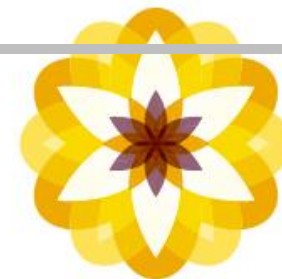
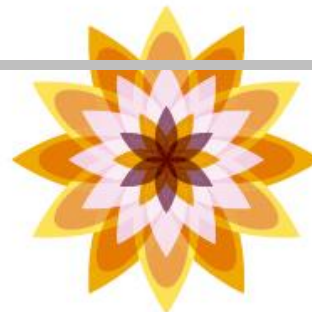


Chapter 07

기본적인 게임 개발 기술



1. 실시간 처리 구현하기

- 게임 소프트웨어는 시간과 함께 처리를 진행합니다.
- 예를 들어, 액션 게임에서는 사용자가 아무것도 하지 않아도 적 캐릭터는 화면 위를 돌아다니며, 배경의 구름이 흐르거나 수면이 움직입니다.
- 제한 시간이 있는 게임이라면 남은 시간이 줄어들기도 합니다.
- 시간 축에 따라 처리가 진행되는 소프트웨어는 실시간 처리를 수행하며 이는 게임 제작에 있어 없어서는 안 될 입력입니다.
- 그렇다면 파이썬으로 실시간 처리를 수행하는 방법을 학습합니다.

1. 실시간 처리 구현하기

■ after() 명령 사용하기

- 파이썬에서는 after() 명령으로 실시간 처리를 수행할 수 있습니다.
- after() 명령 서식은 다음과 같습니다.

after(밀리초 , 실행할 함수명)

- 인수는 '몇 밀리초 후'에 '어떤 함수를 실행하는가'를 결정합니다.
- after() 명령의 인수에 입력하는 함수명 뒤에는 ()를 붙이지 않습니다.
- 예를 들어 다음 코드는 1000밀리초(1초) 후에 count_up이라는 함수를 실행합니다.

after(1000, count_up)

1. 실시간 처리 구현하기

■ after() 명령 사용하기

- 다음 예제에서 숫자를 자동으로 세는 프로그램을 확인하고 실시간 처리 이미지와 연결해 보겠습니다.
- 예제 ex0701_1.py

```
1 import tkinter
2 tmr = 0
3 def count_up( ):
4     global tmr
5     tmr = tmr + 1
6     label['text'] = tmr
7     root.after(1000, count_up)
8
```

tkinter 모듈 импорт
시간 카운트 변수 tmr 선언
실시간 처리 수행 함수 정의
tmr 을 전역 변수로 선언
tmr값 1 증가
라벨에 tmr값 표시
1초 후에 이 함수를 다시 실행

1. 실시간 처리 구현하기

■ after() 명령 사용하기

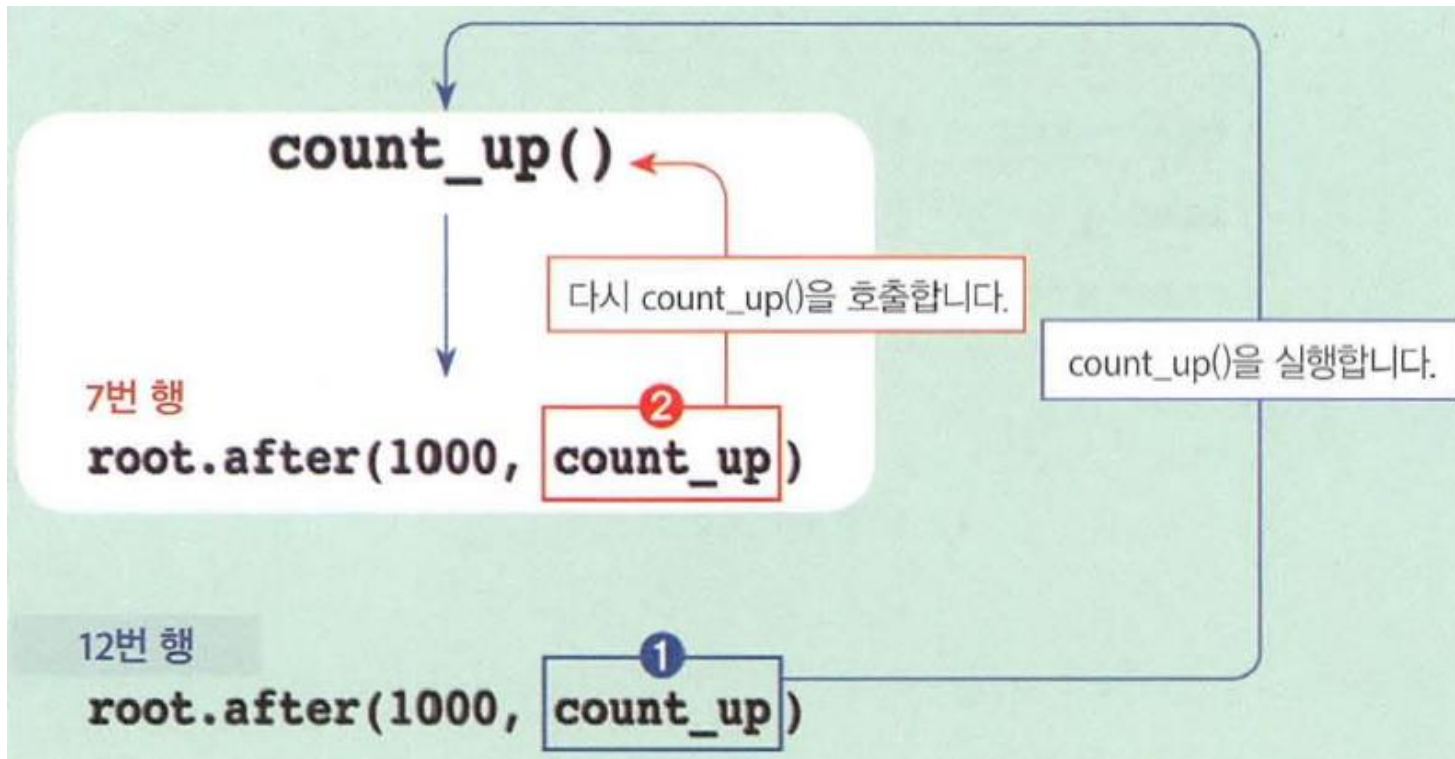
9	root = tkinter.Tk()	윈도우 객체 생성
10	label = tkinter.Label(font=("Times New Roman", 80))	라벨 컴포넌트 생성
11	label.pack()	라벨 컴포넌트 배치
12	root.after(1000, count_up)	1초 후 지정한 함수 호출
13	root.mainloop()	윈도우 표시

- 이 프로그램을 실행하면 윈도우에 표시된 숫자가 1초마다 증가합니다.
- 4번 행의 `global`은 함수 밖에서 정의한 변수값을 함수 안에서 변경할 때 사용하는 명령으로 뒤에서 자세히 설명하겠습니다.
- 3~7번 행에서 `count_up()`이라는 함수를 정의하고 이 함수와 `after()` 명령으로 실시간 처리를 수행합니다.

1. 실시간 처리 구현하기

■ after() 명령 사용하기

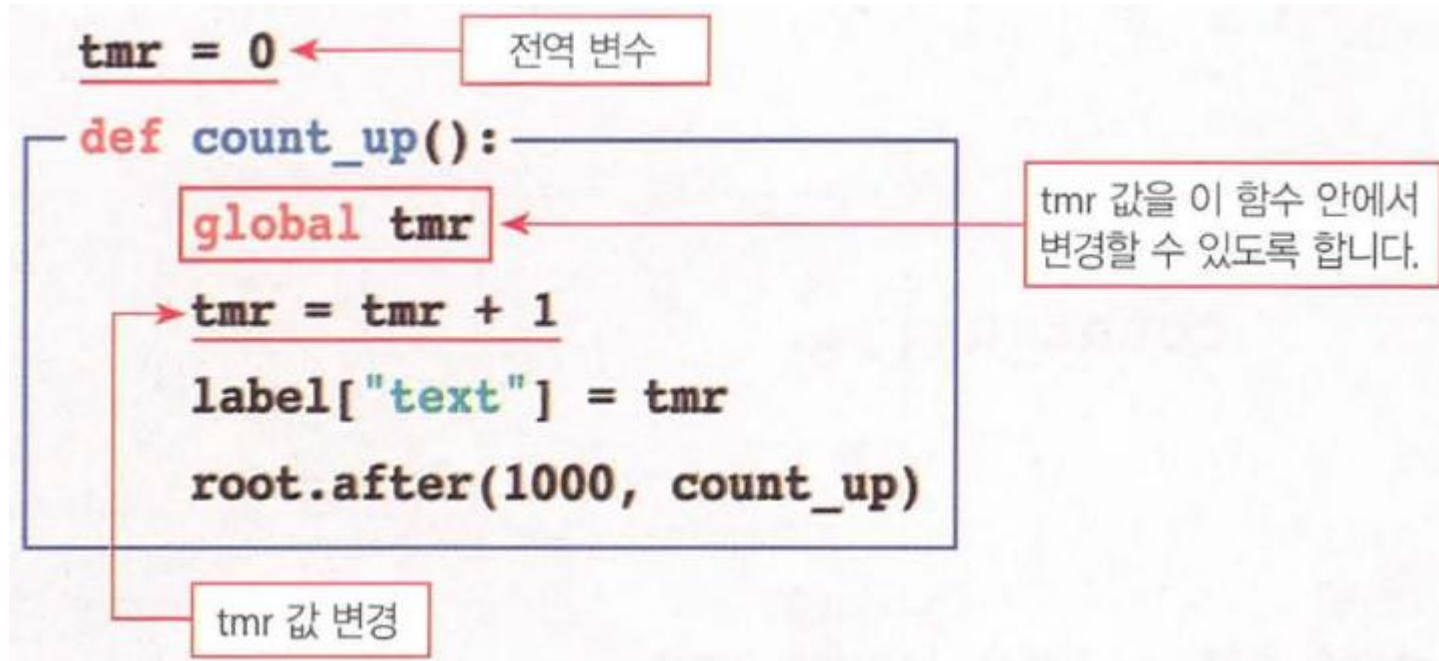
- `count_up()` 함수는 변수 `tmr`을 1씩 증가시키면서 그 값을 라벨에 표시합니다.
- 이 프로그램을 실행하면 먼저 12번 행 `after()` 명령에서 `count_up()`을 호출합니다.
- `count_up()` 함수 안에도 7번 행과 같이 `after()` 명령을 입력했으므로 1초 후 다시 `count_up()`이 호출됩니다.
- 이 처리를 그림으로 표시하면 다음과 같습니다.



1. 실시간 처리 구현하기

■ 전역 변수와 지역 변수

- 함수 외부에서 선언한 변수를 전역 변수(global variable), 함수 내부에서 선언한 변수를 지역 변수(local variable)라고 부릅니다.
- 파이썬에서는 함수 안에서 전역 변수 값을 변경할 때 해당 변수를 global이라고 선언해 주어야 합니다.
- 이번 프로그램에서 변수 tmr은 전역 변수입니다.
- tmr 값을 count_up() 안에서 증가시키므로 다음과 같이 입력합니다.



1. 실시간 처리 구현하기

■ 전역 변수와 지역 변수

- `global` 선언을 하지 않고 다음과 같이 입력하면 `tmr = tmr + 1` 부분에서 에러가 발생합니다.

```
tmr = 0
def count_up():
    tmr = tmr + 1
    label["text"] = tmr
    root.after(1000, count_up)
```

- 또한, 다음과 같이 입력하면 함수 안에서 선언한 `tmr`은 지역 변수가 되어 이 함수를 부를 때마다 값이 0이 됩니다.
- 때문에 시간이 지나도 라벨의 표시는 1인 채로 변하지 않습니다.

```
def count_up():
    tmr = 0;
    tmr = tmr + 1;
    label["text"] = tmr
    root.after(1000, count_up())
```


1. 실시간 처리 구현하기

■ 전역 변수와 지역 변수

- 전역 변수 값은 프로그램이 종료될 때까지 보존되지만, 함수 내 지역 변수 값은 그 함수를 호출할 때마다 초기화됩니다.
- 이는 많은 프로그래밍 언어에 공통되는 중요한 규칙들 중 하나이므로 반드시 기억하도록 합니다.
- 파이썬의 전역 변수에는 또 한 가지 특징이 있습니다.
- 함수 내에서 그 값을 참조만 하는 경우라면 global로 선언할 필요가 없습니다.
- 예를 들어, 다음 프로그램에서는 함수 내에서 orange, apple 값을 변경하지 않으므로 이 두 변수를 global로 선언할 필요는 없습니다.

```
orange = 50
apple = 120
def total_price():
    print(orange + apple)
```

- 또한, 함수 외부에서 선언한 리스트를 함수 내에서 다루는 경우 global로 선언할 필요가 없습니다.
- 리스트의 각 엘리먼트는 어떤 함수에서도 global로 선언하지 않고 값을 바꿀 수 있습니다.

2. 키 입력 받기

- 게임 소프트웨어에서는 어떤 키를 눌렀는지 판단해 그 키 값에 따라 캐릭터를 움직입니다.
- 키를 누른 것을 바로 알 수 있는 프로그램 작성 방법을 설명합니다.
- 이벤트에 관해
 - 사용자가 소프트웨어에 대해 키나 마우스를 조작하는 것을 이벤트(event)라고 부릅니다.
 - 예를 들어, 윈도우에 있는 이미지를 클릭한 경우에는 '이미지에 대한 클릭 이벤트가 발생했다'고 표현합니다.



2. 키 입력 받기

■ bind() 명령을 사용해 얻을 수 있는 이벤트

- 파이썬에서 이벤트를 받을 때는 bind() 명령을 사용합니다.
- bind() 명령을 사용하는 방법은 다음과 같습니다.

bind("<이벤트>", 이벤트 발생 시 실행할 함수명)

- 인수의 함수명은 ()을 붙이지 않고 입력합니다.
- bind() 명령을 사용해 얻을 수 있는 주요 이벤트는 다음과 같습니다.

<이벤트>	이벤트 상세
<KeyPress> 혹은 <Key>	키를 누름
<KeyRelease>	키를 눌렀다가 떼
<Motion>	마우스 포인터 움직임
<ButtonPress> 혹은 <Button>	마우스 버튼 클릭

- <KeyPress>는 단순히 <Key>, <ButtonPress>는 <Button>으로 입력해도 관계없습니다.

2. 키 입력 받기

■ bind() 명령을 사용해 얻을 수 있는 이벤트

- <ButtonPress>는 마우스의 모든 버튼을 감지합니다.
- <Button-1>은 마우스 왼쪽 버튼 클릭만, <Button-2>는 마우스 가운데 버튼 클릭만, <Button -3>는 마우스 오른쪽 버튼 클릭만 감지합니다.
- 다음 코드는 키보드 이벤트를 받는 함수를 작성한 것입니다.

```
def key_down(e):  
    global key  
    key = e.keycode  
    print("KEY: " + str(key))
```

- 인수 e로 이벤트를 얻습니다.
- 이 함수는 키보드 이벤트를 받는 함수이므로 e.keycode에서 키 코드를 얻습니다.
- 인수를 'e'라고 표기 했지만, def 'key_down(event)'처럼 원하는 변수명을 넣을 수도 있습니다.
- 이 경우에는 event.keycode가 키 코드값이 됩니다.

2. 키 입력 받기

■ bind() 명령 사용하기

- 키보드 이벤트를 얻어 어떤 키를 눌렀는지 확인하는 프로그램을 작성해 보겠습니다.
- bind() 명령 사용 방법은 동작 확인 후 설명하겠습니다.
- 예제 ex0702_1.py

```
1 import tkinter
2 key = 0
3 def key_down(e):
4     global key
5     key = e.keycode
6     print("KEY: " + str(key))
7
8 root = tkinter.Tk( )
9 root.title("키 코드 얻기")
10 root.bind("<KeyPress>", key_down)
11 root.mainloop( )
```

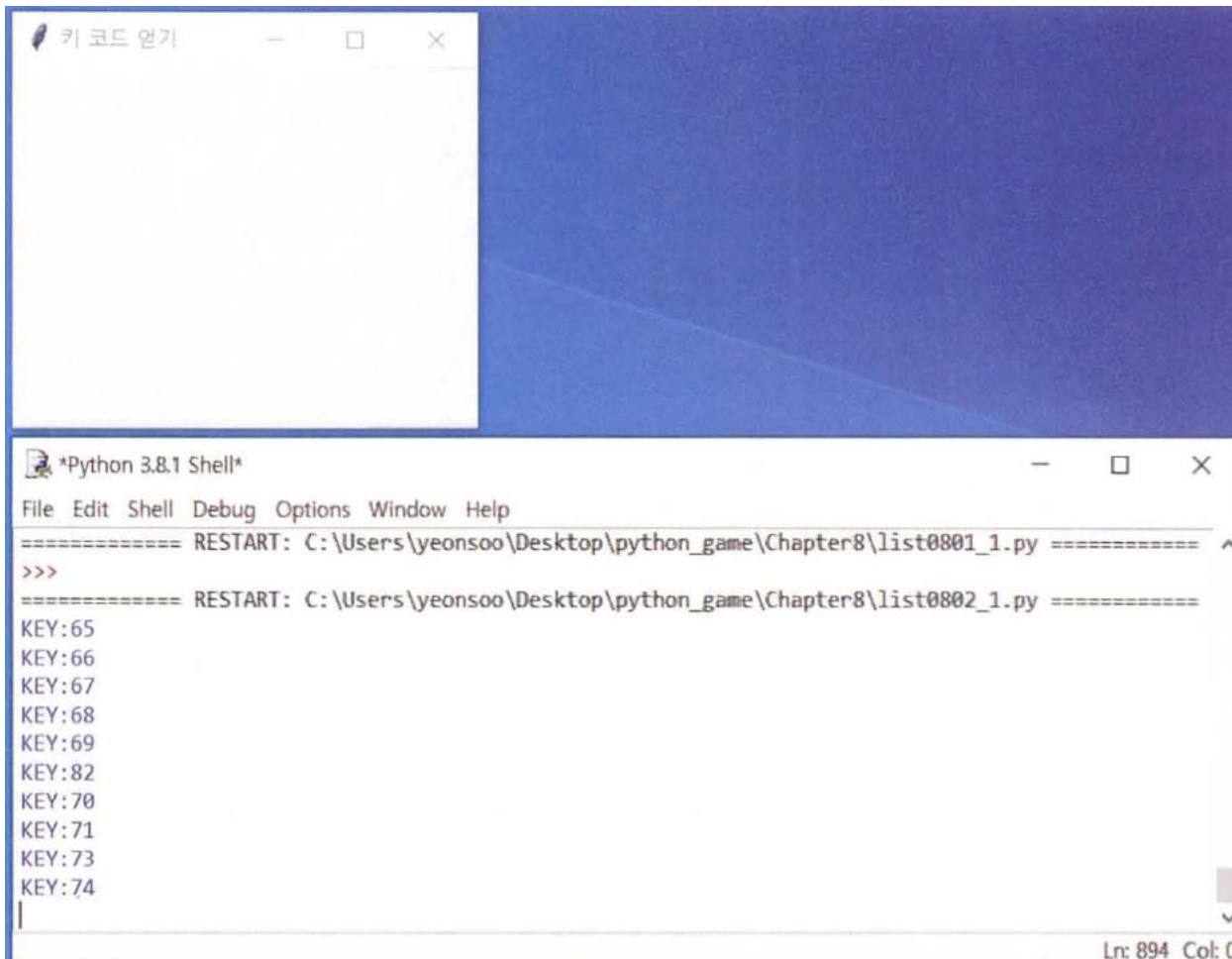
tkinter 모듈 импорт
키 코드 입력 변수 선언
키를 눌렀을 때 실행할 함수 정의
key를 전역변수로 선언
눌려진 키의 코드를 key에 대입
화면에 key값을 출력

윈도우 객체 생성
윈도우 제목 지정
키를 눌렀을 때 실행할 함수 지정
윈도우 표시

2. 키 입력 받기

■ bind() 명령 사용하기

- 이 프로그램을 실행하면 윈도우에는 아무런 표시가 나타나지 않으나, 키보드 키를 누르면 셸 윈도우에 그 키에 해당하는 값(키 코드)이 출력됩니다.



```
*Python 3.8.1 Shell*
File Edit Shell Debug Options Window Help
===== RESTART: C:\Users\yeonsoo\Desktop\python_game\Chapter8\list0801_1.py =====
>>>
===== RESTART: C:\Users\yeonsoo\Desktop\python_game\Chapter8\list0802_1.py =====
KEY:65
KEY:66
KEY:67
KEY:68
KEY:69
KEY:82
KEY:70
KEY:71
KEY:73
KEY:74
|
```

Ln: 894 Col: 0

3. 키 입력에 따라 이미지 움직이기

■ 실시간 키 입력

- 앞 프로그램에서 키 코드를 셸 윈도우에 출력했습니다.
- 이번에는 캐릭터를 움직이는 준비 단계로 윈도우 라벨에 키 코드를 표시하겠습니다.
- 다음 프로그램은 키 입력과 실시간 처리를 동시에 수행하는 프로그램입니다.
- 예제 `ex0703_1.py`

```
1 import tkinter
```

```
2
```

```
3 key = 0
```

```
4 def key_down(e):
```

```
5     global key
```

```
6     key = e.keycode
```

```
7
```

```
8 def main_proc( ):
```

```
9     label['text'] = key
```

```
10    root.after(100, main_proc)
```

tkinter 모듈 импорт

키 코드 입력 변수 선언

키를 눌렀을 때 실행할 함수 정의

key를 전역변수로 선언

눌려진 키의 코드를 key에 대입

실시간 처리를 수행할 함수 정의

라벨에 key 값 표시

0.1초 후 실행할 함수 지정

3. 키 입력에 따라 이미지 움직이기

■ 실시간 키 입력

▪ 예제 ex0703_1.py

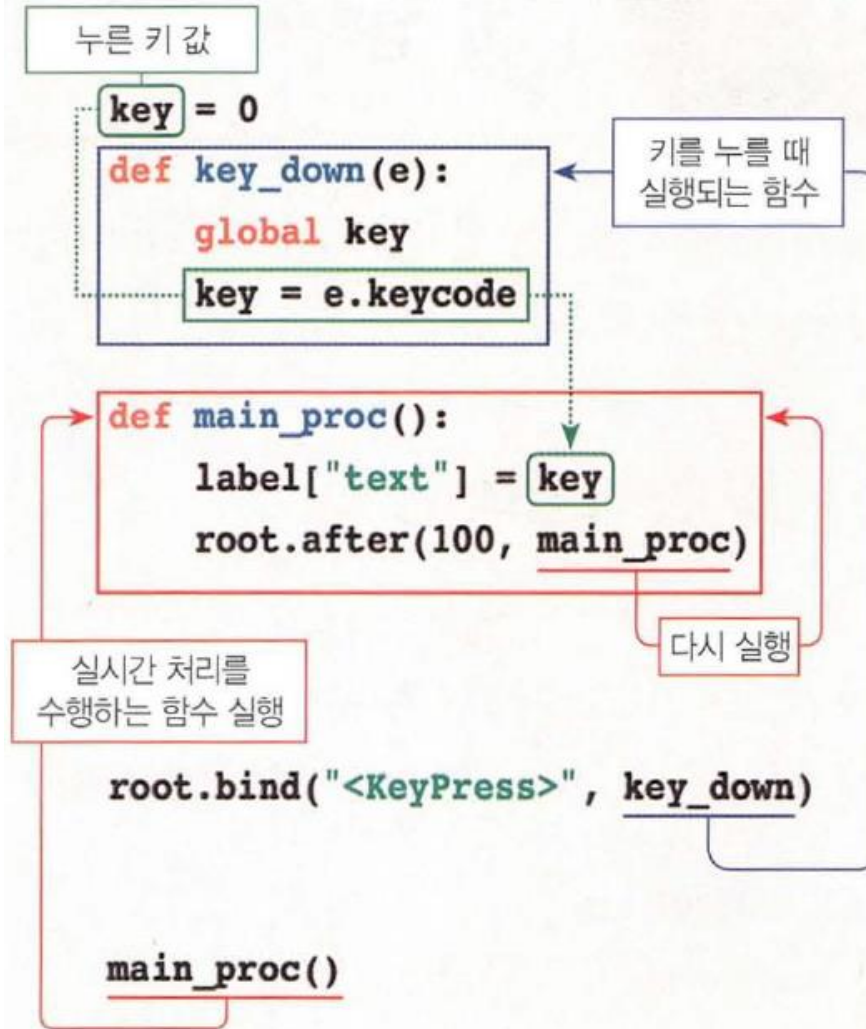
11		
12	<code>root = tkinter.Tk()</code>	윈도우 객체 생성
13	<code>root.title("키 코드 얻기")</code>	윈도우 제목 지정
14	<code>root.bind("<KeyPress>", key_down)</code>	키를 눌렀을 때 실행할 함수 지정
15	<code>label = tkinter.Label(font=("Times New Roman", 80))</code>	라벨 컴포넌트 생성
16	<code>label.pack()</code>	라벨 컴포넌트 배치
17	<code>main_proc()</code>	<code>main_proc()</code> 함수 실행
18	<code>root.mainloop()</code>	윈도우 표시

- 이 프로그램을 실행하면 누른 키의 코드가 윈도우에 표시됩니다.
- 예를 들어, 'space' 키를 누르면 32가 표시됩니다.
- 4~6번 행이 키 이벤트를 얻는 함수, 8~10번 행이 실시간 처리를 수행하는 함수입니다.
- 키 이벤트 얻기는 `bind()` 명령으로 함수를 지정하고, 실시간 처리는 `after()` 명령으로 지정합니다.

3. 키 입력에 따라 이미지 움직이기

■ 실시간 키 입력







- 프로그램의 동작을 그림으로 나타내면 다음과 같습니다.



3. 키 입력에 따라 이미지 움직이기

■ 주요 키 코드

- 파이썬에서 주요 키 코드는 다음과 같습니다.

키	키 코드
방향키 ←↑→↓(순서대로)	37, 38, 39, 40
 키	32
 키	13
알파벳  ~ 	65~90
숫자  ~ 	48~57

3. 키 입력에 따라 이미지 움직이기

■ keysym 값을 사용해 판정하기

- 이번에는 keycode 값이 아닌 keysym 값을 얻는 프로그램을 작성해 보겠습니다.
- 예제 ex0703_2.py

```
1 import tkinter
2
3 key = ""
4 def key_down(e):
5     global key
6     key = e.keysym
7
8 def main_proc( ):
9     label['text'] = key
10    root.after(100, main_proc)
```

tkinter 모듈 импорт

키 코드 입력 변수 선언

키를 눌렀을 때 실행할 함수 정의

key를 전역변수로 선언

눌려진 키 이름을 key에 대입

실시간 처리를 수행할 함수 정의

라벨에 key 값 표시

0.1초 후 실행할 함수 지정

3. 키 입력에 따라 이미지 움직이기

■ keysym 값을 사용해 판정하기

▪ 예제 ex0703_2.py

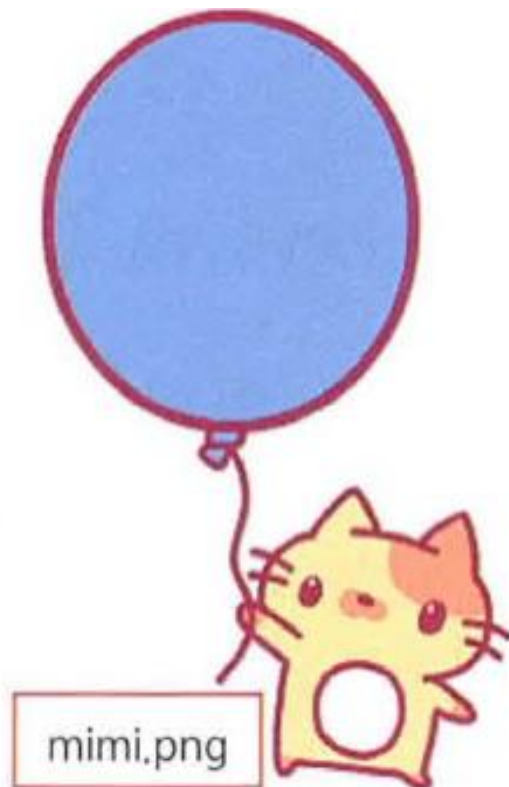
11		
12	<code>root = tkinter.Tk()</code>	윈도우 객체 생성
13	<code>root.title("실시간 키입력")</code>	윈도우 제목 지정
14	<code>root.bind("<KeyPress>", key_down)</code>	키를 눌렀을 때 실행할 함수 지정
15	<code>label = tkinter.Label(font=("Times New Roman", 80))</code>	라벨 컴포넌트 생성
16	<code>label.pack()</code>	라벨 컴포넌트 배치
17	<code>main_proc()</code>	<code>main_proc()</code> 함수 실행
18	<code>root.mainloop()</code>	윈도우 표시

- 이 프로그램에서는 방향키의 '↑'를 누르면 Up, '↓'를 누르면 Down, 'space' 키를 누르면 space, 'Enter' 키나 'return' 키를 누르면 Return이라는 문자가 표시됩니다.
- `keysym`으로 얻은 키 이름은 윈도우, 맥 공통이므로 키 입력은 `keysym` 값으로 판정하는 것이 편리합니다.

3. 키 입력에 따라 이미지 움직이기

■ 실시간으로 캐릭터 움직이기

- 윈도우에 표시된 캐릭터를 방향키를 눌러 상하좌우로 움직이는 프로그램을 확인합니다.
- 새로 등장한 명령은 동작 확인 후 설명합니다.
- 이번 프로그램에서는 다음 이미지를 사용합니다.
- 깃헙 페이지에서 다운로드한 이미지 파일을 예제 코드와 같은 폴더에 넣어주십시오.



3. 키 입력에 따라 이미지 움직이기

■ 실시간으로 캐릭터 움직이기

- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 `ex0703_3.py`

```
1 import tkinter
2
3 key = ""
4 def key_down(e):
5     global key
6     key = e.keysym
7 def key_up(e):
8     global key
9     key = ""
10
11 cx = 400
12 cy = 300
```

tkinter 모듈 импорт

키 코드 입력 변수 선언

키를 눌렀을 때 실행할 함수 정의
key를 전역변수로 선언

눌려진 키 이름을 key에 대입

키를 눌렀을 때 실행할 함수 정의
key를 전역변수로 선언
key에 빈 문자열 대입

캐릭터의 x좌표를 관리할 변수

캐릭터의 y좌표를 관리할 변수

3. 키 입력에 따라 이미지 움직이기

■ 실시간으로 캐릭터 움직이기

- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 `ex0703_3.py`

```
13 def main_proc( ):
14     global cx, cy
15     if key == "Up":
16         cy = cy - 20
17     if key == 'Down':
18         cy = cy + 20
19     if key == "Left":
20         cx = cx - 20
21     if key == "Right":
22         cx = cx + 20
23     canvas.coords("MYCHR", cx, cy)
24     root.after(100, main_proc)
```

실시간 처리를 수행할 함수 정의
cx, cy를 전역 변수로 선언
방향키 [
키 코드 입력 변수 선언
키를 눌렀을 때 실행할 함수 정의
key를 전역변수로 선언
눌려진 키 이름을 key에 대입
키를 눌렀을 때 실행할 함수 정의
key를 전역변수로 선언
key에 빈 문자열 대입

캐릭터의 x좌표를 관리할 변수

3. 키 입력에 따라 이미지 움직이기

■ 실시간으로 캐릭터 움직이기

- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 `ex0703_3.py`

```
25
26 root = tkinter.Tk()
27 root.title("캐릭터 이동")
28 root.bind("<KeyPress>", key_down)
29 root.bind("<KeyRelease>", key_up)
30 canvas = tkinter.Canvas(width=800,
31                          height=600, bg='lightgreen')
32 canvas.pack()
33 img =
34     tkinter.PhotoImage(file='mimi.png')
```

윈도우 객체 생성

윈도우 제목 지정

키를 눌렀을 때 실행할 함수 지정

키를 눌렀다 뗐을 때 실행할 함수 지정

캔버스 컴포넌트 생성

캔버스 컴포넌트 배치

캐릭터 이미지를 변수 `img`에 로딩

3. 키 입력에 따라 이미지 움직이기

■ 실시간으로 캐릭터 움직이기

- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 ex0703_3.py

```
33 canvas.create_image(cx, cy, image=img,  
34 tag="MYCHR")  
35 main_proc( )  
36 root.mainloop( )
```

캔버스에 이미지 표시

main_proc() 함수 실행
윈도우 표시

- 이 프로그램을 실행하면 캐릭터가 표시되고 방향키를 통해 상하좌우로 이동할 수 있습니다.
- 13~24번 행 main_proc() 이 실시간 처리를 수행하는 함수입니다.
- 11~ 12번 행에서 캐릭터 좌표를 관리하는 변수 cx, cy를 전역 변수로 선언합니다.
- main_proc() 내에서는 누른 거에 대응해 cx, cy 값을 증가/감소시킵니다.
- 23번 행의 coords()는 표시 중인 이미지를 새로운 위치로 이동하는 명령입니다.
- coords() 명령의 인수는 태그 명, X 좌표, Y 좌표입니다.

3. 키 입력에 따라 이미지 움직이기

■ 태그

- 32번 행의 `PhotoImage()` 명령으로 이미지를 로딩하고 33번 행의 `create_image()` 명령으로 캔버스에 이미지를 표시합니다.
- 이때 `create_image()` 명령의 인수로 다음과 같이 태그를 지정합니다.

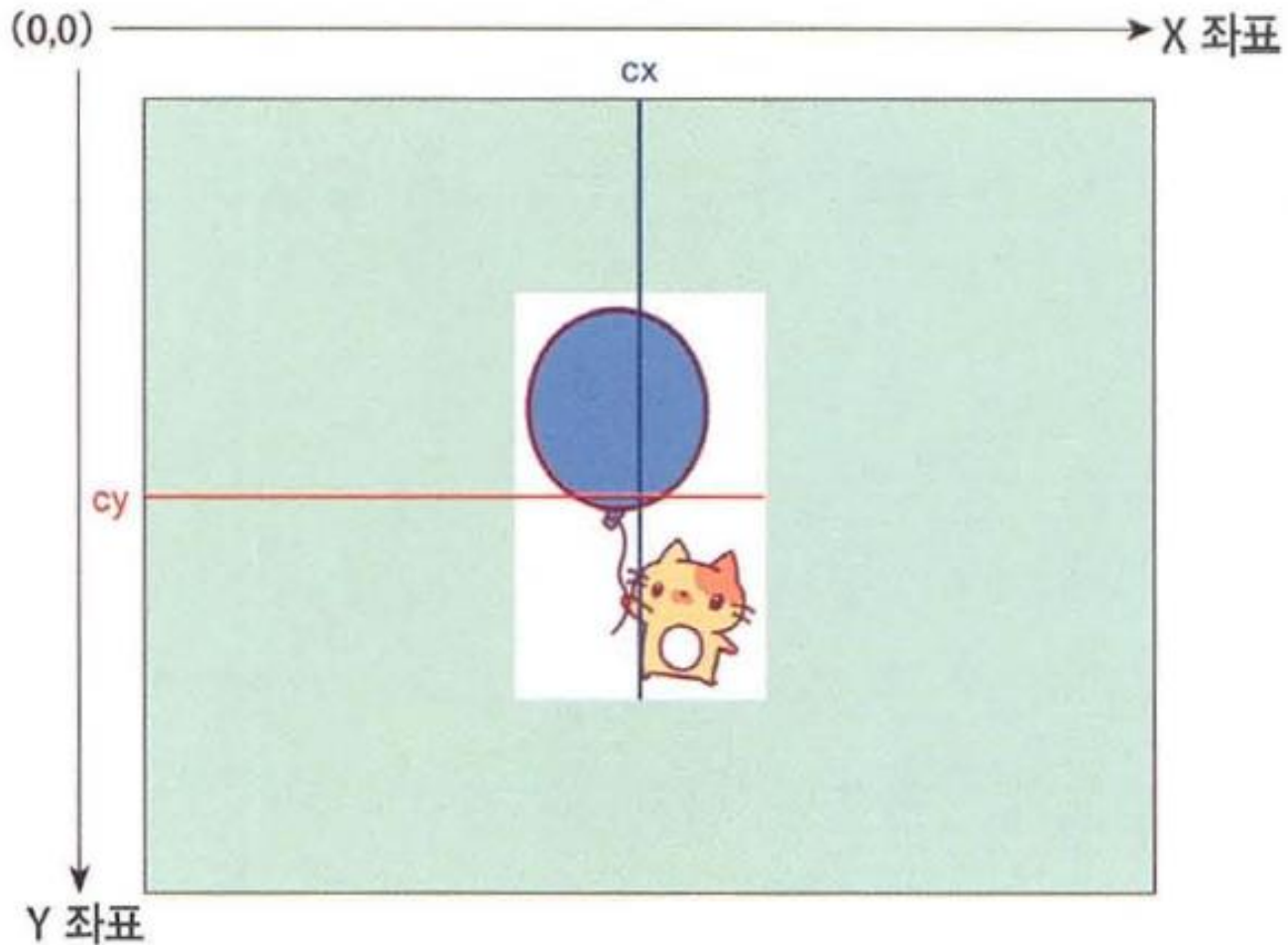
```
canvas.create_image(cx , cy , image=img, tag= "MYCHR")
```

- `tag=` 뒤에 입력한 문자열이 태그 명입니다.
- 태그는 캔버스에 그리는 도형이나 이미지에 붙일 수 있으며, 도형이나 이미지를 움직이거나 지우는 경우에 사용합니다.
- 태그 명은 자유롭게 붙일 수 있으나, 알기 쉽게 붙이도록 합니다.
- 예제에서는 MYCHR이라는 태그 명을 붙였습니다.

3. 키 입력에 따라 이미지 움직이기

■ create_image()의 좌표

- create_image() 명령의 인수인 좌표는 다음 그림과 같이 이미지의 중심 좌표입니다



4. 미로 데이터 정의하기

- 2D(2차원) 화면 구성 게임에서는 배경 데이터를 배열로 관리합니다.
- 파이썬에서는 리스트가 배열에 해당합니다.
- 리스트로 미로를 정의해 윈도우에 표시하는 방법을 설명합니다.
- 다음 절에서는 미로 안을 캐릭터가 걸어 다니도록 해보겠습니다.

4. 미로 데이터 정의하기

■ 2차원 리스트

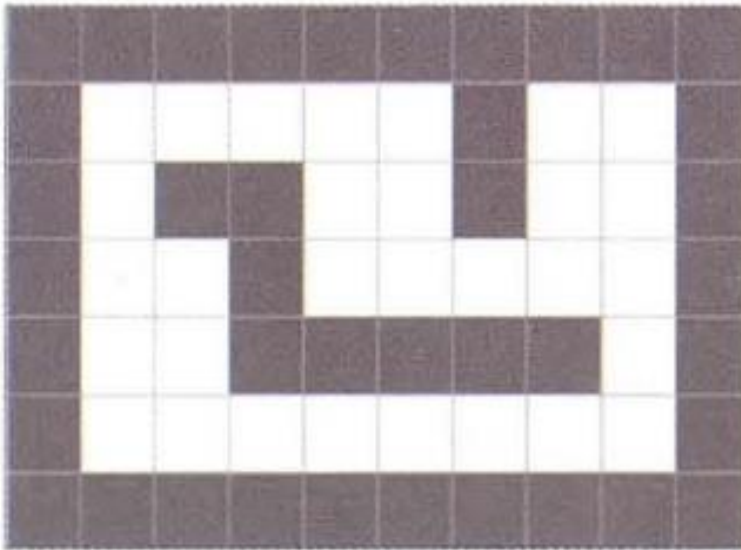
- 미로와 같은 데이터는 2차원 리스트로 정의합니다.
- 2차원 리스트란 가로 방향(열)과 세로 방향(행) 인덱스를 사용해 데이터를 다루는 리스트입니다.
- 가로 방향을 x , 세로 방향을 y 라고 정의할 때 각 엘리먼트의 인덱스는 다음과 같습니다.

$m[0][0]$	$m[0][1]$	$m[0][2]$	$m[0][3]$
$m[1][0]$	$m[1][1]$	$m[1][2]$	$m[1][3]$
$m[2][0]$	$m[2][1]$	$m[2][2]$	$m[2][3]$

- 예를 들면, 오른쪽 아래 모서리의 $m[2][3]$ 에 10을 대입하는 경우 $m[2][3] = 10$ 과 같이 입력합니다.

IT COOKBOOK

- 다음과 같은 미로가 있다고 가정합니다.
- 흰 부분이 바닥, 회색 부분이 벽입니다.
- 이 미로를 2차원 리스트로 정의해 봅시다.



IT COOKBOOK

- 미로를 프로그램에서 다룰 때는 바닥과 벽을 숫자로 바꿉니다.
- 여기에서는 바닥을 0, 벽을 1로 표기합니다.

[illegible]

4. 미로 데이터 정의하기

■ 리스트로 미로 정의하기

- 이 값을 2차원 리스트로 정의합니다.
- 리스트 명은 maze입니다.

```
maze = [  
    [1,1,1,1,1,1,1,1,1,1],  
    [1,0,0,0,0,0,1,0,0,1],  
    [1,0,1,1,0,0,1,0,0,1],  
    [1,0,0,1,0,0,0,0,0,1],  
    [1,0,0,1,1,1,1,1,0,1],  
    [1,0,0,0,0,0,0,0,0,1],  
    [1,1,1,1,1,1,1,1,1,1]  
]
```

리스트 시작 [

각 행은 '[~]'으로 입력합니다.

가장 마지막 행의 ']'에는逗마가 필요
하지 않습니다.

리스트 끝]

4. 미로 데이터 정의하기

■ 리스트로 미로 정의하기

- 리스트로 정의한 미로를 윈도우에 표시합니다.
- 실시간 처리나 키 입력은 고려하지 않고, 미로만 표시하는 프로그램을 만듭니다.
- 이 프로그램에는 for 구문 안에 다른 for 구문을 넣은 2중 반복 for를 사용합니다.
- 2중 반복 for에 관해서는 동작 확인 후 설명합니다.
- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 ex0704_1.py

1	import tkinter	tkinter 모듈 импорт
2	root = tkinter.Tk()	윈도우 객체 생성
3	root.title('미로 표시')	윈도우 타이틀 설정
4	canvas = tkinter.Canvas(width=800, height=560, bg='white')	캔버스 컴포넌트 생성
5	canvas.pack()	캔버스 컴포넌트 배치

4. 미로 데이터 정의하기

■ 리스트로 미로 정의하기

■ 예제 ex0704_1.py

```
6  maze = [  
7      [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
8      [1, 0, 0, 0, 0, 0, 1, 0, 0, 1],  
9      [1, 0, 1, 1, 0, 0, 1, 0, 0, 1],  
10     [1, 0, 0, 1, 0, 0, 0, 0, 0, 1],  
11     [1, 0, 0, 1, 1, 1, 1, 1, 0, 1],  
12     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],  
13     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  
14 ]
```

리스트로 미로 정의

4. 미로 데이터 정의하기

■ 리스트로 미로 정의하기

■ 예제 ex0704_1.py

```
15 for y in range(7):
16     for x in range(10):
17         if maze[y][x] == 1:
18             canvas.create_rectangle(x*80,
19                                     y*80, x*80 + 80, y*80 + 80, fill='gray')
19 root.mainloop()
```

반복, y: 0→1→2→3→4→5→6

반복, x:

0→1→2→3→4→5→6→7→8→9

maze[y][x]가 1, 즉, 벽이라면

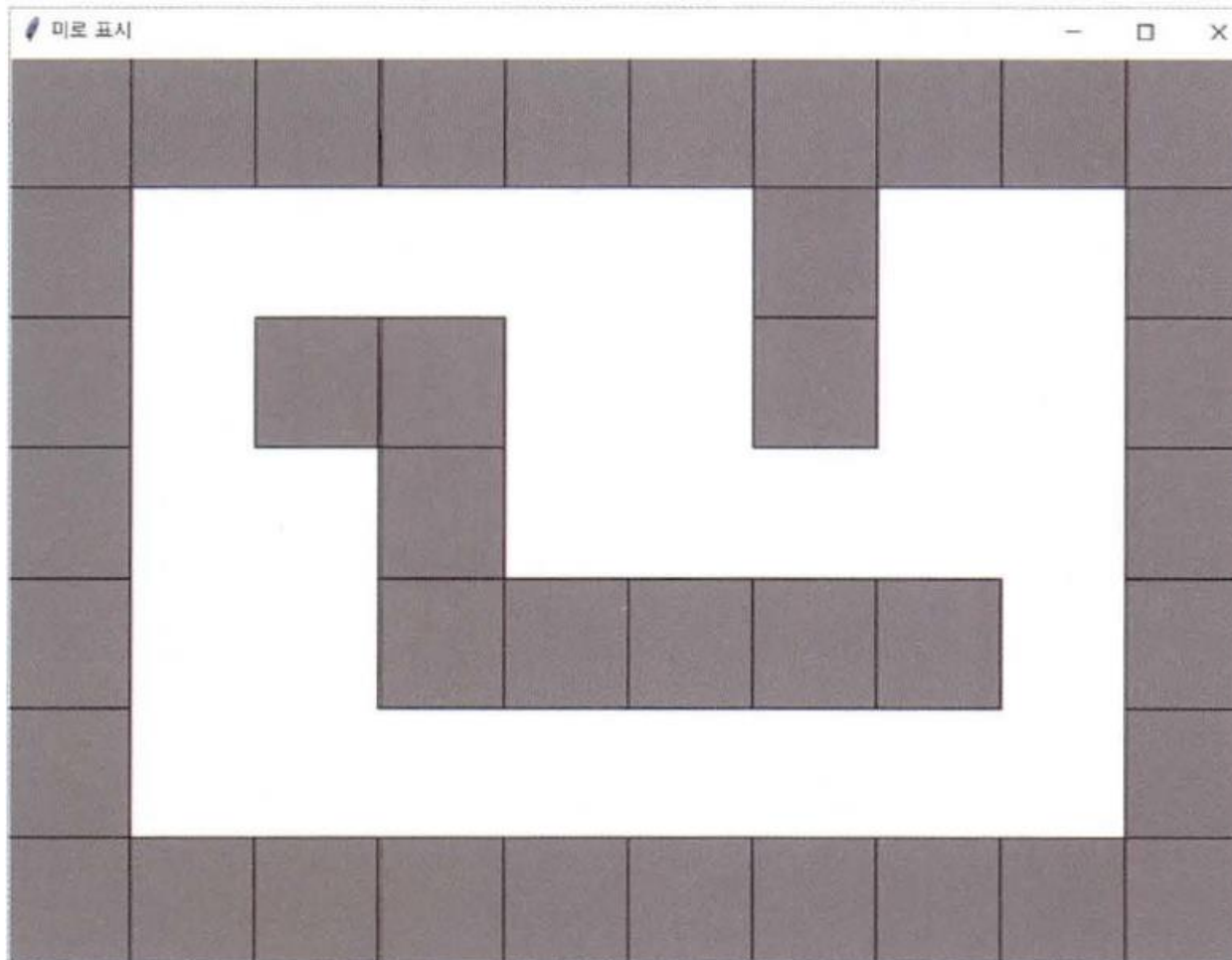
회색 사각형을 그림

윈도우 표시

4. 미로 데이터 정의하기

■ 리스트로 미로 정의하기

- 이 프로그램을 실행하면 다음과 같은 미로가 표시됩니다.



- 15~18번 행이 2중 반복 for 구문입니다.
- 이 구조는 다음과 같이 되어 있습니다.

처리 블록

- 변수 1과 변수 2는 다른 이름을 붙입니다.

4. 미로 데이터 정의하기

■ 2중 반복 for 구문

- 이번 프로그램에서는 변수 1을 y, 변수 2를 x로 합니다.

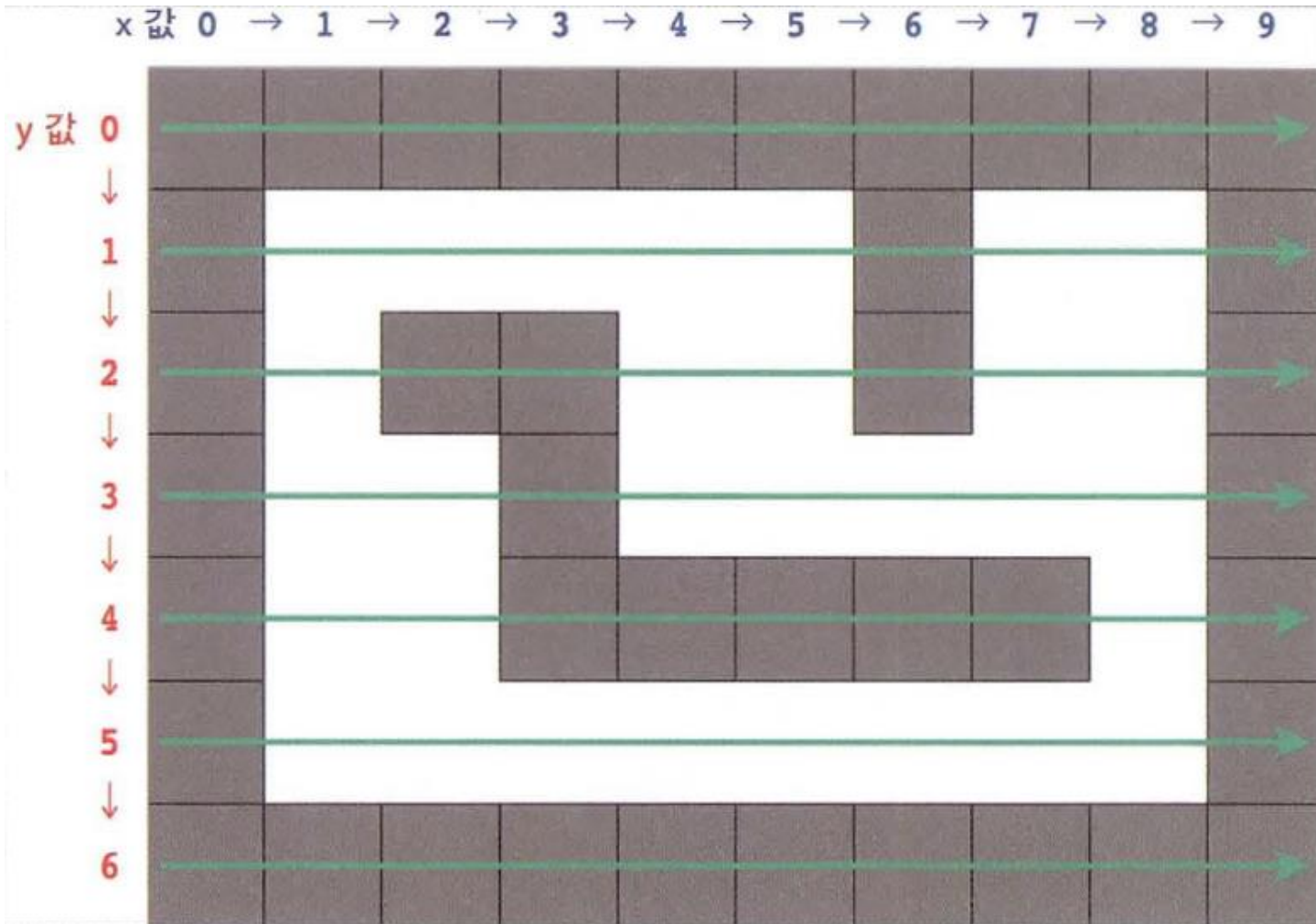
```
for y in range(7):  
    for x in range(10):  
        ~ 처리 ~
```

- y 값은 0→1→2→3→4→5→6으로 바뀝니다.
- 먼저 y 값이 0일 때, x 값이 0→1→2→3→4→5→6→7→8→9로 바뀌면서 처리를 수행합니다.
- x의 반복이 끝나면 y 값이 1이 되고, 다시 x 값이 0→1→2→3→4→5→6→7→8→9로 바뀌면서 처리를 수행합니다.
- 이번 2중 반복에서는 maze[y][x]의 값을 확인해서 1이면 회색 사각형(벽)을 그립니다.

4. 미로 데이터 정의하기

■ 2중 반복 for 구문

- 이를 그림으로 표시하면 다음과 같습니다.



5. 2차원 화면 게임 개발 기초

- 이 절에서는 실시간 처리, 키 입력, 미로 정의와 같은 3가지 지식을 조합해 캐릭터를 움직여 미로를 이동하는 프로그램을 만듭니다.
- 여기에서 학습한 내용은 2D(2차원) 화면으로 구성된 게임을 개발하는 기초가 됩니다.
- 미로 안 걷기
 - 3절의 캐릭터를 방향 키로 움직이는 프로그램, 4절의 미로를 표시하는 프로그램을 모두 조합해 캐릭터가 미로 안을 걷는 프로그램을 만듭니다.
 - 캐릭터를 움직이기 위해 if 구문에서 and를 사용해 2개의 조건을 동시에 판정합니다.
 - 이에 관해서는 동작을 확인 후에 설명하겠습니다.
 - 이번 프로그램에서는 오른쪽 이미지를 사용합니다.
 - 깃헙 페이지에서 이미지를 다운로드하고 예제 코드와 같은 폴더에 저장합니다.



mimi_s.png

5. 2차원 화면 게임 개발 기초

■ 미로 안 걷기

- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 ex0705_1.py

```
1 import tkinter
2
3 key = ""
4 def key_down(e):
5     global key
6     key = e.keysym
7 def key_up(e):
8     global key
9     key = ""
10
```

tkinter 모듈 импорт

키 이름을 입력할 변수 선언

키를 눌렀을 때 실행할 함수 정의

key를 전역 변수로 선언

누른 키 이름을 key에 대입

키를 눌렀다 떼을 때 실행할 함수 정의

key를 전역 변수로 선언

key에 빈 문자열 입력

5. 2차원 화면 게임 개발 기초

■ 미로 안 걷기

- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 ex0705_1.py

```
11 mx = 1
12 my = 1
13 def main_proc():
14     global mx, my
15     if key == "Up" and maze[my - 1][mx] == 0:
16         my = my - 1
17     if key == "Down" and maze[my + 1][mx] == 0:
18         my = my + 1
19     if key == "Left" and maze[my][mx - 1] == 0:
20         mx = mx - 1
```

캐릭터 가로 방향 위치를 관리하는 변수
캐릭터 세로 방향 위치를 관리하는 함수
실시간 처리 수행 함수 정의

mx, my를 전역 변수로 선언

방향키[↑]을 눌렀을 때 위가 통로라면

my 값 1 감소

방향키[↓]을 눌렀을 때 아래가 통로라면

my 값 1 증가

방향키[←]를 눌렀을 때 왼쪽이 통로라면

mx 값 1 감소

5. 2차원 화면 게임 개발 기초

■ 미로 안 걷기

- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 ex0705_1.py

```
21     if key == "Right" and maze[my][mx + 1] == 0:
22         mx = mx + 1
23         canvas.coords("MYCHR", mx * 80 + 40, my * 80 + 40)
24         root.after(300, main_proc)
25
26 root = tkinter.Tk()
27 root.title("미로 안 이동하기")
28 root.bind("<KeyPress>", key_down)
29 root.bind("<KeyRelease>", key_up)
30 canvas = tkinter.Canvas(width=800, height=560,
31                          bg="white")
32 canvas.pack()
```

방향키[→]를 눌렀을 때 오른쪽이 통로라면
mx 값 1 증가
캐릭터 이미지를 새로운 위치로 이동
after() 명령으로 0.3초 후 실행할 함수 지정

윈도우 객체 생성
윈도우 타이틀 설정
bind() 명령으로 키를 눌렀을 때 실행할 함수 정의
bind() 명령으로 키를 뗐을 때 실행할 함수 정의
캔버스 컴포넌트 생성

캔버스 컴포넌트 배치

5. 2차원 화면 게임 개발 기초

```
33 maze = [  
34     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
35     [1, 0, 0, 0, 0, 0, 1, 0, 0, 1],  
36     [1, 0, 1, 1, 0, 0, 1, 0, 0, 1],  
37     [1, 0, 0, 1, 0, 0, 0, 0, 0, 1],  
38     [1, 0, 0, 1, 1, 1, 1, 1, 0, 1],  
39     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],  
40     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  
41 ]  
42 for y in range(7):  
43     for x in range(10):  
44         if maze[y][x] == 1:  
45             canvas.create_rectangle(x * 80, y * 80, x  
46                                     * 80 + 79, y * 80 + 79, fill="skyblue", width=0)  
47 img = tkinter.PhotoImage(file="mimi_s.png")  
48 canvas.create_image(mx * 80 + 40, my * 80 + 40,  
49                     image=img, tag="MYCHR")  
49 main_proc()  
50 root.mainloop()
```

리스트로 미로 정의

반복, y: 0→1→2→3→4→5→6

반복, x: 0→1→2→3→4→5→6→7→8→9

maze[y][x]가 1, 즉, 벽이라면

회색 사각형 그림

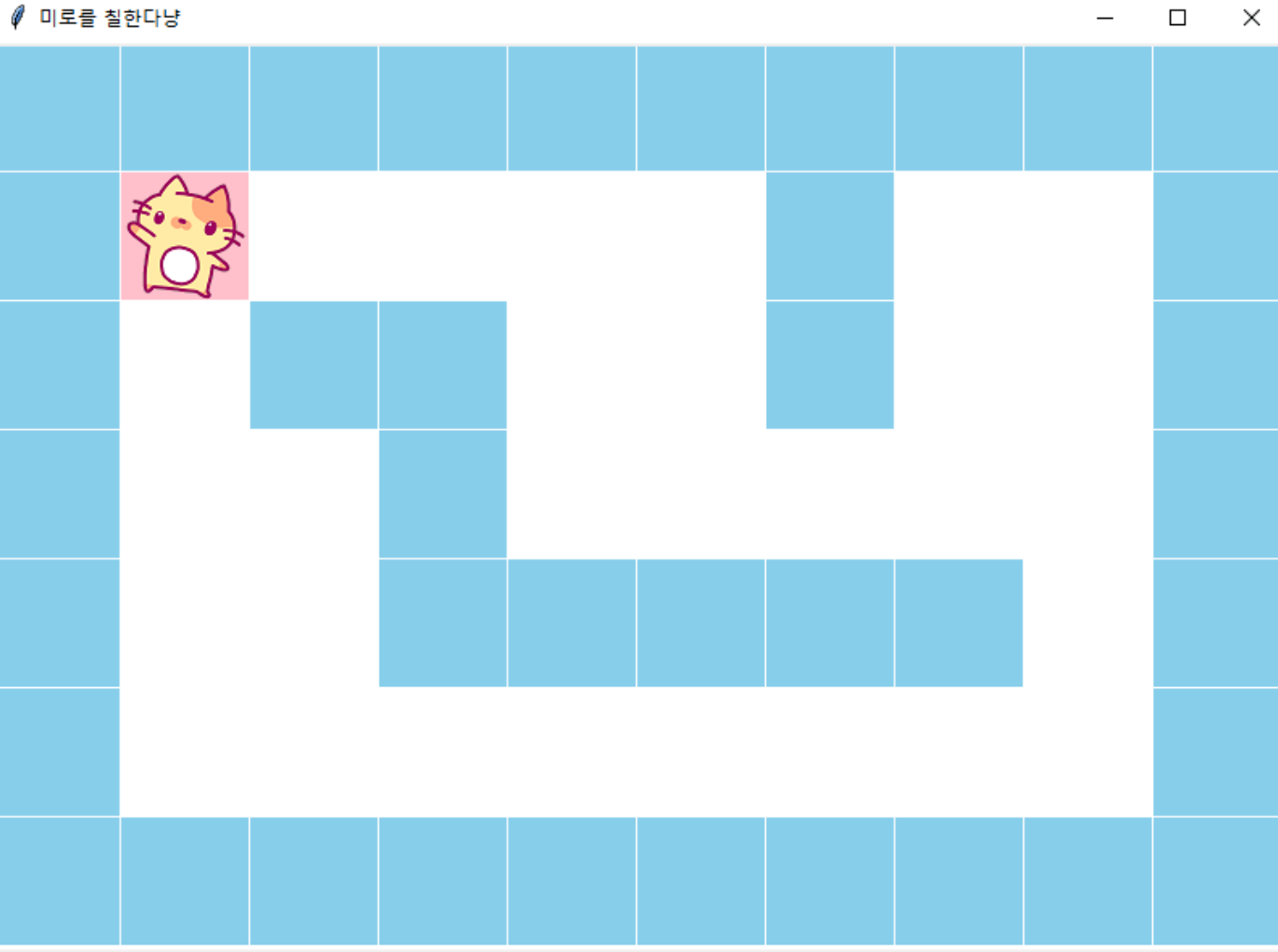
캐릭터 이미지를 변수 img에 로딩
캔버스에 이미지 표시

main_proc() 함수 실행
윈도우 표시

5. 2차원 화면 게임 개발 기초

■ 미로 안 걷기

- 이 프로그램을 실행하면 캐릭터가 표시되고 방향 키로 미로 안을 움직일 수 있습니다.



5. 2차원 화면 게임 개발 기초

■ 미로 안 걷기

- 캐릭터를 움직이는 부분을 별도로 확인해 보겠습니다.
- `main_proc()` 함수 처리 부분입니다.

```
~ 생략 ~
11  mx = 1
12  my = 1
13  def main_proc():
14      global mx, my
15      if key == "Up" and maze[my - 1][mx] == 0:
16          my = my - 1
17      if key == "Down" and maze[my + 1][mx] == 0:
18          my = my + 1
19      if key == "Left" and maze[my][mx - 1] == 0:
20          mx = mx - 1
21      if key == "Right" and maze[my][mx + 1] == 0:
22          mx = mx + 1
23      canvas.coords("MYCHR", mx * 80 + 40, my * 80 + 40)
~ 생략 ~
```

■ 미로 안 걷기

- 3절의 캐릭터를 움직이는 프로그램에서는 캔버스의 캐릭터 좌표를 변수 `cx`, `cy`로 관리했습니다.
- 이번 프로그램에서는 미로의 어느 칸에 있는지를 관리하는 변수를 11~12번 행에서 선언합니다.
- ‘어느 칸’이란 `maze[y][x]`의 인덱스인 `y`와 `x`의 값입니다.
- 15번 행 `if key == "Up" and maze[my - 1][mx] == 0` 조건 분기는 ‘방향키 위를 누른 상태이고 현재 칸의 위 쪽이 바닥이면’이라는 의미입니다.
- `and`를 사용하면 2개 이상의 조건이 동시에 만족하는지 조사할 수 있습니다.
- 이 프로그램에서는 칸 1개의 폭과 높이를 각각 80픽셀로 하고 있으며, 캐릭터를 표시하는 칸은 `canvas.coords("MYCHR", mx * 80 + 40, my * 80 + 40)`으로 입력한 것과 같이, X 좌표가 `mx * 80 + 40`, Y 좌표가 `my * 80 + 40`이 됩니다.
- 각각 40을 더한 것은 지정한 좌표가 이미지의 중심이 되기 때문입니다.

5. 2차원 화면 게임 개발 기초

■ 미로 안 걷기

- 2차원 리스트 `maze[][]`의 인덱스를 그림으로 표시하면 다음과 같습니다.
- 이 프로그램에서는 변수 `mx`와 `my`가 인덱스 값이 됩니다.



5. 2차원 화면 게임 개발 기초

- 이번 프로그램에서는 미로의 바닥과 벽을 0과 1의 숫자로 관리했습니다만, 예를 들어, 들판을 0, 숲을 1, 수면을 2 등으로 데이터 종류를 늘리면 보다 복잡한 게임 세계를 만들 수 있습니다.
- 2차원 평면으로 구성된 게임 소프트웨어의 대부분은 이 프로그램에서 구현한 것과 같이, 배경이나 맵 위에 존재하는 물체를 숫자 값으로 바꿔 게임 세계의 어느 곳에 무엇이 있는지를 관리합니다.

6. 게임 완성하기

- 걸어간 통로를 표시하면서 한 번에 미로 안을 모두 칠한다면 클리어되도록 게임 프로그램을 업그레이드합니다.
- 리스트 값 바꾸기
 - 캐릭터가 지나간 위치를 분홍색으로 칠하도록 합니다.
 - 2차원 리스트로 구현한 미로 데이터 값은 통로가 0, 벽은 1입니다.
 - 지나간 위치의 값은 0에서 2로 바꿉니다.
 - 2로 바뀐 위치에도 들어가지 않으면 후퇴할 수 없으므로 한 번에 그리는 규칙을 실현할 수 있습니다.
 - 이 프로그램은 앞 절의 ex0705_1.py를 수정한 것입니다.

6. 게임 완성하기

■ 리스트 값 바꾸기

- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 ex0706_1.py

```
1 import tkinter
2
3 key = ""
4 def key_down(e):
5     global key
6     key = e.keysym
7 def key_up(e):
8     global key
9     key = ""
10
```

tkinter 모듈 импорт

키 이름을 입력할 변수 선언

키를 눌렀을 때 실행할 함수 정의

key를 전역 변수로 선언

누른 키 이름을 key에 대입

키를 눌렀다 뗄 때 실행할 함수 정의

key를 전역 변수로 선언

key에 빈 문자열 입력

6. 게임 완성하기

■ 리스트 값 바꾸기

■ 예제 ex0706_1.py

```
11 mx = 1
12 my = 1
13 def main_proc():
14     global mx, my
15     if key == "Up" and maze[my - 1][mx] == 0:
16         my = my - 1
17     if key == "Down" and maze[my + 1][mx] == 0:
18         my = my + 1
19     if key == "Left" and maze[my][mx - 1] == 0:
20         mx = mx - 1
21     if key == "Right" and maze[my][mx + 1] == 0:
22         mx = mx + 1
23     if maze[my][mx] == 0:
24         maze[my][mx] = 2
25         canvas.create_rectangle(mx * 80, my * 80,
mx * 80 + 79, my * 80 + 79, fill="pink", width=0)
```

캐릭터 가로 방향 위치를 관리하는 변수
캐릭터 세로 방향 위치를 관리하는 함수
실시간 처리 수행 함수 정의

mx, my를 전역 변수로 선언

방향키[↑]을 눌렀을 때 위가 통로라면
my 값 1 감소

방향키[↓]을 눌렀을 때 아래가 통로라면
my 값 1 증가

방향키[←]를 눌렀을 때, 왼쪽이 통로라면
mx 값 1 감소

방향키[→]를 눌렀을 때 오른쪽이 통로라면
mx 값 1 증가

캐릭터가 있는 장소가 통로라면

리스트 값을 2로 변경

해당 위치를 분홍색으로 칠함

6. 게임 완성하기

■ 리스트 값 바꾸기

■ 예제 ex0706_1.py

```
26 canvas.delete("MYCHR")
27 canvas.create_image(mx * 80 + 40, my * 80 + 40,
image=img, tag="MYCHR")
28 root.after(300, main_proc)
29
30 root = tkinter.Tk()
31 root.title("미로를 칠한다냥")
32 root.bind("<KeyPress>", key_down)
33 root.bind("<KeyRelease>", key_up)
34 canvas = tkinter.Canvas(width=800, height=560,
bg="white")
35 canvas.pack()
36
```

우선 캐릭터 삭제함
다시 캐릭터를 화면에 표시함

0.3초 후 다시 실행할 함수 지정

윈도우 객체 생성

윈도우 타이틀 설정

bind() 명령으로 키를 누를 때 실행할 함수 정의

bind() 명령으로 키를 뗄 때 실행할 함수 정의

캔버스 컴포넌트 생성

캔버스 컴포넌트 배치

6. 게임 완성하기

```
37 maze = [  
38     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
39     [1, 0, 0, 0, 0, 0, 1, 0, 0, 1],  
40     [1, 0, 1, 1, 0, 0, 1, 0, 0, 1],  
41     [1, 0, 0, 1, 0, 0, 0, 0, 0, 1],  
42     [1, 0, 0, 1, 1, 1, 1, 1, 0, 1],  
43     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],  
44     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  
45 ]  
46 for y in range(7):  
47     for x in range(10):  
48         if maze[y][x] == 1:  
49             canvas.create_rectangle(x * 80, y * 80,  
50                                     x * 80 + 79, y * 80 + 79, fill="skyblue", width=0)  
51 img = tkinter.PhotoImage(file="mimi_s.png")  
52 canvas.create_image(mx * 80 + 40, my * 80 + 40,  
53                     image=img, tag="MYCHR")  
54 main_proc()  
55 root.mainloop()
```

리스트로 미로 정의

반복, y: 0→1→2→3→4→5→6

반복, x: 0→1→2→3→4→5→6→7→8→9

maze[y][x]가 1, 즉, 벽이라면

회색 사각형 그림

캐릭터 이미지를 변수 img에 로딩

캔버스에 이미지 표시

main_proc() 함수 실행

윈도우 표시

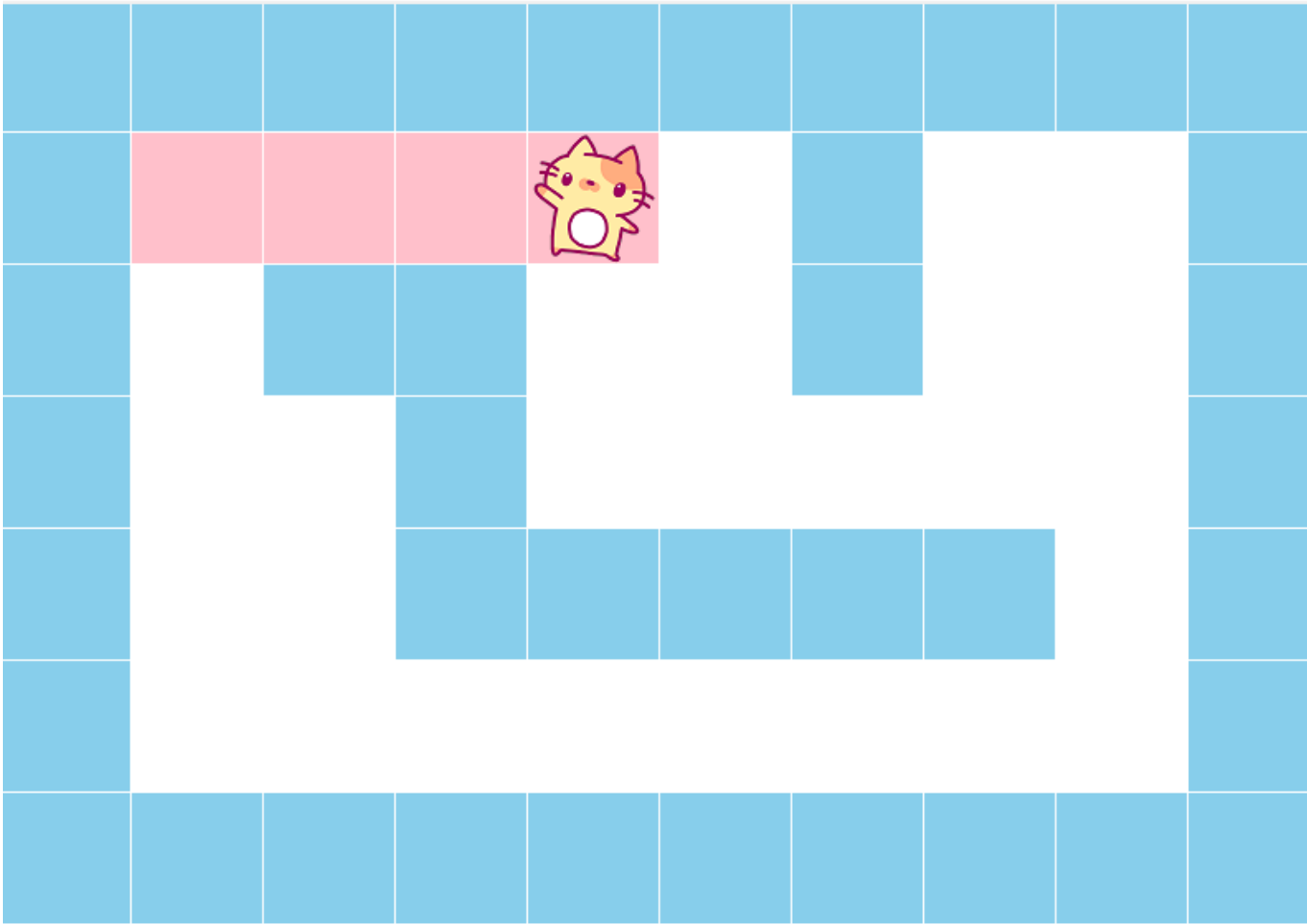
6. 게임 완성하기

■ 리스트 값 바꾸기

- 이 프로그램을 실행하고 캐릭터를 이동하면 길이 분홍색으로 칠해집니다.

미로를 칠한다냥

— □ ×



■ 리스트 값 바꾸기

- 23 ~25번 행에서 바닥을 칠하는 처리를 수행합니다.
- If 구문으로 캐릭터가 위치한 곳의 리스트 값을 확인하고, 값이 0이면 2로 바꾼 뒤 해당 위치를 분홍색으로 칠합니다.
- 26번 행에서 캐릭터를 `delete()` 명령으로 삭제한 뒤 27번 행에서 `create_image()` 명령으로 다시 그립니다.
- `delete()` 명령은 그려진 도형이나 이미지에 붙은 태그를 인수로 지정하면 그 도형이나 이미지를 삭제합니다.
- 이전 프로그램에서 사용했던 `coords()` 명령으로 캐릭터가 분홍으로 겹쳐 칠해져 보이지 않도록 캐릭터를 다시 그립니다.

6. 게임 완성하기

■ 게임 클리어 판정하기

- 다음으로 모든 바닥이 칠해졌는지 판정하는 처리를 추가합니다.
- 판정은 if 구문으로 수행합니다.
- 다음 프로그램을 입력하고 파일 이름을 붙여 저장한 뒤 실행해 봅니다.
- 예제 ex0706_2.py

```
1 import tkinter
2 import tkinter.messagebox
3
4 key = ""
5 def key_down(e):
6     global key
7     key = e.keysym
8 def key_up(e):
9     global key
10    key = ""
11
```

tkinter 모듈 импорт
tkinter.messagebox 모듈 импорт

키 이름을 입력할 변수 선언
키를 눌렀을 때 실행할 함수 정의
key를 전역 변수로 선언
누른 키 이름을 key에 대입

키를 눌렀다 뗐을 때 실행할 함수 정의
key를 전역 변수로 선언
key에 빈 문자열 입력

6 게임 완성하기

```
12 mx = 1
13 my = 1
14 yuka = 0
15 def main_proc():
16     global mx , my , yuka
17     if key == "Up" and maze[my-1][mx] == 0:
18         my = my - 1
19     if key == "Down" and maze[my+1][mx] == 0:
20         my = my + 1
21     if key == "Left" and maze[my][mx-1] == 0:
22         mx = mx - 1
23     if key == "Right" and maze[my][mx+1] == 0:
24         mx = mx + 1
25     if maze[my][mx] == 0:
26         maze[my][mx] = 2
27         yuka = yuka + 1
28         canvas.create_rectangle(mx * 80, my * 80,
mx * 80 + 79, my * 80 + 79, fill="pink", width=0)
29         canvas.delete("MYCHR")
30         canvas.create_image(mx * 80+40 , my * 80 + 40,
image=img, tag ="MYCHR")
```

캐릭터 가로 방향 위치를 관리하는 변수
캐릭터 세로 방향 위치를 관리하는 함수
칠해진 칸을 세는 함수

실시간 처리 수행 함수 정의

mx, my, yuka를 전역 변수로 선언

방향키[↑]을 눌렀을 때 위가 통로라면

my 값 1 감소

방향키[↓]을 눌렀을 때 아래가 통로라면

my 값 1 증가

방향키[←]를 눌렀을 때 왼쪽이 통로라면

mx 값 1 감소

방향키[→]를 눌렀을 때 오른쪽이 통로라면

mx 값 1 증가

캐릭터가 있는 장소가 통로라면

리스트 값 2로 변경

칠한 회수 1 증가

해당 위치를 분홍색으로 칠함

우선 캐릭터를 삭제함

다시 캐릭터를 화면에 표시함

6. 게임 완성하기

■ 게임 클리어 판정하기

■ 예제 ex0706_2.py

```
31     if yuka == 30:
32         canvas.update()
33         tkinter.messagebox.showinfo("축하합니다!", "
모든 바닥을 칠했습니다!")
34     else:
35         root.after(300, main_proc)
36
37 root = tkinter.Tk()
38 root.title("미로를 칠한다냥")
39 root.bind("<KeyPress>", key_down)
40 root.bind("<KeyRelease>", key_up)
41 canvas = tkinter.Canvas(width=800, height=560,
bg="white")
42 canvas.pack()
43
```

30개 칸을 모두 칠했다면
캔버스 업데이트
클리어 메시지 표시

그렇지 않다면
0.3초 후 다시 실행할 함수 지정

윈도우 객체 생성
윈도우 타이틀 설정
bind() 명령으로 키를 누를 때 실행할 함수 정의
bind() 명령으로 키를 뗄 때 실행할 함수 정의
캔버스 컴포넌트 생성

캔버스 컴포넌트 배치

6. 게임 완성하기

```
44 maze = [  
45     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
46     [1, 0, 0, 0, 0, 0, 1, 0, 0, 1],  
47     [1, 0, 1, 1, 0, 0, 1, 0, 0, 1],  
48     [1, 0, 0, 1, 0, 0, 0, 0, 0, 1],  
49     [1, 0, 0, 1, 1, 1, 1, 1, 0, 1],  
50     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],  
51     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  
52 ]  
  
53 for y in range(7):  
54     for x in range(10):  
55         if maze[y][x] == 1:  
56             canvas.create_rectangle(x * 80, y * 80,  
57                                     x * 80 + 79, y * 80 + 79, fill="skyblue", width=0)  
58  
59 img = tkinter.PhotoImage(file="mimi_s.png")  
60 canvas.create_image(mx * 80 + 40, my * 80 + 40,  
61                     image=img, tag="MYCHR")  
62 main_proc()  
63 root.mainloop()
```

리스트로 미로 정의

반복, y: 0→1→2→3→4→5→6

반복, x: 0→1→2→3→4→5→6→7→8→9

maze[y][x]가 1, 즉, 벽이라면

회색 사각형 그림

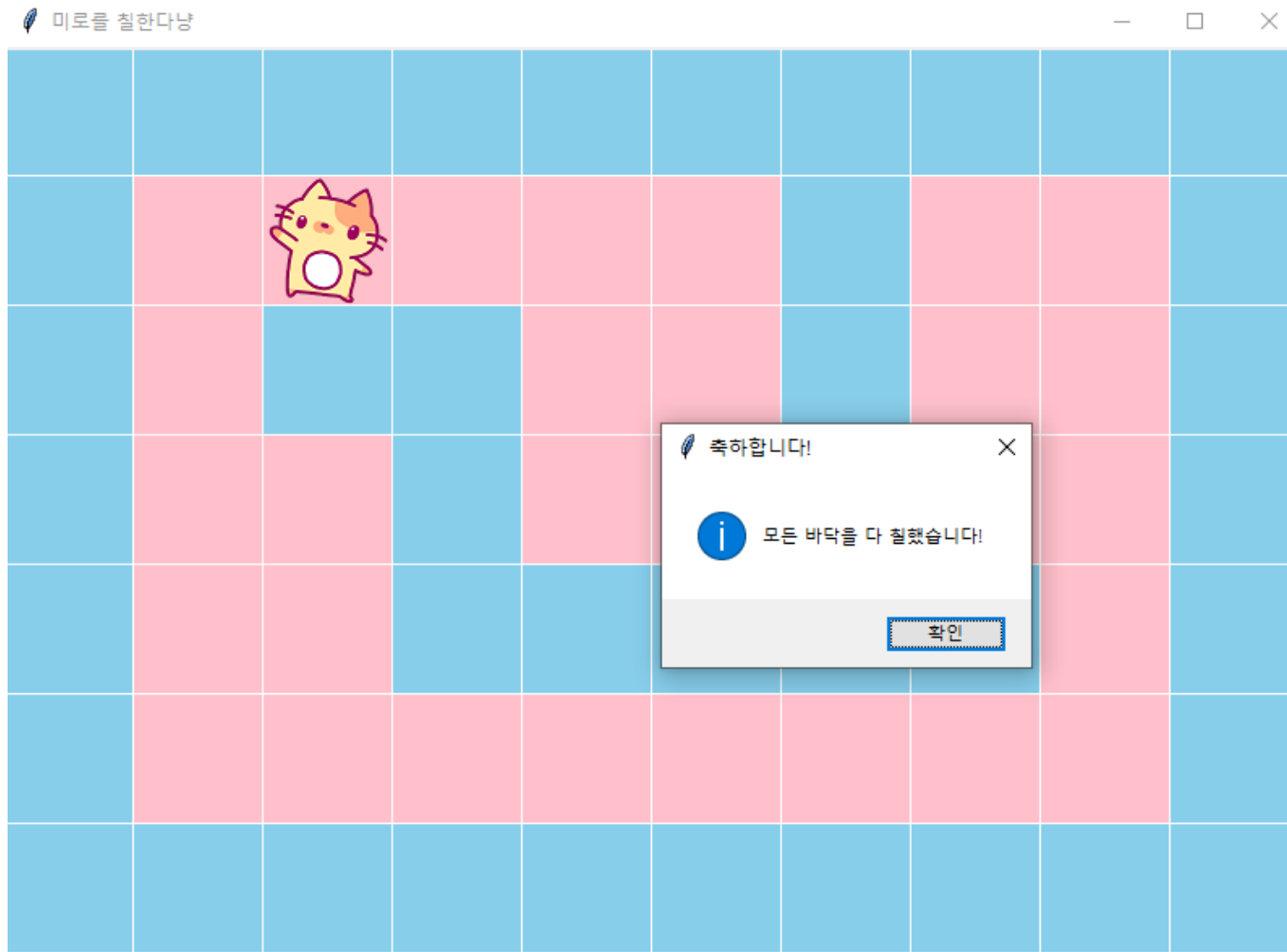
캐릭터 이미지를 변수 img에 로딩
캔버스에 이미지 표시

main_proc() 함수 실행
윈도우 표시

6. 게임 완성하기

■ 게임 클리어 판정하기

- 이 프로그램을 실행하고 모든 길을 칠하면 클리어 메시지가 표시됩니다.



6. 게임 완성하기

■ 게임 클리어 판정하기

- 이 미로의 바닥은 모두 30칸입니다.
- 27번 행에서 색을 칠한 칸을 세고 31번 행 if 구문에서 그 값이 30이 되었다면 클리어 메시지를 표시합니다.
- 32번 행 `canvas.update()`는 캔버스 상 표시를 변경하기 위해 넣은 것입니다.
- `update()` 명령이 없으면 PC에 따라 가장 마지막에 색을 칠한 칸을 그리지 않은 채 메시지 박스를 표시합니다.
- 칠한 칸의 수가 30이 아닌 동안에는 35번 행 `after()` 명령으로 실시간 처리를 계속 수행합니다.

6. 게임 완성하기

■ 다시 시작 처리 추가하기

- 칸을 잘못 칠한 경우 처음부터 다시 시작할 수 있다면 편리합니다.
- 왼쪽 'shift' 키를 누르면 처음으로 되돌릴 수 있도록 프로그램을 수정합니다.
- 예제 ex0706_3.py

```
1 import tkinter
2 import tkinter.messagebox
3
4 key = ""
5 def key_down(e):
6     global key
7     key = e.keysym
8 def key_up(e):
9     global key
10    key = ""
11
12 mx = 1
13 my = 1
14 yuka = 0
```

tkinter 모듈 импорт

tkinter.messagebox 모듈 импорт

키 이름을 입력할 변수 선언

키를 눌렀을 때 실행할 함수 정의

key를 전역 변수로 선언

누른 키 이름을 key에 대입

키를 눌렀다 떼을 때 실행할 함수 정의

key를 전역 변수로 선언

key에 빈 문자열 입력

캐릭터 가로 방향 위치를 관리하는 변수

캐릭터 세로 방향 위치를 관리하는 함수

칠해진 칸을 세는 함수

6. 게임 완성하기

■ 다시 시작 처리 추가하기

```
15 def main_proc():
16     global mx, my, yuka
17     if key == "Shift_L" and yuka > 1:
18         canvas.delete("PAINT")
19         mx = 1
20         my = 1
21         yuka = 0
22         for y in range(7):
23             for x in range(10):
24                 if maze[y][x] == 2:
25                     maze[y][x] = 0
26     if key == "Up" and maze[my - 1][mx] == 0:
27         my = my - 1
28     if key == "Down" and maze[my + 1][mx] == 0:
29         my = my + 1
30     if key == "Left" and maze[my][mx - 1] == 0:
31         mx = mx - 1
32     if key == "Right" and maze[my][mx + 1] == 0:
33         mx = mx + 1
```

실시간 처리 수행 함수 정의

mx, my, yuka를 전역 변수로 선언

왼쪽 Shift 키를 눌렀고 2칸 이상 칠했다면

칠해진 칸 삭제

mx에 1 대입

my에 1 대입(캐릭터를 초기 위치로 되돌림)

yuka에 0 대입

2중 반복, 외측 for 구문

내측 for 구문

칠해진 칸이 있다면

값을 0(칠하지 않은 상태)으로

방향키[↑]을 눌렀을 때 위가 통로라면

my 값 1 감소

방향키[↓]을 눌렀을 때 아래가 통로라면

my 값 1 증가

방향키[←]를 눌렀을 때 왼쪽이 통로라면

mx 값 1 감소

방향키[→]를 눌렀을 때 오른쪽이 통로라면

mx 값 1 증가

6. 게임 완성하기

■ 다시 시작 처리 추가하기

■ 예제 ex0706_3.py

```
34     if maze[my][mx] == 0:
35         maze[my][mx] = 2
36         yuka = yuka + 1
37         canvas.create_rectangle(mx * 80, my *
80, mx * 80 + 79, my * 80 + 79, fill="pink",
width=0, tag="PAINT")
38     canvas.delete("MYCHR")
39     canvas.create_image(mx * 80 + 40, my * 80 +
40, image=img, tag="MYCHR")
40     if yuka == 30:
41         canvas.update()
42         tkinter.messagebox.showinfo("축하합니다!",
"모든 바닥을 칠했습니다!")
```

캐릭터가 있는 장소가 통로라면
리스트 값을 2로 변경
칠한 회수 1 증가
해당 위치를 분홍색으로 칠함

우선 캐릭터 삭제함
다시 캐릭터를 화면에 표시함

30개 칸을 모두 칠했다면
캔버스 업데이트
클리어 메시지 표시

6. 게임 완성하기

다시 시자 처리 추가하기

```
43     else:
44         root.after(300, main_proc)
45
46 root = tkinter.Tk()
47 root.title("바닥을 칠한다냥")
48 root.bind("<KeyPress>", key_down)
49 root.bind("<KeyRelease>", key_up)
50 canvas = tkinter.Canvas(width=800, height=560,
51                          bg="white")
52 canvas.pack()
53
54 maze = [
55     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
56     [1, 0, 0, 0, 0, 0, 1, 0, 0, 1],
57     [1, 0, 1, 1, 0, 0, 1, 0, 0, 1],
58     [1, 0, 0, 1, 0, 0, 0, 0, 0, 1],
59     [1, 0, 0, 1, 1, 1, 1, 1, 0, 1],
60     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
61     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

그렇지 않다면

0.3초 후 다시 실행할 함수 지정

윈도우 객체 생성

윈도우 타이틀 설정

bind() 명령으로 키를 누를 때 실행할 함수 정의

bind() 명령으로 키를 뗄 때 실행할 함수 정의

캔버스 컴포넌트 생성

캔버스 컴포넌트 배치

리스트로 미로 정의

6. 게임 완성하기

■ 다시 시작 처리 추가하기

■ 예제 ex0706_3.py

```
62 for y in range(7):
63     for x in range(10):
64         if maze[y][x] == 1:
65             canvas.create_rectangle(x * 80, y *
66                                     80, x * 80 + 79, y * 80 + 79, fill="skyblue",
67                                     width=0)
68 img = tkinter.PhotoImage(file="mimi_s.png")
69 canvas.create_image(mx * 80 + 40, my * 80 + 40,
70                    image=img, tag="MYCHR")
71 main_proc()
72 root.mainloop()
```

반복, y: 0→1→2→3→4→5→6

반복, x: 0→1→2→3→4→5→6→7→8→9

maze[y][x]가 1, 즉, 벽이라면

회색 사각형 그림

캐릭터 이미지를 변수 img에 로딩
캔버스에 이미지 표시

main_proc() 함수 실행
윈도우 표시

- 왼쪽 'shift' 키를 누르면 최초 상태로 돌아가는 것을 확인할 수 있습니다.

■ 이번 장에서 만든 게임에 관해

- 판매용 게임이라면 타이틀 화면은 물론 여러 스테이지가 있고, 클리어한 스테이지 데이터가 저장되는 등 다양한 기능이 있을 것입니다.
- 현재 시점에서 이런 기능들을 구현하려고 하면 매우 어려운 프로그램이 되므로 미로를 칠하는 게임은 이 정도에서 마무리합니다.
- 이 후 개발을 위해 타이틀 화면을 표시하는 방법과 스테이지 수를 늘리는 방법을 설명합니다.

6. 게임 완성하기

■ 타이틀 화면 추가하기

- 어떤 처리를 수행하고 있는지 관리하는 변수를 준비합니다.
- 이 변수를 인덱스라고 부릅니다.
- 예를 들어, index라는 변수를 준비하고 그 값이 1인 경우에는 타이틀 화면을 처리하고, 2인 경우에는 게임을 처리하도록 합니다.
- 그리고 다음과 같은 프로그램을 작성합니다.

```
if index == 1:
    if 스페이스 키를 눌렀다면:
        게임에 필요한 변수 초기화
        타이틀 표시 삭제
        index에 2 대입
elif index == 2:
    게임 처리
```

- if ~ elif는 여러 조건을 순서대로 확인하는 조건 분기 명령입니다.
- 이 프로그램에서는 'if index == 1'로 index 값이 1인지 확인한 뒤 1이 아니면 'elif index == 2'로 index 값이 2인지 확인합니다.

6. 게임 완성하기

■ 스테이지 수 추가하기

- 스테이지 수를 관리하는 변수를 준비합니다.
- 예를 들면, stage라는 변수를 준비하고 클리어한 경우에는 값을 1 증가시키고, stage 값에 따라 미로 데이터(리스트)를 바꾸고, 캐릭터 위치를 초기 위치로 변경해서 게임을 다시 시작하도록 합니다.

7. 디지털 사진 액자 만들기

- 스마트폰이나 디지털 카메라로 촬영한 사진(디지털 데이터)을 표시하는 디스플레이 장비를 디지털 포토 프레임(디지털 액자)이라고 부릅니다.
- 이 장에서 학습한 실시간 처리를 이용해 디지털 포토 프레임 프로그램을 만들어 봅니다.
- 이미지 데이터를 계속해서 순서대로 표시하는 파이썬 프로그램을 소개합니다.

7. 디지털 사진 액자 만들기

■ ex0707.py

```
1 import tkinter
2
3 pnum = 0
4 def photograph():
5     global pnum
6     canvas.delete("PH")
7     canvas.create_image(400, 300, image=photo
8     [pnum], tag="PH")
9     pnum = pnum + 1
10    if pnum >= len(photo):
11        pnum = 0
12    root.after(7000, photograph)
```

tkinter 모듈 импорт

표시할 이미지 파일 번호 관리 변수
실시간 처리를 수행할 함수 정의
pnum을 전역 변수로 선언
이미지 삭제
이미지 표시

다음 이미지 번호 계산
가장 마지막 이미지까지 수행했다면
첫 번째 번호로 되돌림
7초 후 이 함수 다시 실행

7. 디지털 사진 액자 만들기

■ ex0707.py

```
13 root = tkinter.Tk()
14 root.title("디지털 액자")
15 canvas = tkinter.Canvas(width=800, height=600)
16 canvas.pack()
17 photo = [
18     tkinter.PhotoImage(file="cat00.png"),
19     tkinter.PhotoImage(file="cat01.png"),
20     tkinter.PhotoImage(file="cat02.png"),
21     tkinter.PhotoImage(file="cat03.png")
22 ]
23 photograph()
24 root.mainloop()
```

윈도우 객체 생성
윈도우 타이틀 설정
캔버스 컴포넌트 생성
캔버스 배치
리스트로 이미지 파일 정의

실시간 처리 수행 함수 호출
윈도우 표시

7. 디지털 사진 액자 만들기

■ ex0707.py

- 9번 행의 `len()` 명령으로 `()` 안에 입력한 리스트의 엘리먼트 수를 알 수 있습니다.
- 17~22번 행에서 `photo`라는 리스트에 네 종류의 이미지 파일을 정의했으므로 `len(photo)` 값은 4가 됩니다.
- 9~10번 행의 조건 분기에서 가장 마지막 이미지를 표시한 후 다시 처음 이미지부터 표시되도록 하기 위해 `len()` 명령을 사용합니다.
- 이렇게 하면 리스트에 이미지 파일명을 추가하는 것만으로 프로그램의 다른 부분을 수정하지 않고 모든 이미지를 표시한 뒤, 가장 처음 이미지부터 다시 표시할 수 있습니다.

7. 디지털 사진 액자 만들기

■ ex0707.py

- 이 프로그램을 실행하면 다음과 같이 고양이 그림들이 순서대로 표시됩니다.



- 몇 초마다 이미지를 변경할 것인지는 11번 행 `after()` 명령의 인수로 지정합니다.
- 취미 사진이나 가족 사진, 좋아하는 슬라이드 등으로 오리지널 디지털 액자를 만들어 보기 바랍니다.



Thank You
