

5. 반지름의 길이가 R인 원에 내접하는 정N각형을 그리는 함수를 작성하고 이를 시험하는 프로그램을 작성하라. 단 정N각형은 중심이 화면의 중앙에 있어야 하고, 항상 동쪽끝에서 그리기 시작한다. (R과 N은 키보드에서 입력받는 정수. N은 3 이상임)
6. 터틀 그래픽스와 재귀 호출을 이용해 Koch curve 그리기 프로그램을 작성하라.
7. Koch curve를 이용해 삼각형의 각 선분에 대해 재귀 호출을 반복해 그림 9.4와 같은 Koch 눈송이를 그리는 프로그램을 작성하라.



그림 9.4: Koch 눈송이

9.2 tkinter를 이용한 윈도우 구성

앞의 터틀 모듈을 이용한 그림 그리기는 사용하기 쉽지만, 속도가 느리고 다양한 입력의 처리가 가능하지 않다는 단점이 있다. 최근 대다수의 프로그램은 윈도우상에서 마우스를 이용해 메뉴, 버튼 등을 사용하는 그래픽 사용자 인터페이스(GUI: graphical user interface)를 이용해 구현된다. 이 절에서는 tkinter를 이용한 GUI 프로그래밍 방법에 대해 공부한다.

tkinter는 그래픽 사용자 인터페이스를 만들 때 많이 사용하는 모듈로 유닉스 계열에서 사용되던 Tcl/Tk를 파이썬에서 사용할 수 있게 객체 지향으로 확장한 것이다. Tkinter는 다른 GUI 프레임워크나 툴킷에 비해 지원되는 위젯들이 부족하고 사용자 인터페이스(UI)도 예쁘지 않다는 단점이 있지만, 파이썬 설치할 때 기본적으로 설치되는 모듈이기 때문에 쉽고 간단한 GUI 프로그램을 만들 때 활용된다.

<https://docs.python.org/3/library/tkinter.html>에 보면 tkinter에 대해 설명하고 있다. 명령어 라인에서 `python -m tkinter`라고 실행해 보자. tkinter가 시스템에

올바르게 설치되었으면 간단한 Tk 인터페이스를 보여주는 창이 열리고 설치된 Tcl / Tk 버전도 표시된다.

기본적인 윈도우 만들기

가장 기본적인 윈도우를 만드는 방법을 프로그램 9.2에 나타냈다.

프로그램 9.2: 윈도우 구성

```
from tkinter import *

window = Tk()

# 이 부분에서 필요한 화면을 구성하고 여러 처리를 함

window.mainloop()
```

1행은 파이썬에서 GUI 관련 모듈을 제공해주는 표준 윈도우 라이브러리 tkinter를 포함해 사용하겠다는 의미이며, 3행의 Tk()는 기본이 되는 윈도우(루트 윈도우(Root Window) 또는 베이스 윈도우(Base Window))를 반환받아 이 행이 실행되면 윈도우 창이 출력된다. 5행의 주석 처리된 부분에 앞으로 여러가지 문장들이 추가되어 프로그램이 확장된다. 7행은 3행에서 만들어진 window창에서 키보드 누르기, 마우스 클릭 등의 이벤트가 발생하면 이를 처리하는 데 필요한 이벤트 처리 루프이다. 프로그램 9.2를 실행하면 그림 9.5와 같이 제목이 tk인 비어있는 간단한 윈도우를 볼 수 있다.

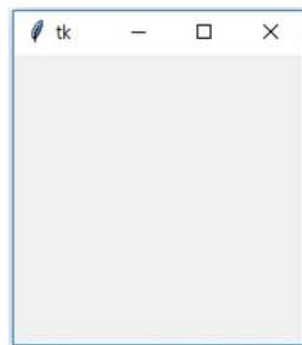


그림 9.5: 기본 윈도우 구성

윈도우의 창 크기 조절

앞의 프로그램을 윈도우 창에 원하는 제목을 표시하고, 원하는 크기를 지정해 창을 만들고, 창의 크기를 바꿀 수 있도록 프로그램 9.3과 같이 수정해 보자.

4행에 title() 메소드를 이용해 윈도우 창에 제목을 표시했고, 5행에 geometry()로 창의 최초 크기를 600×400으로 지정하고, 6행에 resizable()로 창의 가로와 세로의 크기를 변경할 수 있도록(TRUE) 설정했다. FALSE이면 창의 크기변경이 불가능하다.

프로그램 9.3: 크기조절이 가능한 윈도우

```
from tkinter import *

window = Tk()
window.title("크기조절이 가능한 윈도우")
window.geometry("600x400")
window.resizable(width=TRUE, height=TRUE)

window.mainloop()
```

프로그램 9.3을 작성해 실행하면 그림 9.6과 같이 윈도우의 제목이 달라진 것을 알 수 있다. 마우스를 이용해 윈도우의 크기를 변경해 보자.

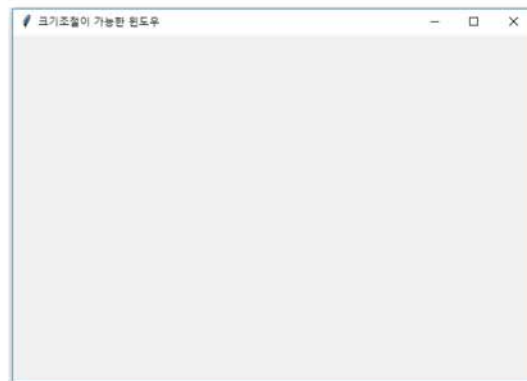


그림 9.6: 크기조절이 가능한 윈도우

9.3 위젯과 기본 활용법

윈도우 창에 나오는 문자, 버튼, 체크박스, 라디오버튼 등을 위젯(widget)이라 하는 데, 위젯을 이용한 프로그램을 작성하려면 각 위젯의 특성을 이해하고 이를 이용할 줄 알아야 한다.

tkinter에서 사용할 수 있는 여러 종류의 위젯을 그림 9.7에 표시하였다. (출전 : <http://pythoncard.sourceforge.net/samples/widgets.html>)

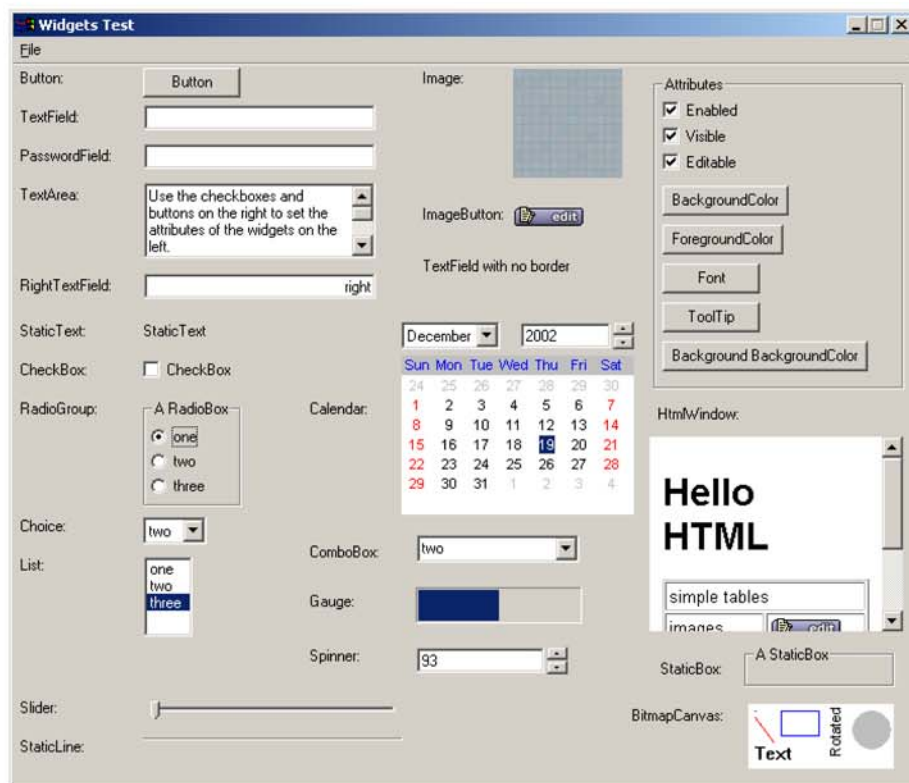


그림 9.7: 여러 종류의 위젯

위젯의 분류

위젯은 단순 위젯과 컨테이너 위젯으로 나누는 데, 컨테이너 위젯은 다른 단순한 위젯들 안에 넣을 수 있는 위젯이다.

단순 위젯에는 Button, Canvas, CheckButton, Entry, Label, Message 등이 있고, 컨테이너 위젯에는 Frame, Toplevel, Label Frame, PanedWindow 등이 있다.

tkinter에서는 다음과 같은 위젯들이 제공된다.

- Label : 텍스트나 이미지를 출력
- Button : 명령을 실행하기 위해 사용되는 일반 버튼
- Checkbutton : 버튼을 클릭하여 여러 값 중 하나를 선택
- Radiobutton : 라디오 버튼을 클릭하여 연결된 변수에 값을 설정하고 모든 다른 라디오 버튼과 연결된 동일 변수값을 삭제
- Menu : 폴다운 메뉴와 팝업 메뉴를 구현하는 데 사용되는 메뉴

배치관리자

위젯을 원하는 위치에 배치하는 관리자는 다음과 같은 3종류가 있다.

- 압축 배치 관리자(pack geometry manager) : 위젯들을 부모 위젯 안에 압축해 배치
- 격자 배치 관리자(grid geometry manager) : 표 형태로 배치
- 절대 배치 관리자(place geometry manager) : 주어진 절대 위치에 위젯을 배치

세 배치 관리자의 차이점을 알기 위해, 뒤이어 나오는 Label() 함수를 이용해 세 레이블을 표시하는 프로그램을 다음과 같이 작성했다. 먼저 압축 배치 관리자는 pack() 함수를 사용한다.

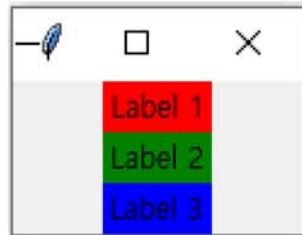
```
from tkinter import *
window = Tk()

label1 = Label(window, text="Label 1", bg='red')
label2 = Label(window, text="Label 2", bg='green')
label3 = Label(window, text="Label 3", bg='blue')

label1.pack()
label2.pack()
label3.pack()
```

```
window.mainloop()
```

실행결과를 다음과 같다.



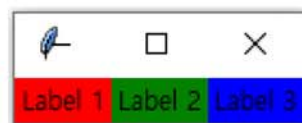
격자 배치 관리자는 `grid()` 함수를 사용한다. `row`, `column` 값을 적절히 바꿔 원하는 곳에 레이블이 배치되는 지 확인하자.

```
from tkinter import *
window = Tk()

label1 = Label(window, text="Label 1", bg='red')
label2 = Label(window, text="Label 2", bg='green')
label3 = Label(window, text="Label 3", bg='blue')

label1.grid(row=0,column=0)
label2.grid(row=0,column=1)
label3.grid(row=0,column=2)

window.mainloop()
```



절대 배치 관리자는 위젯을 고정 위치에 배치하기 위해 `place()` 함수를 사용한다. 형식은 '`place(x=X좌표, y=Y좌표, width=폭, height=높이)`' 이고, `width`와 `height`를 생략하면 위젯의 원래 크기로 표현한다. 값을 적절히 바꿔 보라.

```
from tkinter import *
window = Tk()

label1 = Label(window, text="Label 1", bg='red')
```

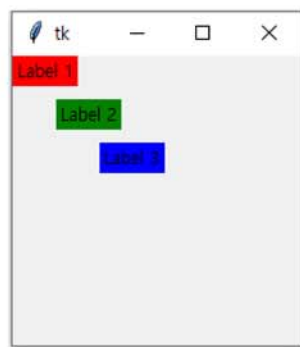
```

label2 = Label(window, text="Label 2", bg='green')
label3 = Label(window, text="Label 3", bg='blue')

label1.place(x=0, y=0)
label2.place(x=30, y=30)
label3.place(x=60, y=60)

window.mainloop()

```



세 배치관리자의 특성을 알기 위해 위의 세 프로그램을 꼭 실행해 보고 순서 또는 숫자를 바꾸어 실행해 세 방법의 차이점을 알 수 있도록 하자.

이 책에서는 pack() 함수를 사용하는 압축 배치 관리자를 주로 사용한다.

레이블

Label(레이블, 라벨)은 문자 또는 이미지를 표현할 수 있는 위젯이다. 사용 형식은 다음과 같다. 옵션에서 위젯을 다양하게 설정할 수 있으며, 이 위젯들은 pack() 함수 등을 호출해 화면에 표시한다.

```
Label(부모윈도우, 옵션..)
```

프로그램 9.4: 레이블의 사용

```

from tkinter import *
window = Tk()

label1 = Label(window, text="Label 1")#, anchor=S)
label2 = Label(window, text="Label 2", font=("Courier", 20), fg="blue")
label3ORI = Label(window, text="Label 3", bg="yellow", width=20, height=2)

```

```
label3 = Label(window, text="Wkslaksllkasaj",bg="yellow", anchor=N, width=1,height=1)

label1.pack()
label2.pack()
label3.pack()

window.mainloop()
```

4-6행은 Label() 함수로 레이블을 만들고, 8-10행은 pack() 함수를 이용해 해당 레이블을 화면에 표시하는 것이다.

옵션 중 text는 표시할 글자의 내용 자체를, font는 글꼴과 크기 지정, fg는 foreground의 약자로 글자 색을, bg는 background의 약자로 배경색을 지정한다. width와 height는 위젯의 폭과 높이를 설정하며 anchor는 위젯이 어느 위치에 자리 잡을지를 지정한다. anchor에 사용할 수 있는 값은 N, NE, E, SE, S, SW, W, NW, CENTER 이고, 기본값은 CENTER이다. 색은 색상의 이름이나 #RRGGBB 문자열을 사용하여 빨강, 초록 및 파랑(RGB) 색상 요소를 명시적으로 지정할 수 있다.

레이블에 이미지 파일 사용

레이블에 글자 대신 이미지를 사용할 수도 있는데, 사용 방법은 다음 프로그램과 같이 PhotoImage() 메소드를 이용해 이미지 파일을 읽는다. 지원되는 이미지 포맷은 gif, png, pgm이고, jpg, bmp 포맷 등은 지원되지 않는다. 적당한 gif 이미지 파일을 구해 test.gif 로 이름을 바꾼 후 다음 프로그램을 실행하면 된다.

프로그램 9.5: 레이블에 이미지 파일 사용

```
from tkinter import *
window = Tk()

photo = PhotoImage(file="test.gif")
label = Label(window, image=photo)

label.pack()

window.mainloop()
```

버튼(Button)

마우스로 클릭하면 놀리는 효과와 함께 지정한 작업이 실행되게 하는 위젯으로 레이블과 다르게 버튼은 눌렀을 때 정해진 작업을 하도록 작업을 해야 한다.

사용 형식은 다음과 같다.

Button(부모윈도우, 옵션..)

다음 프로그램은 버튼을 누르면 (4행의 command 옵션에 quit 명령을 주었으므로) 파이썬 IDLE이 종료되는 프로그램이다.

프로그램 9.6: 버튼 사용 프로그램

```
from tkinter import *
window = Tk()

button = Button(window, text="Program Quit", fg="blue", command=quit)
button.pack()

window.mainloop()
```

다음은 버튼을 누르면 사용자가 미리 정의한 해당 함수가 실행되도록 변경된 프로그램이다. 두 개의 버튼 중 어느 것이든 누르면, 해당되는 함수가 호출되어 일을 수행한다.

프로그램 9.7: 2개의 버튼 사용 프로그램

```
from tkinter import *

def btnFunc1():
    print('Button 1 pressed.')

def btnFunc2():
    print('Button 2 pressed.')

window = Tk()

button1 = Button(window, text="Button1", fg="blue", command=btnFunc1)
button2 = Button(window, text="Button2", fg="red", command=btnFunc2)
button1.pack()
button2.pack()

window.mainloop()
```

버튼에도 이미지를 넣을 수 있다. 다음은 이미지 버튼을 누르면 간단한 메시지가 출력되는 프로그램이다. 4-5행에 간단한 메시지를 출력하는 btnFunc() 함수를 정의하였다.

프로그램 9.8: 이미지 버튼 프로그램

```
from tkinter import *

# function
def btnFunc() :
    print("이미지를 눌렀습니다.")

window = Tk()

photo = PhotoImage(file="test.gif")
button1 = Button(window, image=photo, command=btnFunc)

button1.pack()

window.mainloop()
```

체크버튼

체크버튼(checkbutton)은 사용자가 클릭해 체크된 상태와 체크되지 않은 상태 중의 하나로 만들 수 있는 위젯이다. 즉 체크 박스의 켜기/끄기를 위해 사용되는 위젯이다.

사용 형식은 다음과 같다.

```
Checkbutton(부모 윈도우, 옵션...)
```

다음은 체크버튼을 켜거나 끄면, 그에 따른 메시지를 화면에 출력하는 프로그램이다.

프로그램 9.9: 체크 버튼 프로그램

```
from tkinter import *

window = Tk()

def chkFunc() :
    if chk.get() == 0 :
        print("check box is off")
    else :
        print("check box is on")

chk = IntVar()
```

```

check = Checkbutton(window, text="Click", variable=chk, command=chkFunc)
check.pack()

window.mainloop()

```

10행은 IntVar() 함수를 사용해 정수형 chk 변수를 준비한다. 11행은 Checkbutton() 옵션 중에서 variable에 chk 변수를 사용하는데, 체크버튼을 켜면 chk에 1이, 끄면 chk에 0이 대입된다. 또한 체크버튼을 켜거나 끄면 chkFunc() 함수가 실행된다. 4행-8행의 chkFunc() 함수는 chk.get() 함수를 이용해 체크버튼에 설정된 값을 가져와 해당 메시지를 출력한다. 프로그램을 실행해 체크버튼을 몇 번 켜고 꺼보자.

라디오 버튼

라디오 버튼(radio button)은 체크 박스와 비슷하지만 하나의 그룹 안에서는 하나만을 선택할 수 있다는 점이 다르다. 즉 여러 개 중 하나를 선택하는 데 사용되는 위젯이다. 사용 형식은 다음과 같다.

```
Radiobutton(부모 윈도우, 옵션...)
```

프로그램 9.10: 라디오 버튼 프로그램

```

from tkinter import *
win = Tk()

def myFunc() :
    if var.get() == 1 :
        label.configure(text="cat")
    elif var.get() == 2 :
        label.configure(text="dog")
    else :
        label.configure(text="bird")

var = IntVar()
label = Label(win, text="Favorite pet", fg="blue")
b1 = Radiobutton(win, text="cat", variable=var, value=1, command=myFunc)
b2 = Radiobutton(win, text="dog", variable=var, value=2, command=myFunc)
b3 = Radiobutton(win, text="bird", variable=var, value=3, command=myFunc)

```

```
label.pack()
b1.pack()
b2.pack()
b3.pack()

win.mainloop()
```

라디오버튼을 선택하면 var에는 value에 할당된 값이 들어간다. 즉 b1 버튼을 선택하면 var에는 1이 대입된다. 4행-10행의 myFunc() 함수는 var 변수의 값에 의해 레이블의 텍스트를 변경한다. 프로그램을 실행해 라디오 버튼을 눌러 변화를 살펴보자.

엔트리 위젯

엔트리 위젯은 사용자로부터 입력을 받을 때 필요하다.

다음 프로그램은 임의의 마일 수를 입력한 후 변환 버튼을 누르면 킬로미터로 변환된 결과가 화면에 표시되는 프로그램이다. 앞에서 나온 레이블, 버튼과 함께 엔트리 위젯이 사용되는 예이다.

프로그램 9.11: 엔트리 위젯 사용 프로그램

```
#entryComp.py

from tkinter import *

def comp():
    mile = float(entry1.get())
    km = 1.609 * mile
    entry1.delete(0, 50)
    entry1.insert(0, str(km))

window = Tk()

label1 = Label(window, text="Convert mile to km")
entry1 = Entry(window)
button1 = Button(window, text="Convert", command=comp)

label1.pack(side=TOP)
entry1.pack(side=TOP)
button1.pack(side=TOP)
```

```
window.mainloop()
```

12행의 엔트리 위젯에서 마일 수를 입력받은 후 13행의 Convert 버튼을 누르면 3행의 comp() 함수가 실행된다. 4행에서 입력된 수를 get() 메소드로 가져와 실수로 변환 후, 5행의 변환식으로 킬로미터를 계산한다. 6행에서 앞에서 입력받은 엔트리의 내용을 지운 후, 7행의 insert() 메소드로 엔트리에 출력한다.

9.4 위젯 배치와 크기 조절

윈도우 창에 위젯이 여러 개 표시될 때는 위젯의 위치와 크기 조절을 잘 해야 한다. 이 절에서는 pack() 함수의 옵션을 이용해 수평 정렬, 수직 정렬 등을 하는 방법을 알아 보자.

수평으로 정렬

파이썬에서 pack() 함수를 사용해 위젯을 출력할 때 pack() 함수의 옵션 중 수평으로 정렬하는 방법으로 'side = LEFT, RIGHT' 를 사용한다.

프로그램 9.12: 위젯을 수평으로 정렬

```
from tkinter import *
window = Tk()

b1 = Button(window, text="Button1")
b2 = Button(window, text="Button2")
b3 = Button(window, text="Button3")

b1.pack(side=LEFT)
b2.pack(side=LEFT)
b3.pack(side=LEFT)

window.mainloop()
```

8행-10행의 side=LEFT 옵션은 왼쪽부터 채우라는 의미이다. side=RIGHT 옵션을 사용해 재실행해 어떻게 달라지는 지 확인하자.

수직으로 정렬

파이썬에서 `pack()` 함수를 사용해 위젯을 출력할 때 `pack()` 함수의 옵션 중 수직으로 정렬하는 방법으로 `'side=TOP, BOTTOM'` 를 사용한다.

프로그램 9.12의 8-10행을 `'side=TOP'` 과 `'side=BOTTOM'` 으로 변경한 후 각각의 실행 결과를 확인하자.(?는 1,2,3을 뜻함)

```
button?.pack(side=TOP)
```

위젯의 폭 맞추기

위젯의 폭을 윈도우 창의 폭에 맞추기 위해 프로그램 9.12의 8-10행을 다음과 같이 변경해 실행해 보자.

```
button?.pack(side=TOP, fill=X)
```

위젯 사이에 여백 주기

위젯 사이에 여백을 주려면 `'padx=픽셀값'` 또는 `'pady=픽셀값'` 방식을 사용한다. 프로그램 9.12의 8-10행을 다음과 같이 변경해 실행해 보자.

```
button?.pack(side=TOP, fill=X, padx=10, pady=10)
```

위젯 내부에 여백 주기

`'ipadx=픽셀값'` 또는 `'ipady=픽셀값'` 방식을 사용한다.

프로그램 9.12의 8-10행을 다음과 같이 변경해 실행해 결과가 어떻게 다른 지 확인하자.

```
button?.pack(side=TOP, fill=X, ipadx=10, ipady=10)
```

9.5 이벤트의 처리

윈도우 프로그램에서 키보드 또는 마우스를 누르면 해당되는 이벤트(event)가 발생되었다고 한다. `mainloop()` 함수는 이러한 이벤트가 발생하기를 기다리는 함수이고, 이

벤트가 발생되면 발생한 이벤트의 종류에 따라 각 해당 응용 프로그램이 이를 적절히 처리하도록 프로그램이 작성되어야 한다.

키보드 이벤트와 마우스 이벤트를 처리하는 프로그램을 만들어 보자.

키보드 이벤트 처리

키보드 이벤트는 위젯에서 키보드가 눌리면 발생하는 이벤트이며 대표적인 것은 <Key> 이벤트이다.

다음 프로그램은 엔터키와 위 화살표를 누르면 해당 이벤트가 발생한 것을 알려주는 간단한 프로그램이다.

프로그램 9.13: 엔터키와 위 화살표키 이벤트 처리

```
from tkinter import *

def callback1(event):
    print('You pressed the enter key.')

def callback2(event):
    print('You pressed the up arrow.')

window = Tk()

window.bind('<Return>', callback1)
window.bind('<Up>', callback2)

window.mainloop()
```

다음은 뒤에 나오는 메시지를 이용해 어떤 키가 눌렸는 지 확인할 수 있는 프로그램이다.

프로그램 9.14: 키보드 이벤트 처리

```
from tkinter import *
from tkinter import messagebox

def keyEvent(event) :
    messagebox.showinfo("KeyEvent", "Key pressed: "+chr(event.keycode))

window = Tk()
```

```

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

window.mainloop()

```

keyEvent() 함수는 키보드가 눌릴 때 작동하는 함수를 정의하는 데 event.keycode 에는 눌린 키의 숫자값이 들어있으므로 chr() 함수를 사용하면 문자로 변환되어, 눌린 키가 메시지 창에 나온다(영문자 A- Z, 0 - 9).

마우스 이벤트 기본 처리

마우스를 사용할 때 여러 종류의 이벤트가 발생하는 데, 버튼을 클릭할 때, 더블 클릭할 때, 버튼에서 떼릴 때, 버튼을 누른 채 드래그할 때 등이다.

이런 이벤트를 다음과 같이 요약할 수 있다.(아래에서 [n]은 마우스 왼쪽, 가운데, 오른쪽 버튼이 1, 2, 3 이다.)

- 마우스 클릭할 때는 <Button-[n]>(또는 모든 버튼 공통은 <Button>)
- 마우스 더블 클릭할 때는 <Double-Button-[n]>(또는 모든 버튼 공통은 <Double-Button>)
- 마우스 떼었을 때는 <ButtonRelease-[n]>(또는 모든 버튼 공통은 <ButtonRelease>)
- 마우스 드래그 할 때는 <B[n]-Motion> 이다.

마우스 이벤트가 처리되는 기본 형식은 다음과 같다.

```

def 이벤트처리함수(event) :
    # 이벤트 발생시 처리할 내용
    #
    위젯.bind("마우스이벤트", 이벤트처리함수)

```

마우스 버튼 클릭 시 처리하는 간단한 코드가 다음과 같다.

프로그램 9.15: 마우스 버튼 클릭 시 처리 프로그램

```

from tkinter import *

def clickLeft(event) :
    print("left button clicked")
def clickMiddle(event) :
    print("middle button clicked")
def clickRight(event) :
    print("right button clicked")

window = Tk()

window.bind("<Button-1>", clickLeft)
window.bind("<Button-2>", clickMiddle)
window.bind("<Button-3>", clickRight)

window.mainloop()

```

3-8행은 마우스 이벤트 시에 작동할 함수를 정의한다. 12-14행은 window.bind() 함수에 해당 마우스 버튼 클릭 시에 나오는 이벤트에 따라 3-8행의 함수 이름을 지정한다. 윈도우 창을 의미하는 window 변수를 사용했으므로, 윈도우 창 아무 곳이나 클릭해도 bind() 함수가 동작한다.

이벤트 매개변수의 활용

이벤트 매개변수에는 마우스 클릭시의 좌표, 마우스 버튼의 번호 등이 포함된다.

다음은 마우스를 클릭할 때마다, 어떤 마우스 버튼이 어느 좌표에서 클릭되었는지 보여주는 프로그램이다.

프로그램 9.16: 마우스 이벤트 매개변수의 활용

```

from tkinter import *

def clickButton(event) :
    msg = ""
    if event.num == 1 :
        msg += "Left button clicked at ("
    elif event.num == 2 :
        msg += "Middle button clicked at ("

```

```

elif event.num == 3 :
    msg += "Right button clicked at ("
msg += str(event.y) + "," + str(event.x) + ")"
label1.configure(text=msg)

def doubleButton(event) :
    msg = ""
    if event.num == 1 :
        msg += "Left button double-clicked at ("
    elif event.num == 2 :
        msg += "Middle button double-clicked at ("
    elif event.num == 3 :
        msg += "Right button double-clicked at ("
    msg += str(event.y) + "," + str(event.x) + ")"
    label1.configure(text=msg)

window = Tk()

window.geometry("300x300")
label1 = Label(window, text="message area" )
label1.pack(expand=1, anchor=CENTER)
window.bind("<Button>", clickButton)
window.bind("<Double-Button>", doubleButton)

window.mainloop()

```

3-12행은 마우스 클릭 시 실행될 이벤트 함수를 정의한 것이고, 14-23행은 마우스 더블 클릭 시 실행될 이벤트 함수를 정의한다. 30-31행에서 마우스를 클릭 또는 더블 클릭하면 호출하는 함수를 각각 바인드 한다. event.num은 마우스의 버튼을 의미하며 event.x와 event.y는 클릭한 위치의 좌표이다.

9.6 메세지 박스

메세지 박스(MessageBox)는 프로그램에서 주의나 경고 등의 메세지를 알린 다음 없어지는 윈도우의 한 종류로 다음과 같이 import 한 후 사용한다.

```
from tkinter import messagebox
```

프로그램 9.17은 버튼이 눌릴 때 끝낼 것인지 묻는 내용이다.

프로그램 9.17: askquestion() 메세지 박스 사용 프로그램

```

from tkinter import *
from tkinter import messagebox

def myFunc():
    messagebox.askquestion(title= 'Quit?', message= 'want to quit?' )

window = Tk()

button1 = Button(window, text="button", fg="red", command= myFunc)
button1.pack()
window.mainloop()

```

프로그램 9.17을 실행하면 그림 9.8과 같은 결과를 얻는다.

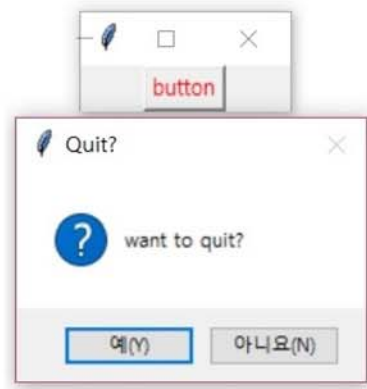


그림 9.8: askquestion() 함수 사용

위에서 사용한 askquestion() 외에 다음과 같은 메세지 박스도 있다. 프로그램 9.17을 수정해 이 두 메세지 박스도 시험해 보자.

```

showinfo(title= 'Info' , message= 'Mesg for you' )
showwarning(title= 'Warning' , message= 'Unsupported format' )

```

프로그램 9.18에 모든 메세지박스의 사용 예를 표시했다. 프로그램을 실행해 각 메세지 박스가 어떤 모양인지 확인해 보자.

프로그램 9.18: 모든 메시지 박스

```

from tkinter import *
from tkinter import messagebox as mb

window = Tk()

mb.showinfo(title='showinfo', message='showinfo')
mb.askquestion(title='askquestion', message='askquestion')
mb.showwarning(title='showwarning', message='showwarning')
mb.askokcancel(title='askokcancel', message='askokcancel')
mb.askretrycancel(title='retrycancel', message='askretrycancel')
mb.asksynccancel(title='yesnocancel', message='asksynccancel')
mb.showerror(title='showerror', message='showerror')

window.mainloop()

```

9.7 대화상자

대화상자에는 간단한 대화상자와 파일 선택 대화상자가 있다.

간단한 대화상자

정수, 실수 또는 문자를 입력받을 때 사용하는 간단한 대화상자가 있다. 각 함수의 이름은 askinteger(), askfloat() 또는 askstring()인데 이들 함수를 사용하려면 다음과 같이 import 한 후 사용한다.

```

from tkinter.simpledialog import *

```

프로그램 9.19: 간단한 대화상자

```

from tkinter import *
from tkinter.simpledialog import *

window = Tk()

val = askinteger("int", "type int number", minvalue=1, maxvalue=6)
#print(val*10)
val = askfloat("float", "type float number")
#print(val*10)
val = askstring("string", "type string")

```

```
#print(val)
window.mainloop()
```

파일 선택 대화상자

파일의 이름을 입력하는 대신 파일을 쉽게 선택해 사용할 수 있게 해주는 대화상자가 파일 선택 대화상자(FileDialog)이다. 많은 프로그램들이 파일을 열거나, 파일로 저장할 때 파일 선택 대화상자를 사용한다. 다음과 같이 import 한 후 사용한다.

```
from tkinter.filedialog import *
```

파일 선택 대화상자는 표 9.1에 제시된 것과 같은 대화상자들이 있다.

표 9.1: 파일 선택 대화상자

대화상자	설명
askopenfilename	파일선택 대화상자
askopenfilenames	하나 이상의 파일을 선택할 수 있는 대화상자
asksaveasfilename	파일 저장 대화상자
askdirectory	디렉토리를 선택하는 대화상자

askopenfilename()과 asksaveasfilename()의 반환값은 선택한 파일의 이름이다. 선택된 이름이 없으면 반환값이 없다. askdirectory() 함수는 선택된 디렉터리의 이름을 반환한다. 사용 예는 다음과 같다.

```
from tkinter.filedialog import askopenfilename
filename = askopenfilename()
```

파일의 확장자가 일정한(예를 들어 GIF) 파일만 입력받으려면 다음과 같이 지정해 사용할 수 있다.

```
askopenfilename(filetypes=(("GIF 파일", "*.gif"), ("모든 파일", "*.*")))
```

PhotoImage와 파일 열기 대화 상자를 이용해 임의의 GIF 이미지를 선택해 화면에 표시하는 프로그램을 작성해 보자.

프로그램 9.20: 파일 열기 대화 상자 이용 이미지 보이기 프로그램

```

from tkinter import *
from tkinter.filedialog import askopenfilename

window = Tk()

window.title("Show image with PhotoImage")
fname = askopenfilename(parent=window, filetypes = (("GIF 파일",
↳ "*.gif"), ("png 파일", "*.png"), ("모든 파일", "*.*")))#filetypes =
↳ (("GIF 파일", "*.gif"), ("모든 파일", "*.*")))
photo = PhotoImage(file=fname)
w = Label(window, image=photo)
w.pack()

window.mainloop()

```

9.8 메뉴

대부분의 GUI 프로그램은 메뉴를 사용해 작성된다.

프로그램 9.21은 [File] 메뉴 아래에 [Menu1]과 [Menu2]의 하위 메뉴가 보이도록 작성되었다.

프로그램 9.21: 메뉴 만들기 프로그램

```

from tkinter import *

window = Tk()
window.title("window with menu")
window.geometry("600x400")
mainMenu = Menu(window)
window.config(menu = mainMenu)

fileMenu = Menu(mainMenu, tearoff=0)
mainMenu.add_cascade(label = "File", menu = fileMenu)
fileMenu.add_command(label = "Menu1")
fileMenu.add_separator()
fileMenu.add_command(label = "Menu2")

window.mainloop()

```

Menu(window) 함수를 이용해 mainMenu를 창에 만들었고, Menu(mainMenu, tearoff=0)로 File 메뉴를 만들었다. (tearoff=0을 없애면 어떻게 되는 지 시험해 보라.) fileMenu에 붙이는 여러 메소드(add_cascade(), add_command())를 이용해 File 메뉴의 서브메뉴 두 개를 만들었다.

프로그램 9.21을 실행하면 그림 9.9와 같은 결과가 나온다.

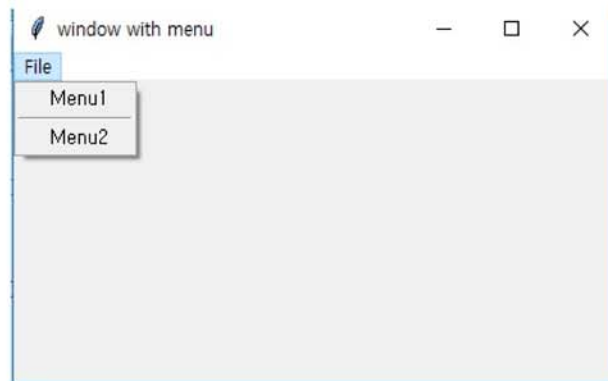


그림 9.9: 메뉴 프로그램의 실행

그림 9.9처럼 실행된 프로그램의 메뉴 버튼을 눌러 선택해도 동작이 실제로 일어나지는 않는다. 메뉴 선택에 따른 동작이 정의되지 않았기 때문이다.

이번에는 메뉴를 선택하면 어떤 동작을 할 수 있도록 코드를 추가해 보기로 하자.

메뉴 선택에 따른 동작

PhotoImage를 이용해 임의의 이미지 파일을 선택하면 그 파일을 열어 화면에 표시하는 프로그램을 작성해 보자.

프로그램 9.22: 이미지 보이기 프로그램

```
from tkinter import *
from tkinter.filedialog import askopenfilename

def imgOpen():
    fname = askopenfilename(parent=window, filetypes = (("GIF 파일",
        ↳ "*.gif"), ("모든 파일", "*.*")))
    print(fname)
    photo = PhotoImage(file=fname)
```

```
w = Label(window, image=photo)
w.photo = photo
w.pack()
window.mainloop()

def ftExit():
    window.quit()
    window.destroy()

window = Tk()
window.title("Show image with PhotoImage")
#window.geometry("400x200")
mainMenu = Menu(window, tearoff=0)
window.config(menu = mainMenu)

fileMenu = Menu(mainMenu)
mainMenu.add_cascade(label = "File", menu = fileMenu)
fileMenu.add_command(label = "Open", command = imgOpen)
fileMenu.add_separator()
fileMenu.add_command(label = "Exit", command = ftExit)

window.mainloop()
```