

# 기술 (Technology)

## 9.1 화재경보기 작동하기

프로그램 p09-01 ... 화재경보기 작동하기

관련 학습 ... 터틀의 키보드 이벤트 처리

변수의 범위와 지역변수, 전역변수

## 9.2 7세그먼트 LED를 이용한 숫자 표시하기

### 9.2.1 7세그먼트 형식의 숫자 이미지 표시하기

프로그램 p09-02 ... 7세그먼트 형식의 숫자 이미지 표시하기

관련 학습 ... 터틀의 모양 등록 및 변경

터틀의 모양으로 등록이 가능한 이미지의 종류

### 9.2.2 7세그먼트 개별 LED를 이용한 숫자 표시하기

프로그램 p09-03 ... 7세그먼트 단자를 이용한 LED 숫자 표시하기

관련 학습 ... 리스트로 1차원 배열과 2차원 행렬 표현하기

Thinking!

잠깐! Coding

**Coding! Programming**



1. 다음 프로그램의 결과를 예측하여 작성해보자.

```
a = b = c = 1

def func():
    a = b = c = 2
    print("func:", a, b, c)

print("main:", a, b, c)
func()
print("main:", a, b, c)
```



함수 내부에서 변수를 global로 선언하지 않고 전역변수와 같은 이름의 지역변수가 사용되면 지역 변수를 우선하여 참조한다. 지역변수는 함수의 실행이 종료되면 삭제되므로 함수 외부에서는 존재하지 않는다.

2. 다음 프로그램의 결과를 예측하여 작성해보자.

```
a = b = c = 1

def func():
    global a, b, c
    a = b = c = 2
    print("func:", a, b, c)

print("main:", a, b, c)
func()
print("main:", a, b, c)
```



함수 내부에서 변수를 global로 선언하면 전역변수를 참조하게 되고, 함수 내부에서 전역변수의 값을 변동할 경우 함수를 종료하더라도 변경된 값은 그대로 유지된다.

### 3. 다음 프로그램의 결과를 예측하여 작성해보자.

```
a = b = c = 1

def func1():
    a = b = c = 2
def func2():
    global a, b
    a = b = 3
    c = 3

print(a, b, c)
func1()
print(a, b, c)
func2()
print(a, b, c)
```



함수 내부에서 변수를 global로 선언하지 않으면 지역변수이므로 함수 실행이 종료되면 해당 지역 변수는 삭제되어 함수 외부에서는 존재하지 않고 전역변수만 남게 된다. global로 선언을 한 변수는 전역변수로 선언되어 함수 내에서 함수 외부의 전역변수 값을 변경할 수 있다.

4. 두 정수를 전달받아 나누기하여 전역변수를 통해 몫과 나머지를 반환하는 `div_qr()` 함수를 선언해보자. 두 정수를 입력받아 `div_qr()` 함수를 호출해보자.

정수1 : 7

정수2 : 3

몫: 2 나머지: 1



몫과 나머지에 해당하는 전역변수를 미리 선언해둔 후, 함수 내부에서 `global` 예약어로 전역변수로 선언하고 값을 대입한다.

5. ‘Basic Coding 4번 문제’와 같이 전역변수를 통하여 결과를 반환받을 경우의 문제점은 무엇인지 생각해보자. 그리고 ‘7장 Coding? Programming!’의 ‘Enhancement Coding 6번 문제’를 참고하여, 전역변수를 사용하지 않고 튜플 방식으로 몫과 나머지를 반환하도록 `div_qr()` 함수를 수정하여 호출해보자.

정수1 : 7

정수2 : 3

몫: 2 나머지: 1



함수 내부에서 튜플 방식을 이용하여 반환할 경우 `return (몫, 나머지)` 형식으로 반환하고, 함수를 호출하여 함수의 결과를 반환받을 때 `(몫, 나머지) = div_qr()` 형식으로 작성한다.

6. 리스트를 이용한 1차원 배열 형태로 21, 7, 43, 65, 2, 8, 72, 52, 9의 값을 선언하고 해당 리스트 내의 모든 값을 출력해보자.

```
21 7 43 65 2 8 72 52 9
```



리스트를 이용한 1차원 배열 선언은 `data = [ 21, ..., 9 ]` 형태로 선언한다. `for` 문을 이용한 반복에서 리스트의 크기는 `len(data)`을 통하여 계산하며, 리스트 내의 원소는 `data[i]` 형태로 참조한다.

7. 'Basic Coding 6번 문제'를 참고하여, 리스트를 이용한 1차원 배열 형태로 21, 7, 43, 65, 2, 8, 72, 52, 9의 값을 선언하고 해당 리스트 내의 모든 값을 역순으로 출력해보자.

```
9 52 72 8 2 65 43 7 21
```



리스트를 이용한 1차원 배열 선언은 `data = [ 21, ..., 9 ]` 형태로 선언한다. `for` 문을 이용한 반복에서 범위를 `range(len(data)-1, -1, -1)`로 정하여 반복한다.



8. 리스트를 이용한 2차원 배열 형태로 1행에 21, 7, 43, 65를 2행에 2, 8, 72, 52의 값을 선언하고 해당 리스트 내의 값을 행과 열의 형태에 맞게 출력해보자.

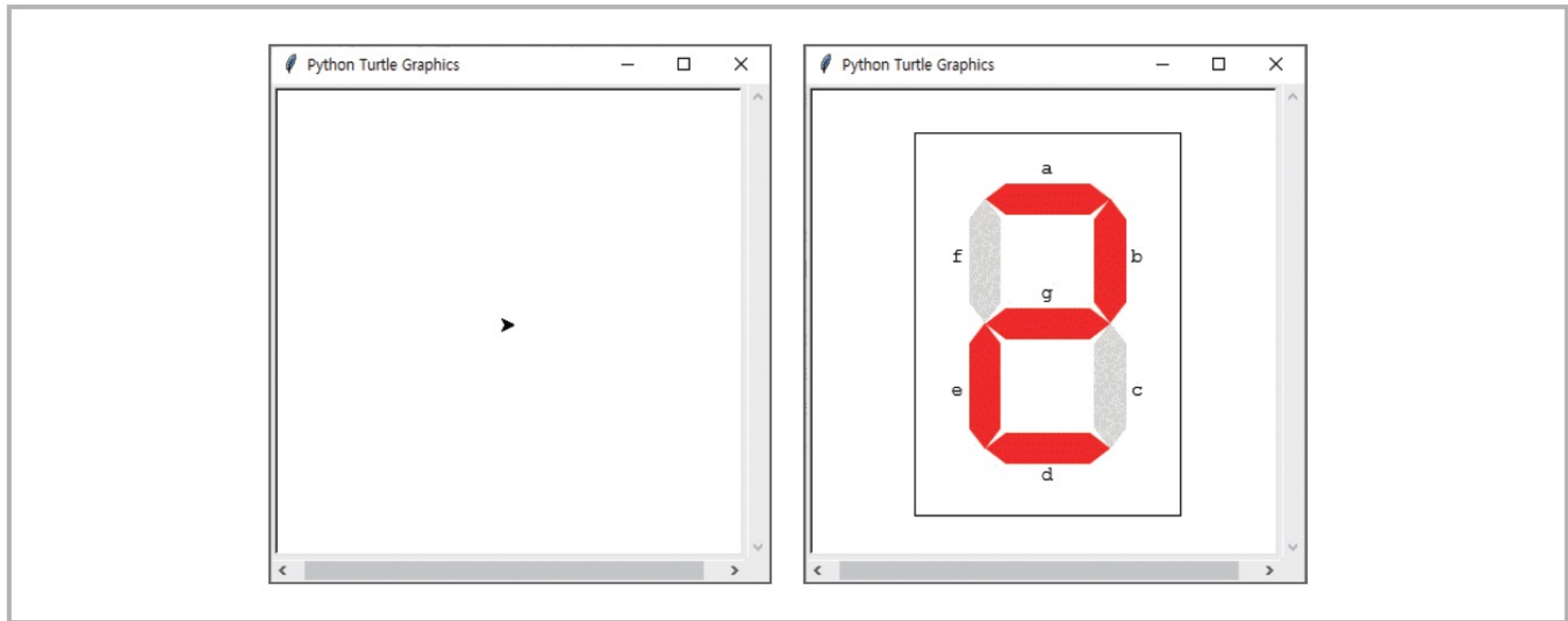
```
21 7 43 65
```


```
2 8 72 52
```



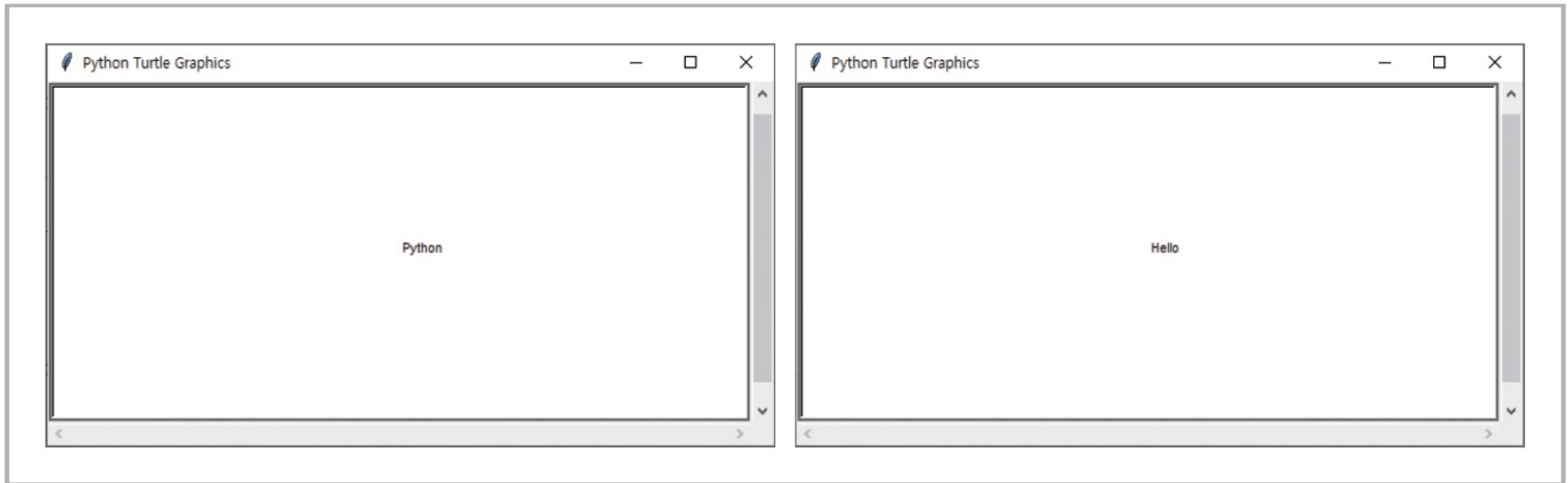
리스트를 이용한 2차원 배열 선언은 `data = [[21, ...], [2, ...]]` 형태로 선언한다. `for` 문을 이용한 반복에서 리스트의 행의 크기는 `len(data)`, 열의 크기는 `len(data[0])`을 통해 계산하며, 리스트 내의 원소는 `data[i][j]` 형태로 참조한다.

9. default\_shape 리스트의 원소로 “arrow”, “turtle”, “circle”, “square”, “triangle”, “classic”을 선언하고, user\_shape 리스트의 원소로 “7s0.gif”, “7s1.gif”, “7s2.gif”를 선언해보자. 그리고 for 문을 이용하여 default\_shape 리스트의 원소값을 이용하여 터틀의 모양을 순서대로 변경하고, user\_shape 리스트의 원소값을 이용하여 터틀의 모양을 순서대로 등록하고 변경해보자.



 터틀의 addshape() 함수를 통해 터틀의 모양을 등록하고, 등록된 터틀의 모양은 shape() 함수를 통해 변경한다.

10. 터틀 스크린에서 p 키를 누르면 “Python”, h 키를 누르면 “Hello”를 중앙 지점(x:0, y:0)에 출력하고, c 키를 누르면 스크린 내의 모든 흔적을 지워보자. 단, 터틀은 터틀 스크린에서 숨긴다.



 p 키, h 키, c 키를 눌렀을 때 처리할 콜백 함수를 `key_p()`, `key_h()`, `key_c()` 함수로 선언한다. 중앙 지점(x:0, y:0) 위치로의 이동은 터틀의 `home()` 함수 또는 `goto()` 함수를 사용한다.

1. 리스트를 이용한 1차원 배열 형태로 21, 7, 43, 65, 2, 8, 72, 52, 9의 값을 선언하고, 정수를 입력받아 리스트 내의 값을 검색해보자. 값을 찾으면 “위치 : n”으로 출력하고, 찾지 못하면 “찾지 못함”이라고 출력해보자.

찾을 값 : 72

위치 : 6

찾을 값 : 77


찾지 못함



값을 찾았는지 찾지 못하였는지를 판별할 때, 값을 찾으면 리스트의 0~크기-1 범위에 찾은 위치가 있게 되며, 반복이 종료된 후 위치가 리스트의 크기 값에 해당하므로 이 경우는 찾지 못한 경우이다.

2. ‘Enhancement Coding 1번 문제’를 참고하여, 리스트를 이용한 1차원 배열 형태로 21, 7, 43, 65, 2, 8, 72, 52, 9의 값을 선언하고, 정수를 입력받아 해당 정수의 배수에 해당하는 값을 검색해보자. 값을 찾으면 “위치 : n, 값 : m” 형식으로 출력하고, 찾지 못하면 “찾지 못함”이라고 출력해보자.

```
찾을 배수 : 10
찾지 못함
찾을 배수 : 3
위치 : 0 값 : 21
위치 : 6 값 : 72
위치 : 8 값 : 9
```

 입력한 값으로 리스트 내의 값을 % 연산한 결과가 0이면 입력한 값의 배수에 해당하는 숫자이다.

3. 리스트를 이용한 2차원 배열 형태로 1행에 21, 7, 43, 65를 2행에 2, 8, 72, 52의 값을 선언하고, 정수를 입력받아 리스트 내의 값을 검색해보자. 값을 찾으면 “위치 : r행 c열”로 출력하고, 찾지 못하면 “찾지 못함”이라고 출력해보자.

찾을 값 : 72


위치 : 2행 3열

찾을 값 : 43

위치 : 1행 3열

찾을 값 : 77

찾지 못함

 값을 찾았는지 찾지 못하였는지를 판별할 때, 값을 찾으면 리스트의 ‘0~크기-1’ 행 범위와 ‘0~data[0] 크기-1’ 열 범위에 찾은 위치가 있게 되며, 반복이 종료된 후 해당 행 범위와 열 범위 내에 위치하지 않으면 찾지 못한 것이다.

4. 리스트를 이용한 1차원 배열 형태로 3, 2, 7, 4, 1의 값을 리스트 a에 선언하고, 8, 3, 1, 2, 5, 6의 값을 리스트 b에 선언해보자. 그리고 두 리스트 a와 b의 공통원소를 찾아 출력해보자.

공통원소 : 3 2 1



바깥쪽 for 문은 `range(len(a))`로 반복하고, 안쪽 for 문은 `range(len(b))`로 반복하면서 `a[i]`의 값과 `b[j]`의 값이 같으면 공통원소이므로 값을 출력한다.

5. 리스트를 이용한 1차원 배열 형태로 0, 1의 값을 리스트 a에 선언하고, 0, 1의 값을 리스트 b에 선언해보자. 그리고 두 리스트 a와 b 각 원소에 대한 AND OR 연산 결과를 출력해보자.

a	b	and	or
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1



바깥쪽 for 문은 `range(len(a))`로 반복하고, 안쪽 for 문은 `range(len(b))`로 반복하면서 `a[i]`의 값과 `b[j]`의 값에 대해 and 연산, or 연산을 한다.



6. 'Basic Coding 8번 문제'를 참고하여, 리스트를 이용한 2차원 배열 형태로 1행에 21, 7, 43, 65를 2행에 2, 8, 72, 52의 값을 선언하고 해당 리스트 내의 값을 전치행렬 형태로 출력해보자.

행렬

21 7 43 65

2 8 72 52

전치행렬

21 2

7 8

43 72


65 52



리스트를 이용한 2차원 배열 선언은 `data = [[21, ...], [2, ...]]` 형태로 선언한다. 전치행렬 형태로 출력하기 위하여, 바깥쪽 `for` 문을 `range(len(data[0]))`으로 반복하고, 안쪽 `for` 문을 `range(len(data))`로 반복한다.

7. 리스트를 이용한 1차원 배열 형태로 10, 20, 30, 40, 0의 값을 리스트 a에 선언하고, 각 값을 마지막 0의 위치에 모두 더해 출력해보자.

$$10 + 20 + 30 + 40 = 100$$

 리스트를 이용한 1차원 배열 선언은 `data = [10, 20, 30, 40, 0]` 형태로 선언한다. `for` 문을 `range(len(data)-1)`로 반복하면서 `data[len(data)-1]`의 값에 `data[i]`의 값들을 더한다.

8. 리스트를 이용한 2차원 배열 형태로 다음 표의 값들을 선언하고, 행의 합과 열의 합을 각각의 빈칸에 구하여 출력해보자.

10	20	30	40	
50	60	70	80	
90	100	110	120	
130	140	150	160	
170	180	190	200	

10 20 30 40 100  
 50 60 70 80 260  
 90 100 110 120 420  
 130 140 150 160 580  
 170 180 190 200 740  
 450 500 550 600 2100

10	20	30	40	
50	60	70	80	
90	100	110	120	
130	140	150	160	
170	180	190	200	



리스트를 이용한 2차원 배열 선언은 `data = [[10, ..., 0], [50, ..., 0], ..., [0, ..., 0]]` 형태로 선언한다. 바깥쪽 for 문을 `range(len(data)-1)`로 반복하고, 안쪽 for 문을 `range(len(data[0]-1))`로 반복하면서 각 행의 마지막 열에 해당 행의 각 열의 값들을 더한다. 마지막 행의 값들을 더하기 위하여 for 문을 `range(len(data[0]-1))`로 반복하면서 마지막 행의 마지막 열에 각 열의 값들을 더한다.

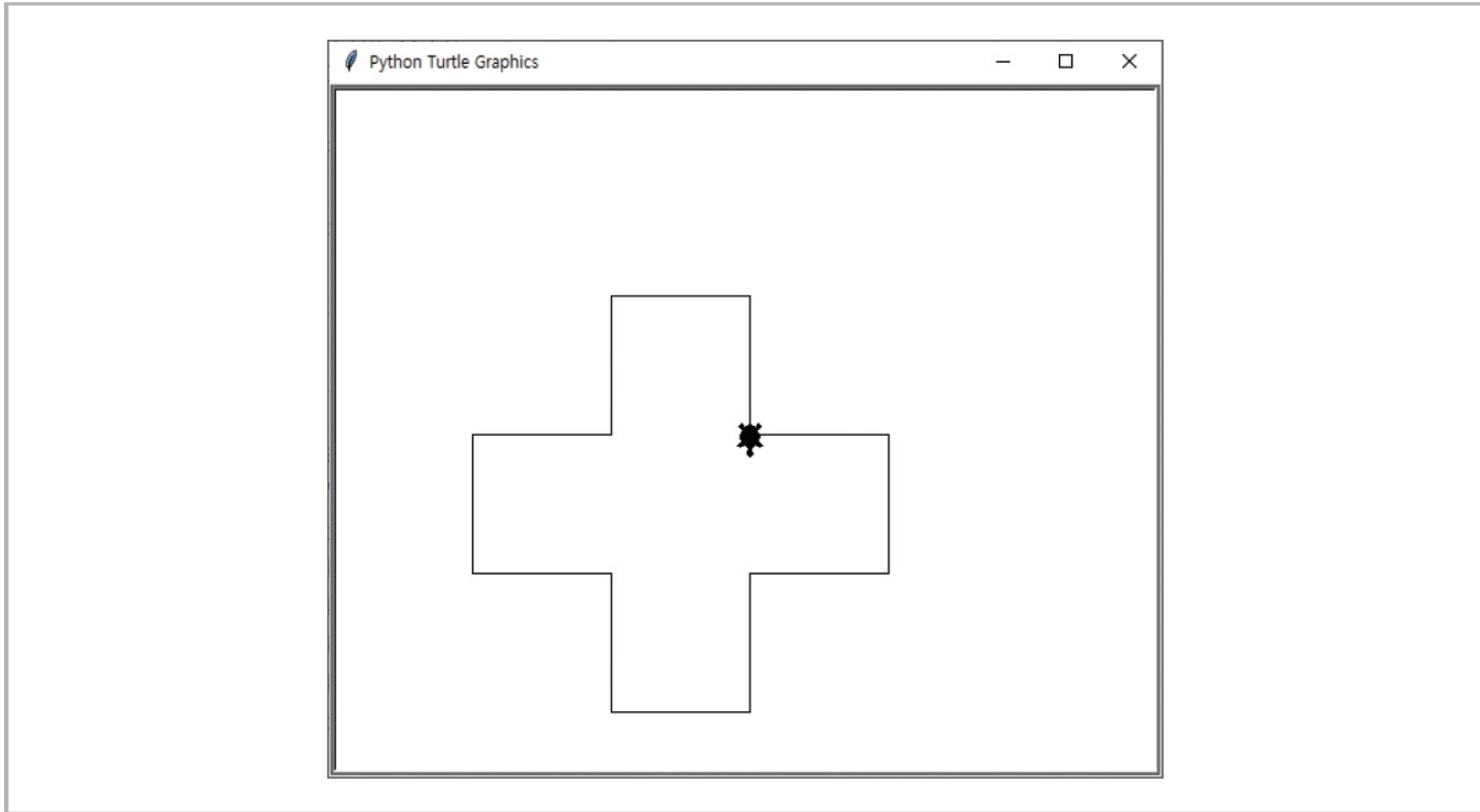
9. '프로그램 p09-03'의 공통 캐소드 타입에 의한 숫자 0~9에 해당하는 개별 LED 단자 설정 리스트 s7seg\_num의 내용을 공통 애노드 타입으로 변환하여 리스트 s7seg\_num\_anode에 저장해보자. 그리고 리스트 s7seg\_num과 s7seg\_num\_anode의 값을 행렬의 형태에 맞게 출력해보자.

s7seg_num	s7seg_num_anode
1 1 1 1 1 1 0	0 0 0 0 0 0 1
0 1 1 0 0 0 0	1 0 0 1 1 1 1
1 1 0 1 1 0 1	0 0 1 0 0 1 0
1 1 1 1 0 0 1	0 0 0 0 1 1 0
0 1 1 0 0 1 1	1 0 0 1 1 0 0
1 0 1 1 0 1 1	0 1 0 0 1 0 0
1 0 1 1 1 1 1	0 1 0 0 0 0 0
1 1 1 0 0 0 0	0 0 0 1 1 1 1
1 1 1 1 1 1 1	0 0 0 0 0 0 0
1 1 1 1 0 1 1	0 0 0 0 1 0 0



s7seg\_num\_anode = s7seg\_num와 같이 단순한 대입을 통해 리스트를 만들 경우 s7seg\_num\_anode의 원소 내용을 변경하면 s7seg\_num의 원소 내용도 함께 변경된다. 이 문제를 방지하기 위하여 import copy 후, s7seg\_num\_anode = copy.deepcopy(s7seg\_num)를 실행하면 s7seg\_num\_anode의 원소 내용이 변경되더라도 s7seg\_num은 영향을 받지 않는다.

10. 터틀 스크린에서 거북이를 네 방향의 방향키로 움직이면 해당 방향으로 거북이를 회전시키고 forward 방향으로 10만큼 움직이면서 선을 그려보자. 또한, r 키를 누르면 스크린 내의 모든 흔적을 지우고 스크린에서 터틀의 처음 위치인 중앙 지점(x:0, y:0)으로 이동한다.



터틀의 `setheading()` 함수를 통해 거북이를 네 방향으로 회전시킬 수 있으며, `setheading()` 함수의 인수로 0, 90, 180, 270이 주어지면 동(0), 서(90), 남(180), 북(270) 각 방향으로 회전한다. 중앙 지점(x:0, y:0) 위치로의 이동은 터틀의 `home()` 함수 또는 `goto()` 함수를 사용한다.