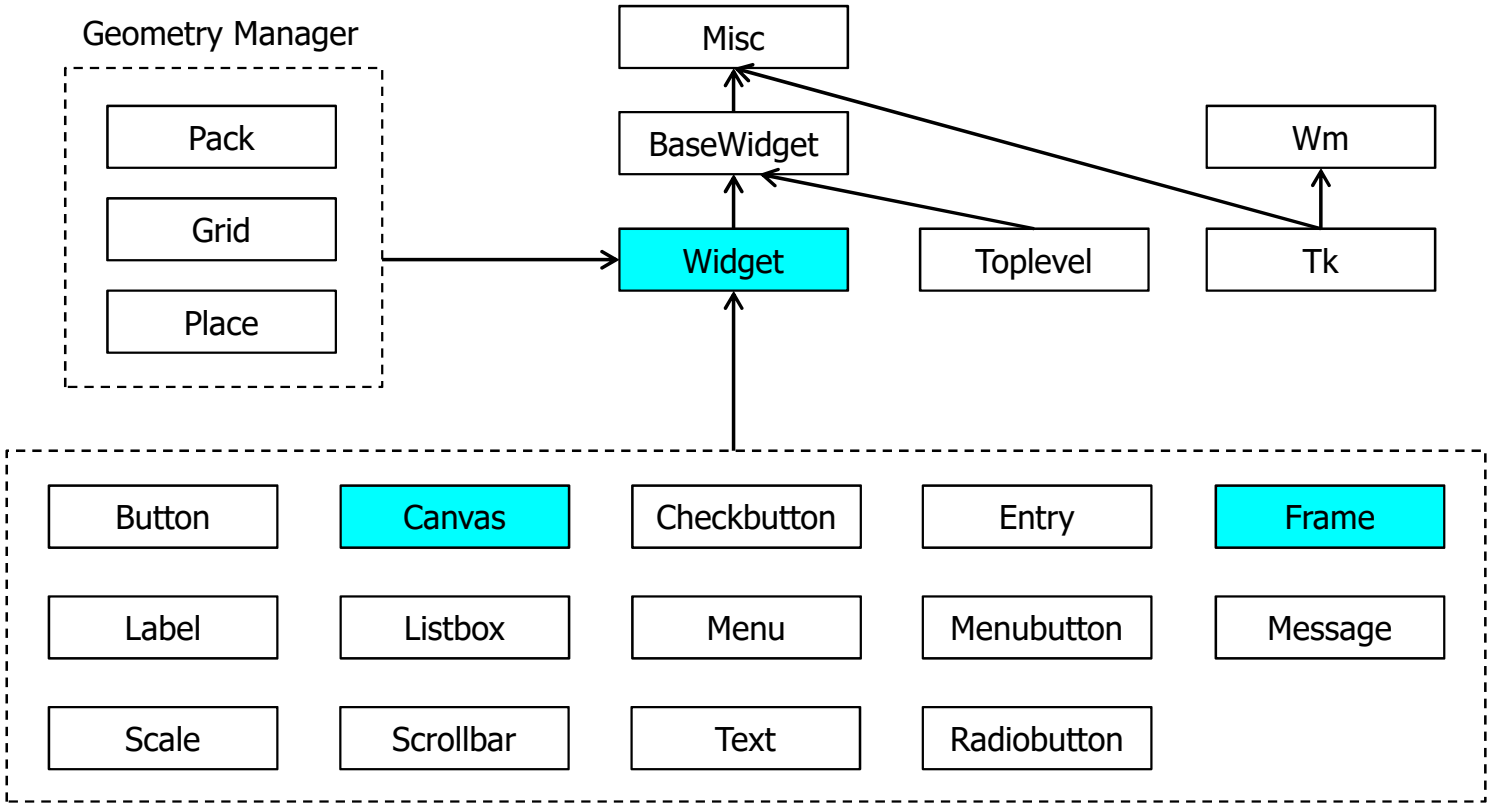
- Tcl(tool command language)은 범용 스크립트 언어
- Tk(toolkit)는 기본적인 GUI 위젯 라이브러리를 제공하는 오픈소스 크로스 플랫폼 GUI toolkit으로 많은 프로그래밍 언어에서 GUI 프로그래밍 인터페이스로 사용됨
- Tcl/Tk는 Tcl과 Tk가 함께 통합되어 있음
- tkinter 패키지는 Tcl/Tk를 사용할 수 있도록 파이썬 인터페이스(wrapper)를 제공하며, tkinter를 사용하여 Tcl 스크립트 언어를 실행할 수 있으며, GUI 프로그래밍을 할 수 있음
- 파이썬 프로그램에서는 주로 tkinter를 사용하여 Tk GUI 프로그래밍을 구현
- 파이썬 프로그램 환경인 IDLE의 사용자 인터페이스는 tkinter를 사용하여 구현되어 있음
- tkinter 패키지를 사용하기 위해서는 import tkinter로 tkinter 패키지를 설치하여야 하며, tkinter.TclVersion 속성값으로 버전을 확인할 수 있음

tkinter - window

◆ Top-level 윈도우 관련 메소드

메소드	설명
iconify(), deiconify()	윈도우를 아이콘화 (iconify)하여 작게 만들거나, 확장 시킴
geometry(newGeometry = None)	윈도우의 가로 및 세로 크기를 설정하며, 스크린에서의 위치를 설정. 포멧 "{width}x{height}{±x}{±y}" : 가로(width), 세로(height), 스크린에서의 위치 좌표 (x, y), x/y 앞에 +표시는 스크린의 왼쪽/위쪽 기준, -표시는 스크린의 오른쪽/아래쪽 기준
maxsize(width=None, height=None)	윈도우의 최대 크기를 설정
minsize(width=None, height=None)	윈도우의 최소 크기를 설정
overrideredirect(flag=None)	윈도우 덮어쓰기에 대한 설정. 만약 flag = True이면 윈도우의 이동, 크기 조절, 아이콘화, 숨기기 등이 실행되지 않게 하며, 만약 flag = False이면, 윈도우 이동 및 변경이 가능하게 함.
resizable(width=None, height=None)	윈도우 크기 조절에 대한 설정. 만약 width = False이면 윈도우 가로 크기의 변경 금지. 만약 height = False이면 윈도우의 세로 크기 변경 금지
title(text=None)	윈도우의 제목을 주어진 문자열로 설정
withdraw()	윈도우를 숨김. 윈도우의 표시는 iconify()와 deiconify()에 의해 제어됨

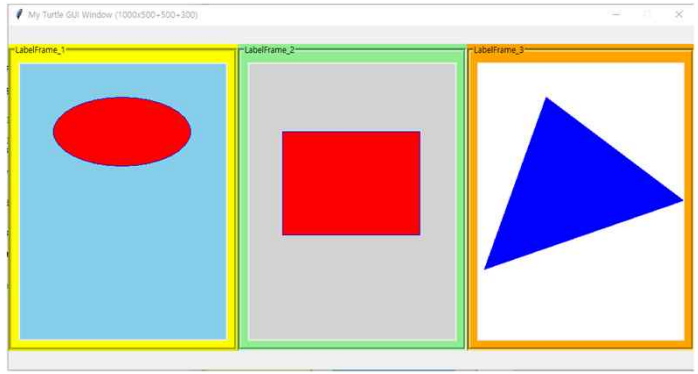
class Widget



tkinter main window 생성 및 설정

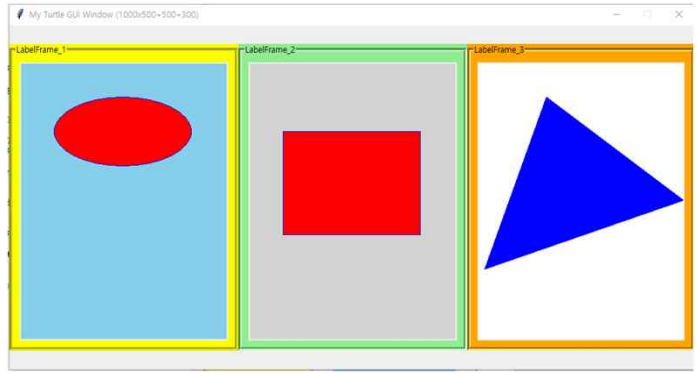
◆ Window, frame, canvas

```
# tkinter - window, frame, canvas
from tkinter import *

window = Tk()
window.title("My tkinter GUI Window (1000x500+500+300)")
window.geometry("1000x500+500+300") # width x height + pos_x + pos_y
print("window.geometry() : ", window.geometry())
window.resizable(0,0)
window.wm_attributes("-topmost", 1) # show at topmost (front)

label_frame_1 = LabelFrame(window, bg="yellow", cursor="plus", bd=5,\
            padx=10, pady=10, text="LabelFrame_1", relief=GROOVE)
label_frame_1.pack(side="left")
canvas_11 = Canvas(label_frame_1, bg="skyblue", width=300, height=400)
canvas_11.pack() # myCanvas.pack(side="top")

canvas_11.create_oval(50, 50, 250, 150, fill="red", outline="blue")
```



◆ window, frame, canvas (2)

```
# tkinter - window, frame, canvas (2)

label_frame_2 = LabelFrame(window, bg="light green", cursor="plus", bd=5,\
            padx=10, pady=10, text="LabelFrame_2", relief=GROOVE)
label_frame_2.pack(side="left")
canvas_21 = Canvas(label_frame_2, bg="light grey", width=300, height=400)
canvas_21.pack()

canvas_21.create_rectangle(50, 100, 250, 250, fill="red", outline="blue")

label_frame_3 = LabelFrame(window, bg="orange", cursor="plus", bd=5,\
            padx=10, pady=10, text="LabelFrame_3", relief=GROOVE)
label_frame_3.pack(side="left")
canvas_31 = Canvas(label_frame_3, bg="white", width=300, height=400)
canvas_31.pack()

canvas_31.create_polygon(100, 50, 10, 300, 300, 200, fill="blue", outline="blue")
```



윈도우 widget 클래스 기능 (1)

클래스	설명
Frame	컨테이너 위젯이며, 테두리와 배경을 설정할 수 있음. 응용 프로그램의 위젯들을 그룹으로 가질 수 있고, 다이얼로그 기능을 구성할 수 있음.
Toplevel	윈도우의 맨 위에 표시되어 있는 컨테이너 위젯.
LabelFrame	테두리와 제목을 가지는 프레임 위젯.
Canvas	구조화된 그래픽 위젯. 그래프나 도형을 생성하고, 그래픽 편집 기능과 사용자 정의 위젯을 생성하기 위하여 사용됨.
PanedWindow	크기를 조절할 수 있는 윈도우 창을 하위 위젯으로 포함하는 컨테이너 위젯.
Label	텍스트 문자열이나 이미지를 표시하는 위젯
Text	형식에 따라 표시되는 문자열. 다양한 스타일과 속성으로 포멧을 지정하는 문자열 표시. 이미지나 윈도우에 내장되어 사용될 수 있음
Entry	문자열 입력 필드
Spinbox	지정된 구간이나 순서가 설정된 집합으로 부터 항목을 선정할 수 있는 변형된 엔트리 위젯
Button	버튼 (Button): 단순 버튼이며 명령어나 다른 동작을 실행하게 함. 체크버튼 (CheckButton): 선택 또는 비선택의 두가지 값을 가질 수 있으며, 버튼을 클릭하여 이 상태를 전환 (toggle)시킬 수 있음. 메뉴버튼 (Menubutton): pulldown 메뉴를 구현. 라디오 버튼(Radiobutton): 다수의 선택가능 값 중에서 하나만 선택되도록 함. 버튼을 클릭하면 그 항목이 설정되며, 동일한 변수에 관련된 다른 라디오 버튼을 해제시킴.

윈도우 widget 클래스 기능 (2)

클래스	설명
Listbox	설정가능한 대안들을 열거함. Listbox는 라디오 버튼이나 체크 리스트로 구성될 수 있음.
Menu	메뉴 창. pulldown 또는 popup 메뉴를 구현하기 위하여 사용됨.
Message	문자열을 표시하며, label 위젯과 유사하나 주어진 넓이 또는 가로-세로 비율에 따라 자동적으로 줄 바꿈 기능이 제공됨.
Scale	슬라이드를 옮겨서 값을 설정할 수 있게 함.
Scrollbar	캔버스, 엔트리, 리스트 박스 및 텍스트 위젯에서 사용할 수 있는 표준 스크롤 바.

◆ Entry widget

Entry 속성 및 관련함수	설 명
focus()	엔트리 위젯에서 직접 키보드 입력이 가능하도록 설정
bind()	엔트리 위젯에서 입력 값이 특정 내용 (예: 엔터키 : <Return>)인 이벤트가 발생할 때 실행되는 함수를 바인딩

◆ Button 속성

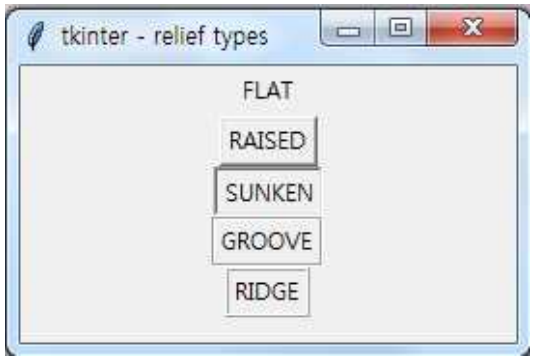
Button 속성 및 관련함수	설 명
text	버튼에 표시되는 문자열
command	버튼이 눌려지는 발생되는 이벤트에서 실행되는 함수를 바인딩

파이썬 색상 (Python Colors)

frame, canvas

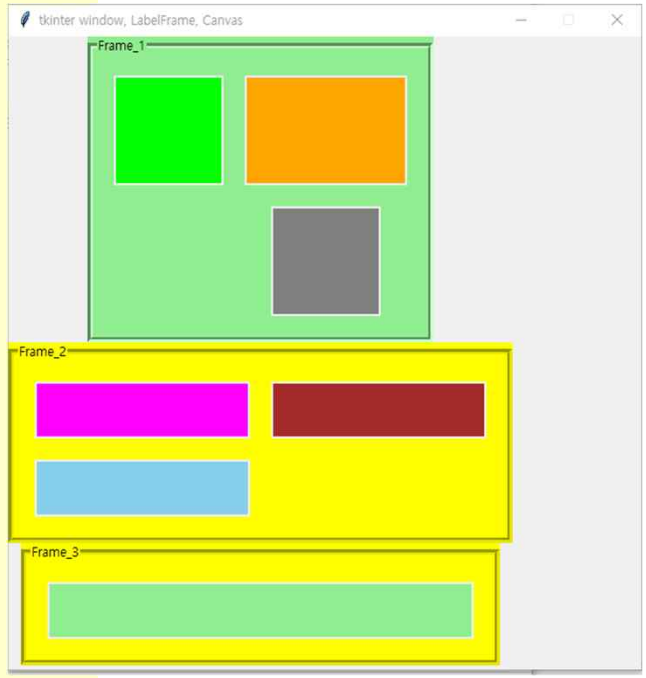
◆ Frame의 주요 속성

속 성	설 명
bd, borderwidth	프레임의 테두리 두께, 기본값은 0
bg, background	배경색, 기본값은 light gray
cursor	프레임에서 마우스 커서, 'arrow', 'man', 'plus', 'circle', 'dot', heart' 등
width, height	프레임의 가로와 세로 길이 grid_propagate(0)를 호출할 때에만 유효
padx, pady	프레임의 가로와 세로 패딩 화소 수
relief	프레임의 모습 스타일: FLAT(기본값), RAISED, SUNKEN, GROOVE, RIDGE



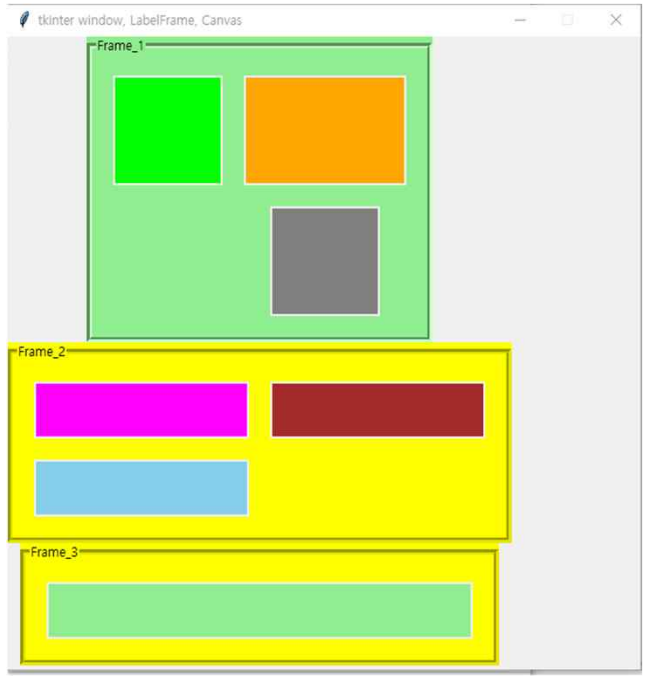
window내부에 frame 및 canvas 설치

```
# tkinter - window configurations (1)

import tkinter
from tkinter import *

window = Tk()
window.title("tkinter window, LabelFrame, Canvas")
window.geometry("600x600+200+100")
# "{w}x{h}{±x}{±y}", +/-x : from left/right end, +/-y: from top/bottom
window.resizable(width=False, height=False) # prohibits window resizing

frame1 = LabelFrame(window, bg="light green", cursor="heart",\
                bd=5, padx=10, pady=10, text="Frame_1", relief=GROOVE)
frame1.grid(row=0, column=0)
frame2 = LabelFrame(window, bg="yellow", cursor="man", bd=5,\
                padx=10, pady=10, text="Frame_2", relief=GROOVE)
frame2.grid(row=1, column=0)
frame3 = LabelFrame(window, bg="yellow", cursor="plus", bd=5,\
                padx=10, pady=10, text="Frame_3", relief=GROOVE)
frame3.grid(row=2, column=0)

canvas_11 = Canvas(frame1, bg="lime", width=100, height=100)
canvas_11.grid(row=0, column=0, fill=None, padx=10, pady=10)
canvas_12 = Canvas(frame1, bg="orange", width=150, height=100)
canvas_12.grid(row=0, column=1, fill=None, padx=10, pady=10)
canvas_13 = Canvas(frame1, bg="grey", width=100, height=100)
canvas_13.grid(row=1, column=1, fill=None, padx=10, pady=10)
```



window내부에 frame 및 canvas 설치

```
# tkinter - window configurations (2)

canvas_21 = Canvas(frame2, bg="magenta", width=200, height=50)
canvas_21.grid(row=0, column=0, fill=None, padx=10, pady=10)
canvas_22 = Canvas(frame2, bg="brown", width=200, height=50)
canvas_22.grid(row=0, column=1, fill=None, padx=10, pady=10)
canvas_23 = Canvas(frame2, bg="sky blue", width=200, height=50)
canvas_23.grid(row=1, column=0, fill=None, padx=10, pady=10)

canvas_31 = Canvas(frame3, bg="light green", width=400, height=50)
canvas_31.grid(row=0, column=0, fill=None, padx=10, pady=10)
```



위젯들의 배치 관리자 (layout manager)

◆ widget들의 layout manager

배치 관리자		예제 및 설명
압축배치	pack()	pack(fill=BOTH, expand=1)
격자배치	grid()	grid(row=x, column=y, sticky=NSEW)
절대위치배치	place()	place(x=100, y=100)

◆ grid() 메소드의 주요 옵션

grid() options	설 명
column	widget을 삽입할 열 (column), 기본값 0으로부터 시작
row	widget을 삽입할 행 (row), 0으로부터 시작, 기본값은 비어있는 첫 행
columnspan	widget이 차지할 열의 개수, 기본값 1
rowspan	widget이 차지할 행의 개수, 기본값 1
in	widget이 놓일 widget, 기본값은 parent
padx, pady	셀 주위의 패딩 화소, 기본값은 0
sticky	widget을 셀에 어떻게 붙일 것인지 지정. S, N, E, W를 조합하여 지정. 기본 값은 중앙에 위치, NSEW는 셀의 경계에 붙임

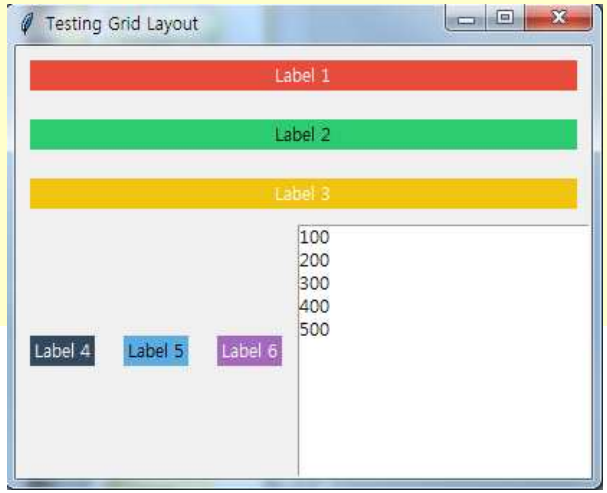
위젯들의 배치 관리자 (layout manager)

◆ place() 메소드의 주요 옵션

place() options	설 명
anchor	widget의 기준 위치, N, E, S, W로 지정. 기본값은 NW, NE, SE, SW, CENTER 등 설정 가능
bordermode	위젯의 테두리를 INSIDE, OUTSIDE
x, y	widget의 위치
height, width	widget의 가로, 세로의 크기
relx, rely	parent widget의 상대 위치 0.0~1.0
relheight, relwidth	parent widget의 상대 크기 0.0~1.0

압축 배치 – pack()

```
# tkinter - pack layout
from tkinter import *
window = Tk()
window.title("Testing Pack Layout")
window.geometry('400x300+100+200')

# fill option
label1 = Label(window, text="Label 1", bg="#E74C3C", fg="white").pack(fill=X, padx=10, pady=10)
label2 = Label(window, text="Label 2", bg="#2ECC71", fg="black").pack(fill=X, padx=10, pady=10)
label3 = Label(window, text="Label 3", bg="#F1C40F", fg="white").pack(fill=X, padx=10, pady=10)

# side option
label4 = Label(window, text="Label 4", bg="#34495E", fg="white").pack(fill=X, padx=10, pady=10, side=LEFT)
label5 = Label(window, text="Label 5", bg="#5DADE2", fg="black").pack(fill=X, padx=10, side=LEFT)
label6 = Label(window, text="Label 6", bg="#A569BD", fg="white").pack(fill=X, padx=10, side=LEFT)

# expand option
listbox = Listbox(window)
listbox.pack(fill=BOTH, expand=1)

L = [100, 200, 300, 400, 500]
for i in range(len(L)):
    listbox.insert(END, str(L[i]))

mainloop()
```

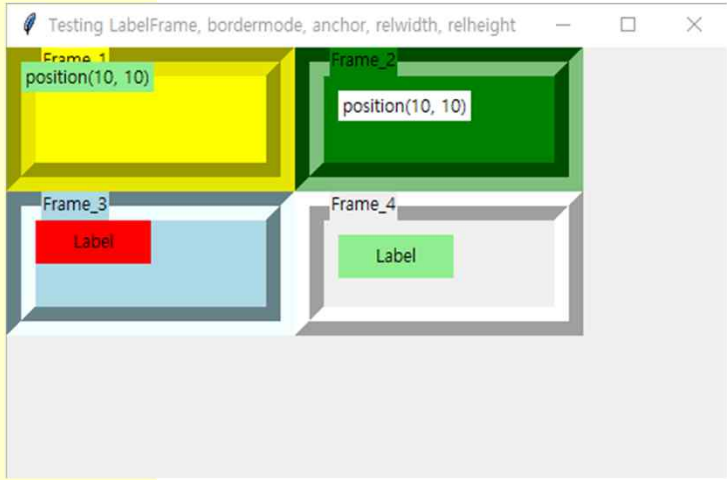


격자 배치 – grid()

```
# Frame with Label and Entry Objects, using grid()
from tkinter import *

def fetch(cells):
    print("Input data: ")
    for i, e in enumerate(cells):
        print("{0}:{1}".format(fields[i], e.get()))

def make_form(fields):
    cells = []
    for r, field in enumerate(fields):
        label = Label(window, width=10, text = field)
        entry = Entry(window)
        label.grid(row=r, column=0, sticky=NSEW)
        entry.grid(row=r, column=1, sticky=NSEW)
        cells.append(entry)
    return cells

if __name__ == "__main__":
    window = Tk()
    window.title("Input Dialog with Label and Entry Objects")
    fields = ("Name", "Age", "Address")
    cells = make_form(fields)
    window.bind("<Return>", (lambda event, e=cells: fetch(e)))
    Button(window, text="Fetch", bg="green",
           command = (lambda e=cells: fetch(e))).grid(row=3, column=0, sticky=NSEW)
    Button(window, text="Quit", bg="Red",
           command = quit).grid(row=3, column=1, sticky=NSEW)
    print("grid size: ", window.grid_size())
    window.grid_columnconfigure(1, weight=1)
    window.grid_rowconfigure(0, weight=1)
    window.mainloop()
```

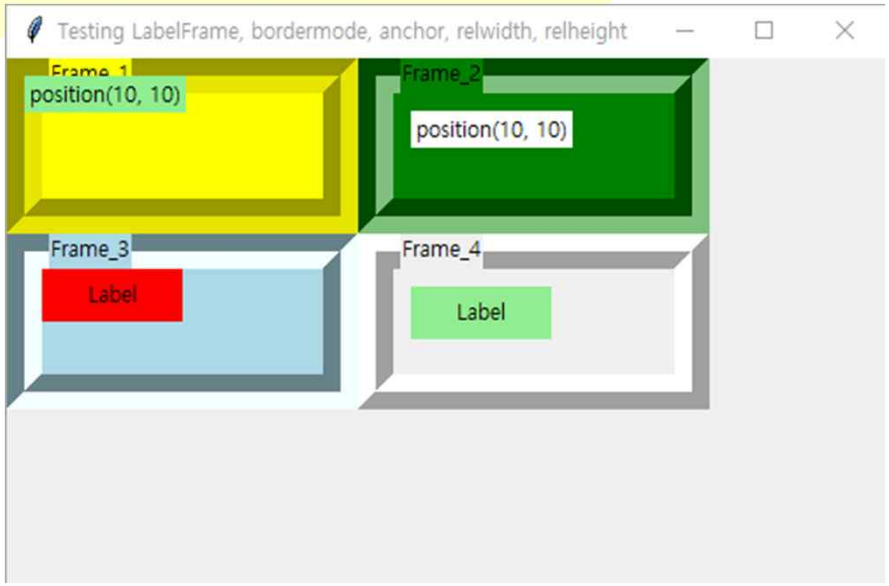
절대위치 배치 – place()

```
# tkinter place() layout - testing bordermode, anchor, relwidth, relheight

from tkinter import *

window = Tk()
window.title("Testing LabelFrame, bordermode, anchor, relwidth, relheight")
window.geometry("500x300+100+100")

f1 = LabelFrame(window, borderwidth=20, relief=GROOVE,\
           width=200, height=100, text="Frame_1", bg="yellow")
f1.grid(row=0, column=0)
l1 = Label(f1, text="position(10, 10)", bg="light green")
l1.place(x=10, y=10, bordermode="outside")

f2 = LabelFrame(window, borderwidth=20, relief=GROOVE,\
           width=200, height=100, text="Frame_2", bg="green")
f2.grid(row=0, column=1)
l2 = Label(f2, text="position(10, 10)", bg="white")
l2.place(x=10, y=10, bordermode="inside")

f3 = LabelFrame(window, borderwidth=20, relief=GROOVE,\
           width=200, height=100, text="Frame_3", bg="light blue" )
f3.grid(row=1, column=0)
l3 = Label(f3, text="Label", bg="red")
l3.place(bordermode="inside", anchor=NW, relwidth=0.5, relheight=0.5)
```



절대위치 배치 – place()

```
# tkinter place() layout - testing bordermode, anchor, relwidth, relheight (2)

f4 = LabelFrame(window, borderwidth=20, relief=RIDGE, width=200, height=100,\
                text="Frame_4" )
f4.grid(row=1, column=1)
l4 = Label(f4, text="Label", bg="light green")
l4.place(x=10, y=10, bordermode="inside", relwidth=0.5, relheight=0.5)

window.mainloop()
```



tkinter GUI 프로그래밍

- canvas 기반의 다양한 도형 (arc, oval, polygon) 그리기
- canvas의 text 출력
- widget (label, button, menu) 사용

캔버스 생성 및 활용

◆ 캔버스 객체 메소드 (1)

canvas method	설 명
after(delay_ms, callback=None, *args)	delay_ms 밀리초 후에 callback 함수 호출; callback=None이면 sleep(delay_ms)으로 대기
bind(sequence=None, func=None, add=None)	sequence 이벤트에 대한 핸들러 func바인딩, add='+'이면 기존 핸들러에 추가
bind_all(sequence=None, func=None, add=None)	응용 프로그램의 모든 위젯에 이벤트 핸들러 바인딩
bind_class(className, sequence=None, func=None, add=None)	className 이름의 모든 위젯에 이벤트 핸들러 func 바인딩
config(option=value, ...) configure(option=value, ...)	옵션 속성변경, w[option] = value
create_arc(bbox, **options)	원호 (arc) 생성
create_bitmap(position, **optiion)	비트맵 생성
create_image()	이미지 생성
create_line(coords, **options)	선 생성
create_oval(bbox, **options)	타원 생성
create_polygon(coords, **options)	다각형 생성
create_rectangle(bbox, **options)	사각형 생성
create_text(position, **options)	텍스트 생성

캔버스 생성 및 활용

◆ 캔버스 객체 메소드 (2)

canvas method	설 명
create_window(position, **options)	윈도우 생성
pos = canvas. coords(obj_name, pos)	obj_name 객체의 현재 위치: pos[0]: x_0, pos[1]: y_0, pos[2]: x_1, pos[3]: y_1
destroy()	위젯 w와 그 위젯 상의 객체들을 모두 삭제
move(item, dx, dy)	item 객체를 dx, dy만큼 이동
update()	디스플레이를 갱신 (update)
update_idletasks()	다시 그리기 (redrawing), 크기 변경 (resizing) 등 디스플레이 갱신이 다음 갱신 전에 처리되도록 강제
winfo_geometry()	스크린상의 캔버스 위치 및 크기를 문자열로 반환 w × h ± x ± y
winfo_width(), winfo_height()	픽셀단위의 가로와 세로크기
winfo_x(), winfo_y()	부모로부터 좌상단 (left, top)의 상대 위치
mainloop()	이벤트 처리 반복 구간을 시작
quit()	이벤트 처리 반복 구간을 중지

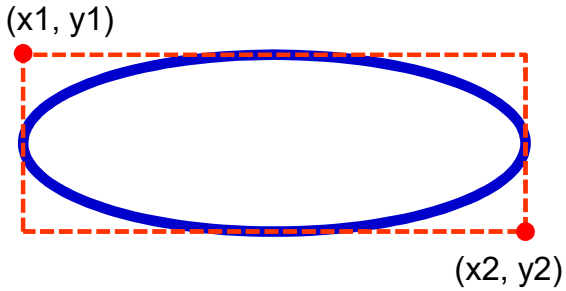
Canvas 객체 메소드

◆ canvas 생성

- canvas = Canvas(window, bg="light blue", width=600, height=400, relief="solid", bd = 2)
- canvas.pack(expand=NO, side="bottom")

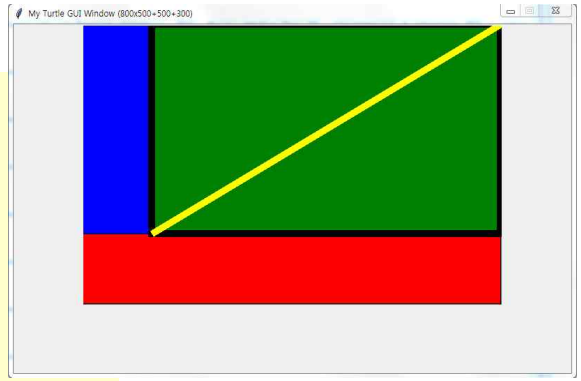
◆ 타원형 생성

- canvas.create_oval(x1, y1, x2, y2, fill_color, tag)



tkinter GUI programming - canvas

◆ canvas.create_rectangle(), create_line()

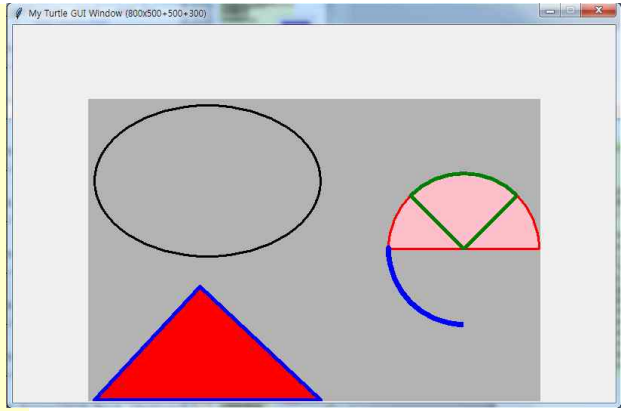



```
# tkinter_GUI -  create_rectangle, create_line
from tkinter import *

window = Tk()
window.title("My tkinter GUI Window (800x500+500+300)")
window.geometry("800x500+500+300") # width, height, position
print("windowt.geometry() : ", window.geometry())
window.resizable(0,0)
window.wm_attributes("-topmost", 1) # show at topmost (front)
myCanvas = Canvas(window, bg="grey40", width=600, height=400) #bg : grey40, yellow
myCanvas.pack() # myCanvas.pack(side="top")
myCanvas.create_rectangle(0, 0, 100, 300, fill="blue")
myCanvas.create_rectangle(0, 300, 600, 400, fill="red")
myCanvas.create_rectangle(100, 0, 600, 300, fill="green", width=10)
myCanvas.create_line(100, 300, 600, 0, fill="yellow", width=10)
```

tkinter GUI programming - canvas

◆ canvas.create_polygon(), _oval(), _arc()

```
# tkinter_GUI – canvas.create_rectangle, create_line
from tkinter import *

window = Tk()
window.title("My tkinter GUI Window (800x500+500+300)")
window.geometry("800x500+500+300") # width, height, position
print("window.geometry() : ", window.geometry())
window.resizable(0,0)
window.wm_attributes("-topmost", 1)
    # show at topmost (front)
myCanvas = Canvas(window, bg="grey70", width=600, height=400)
    #bg : grey40, yellow
myCanvas.pack(side="bottom")
    # myCanvas.pack(side="top")
myCanvas.create_oval(10, 10, 310, 210, fill="", width=3)
myCanvas.create_polygon(10, 400, 310, 400, 150, 250,₩
     fill="#ff0000", outline="#0000ff", width=5) # fill = 'red', outline = 'blue'
myCanvas.create_arc(400, 100, 600, 300, start = 0, extent=180, fill="pink",
    outline="red", width=3)
myCanvas.create_arc(400, 100, 600, 300, start = 180, extent=90, ₩
    outline="blue", style=ARC, width=7)
myCanvas.create_arc(400, 100, 600, 300, start = 45, extent=90,₩
    outline="green", style=PIESLICE, width=5)
```



tkinter GUI programming - canvas

◆ canvas.create_text()

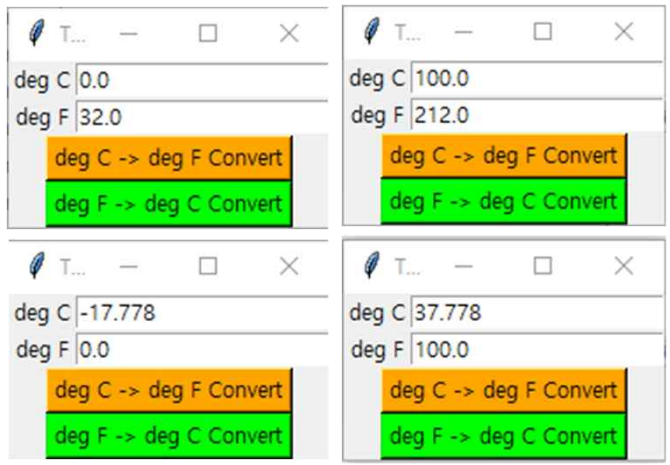

```
# tkinter_GUI – canvas. create_text()
from tkinter import *

window = Tk()
window.title("My tkinter GUI Window (800x500+500+300)")
window.geometry("800x500+500+300") # width, height, position
print("window.geometry() : ", window.geometry())
window.resizable(0,0)
window.wm_attributes("-topmost", 1) # show at topmost (front)
myCanvas = Canvas(window, bg="grey70", width=700, height=400) #bg : grey40, yellow
myCanvas.pack(side="bottom") # myCanvas.pack(side="top")
myCanvas.create_text(150, 50, text="(150, 50), times 10pt", fill="red", font=("times", 10))
myCanvas.create_text(150, 75, text="(150, 75), times 10pt bold", fill="red", font=("times 10 bold"))
myCanvas.create_text(150, 100, text="(150, 100), arial 12pt", fill="blue", font=("arial 12"))
myCanvas.create_text(150, 125, text="(150, 125), arial 12pt bold", fill="blue", font=("arial 12 bold"))
myCanvas.create_text(150, 150, text="(150, 150), 맑은 명조 15pt", fill="green", font=("맑은명조 15"))
myCanvas.create_text(150, 175, text="(150, 175), 맑은 명조 15pt bold", fill="green", font=("맑은명조 15 bold"))
myCanvas.create_text(150, 200, text="(150, 200), tahoma 15pt", fill="green", font=("tahoma 15"))
myCanvas.create_text(150, 225, text="(150, 225), tahoma 15pt bold", fill="green", font=("tahoma 15 bold"))
myCanvas.create_text(150, 250, text="(150, 250), tahoma 15pt italic bold ", fill="green", font=("tahoma 15 italic bold"))
myCanvas.create_text(10, 275, text="(10, 275), tahoma 15pt italic bold anchor=W", fill="green",₩
    font=("tahoma 15 italic bold"), anchor=W)
myCanvas.create_text(10, 300, text="(10, 300), tahoma 10pt italic bold anchor=W", fill="green",₩
    font=("tahoma 10 italic bold"), anchor=W)
myCanvas.create_text(10, 325, text="(10, 325), tahoma 20pt italic bold anchor=W", fill="green",₩
    font=("tahoma 20 italic bold"), anchor=W)
```

tkinter Label, Entry, Button, Scale 활용

◆ 섭씨-화씨 온도 변환 계산기 구현

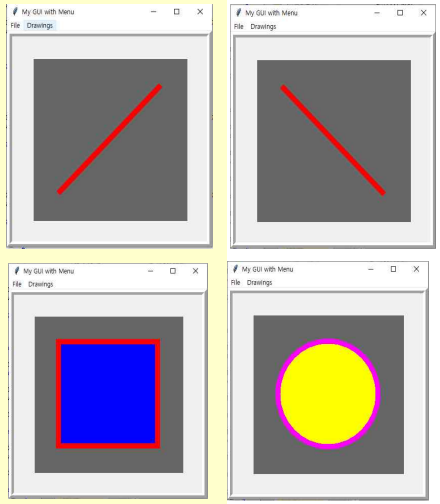
```
#  Temperature Converter (1)

from tkinter import *

class App:
    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        Label(frame, text = 'deg C').grid(row=0, column=0)
        self.c_var = DoubleVar()
        Entry(frame, textvariable=self.c_var).grid(row=0, column=1)
        Label(frame, text='deg F').grid(row=1, column=0)
        self.f_var = DoubleVar()
        Entry(frame, textvariable=self.f_var).grid(row=1, column=1)
        button_CF = Button(frame, text='deg C -> deg F Convert',\
           bg = "orange", command=self.convert_CF)
        button_CF.grid(row=2, columnspan=2)
        button_FC = Button(frame, text='deg F -> deg C Convert',\
           bg = "lime", command=self.convert_FC)
        button_FC.grid(row=3, columnspan=2)

    def convert_CF(self):
        c = self.c_var.get()
        f = c * 1.8 + 32
        f_round = round(f, 3)
        self.f_var.set(f_round)
```

```
#  Temperature Converter (2)

    def convert_FC(self):
        f = self.f_var.get()
        c = (f - 32) / 1.8
        c_round = round(c, 3)
        self.c_var.set(c_round)
#--------------------------------
root = Tk()
root.wm_title('Temp Converter')
app = App(root)
root.mainloop()
```



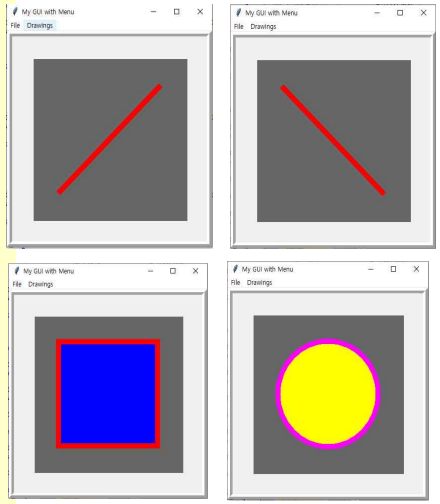
Menu Widget

```
# tkinter - Menu Widget

from tkinter import *
from tkinter import messagebox

def onFileNew():
    print("select New")

def onFileExit():
    result = messagebox.askquestion("Exit Confirm", "Quit ?")
    if result == "yes":
        window.destroy()

def onGraphic():
    global fr, cv
    cv.delete("all")
    DT = drawType.get()
    print("drawType = ", DT)
    if DT == 1:
        print(" (line_slash)")
        cv.create_line(50, 250, 250, 50, fill='red', width=10)
    elif DT == 2:
        print(" (line_back_slash)")
        cv.create_line(50, 50, 250, 250, fill='red', width=10)
    elif DT == 3:
        print(" (Rectangle)")
        cv.create_rectangle(50, 250, 250, 50, outline='red', fill='blue', width=10)
    elif DT == 4:
        print(" (Oval)")
        cv.create_oval(50, 50, 250, 250, outline='magenta', fill = 'yellow', width=10)
    else:
        print(" (Undefined type)")
```



Menu Widget

```
def makeMenu(master):
    menuBar = Menu(master)
    master.config(menu = menuBar)

    filemenu = Menu(menuBar, title = "file")
    filemenu.add_command(label="New ...", command=onFileNew)
    filemenu.add_command(label="Exit", command=onFileExit)
    menuBar.add_cascade(label="File", menu=filemenu)

    drawing = Menu(menuBar, tearoff=0)
    drawing.add_radiobutton(label="Line_slash",
        variable=drawType, value=1, command=onGraphic)
    drawing.add_radiobutton(label="Line_back_slash",
        variable=drawType, value=2, command=onGraphic)
    drawing.add_radiobutton(label="Rectangle",
        variable=drawType, value=3, command=onGraphic)
    drawing.add_radiobutton(label="Oval",
        variable=drawType, value=4, command=onGraphic)
    menuBar.add_cascade(label="Drawings", menu=drawing)

if __name__ == "__main__":
    global fr, cv
    window = Tk()
    window.title("My GUI with Menu")
    window.geometry("400x400+100+100")
    drawType = IntVar();
    drawType.set(1)
    makeMenu(window)
    fr = Frame(window, borderwidth=10, relief=RIDGE, width=300, height=300)
    fr.pack(expand=YES, fill=BOTH)
    cv = Canvas(fr, bg="grey40", width=300, height=300)
    #cv.pack(expand=True, fill="both")
    cv.place(x=40, y=40)

    window.mainloop()
```



tkinter Label, Entry, Button, Scale 활용

◆ 스톱워치 만들기


```
# class StopWatch  (1)
import time
from tkinter import *

class StopWatch():
    def __init__(self, window):
        self.window = window
        self.frame = LabelFrame(window, text="My Simple Stop Watch", relief=GROOVE)
        self.frame.grid(row=0, column=0)
        self.start_time = time.time()
        self.stop_time = time.time()
        self.elapsed_time = 0.0
        self.prev_elapsed_time = 0.0
        self.running = False
        self.timeText = Label(self.frame, height = 5, text="{:>7s}".format("0.000"), font=("Arial 40 bold"))
        self.timeText.pack(side = TOP)
        self.startButton = Button(self.frame, width=10, text="Start", bg="green", command=self.start)
        self.startButton.pack(side = LEFT)
        self.stopButton = Button(self.frame, width=10, text="Stop", bg="red", command=self.stop)
        self.stopButton.pack(side = LEFT)
        self.resetButton = Button(self.frame, width=10, text="Reset", bg="yellow", command=self.reset)
        self.resetButton.pack(side = LEFT)
```

```
# class StopWatch  (2)

    def runTimer(self):
        if (self.running):
            self.cur_time = time.time()
            time_diff = self.cur_time - self.start_time
            self.elapsed_time = time_diff + self.prev_elapsed_time
            self.timeText.configure(text="{:7.3f}".format(self.elapsed_time))
        self.window.after(10, self.runTimer)

    def start(self):
        if (self.running != True):
            self.running = True
            self.start_time = time.time()
            self.prev_elapsed_time = self.elapsed_time

    def stop(self):
        self.running = False
        self.prev_elapsed_time = self.elapsed_time

    def reset(self):
        self.running = False
        self.elapsed_time = 0.0
        self.prev_elapsed_time = 0.0
        self.timeText.configure(text="{:>7s}".format("0.000"))

if __name__ == '__main__':
    window = Tk()
    window.title("My Simple Stop Watch")
    stop_watch = StopWatch(window)
    stop_watch.runTimer()
    window.mainloop()
```

tkinter 키보드 및 마우스 이벤트 처리

tkinter 주요 이벤트

◆ tkinter 주요 이벤트

이벤트 구분	이벤트 유형	설 명
키보드	KeyPress, Key	키 누름
	KeyRelease	키 뗌
마우스	Button	마우스 버튼 누름 (press)
	ButtonRelease	마우스 버튼 뗌 (up, release)
	Motion	마우스를 움직임
	B1-Motion, B3-Motion	마우스 버튼 1 (left), 마우스 버튼 3(right)
	Enter	마우스 포인터가 위젯으로 들어감
	Leave	마우스 포인터가 위젯을 벗어남
위젯	Configure	위젝의 크기가 변경됨
	Destroy	위젯이 삭제됨
	Expose	위젯 또는 응용 프로그램의 일부가 다른 윈도우에 가렸다 다시 보여짐

주요 키보드 입력 문자열 (keysym)

◆ 키보드 입력키의 문자열 (keysym)

키보드 입력 키의 문자열 (keysym)	설 명
Alt, Alt_L	왼쪽 Alt 키
Control, Control_L	왼쪽 Control 키
Shift_L, Shift_R	왼쪽 Shift 키
Tab, Escape	Tab, Esc 키
BackSpace	Backspace 키
Cancel, Pause	Break, Pause 키
Caps_Lock, Num_Lock	Caps_Lcok, Num_Lock
Insert, Delete, Home, End	Ins, Del, Home, End
Execute, Print	SysReq, PrintScreen
Left, Right, Down, Up	상, 하, 좌, 우 화살표 키
Next, Prior	Page Down, Page Up 키
A, ..., Z, a, ..., z, 0, ..., 9	영문 대소문자, 숫자 키
F1, ..., F12	Function 키
Return, Space	Enter, Space 키
KP_0, ..., KP_9	키패드 0~9
KP_Add, KP_Subtract, KP_Multiply, KP_Divide	키패드 +, -, *, /
KP_Left, KP_Right, KP_Down, KP_Up	키패드 상, 하 , 좌, 우 키
KP_Insert, KP_Delete, KP_Decimal	키패드 Insert, delete, 마침표
KP_Enter, KP_End, KP_Home, KP_Begin	키패드 Enter, End, Home, Begin
KP_Next, KP_Prior	키패드 Page Up, Page Down

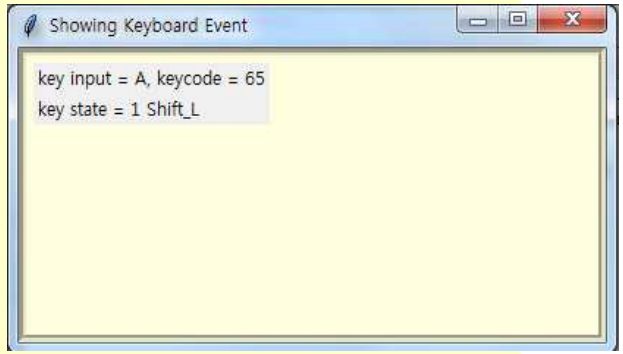
키보드 이벤트 객체

◆ 키보드 이벤트 객체의 속성

키보드 이벤트 속성	설 명
event.char	KeyPress, KeyRelease 이벤트에서의 ASCII 문자
event.keycode	KeyPress, KeyRelease 이벤트에서 숫자 코드값, 대소문자 구별 없음
event.keysym	특수키를 포함한 KeyPress, KeyRelease 이벤트에서 키의 문자열
event.width, event.height	Configure 이벤트에서 widget의 가로와 세로 크기
event.state	modifier 키의 상태, mask 비트 연산으로 체크 확인 0x0001 (shift), 0x0002(Caps_Lock), 0x0004(Ctrl), 0x0008 (Alt_L), 0x0010 (Num_Lock), 0x0100(Button1), 0x400(Button 3)

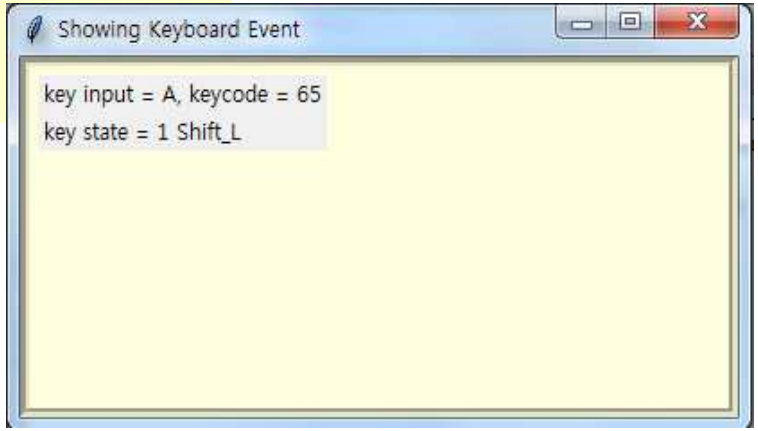
키보드 이벤트 처리 예제

```
# Handling Keyboard Events (1)
from tkinter import *

def keyEvent(event):
    if len(event.char) == 0:
        input_char = event.keysym
    else:
        input_char = event.char
    label_ky.configure(text="key input = {}, keycode = {}".format(input_char, event.keycode))
    label_ky.update
    ky_state = ""
    if event.state & 0x0001:
        ky_state += "Shift_L "
    if event.state & 0x0002:
        ky_state += "Caps_Lock "
    if event.state & 0x0004:
        ky_state += "Ctrl "
    if event.state & 0x0008:
        ky_state += "Alt_L "
    if event.state & 0x0010:
        ky_state += "Num_Lock "
    if event.state & 0x0080:
        ky_state += "Alt_R "
    label_state.configure(text="key state = {} {}".format(event.state, ky_state))
    label_state.update
```



키보드 이벤트 처리 예제

```
# Handling Keyboard Events (2)
#-----------------------------------
# main loop
window = Tk()
window.geometry("400x200+100+100")
window.title("Showing Keyboard Event")
frame = Frame(window, bg="light yellow", bd = 5, padx=5, pady=5, relief=GROOVE)
frame.pack(expand=YES, fill=BOTH)
label_ky = Label(frame, text="key input = {}".format(""),)
label_ky.update()
label_ky.grid(row=0, column=0, sticky=NSEW)
label_state = Label(frame, text="key state = {}".format(""), anchor="w")
label_state.update()
label_state.grid(row=1, column=0, sticky=NSEW)

window.bind("<Key>", keyEvent)
window.mainloop()
```



마우스 이벤트

◆ 마우스 이벤트 속성 및 이벤트 종류 (1)

구분	마우스 이벤트 속성 및 종류	설 명
마우스 이벤트 속성	event.state	마우스 입력 modifier의 상태, mask 비트 연산으로 체크 확인: 0x0100(Button1), 0x400(Button 3)
	event.num	마우스 버튼 메시지에서의 버튼 번호
	event.widget	이벤트가 발생한 위젯
	event.x, event.y	마우스 이벤트에서 마우스의 위치 (위젯의 upper-left가 원점)
	event.x_root, event.y_root	마우스 이벤트에서 마우스의 위치 (스크린의 upper-left가 원점)
마우스 이벤트 종류	<Button-1>	마우스 왼쪽 버튼을 누를 때
	<Button-2>	마우스 휠 버튼을 누를 때
	<Button-3>	마우스 오른쪽 버튼을 누를 때
	<Button-4>	scroll up
	<Button-5>	scroll down
	<MouseWheel>	mouse wheel 회전

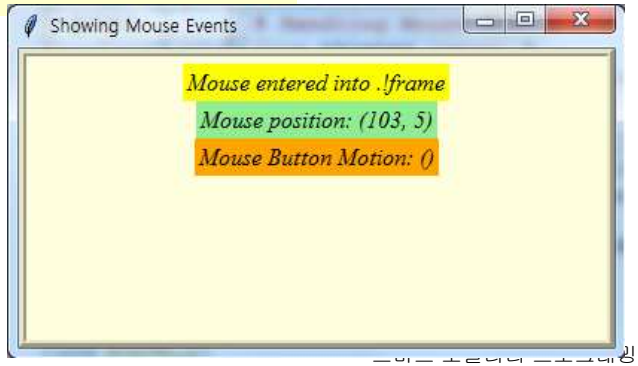
마우스 이벤트

◆ 마우스 이벤트 속성 및 이벤트 종류 (2)

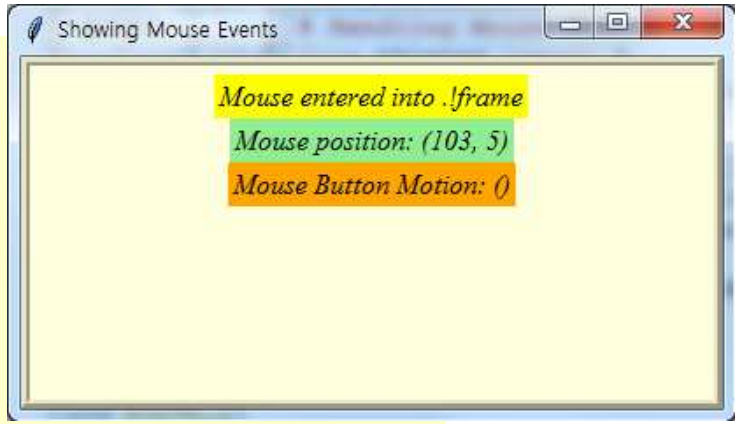
구분	마우스 이벤트 속성 및 종류	설 명
마우스 이벤트 종류	<B1-Motion>	마우스 왼쪽 버튼을 누르고 이동할 때
	<B2-Motion>	마우스 휠 버튼을 누르고 이동할 때
	<B3-Motion>	마우스 오른쪽 버튼을 누르고 이동할 때
	<ButtonRelease-1>	마우스 왼쪽 버튼을 뗄 때
	<ButtonRelease-2>	마우스 휠 버튼을 뗄 때
	<ButtonRelease-3>	마우스 오른쪽 버튼을 뗄 때
	<Double-Button-1>	마우스 왼쪽 버튼을 더블 클릭할 때
	<Double-Button-2>	마우스 휠 버튼을 더블 클릭할 때
	<Double-Button-3>	마우스 오른쪽 버튼을 더블 클릭할 때
	<Enter>	위젯 안으로 마우스 포인터가 들어 왔을 때
	<Leave>	위젯 밖으로 마우스 포인터가 나갔을 때
	<FocusIn>	위젯 안으로 Tab키를 사용하여 들어 왔을 때
	<FocusOut>	위젯 안으로 Tab키를 사용하여 나갔을 때
	<Configure>	위젯 모양이 수정되었을 때

마우스 이벤트 핸들링

```
# Handling Mouse Events (1)
from tkinter import *

def mouse_enter(event):
  label_mouse_enter.configure(text = "Mouse entered into {}".format(event.widget))
  label_mouse_enter.update()

def mouse_leave(event):
  label_mouse_enter.configure(text = "Mouse left from {}".format(event.widget))
  label_mouse_enter.update()

def move_position(event):
  label_pos.configure(text = "Mouse position: ({}, {})".format(event.x, event.y))
  label_pos.update()

def button_1_pressed(event):
  label_mouse_btn.configure(text = "Mouse button {} is pressed".format(event.num))
  label_mouse_btn.update()

def button_1_released(event):
  label_mouse_btn.configure(text = "Mouse button {} is released".format(event.num))
  label_mouse_btn.update()
```



마우스 이벤트 핸들링

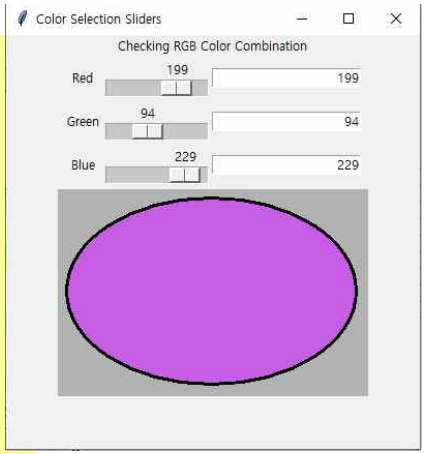



```
# Handling Mouse Events (2)

# main loop
window = Tk()
window.geometry("400x200+100+100")
window.title("Showing Mouse Events")
frame = Frame(window, bg="light yellow",\
 bd=5, padx=5, pady=5, relief=RIDGE)
frame.bind('<Enter>', mouse_enter)
frame.bind('<Leave>', mouse_leave)
frame.bind('<Motion>', move_position)
frame.bind('<Button-1>', button_1_pressed)
frame.bind('<ButtonRelease-1>', button_1_released)
frame.pack(expand=YES, fill=BOTH)

mouse_motion = ""
label_mouse_enter = Label(frame, text = "Mouse motion: ({})".format(mouse_motion))
label_mouse_enter.config(bg='yellow', font=('times', 12, 'italic'))
label_mouse_enter.pack()

pos_x = pos_y = 0
label_pos = Label(frame, text = "Mouse position: ({}, {})".format(pos_x, pos_y))
label_pos.config(bg='lightgreen', font=('times', 12, 'italic'))
label_pos.pack()

btn_motion = ""
label_mouse_btn = Label(frame, text = "Mouse Button Motion: ({})".format(btn_motion))
label_mouse_btn.config(bg='orange', font=('times', 12, 'italic'))
label_mouse_btn.pack()

window.mainloop()
```

Color Chooser

```
#Python ColorChooser

from tkinter import *
import tkinter.colorchooser as cc

class App:
    def __init__(self, master):
        b=Button(master, text='Color', command=self.ask_color).pack()

    def ask_color(self):
        (rgb, hx) = cc.askcolor()
        print("rgb=" + str(rgb) + " hx=" + hx)

window = Tk()
app = App(window)
window.mainloop()
```

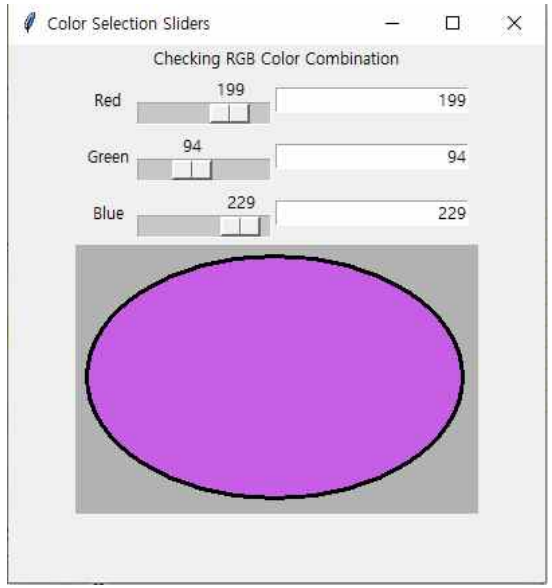
GUI with Slider




```
# Color Control with Slider (1)

from tkinter import *

class App:
    def __init__(self, win):
        frame = Frame(win)
        frame.pack()

        Label(frame, text="Checking RGB Color Combination").grid(row=0, column=0, columnspan=3)
        Label(frame, text='Red').grid(row=1, column=0)
        Label(frame, text='Green').grid(row=2, column=0)
        Label(frame, text='Blue').grid(row=3, column=0)

        scaleRed = Scale(frame, from_=0, to=255, orient=HORIZONTAL, command=self.updateRed)
        scaleRed.grid(row=1, column=1)
        scaleGreen = Scale(frame, from_=0, to=255, orient=HORIZONTAL, command=self.updateGreen)
        scaleGreen.grid(row=2, column=1)
        scaleBlue = Scale(frame, from_=0, to=255, orient=HORIZONTAL, command=self.updateBlue)
        scaleBlue.grid(row=3, column=1)

        self.red_var = IntVar()
        self.green_var = IntVar()
        self.blue_var = IntVar()
        self.red = 0
        self.green = 0
        self.blue = 0
```

GUI with Slider

```
# Color Control with Slider (2)

        Entry(frame, textvariable=self.red_var, justify='right').grid(row=1, column=2)
        Entry(frame, textvariable=self.green_var, justify='right').grid(row=2, column=2)
        Entry(frame, textvariable=self.blue_var, justify='right').grid(row=3, column=2)

        self.canvas = Canvas(win, bg="grey70", width=300, height=200)
        self.canvas.pack()
        self.oval = self.canvas.create_oval(10, 10, 290, 190, fill="white", width=3)
        color = "#%02x%02x%02x"%(self.red, self.green, self.blue)
        self.canvas.itemconfig(self.oval, fill=color)

    def updateRed(self, duty):
        self.red = int(duty)
        self.red_var.set(int(duty))
        color = "#%02x%02x%02x"%(self.red, self.green, self.blue)
        self.canvas.itemconfig(self.oval, fill=color)
    def updateGreen(self, duty):
        self.green = int(duty)
        self.green_var.set(int(duty))
        color = "#%02x%02x%02x"%(self.red, self.green, self.blue)
        self.canvas.itemconfig(self.oval, fill=color)
    def updateBlue(self, duty):
        self.blue = int(duty)
        self.blue_var.set(int(duty))
        color = "#%02x%02x%02x"%(self.red, self.green, self.blue)
        self.canvas.itemconfig(self.oval, fill=color)

if __name__ == "__main__":
    window = Tk()
    window.geometry("400x400")
    window.wm_title('Color Selection Sliders')
    app = App(window)
    window.mainloop()
```

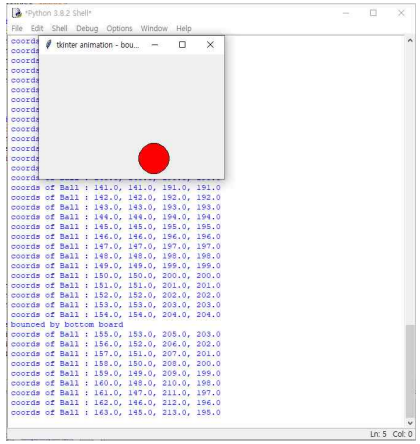


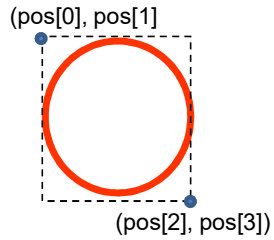
tkinter 기반 애니메이션

Bouncing Ball with tkinter animation, canvas.move()

◆ canvas.move(), update()

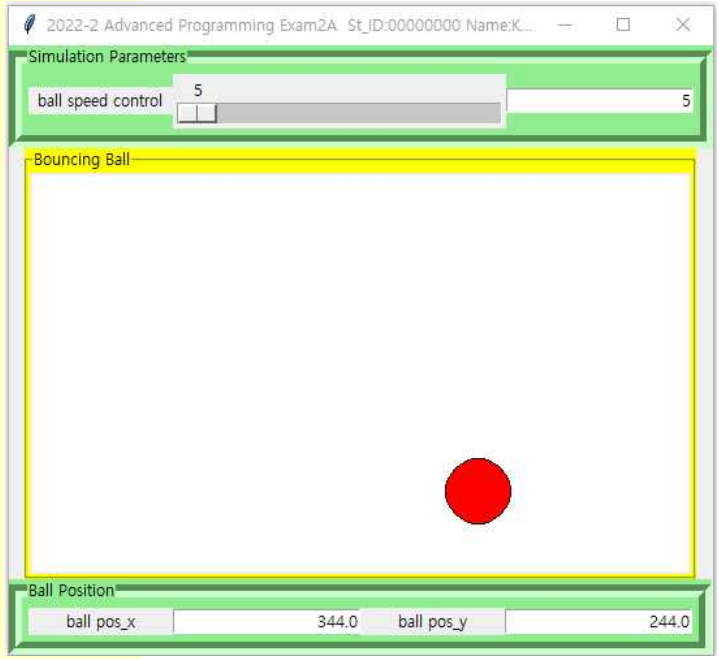
```
# tkinter animation with bouncing ball (1)
from tkinter import *

def init():
    global window, canvas, DX, DY
    window = Tk()
    window.title("tkinter animation - bouncing ball")
    canvas = Canvas(window, width=300, height=200)
    canvas.pack(expand=YES, fill=BOTH)
    canvas.create_oval(0, 0, 50, 50, fill="red", tags="myBall")
    DX = 1 # DX = 5
    DY = 1 # DY = 5

def animate():
    global DX, DY
    canvas.move("myBall", DX, DY)
    pos = canvas.coords("myBall")
    print("coords of Ball : {}, {}, {}, {}".format(pos[0], pos[1], pos[2], pos[3]))
    if pos[0] <= 0 :
        print("bounced by left board ")
        DX *= -1
    if pos[2] >= canvas.winfo_width():
        print("bounced by right board")
        DX *= -1
    if pos[1] <= 0 :
        print("bounced by ceil board")
        DY *= -1
    if pos[3] >= canvas.winfo_height():
        print("bounced by bottom board")
        DY *= -1
    canvas.update_idletasks()
    canvas.update()
    canvas.after(50, animate)

if __name__ == '__main__':
    init()
    animate()
    mainloop()
```



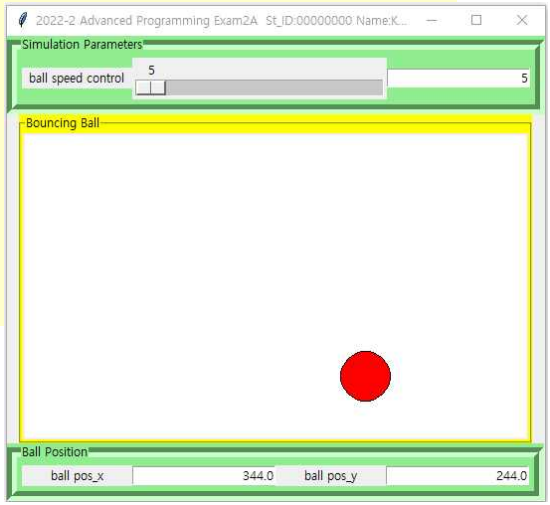


tkinter GUI 기반 애니메이션

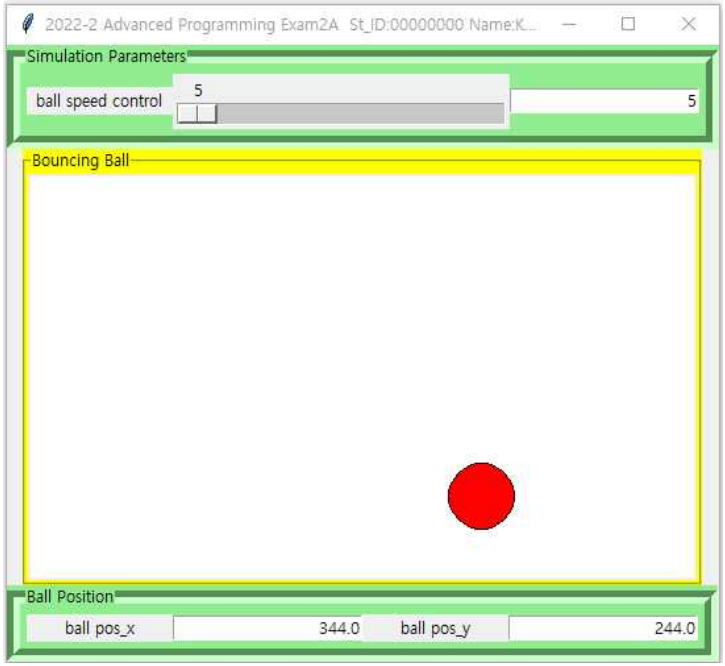
```
# bouncing ball animation with tkinter (1)
from tkinter import *

BALL_SPEED_LIMIT = 100
INITIAL_SPEED = 5
                        CANVAS_WIDTH, CANVAS_HEIGHT = 500, 300
cx, cy = CANVAS_WIDTH / 2, CANVAS_HEIGHT / 2
bd_margin = 1
class BouncingBallSimulator():
    def __init__(self, win):
        self.radius = 50
        self.speed = INITIAL_SPEED
                                              self.dx, self.dy = 1, 1
        self.radius_var = IntVar()
        self.speed_var = IntVar()
                                    self.ball_pos_x, self.ball_pos_y = cx, cy
        self.ball_pos_x_var = IntVar()
        self.ball_pos_y_var = IntVar()

        fr1 = LabelFrame(win, text="Simulation Parameters", bg="light green“,\
           bd=10, padx=5, pady=5, relief=GROOVE)
        fr1.grid(row=0, column=0)
        Label(fr1, text="ball speed control", width=15).grid(row=0, column=0)
        scale_speed = Scale(fr1, length = 250, from_= INITIAL_SPEED,\
           to=BALL_SPEED_LIMIT, orient=HORIZONTAL, command=self.updateSpeed)
        scale_speed.grid(row=0, column=1)
        Entry(fr1, textvariable=self.speed_var, justify='right').grid(row=0, column=2)
        self.speed_var.set(self.speed)
```



```
#  bouncing ball animation with tkinter (2)

        fr2 = LabelFrame(win, text="Bouncing Ball", bg="yellow", bd=2, padx=2, pady=2, relief=GROOVE)
        #fr2 = Frame(win, bg="yellow", relief=GROOVE)
        fr2.grid(row=1, column=0)
        self.canvas = Canvas(fr2, width=CANVAS_WIDTH, height=CANVAS_HEIGHT, bd=2, bg="white")
        self.canvas.grid(row=0, column=0)
        self.canvas.create_oval(cx - self.radius/2, cy - self.radius/2, cx + self.radius/2, cy + self.radius/2,\
            fill="red", tags="myBall")

        fr3 = LabelFrame(win, text="Ball Position", bg="light green", bd=10, padx=5, pady=5, relief=GROOVE)
        fr3.grid(row=2, column=0)
        Label(fr3, text="ball pos_x", width=15).grid(row=0, column=0)
        Entry(fr3, textvariable=self.ball_pos_x_var, justify='right').grid(row=0, column=1)
        Label(fr3, text="ball pos_y", width=15).grid(row=0, column=2)
        Entry(fr3, textvariable=self.ball_pos_y_var, justify='right').grid(row=0, column=3)

    def updateSpeed(self, duty):
        self.speed = int(duty)
        self.speed_var.set(self.speed)
```



```
# Exam2A - bouncing ball animation with tkinter (3)

    def animate(self):
        self.canvas.move("myBall", self.dx, self.dy)
        pos = self.canvas.coords("myBall")
        #print("coords of Ball : {}, {}, {}, {}".format(pos[0], pos[1], pos[2], pos[3]))
        self.ball_pos_x_var.set(pos[0] + self.radius / 2)
        self.ball_pos_y_var.set(pos[1] + self.radius / 2)
        if pos[0] <= (0 + bd_margin):
            print("bounced by left board ")
            self.dx *= -1
        if pos[2] >= CANVAS_WIDTH: #self.canvas.winfo_width():
            print("bounced by right board")
            self.dx *= -1
        if pos[1] <= (0 + bd_margin):
            print("bounced by ceil board")
            self.dy *= -1
        if pos[3] >= CANVAS_HEIGHT: #self.canvas.winfo_height():
            print("bounced by bottom board")
            self.dy *= -1
        self.canvas.update_idletasks()
        self.canvas.update()
        self.canvas.after((BALL_SPEED_LIMIT - self.speed), self.animate)

if __name__ == '__main__':
    window = Tk()
    window.title("Bouncing Ball Animation with tkinter")
    sim_animation = BouncingBallSimulator(window)
    sim_animation.animate()
    mainloop()
```



Homework 9

Homework 9.1

9.1 터틀그래픽을 사용한 다각형 그리기

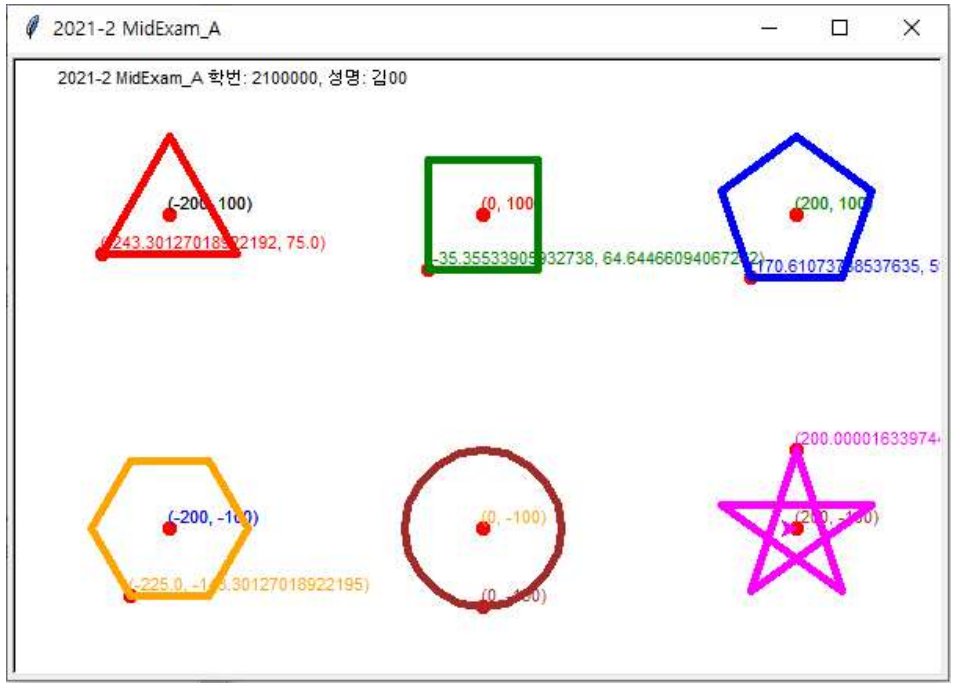
(1) 요구사항

- 지정된 위치에 원을 그리는 drawCircle(t, center_pos, radius, color) 파이썬 함수를 작성하라. 전달되는 인수 t는 파이썬 터틀 객체이며, center_pos는 (center_x, center_y)의 중심점 좌표값의 튜플이고, radius는 원의 반지름이다.
- 지정된 꼭지점 개수의 다각형을 그리는 drawPolygon(t, center_pos, num_vertex, radius, color) 파이썬 함수를 작성하라. 전달되는 인수 t는 파이썬 터틀 객체이며, center_pos는 (center_x, center_y)의 중심점 좌표값의 튜플이고, num_vertex는 다각형의 꼭지점 개수, radius는 다각형의 외접원의 반지름이다.
- 지정된 위치에 별 (star)를 그리는 drawStar(t, center_pos, radius, color) 파이썬 함수를 작성하라. 전달되는 인수 t는 파이썬 터틀 객체이며, center_pos는 (center_x, center_y)의 중심점 좌표값의 튜플이고, radius는 별에 외접하는 원의 반지름이다.
- 각 도형의 지정된 중심점 (center_x, center_y)에는 굵기 5의 붉은색 점을 표시하고, 중심 좌표를 표시하라.
- 다각형의 아랫면이 수평이 되도록 하며, 별의 중앙 꼭지점이 위로 향하게 할 것.
- 각 도형의 시작점 (start_x, start_y)을 math 모듈의 삼각함수 메소드를 사용하여 계산하고, 시작 위치에는 굵기 5의 붉은색 점을 표시하고, 좌표를 표시하라.
- 터틀그래픽의 커서의 마지막 위치는 해당 도형의 중심점이 되도록 이동할 것.
- main() 함수에서는 6개 다각형의 정보 (center_x, center_y, shape, radius, color)를 튜플 리스트로 구성하고, 이 정보에 따라 drawPolygon()을 호출하여 지정된 다각형을 그리도록 할 것.
- main() 함수는 __name__ 변수가 "__main__"일 때 실행될 수 있도록 구성할 것.
- 터틀그래픽 출력 좌측 윗부분에 본인 학번과 출력 할 것.

(2) main()

```
L_drawings =₩
[(-200, 100, "triangle", 50, "red"), (0, 100, "square", 50, "green"), (200, 100, "pentagon", 50, "blue"),₩
(-200, -100, "hexagon", 50, "orange"), (0, -100, "circle", 50, "brown"), (200, -100, "star", 50, "magenta")]
```

(3) 그래픽 출력 (예시)



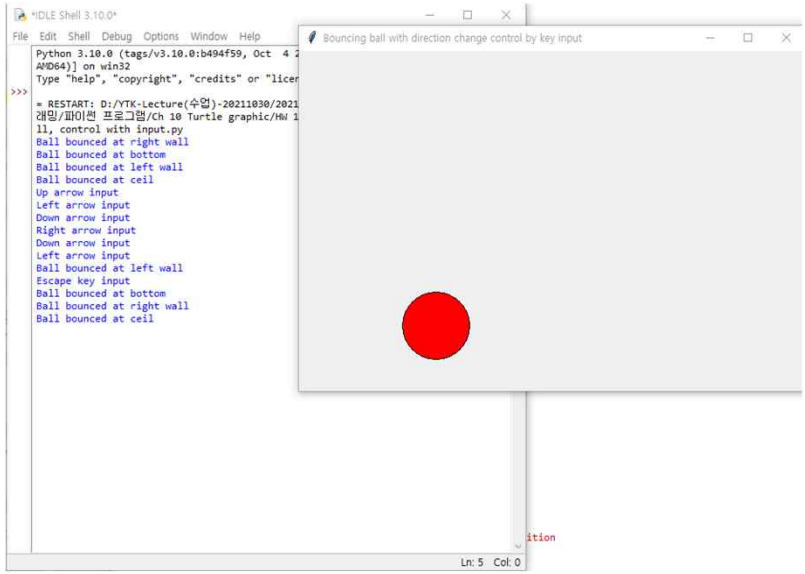
Homework 9.2

9.2 터틀 그래픽 키보드 이벤트 기반의 Bouncing Ball 이동 제어

(1) 요구사항

- 당구대 위에서 공이 벽에 반사되며, 돌아다니는 시뮬레이션 하는 프로그램을 파이썬 터틀 그래픽의 애니메이션 기능을 사용하여 구현하라.
- 당구대의 크기는 (500 x 600)로 설정하고, 당구대 테두리를 표시할 것. 공의 크기는 radius=50, 공의 초기 위치는 당구대 좌측 중간, 공의 초기 이동 방향은 우측/하향 45도로 설정할 것.
- 공의 표면이 당구대 테두리로 들어가지 않게 할 것.
- Bouncing Ball의 이동을 키보드 입력으로 방향 전환
- Ball이 초기에 지정된 방향으로 일정 속도로 진행하게 하고, 벽에 반사되며 이동하는 중에 키보드의 화살표 방향키(up, down, left, right)와 Esc (대각선 방향 이동) 입력에 의한 키 보드 이벤트가 발생하면 이동 방향을 전환을 하여 계속 움직이는 애니메이션 프로그램을 작성하라.

(2) 실행 예시



Homework 9.3

9.3 학생 정보 입력 GUI

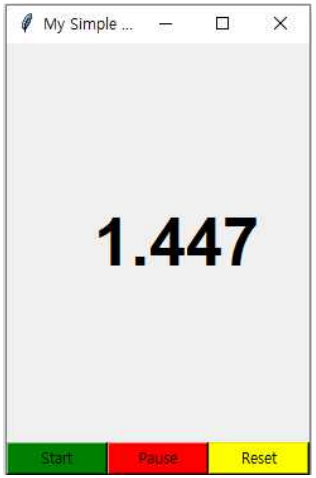
- 학생의 이름과 국어, 영어, 수학 과목별 성적을 입력 받아 반환하여 주는 tkinterGUI 프로그램을 작성하라.
- 이 tkinterGUI 프로그램의 title은 "Input Dialog with Label and Entry Objects"로 설정하라.
- GUI에 포함되는 객체들은격자 배치 (grid layout)를 사용하여 아래 예시와 같이 구성하라.
- "Fetch"버튼을 누르면, 각 입력 엔트리의 값을 읽어 출력하여 주며, "Quit" 버튼을 누르면 GUI 프로그램을 종료하게 한다.
- 실행 예:


```
Input data:
Name:Park
Korean:95
English:90
Math:98
```

Homework 9.4

9.4 Stop Watch

- Start, Pause, Reset 메뉴를 가지는 스톱워치를 tkinter GUI 프로그램으로 구현하라.
- 이 스톱 워치는 소수점 이하 3자리를 사용하여 milli-second 단위 표시의 경과 시간을 알려 준다.
  Reset 버튼은 전체 시간을 0초로 리셋시키며, Start 버튼은 경과된 시간을 측정하도록 한다.
- Pause 버튼은 경과시간 측정을 일시 정지시키며, pause 상태에서 다시 Start 버튼을 누르면 앞서 측정되었던 경과 시간에 추가하여 경과시간을 측정하여 준다.
- 실행 예:



Homework 9.5

9.5 Bouncing Ball with tkinter

- 600 x 400 픽셀 크기의 테이블 위에서 공이 벽면에 반사되며 이동하는 시뮬레이션 기능을 파이썬 tkinter 그래픽 프로그램으로 구현하라. 공의 크기는 지름 80 픽셀로 하며, 공은 색깔은 붉은색으로 할 것.
- 공의 최초 시작 위치는 테이블의 좌측 상단에서 우측 하단 45° 방향으로 이동하게 하며, 벽면에서 반사되게 할 것.
- 공이 이동 중에 키보드로 부터 화살표 키 입력을 받으면, 수평 또는 수직 방향으로 이동하도록 할 것.
- Esc (escape) 키를 입력 받으면 45 ° 방향으로 이동하게 할 것.
- 키 입력을 쉽게 파악할 수 있도록 Python shell에 출력할 것.
- 예) Animation 실행 예: