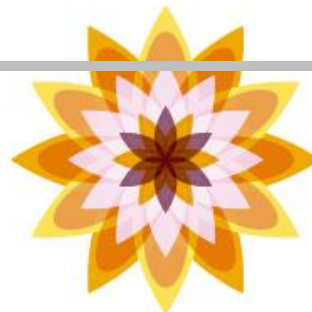
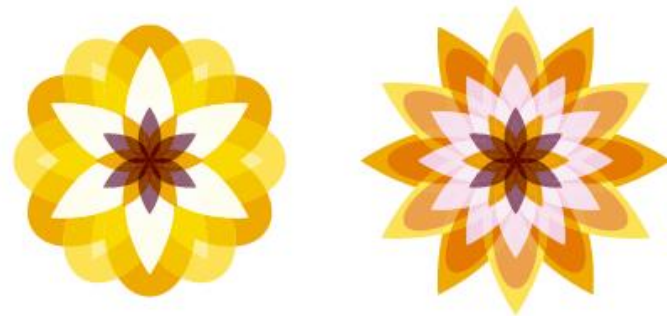


Chapter 05

빛방울 공장



1. 비내리기



새 파일

rain.py

```
1 import pygame, sys
2 from pygame.locals import *
3 pygame.init()
4 pygame.display.set_caption("rain")
5 screen = pygame.display.set_mode((1000,600))
6 clock = pygame.time.Clock()
7 rain_y = 0
8 rain_x = 400
9
10 while 1:
11     clock.tick(60)
12     for event in pygame.event.get():
13         if event.type == pygame.QUIT:
14             sys.exit()
15     screen.fill((255,255,255))
16     rain_y += 4
17     pygame.draw.line(screen,(0,0,0),(rain_x,rain_y),(rain_x,rain_y+5),1)
18
19     pygame.display.update()
```

실행

2. 빗방울 만들기

```
rain_y += 4
```

- 비는 아래로 떨어지므로, 변수를 이용해 y좌표를 바꿔야 합니다.
- 변수 이름을 rain_y라고 정했습니다.
- rain_y는 매 게임 루프마다 4씩 증가합니다.

2. 빗방울 만들기

```
pygame.draw.line(screen, (0,0,0), (rain_x, rain_y), (rain_x, rain_y+5), 1)
```

- 직선을 그리는 함수입니다.
- 이 함수는 우리가 이미 배운 원 그리기 또는 직사각형 그리기 함수와 비슷합니다
- 인자는 5개입니다.
- Screen
 - 직선을 그리는 장소를 가리킵니다. 바로 스크린이지요.
- (0, 0, 0)
 - RGB로 표시한 직선의 색입니다. 다른 색으로 바뀌도 됩니다.
- (rain_x, rain_y)
 - 직선이 시작되는 점의 좌표입니다.

2. 빗방울 만들기

```
pygame.draw.line(screen, (0,0,0), (rain_x, rain_y), (rain_x, rain_y+5), 1)
```

■ (rain_x, rain_y+5)

- 직선이 끝나는 점의 좌표입니다.
- 빗방울이 떨어지는 동안 rain_y의 값은 바뀌지만, rain_x의 값은 변하지 않습니다.
- 따라서 직선이 끝나는 점의 X좌표는 rain_x 그대로지만, y좌표에는 rain_y에 5픽셀만큼 더해야 합니다.
- 이렇게 하면 세로 길이가 5픽셀인 직선이 그려집니다.

■ 1

- 1은 직선의 두께입니다. 지금은 1로 했지만 넓힐 수도 있습니다.

3. 여기저기에서 떨어지는 빗방울

- rain_x의 값은 바뀌지 않습니다.
- 빗방울의 x좌표는 항상 같다는 말입니다.
- 따라서 빗방울은 항상 같은 곳에서 떨어지지요.
- 그럴 거면 rain_x라는 변수를 만들지 말고, 숫자 400이라고 써도 괜찮았을 것입니다.
- 빗방울이 여기저기에 떨어지도록 하려면 여덟 번째 줄의 rain_x = 400을 아래처럼 고쳐야 합니다.

```
rain_x = 400 random.randint(0,1000)
```

- randint() 함수는 괄호 안의 두 수 사이 임의의 수 하나를 불러옵니다.
- 양쪽 끝 숫자들도 포함하므로, 0이나 1000도 선택될 수 있습니다.
- randint()는 파이썬의 random 모듈 안에 저장돼 있습니다. (모듈은 함수 모음이라고 생각하면 됩니다.)

3. 여기저기에서 떨어지는 빗방울

- 하지만 우리가 아직 파이썬의 랜덤(random) 모듈을 가져오지 않았기 때문에, 이 코드는 작동하지 않습니다.
- 프로그램의 가장 첫 줄에서 모듈을 가져올 때 random 모듈도 가져오도록 고칩니다.

```
import pygame, sys, random
```

4. 클래스와 인스턴스

- 빗방울을 많이 만들려면 어떻게 해야 할까요?
- `rain_x1, rain_x2, rain_x3,` 처럼 계속 변수를 추가해서 빗방울을 많이 만들 수도 있습니다.
- 이렇게 많은 변수를 쓰면 `pygame.draw.line()` 함수도 그만큼 많이 써야겠지요.
- 이렇게 해도 되지만 시간이 많이 걸리고 매우 지루합니다.
- 우리는 이런 식으로 하지 않을 거예요.
- 대신 공장을 세울 것입니다. 프로그래밍에서는 클래스(class)라고 부릅니다.
- 클래스를 사용하면 복제본을 많이 만들 수 있습니다.
- 각 복제본은 클래스의 인스턴스(Instance)라고 불립니다.
- 빗방울 공장이라는 클래스에서 만들어지는 빗방울 하나하나가 각각 인스턴스입니다.
- 그러니 하나의 클래스에서 여러 인스턴스를 만들 수 있지요.
- 각각의 인스턴스인 빗방울들은 서로 다릅니다. 각각 다른 곳에서 떨어지게 할 수 있지요.

4. 클래스와 인스턴스

이 빗방울들이
빗방울 클래스에서 만든 인스턴스라는 거야?



인스턴스 빗방울이와...



5. 파이썬 프로그램의 기본 구조

- 클래스라는 개념을 배웠으니, 클래스가 있는 파이썬 프로그램의 기본 구조에 대해 알아보시다.

셋업(Set up)	모듈을 불러오고, pygame을 초기화시키고, 변수를 만듭니다.
클래스(Classes)	다양한 객체들을 제어합니다.
리스트(Lists)	모든 객체들의 움직임을 기록하는 곳입니다.
게임 루프(Game loop)	실제로 게임이 실행되는 곳입니다.

- 셋업 부분은 프로그램에 필요한 것들을 불러오고, 준비하는 곳입니다.
- 모듈을 불러오거나 import, 변수의 초기값을 정하거나, 이미지나 사운드 파일이 어디에 있는지 적어 두는 곳이지요.
- 들여쓰기는 하지 않습니다.

5. 파이썬 프로그램의 기본 구조

- 클래스는 다양한 객체를 정의하고, 그 객체가 사용하는 함수를 정의하는 곳입니다.
- 여러 클래스를 만들 때, 보통 순서는 상관없습니다.
- 빗방울 클래스를 먼저 쓰든, 구름 클래스를 먼저 쓰든 상관없다는 말입니다. (물론 예외도 있지만, 그럴 때는 따로 이야기해 줄게요.)
- 클래스 안의 함수는 모두 한 번씩 들여쓰지요.
- 함수 역시 어느 걸 먼저 쓰는지는 크게 중요하지 않습니다. (역시 또 예외는 있지만, 이것도 그때마다 따로 이야기해 줄게요.)
- 리스트 부분은 클래스의 인스턴스를 만드는 곳입니다.
 - 들여쓰기는 하지 않습니다.
- 게임 루프는 `while 1:`로 시작합니다.
- 앞에서 만든 함수들을 불러오고 실행시키는 곳이므로 들여쓰기에 특히 주의해야 합니다.

5. 파이썬 프로그램의 기본 구조

- 코드를 쓸 때 한 줄을 빈 줄로 띄워도 되고, 띄우지 않아도 됩니다.
- 파이썬은 줄 띄우기에는 민감하지 않습니다. 여러분 보기 좋은 대로 하세요.
- 옆쪽 음영 표시 된 코드를 추가하고, 취소선 표시된 곳은 삭제하여 빗방울 클래스를 만듭시다.

```
import pygame, sys, random
from pygame.locals import *
pygame.init()
pygame.display.set_caption("rain")
screen = pygame.display.set_mode((1000,600))
clock = pygame.time.Clock()
rain_y = 0
rain_x = random.randint(0,800)
raindrop_spawn_time=0
```

셋업

5. 파이썬 프로그램의 기본 구조

```
class Raindrop:
    def __init__(self):
        self.x = random.randint(0,1000)
        self.y = -5

    def move(self):
        self.y += 7

    def draw(self):
        pygame.draw.line(screen,(0,0,0),(self.x,self.y),(self.x,self.y+5),1)
```

클래스

```
raindrops = []
```

리스트

5. 파이썬 프로그램의 기본 구조

게임 루프

```
while 1:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type == QUIT:
            sys.exit()

    raindrops.append(Raindrop())
    screen.fill((255,255,255))
    rain_y += 4
    pygame.draw.line(screen, (0,0,0), (rain_x, rain_y), (rain_x, rain_y+5), 1)
    for raindrop in raindrops:
        raindrop.move()
        raindrop.draw()

    pygame.display.update()
```

실행

6. 빗방울 클래스

```
class Raindrop:
    def __init__(self):
        self.x = random.randint(0,1000)
        self.y = -5

    def move(self):
        self.y += 7

    def draw(self):
        pygame.draw.line(screen,(0,0,0),(self.x,self.y),(self.x,self.y+5),1)
```

- 클래스는 하나의 코드 덩치이므로 1번째 줄 아래의 모든 줄을 들여쓰기 해야 합니다.
- 클래스를 만들려면 먼저 "class"라고 쓰고, 이어 클래스의 이름을 쓴 다음 콜론(:)을 씁니다.
- 클래스 이름은 일반적으로 대문자로 시작합니다.
- 인스턴스를 쓸 때는 소문자로 시작합니다.
- 빗방울(raindrop) 1개는 빗방울 클래스(class Raindrop)의 인스턴스입니다.

6. 빗방울 클래스

```
class Raindrop:
    def __init__(self):
        self.x = random.randint(0,1000)
        self.y = -5

    def move(self):
        self.y += 7

    def draw(self):
        pygame.draw.line(screen,(0,0,0),(self.x,self.y),(self.x,self.y+5),1)
```

- `__init__` 는 밑줄을 2칸 쓰고, `init`라고 쓰고, 다시 밑줄을 2칸 쓴 것입니다.
- 이 클래스는 결국 빗방울과 관련된 모든 것을 제어하는 함수의 목록일 뿐입니다.
- 함수를 사용하는 방법과 `self`라는 단어의 뜻을 알아보겠습니다.

6-1. __init__() 함수

```
def __init__(self):
```

- 빗방울 클래스의 첫 번째 함수는 __init__() 입니다.
- 다른 함수 이름은 마음대로 붙여도 되지만, 첫 번째 함수는 반드시 __init__()라고 불러야 합니다.
- init는 시작하다initiate의 줄임말입니다.
- 지금 하려는 일은 __init__()함수를 정의하는 것입니다.
- 새로운 빗방울을 만들면 파이썬은 빗방울 클래스의 __init__()함수를 찾아 이 함수가 시키는 대로 할 것입니다.

- 파이썬은 새로운 빗방울이 만들어질 때마다 이름을 붙입니다.
- 파이썬만 사용하는 이름이지요.
- 우리는 어떤 이름이 사용되는지 알 수 없지만 어쨌든 첫 번째로 만들어진 빗방울의 이름이 1번 빗방울이라고 가정해 봅시다.
- 파이썬은 1번 빗방울이 빗방울 클래스에 속한다는 사실을 알기 때문에 빗방울 클래스를 가져옵니다.
- 그리고 “self”라는 단어를 보면 그걸 1번 빗방울이라는 이름으로 바꾸고, 거기에 대해 작업합니다.
- 1번 빗방울은 빗방울 클래스로부터 나온 거니까 빗방울 클래스 안의 속성들을 1번 빗방울에게 적용하는 거지요.

- 두 번째 빗방울이 만들어지면 2번 빗방울이라고 이름을 붙입니다.
- 파이썬이 1번 빗방울에 대해 할 일을 모두 마치면, 모든 self를 2번 빗방울로 바꾸고 2번 빗방울에 대해서 작업합니다.
- 이런 방식으로 파이썬은 하나의 빗방울 클래스를 다른 모든 빗방울에 대해 계속 불러와서 작업합니다.
- 요컨대, 각각 다른 모든 인스턴스에 대해 계속 같은 빗방울 클래스를 적용하는 거지요.

6-3. self.x, self.y

- __init__()함수의 나머지 부분을 살펴봅시다.

```
def __init__(self):  
    self.x = random.randint(0,1000)  
    self.y = -5
```

- 새로운 빗방울을 만들면 파이썬은 이 빗방울에게 이름을 붙입니다.
- 4번 빗방울이라고 해봅시다.
- __init__()함수는 self를 4번 빗방울로 바꾸고 x좌표와 y좌표를 줍니다.
- 4번 빗방울.x는 0부터 1000 사이의 임의의 수가 됩니다.
- 4번 빗방울.y는 -5가 되겠지요. -5라고 한 이유는 스크린의 가장 위쪽 끝보다 조금 위에 있어야 하기 때문입니다.
- 한마디로, 스크린 밖에서 대기하고 있는 셈이죠.
- 새로운 빗방울이 만들어질 때마다 이 과정이 반복됩니다.
- 5번 빗방울도 x와 y좌표를 갖게 될 거고, 그 다음 빗방울들도 마찬가지입니다.

6-4. move(), draw() 함수

- 빗방울 클래스 안에 들어가는 move(), draw() 함수 둘 다 def를 사용하여 만들었습니다.

```
def move(self):  
    self.y += 7
```

- Move() 함수는 빗방울이 어떻게 움직여야 하는지 정합니다.
- 게임 루프에서 move() 함수를 부르면 무슨 일이 벌어질까요?
- self.y는 빗방울의 y좌표입니다. 모든 빗방울의 y좌표가 아니라 딱 하나, 특정한 빗방울의 y좌표지요.
- move() 함수는 이 특정 빗방울의 y좌표를 7만큼 더합니다.
- 아래 방향이 양수 방향이므로 빗방울은 아래로 떨어집니다.

6-4. move(), draw() 함수

```
def draw(self):  
    pygame.draw.line(screen, (0,0,0), (self.x, self.y), (self.x, self.y+5), 1)
```

- draw() 함수입니다. 이 함수는 파이게임의 draw.line() 함수를 사용합니다.
- 앞에서 직선을 그릴 때 사용했던 함수입니다.
- 여기서는 self.x와 self.y를 사용해 직선의 시작점과 끝점의 좌표를 정합니다.
- 이전에는 draw.line() 함수를 사용해 직선 1개만 그렸지만, 이제 다른 직선 여러 개를 그릴 수 있게 됐습니다.
- 1번 빗방울, 2번 빗방울, 3번 빗방울에 대해 계속 self를 사용하여 직선을 그리니까요.
- 같은 코드를 여러 번 사용하는 것이지요. 이것이 클래스의 장점입니다.

7. 리스트

- 아래 코드는 클래스 아래 게임 루프 위에 있습니다.

```
raindrops = []
```

- 1번 빗방울, 2번 빗방울 이것들이 어디 있는지 알 필요가 있습니다.
- 빗방울들은 땅에 떨어지면 사라져야 하니까요.
- 그러기 위해 클래스의 모든 인스턴스를 넣을 리스트(list)를 만들 거예요.
- 이 리스트의 이름은 빗방울들(raindrops) 이라고 정합니다.
- 리스트의 이름은 일반적으로 클래스 이름에서 맨 앞의 문자를 소문자로 바꾼 뒤, 복수형으로 붙입니다.
- 리스트의 내용은 대괄호 사이에 들어 있는 것 전체입니다. 지금은 아무것도 없지요.

7. 리스트

- 리스트 안에 뭔가를 직접 추가하는 방법은 나중에 배우겠습니다.
- 지금은 그냥 파이썬한테 1번 빗방울, 2번 빗방울 ...) .. 로 리스트를 채우라고 할 겁니다.
- 우리가 이 런 항목들을 직접 볼 수는 없지만, 파이썬은 기록해 둘 것입니다.
- 리스트는 꽤 직관적이면서도 매우 유용합니다.
- 리스트의 항목 번호는 항상 0에서 시작합니다.
- 우리의 첫 번째 빗방울은 파이썬한테는 0번째 빗방울이었을 거예요.
- 리스트에 4개의 항목이 들어 있으면, 항목 번호는 0, 1, 2, 3입니다.
 - 예를 들어 [a, b, c]라는 리스트에서 a는 0번째 항목, b는 1 번째 항목, c는 2번째 항목입니다.

8. 새 빗방울

```
raindrops.append(Raindrop())
```

- 이 코드는 빗방울 클래스의 인스턴스를 빗방울들 리스트에 추가했습니다.
'append'는 추가하라는 뜻이지요.
- 이 코드를 게임 루프에 써서, 1번의 게임 루프마다 1개의 빗방울이 리스트에 추가되도록 했습니다.
- 루프 1번에 2개의 빗방울을 넣어서 비가 2배로 더 오게 하고 싶다면 아래와 같이 코드를 2번 쓰면 됩니다.

```
raindrops.append(Raindrop())  
raindrops.append(Raindrop())
```

- 5번 넣으면 5배로 오겠지요.

8. 새 빗방울

- 다른 방법도 있습니다.
- 현재 우리가 만든 우주의 속도는 시계 함수로 제한된 상태입니다.
- 우리는 60으로 맞췄지요. 이 말인즉, 1초에 60번 루프가 돈다는 뜻입니다.
- 이 코드를 지우면 프로그램이 보다 빨라질 테고, 그러면 비를 더 많이 오게 할 수도 있습니다.
- 맨 앞에 # 표시를 붙여서 시계 함수 코드가 실행되지 않게 할 수도 있고요.
`# clock.tick(60)`
- 이렇게 해 두면 파이썬은 그 코드를 무시합니다.
- 코드 자체를 지우는 것보다 이렇게 실행을 멈춰 두는 것이 좋을 때가 있습니다. 실험할 때나 버그를 고칠 때 말입니다.
- 나중에 # 표시를 지우면 그 코드가 다시 실행됩니다.
- 하지만 시계를 멈추는 것보다 루프마다 빗방울을 추가하는 쪽을 추천합니다.

9. 모든 빗방울들에 대해 함수 실행

- 빗방울들 리스트의 개별 빗방울에 대해 빗방울 클래스의 `move()` 함수, `draw()` 함수를 적용했습니다.

```
for raingroup in raingroups:  
    raingroup.move()  
    raingroup.draw()
```

- 이 코드는 게임 루프 안 `screen.fill()` 밑에 썼습니다. 스크린을 색칠하고 그 위에 빗방울을 그려야 하니까요.
- `for` 루프가 빗방울(`raingroup`)이라는 변수를 만들면 이 빗방울 변수는 빗방울들 리스트의 첫 번째 항목과 같아집니다. 즉, 0번 빗방울이 됩니다.
- 이 0번 빗방울에 대해 `move()` 함수와 `draw()` 함수가 실행됩니다.

9. 모든 빗방울들에 대해 함수 실행

- 빗방울들 리스트의 개별 빗방울에 대해 빗방울 클래스의 `move()` 함수, `draw()` 함수를 적용했습니다.

```
for raindrop in raindrops:  
    raindrop.move()  
    raindrop.draw()
```

- 그 후 루프는 시작점으로 돌아갑니다. 루프는 리스트의 다음 항목인 1번 빗방울을 가지고 와 빗방울 변수를 그것과 같게 만들고, 함수를 실행합니다.
- 0번 빗방울에 했던 일을 1번 빗방울에 대해 반복하는 거지요.
- 루프는 리스트에 있는 모든 항목, 즉 모든 빗방울에 대해 두 함수를 전부 실행할 때까지 반복됩니다.
- 그러면 우리는 수많은 빗방울을 갖게 되지요.

9-1. for 루프

- for 루프는 메인 게임 루프 안에서 실행되는 작은 프로그램 루프입니다.
- 항상 'for 어떤 것 in 어떤 것'이라는 구조를 갖습니다.
- 보통 여러 항목을 가진 리스트 안 각 항목에 대해 몇 줄의 코드를 반복 실행시키는 데 사용됩니다.
- 예를 들어 봅시다.

```
for i in range(5, 10):  
    print (i * 10)
```

- Range() 함수는 5부터 9까지의 정수 리스트를 만듭니다.
- For 루프는 그 리스트 안 각각의 정수에 대해 2번째 줄을 실행합니다.
- 따라서 50부터 90까지 5개의 숫자가 표시됩니다

10. 페퍼로니

- 지금까지 우리는 빗방울(raindrop)이라는 단어를 세 가지로 사용했습니다.
 - 대문자로 시작하는 빗방울(Raindrop)은 클래스의 이름입니다.
 - 소문자로 시작하고 복수형인 빗방울들(raindrops)은 리스트의 이름입니다
 - 소문자로 시작하고 단수형인 빗방울(raindrop)은 for 루프에 있습니다.
- 마지막 빗방울은 갑자기 어디서 튀어나온 것처럼 보입니다.
- 사실 그렇습니다. 이 빗방울은 for 루프 안에서만 사용되는 변수입니다.
- 아무 이름이나 붙여도 괜찮습니다.

10. 페퍼로니

- 만약 for 루프를 다음과 같이 썼다고 해봅시다.

```
for pepperoni in raindrops:  
    pepperoni.move()  
    pepperoni.draw()
```

- 여기서 for 루프는 페퍼로니(pepperoni)라는 변수를 만들고, 이를 '빗방울 리스트(raindrops)'의 첫 번째 항목과 같게 합니다.
- for 루프는 빗방울 리스트의 첫 번째 항목을 빗방울 클래스에서 가져왔다는 것을 알고 있지요.
- 따라서 페퍼로니에 대해 빗방울 클래스에 있는 2개의 함수를 실행합니다.
- 그 다음 빗방울 리스트의 두 번째 항목에 대해 같은 일을 반복합니다.
- 페퍼로니는 for 루프 안에서만 사용됩니다.
- 따라서 굳이 빗방울(raindrop)이라고 이름을 안 붙이고, 페퍼로니라는 이름을 붙여도 상관없습니다.

11. 빗방울 지우기 step1

- for 루프는 리스트 안 여러 항목에 대해 여러 함수를 실행합니다.
- 그런데 문제가 있습니다.
- 일단 만들어진 빗방울들은, 떨어진 후에도 사라지지 않는다는 점입니다.
- for 루프는 떨어진 빗방울들의 좌표를 계속 업데이트합니다.
- 방치하면 리스트가 너무 길어져서 컴퓨터가 감당하기 어려울 만큼 많은 메모리를 사용해야 합니다.
- 그러면 파이썬은 결국 부르르 떨면서 멈추겠지요.
- 그렇다고 for 루프 안에서 리스트의 항목들을 지울 수도 없습니다.
- for 루프는 리스트의 항목을 하나하나씩 확인하면서 돌기 때문에 항목을 지우면 해매게 됩니다.

11. 빗방울 지우기 step1

- 그럼 문제 해결 방법이 없는 걸까요?
- 그럴리가요! 문제 해결을 위해 for 루프를 while 루프로 바꾸겠습니다.
- While 루프를 사용하면 리스트의 항목들을 지울 수 있습니다.
- 먼저 빗방울 클래스에서 새로운 함수를 만들어 봅시다.
- 이 새로운 함수는 빗방울 클래스 안, move() 함수 밑에 쓰세요.
- 가장 먼저 __init__()만 나온다면 다른 함수들의 순서는 상관없습니다.
- 보통 draw() 함수를 가장 나중에 쓰지만 꼭 그럴 필요는 없습니다.

```
def off_screen(self):  
    return self.y > 800
```

- 이 함수는 불Boolean 대수 값을 돌려줍니다.
- self.y가 800보다 크면 off_screen()은 참값을 돌려줍니다.
- 그렇지 않으면 off_screen()은 거짓값을 돌려줍니다.

11-1. 불 대수

- 프로그래밍에서 자주 쓰이는 불 대수는 '참' 아니면 '거짓'만 돌려주는 식입니다 (참은 1, 거짓은 0이라고 쓰기도 합니다)

- if 조건문을 예로 들어 봅시다.

```
if x == 8:  
    y += 5
```

- 1 번째 줄은 불 대수 값을 찾으라는 뜻입니다. x가 8인지 아닌지 알려 달라는 것이지요.
- 2번째 줄은 만약 1 번째 줄이 참이면 y에 5를 더하라는 뜻입니다. 만약 거짓이면 아무것도 하지 않습니다.
- 그런데 if 뒤에 조건이 여러 개인 경우도 있습니다.

■ and로 연결된 경우

```
if event.type == KEYDOWN and event.key == K_q and menu == "game":
```

- 이 코드는 and로 연결된 3개의 서로 다른 진술이 각각 참인지를 묻습니다.
- 모든 진술이 참일때만 다음 줄을 실행합니다.
- 마치 전자공학에서의 AND 게이트(2개의 입력이 모두 참일 때만 참을 돌려주는 게이트)와 같지요.

■ or로 연결된 경우

```
if self.x < 0 or self.x > 1000:
```

- 이 코드는 or로 연결된 2개의 진술 중 참인 것이 있는지를 묻고 있습니다.
- 만약 둘 중 하나라도 참이면 다음 줄을 실행합니다.

■ “아니면(not)”을 사용하여 참 또는 거짓을 반대로 바꾸기도 합니다.

```
if not A == 30
```

- 이 코드는 A가 30이 아닐 때 참값을 돌려줄 것입니다

12. 삭제를 위한 while 루프

- while 루프가 for 루프보다 복잡해 보이지만, 이해하면 꽤 쉽게 사용할 수 있습니다.
- While 루프의 장점은 리스트의 항목들이 필요 없어지면 지울 수 있다는 것입니다.

12. 삭제를 위한 while 루프

- 음영 표시한 코드를 추가하세요.

```
while 1:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type == QUIT:
            sys.exit()

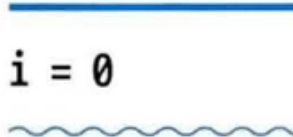
    raindrops.append(Raindrop())

    screen.fill((255,255,255))

    for raindrop in raindrops:
        raindrop.move()
        raindrop.draw()
    i = 0
    while i < len(raindrops):
        raindrops[i].move()
        raindrops[i].draw()
        i += 1

    pygame.display.update()
```

12. 삭제를 위한 while 루프



```
i = 0
```

- `i`라는 변수의 값을 0으로 정합니다.
- 이 변수는 빗방울들 리스트의 항목 번호를 셀 때 사용합니다.
- 계속 말하지만 꼭 `i`라고 쓸 필요는 없습니다.
- 다른 데서 이미 사용한 이름만 아니면 어떤 이름을 써도 상관없습니다.
- `avocado=0`이라고 써도 됩니다.
- 루프를 도는 과정을 반복(iteration)이라고 부르기 때문에, 보통 `i`라고 쓰긴 하지만요.
- 만약 `i`를 이미 썼다면 `j`, 아니면 `k`라고 써도 됩니다.

12. 삭제를 위한 while 루프

```
while i < len(raindrops):
```

- while 루프의 시작을 나타내는 코드입니다.
- len() 함수는 리스트의 길이를 알려 줍니다.
- 빗방울들(raindrops)는 우리가 사용하는 리스트의 이름입니다.
- 지금 i는 0입니다. 1가 빗방울들 리스트의 길이보다 작은 동안에는 참이므로 그 다음 줄들이 실행됩니다.
- 만약 리스트 안에 아무것도 없어서 위 코드가 참값을 돌려주지 않으면, 게임 루프는 그냥 다음 코드로 넘어갑니다.

12. 삭제를 위한 while 루프

```
raindrops[i].move()  
raindrops[i].draw()
```

- 위 두 줄의 코드는 빗방울 클래스에 있는 2개의 함수를 실행시킵니다.
- for 루프 안에서 두 함수를 실행시키는 코드와 비슷해 보이지만 약간 다릅니다.
- for 루프 안에서는 빗방울(raindrop)이라는 변수를 만들고, 차례대로 리스트 안에 있는 빗방울과 같게 했습니다.
- 여기서는 항목 번호 *i*를 빗방울들 리스트에서 가져왔습니다.
- 그래서 `raindrops[i]` 라고 쓴 것입니다.
- `raindrops[i]`는 빗방울 리스트의 항목 번호 *i*를 뜻합니다.

12. 삭제를 위한 while 루프

```
i += 1
```

- 이제 `i`에 1을 더합니다. `while` 루프가 시작 지점으로 돌아가면 `i`는 1이 됩니다.
- 1의 값이 리스트 항목들의 수보다 작은 지를 확인함으로써 리스트에 남은 항목이 있는지 확인합니다.
- 다음으로 넘어가서 리스트에 있는 두 번째 항목인 `raindrops[i]`을 움직이고 그림니다.
- 만약 리스트 안에 6개의 항목이 있다고 가정하면 0, 1, 2, 3, 4, 5번이라고 셀 수 있습니다.
- `i`는 0부터 시작해서 5번까지 루프를 돕니다. 그 다음 루프에서는 `i`는 6이 될 것입니다. `while` 루프의 1번째 줄은 `while i < len(raindrops):`입니다.
- `i`는 6이고 `len(raindrops)`도 6이므로, `6 < 6`은 거짓이 됩니다. 그러면 `while` 루프는 멈추고, 게임 루프는 다음 섹션으로 넘어갑니다.

12. 삭제를 위한 while 루프

```
i += 1
```

- 우리가 `raindrops.append(Raindrop())` 이라는 코드를 써서 리스트에 1개의 항목을 추가하면, 리스트의 길이는 1만큼 증가합니다.
- 그리고 추가한 항목은 리스트의 가장 끝으로 갑니다.

13. 빗방울 지우기 Step2

- 지금까지 리스트에 빗방울들을 추가하고, 루프가 리스트를 돌면서 차례대로 각 빗방울을 움직이고 그리는 것을 배웠습니다.
- 스크린 밖으로 나간 빗방울들을 지우기 위해 `off_screen()` 함수를 만들었지요.
- 아래 음영으로 표시된 부분은 떨어진 빗방울들을 지우는 코드입니다.

```
i=0
while i < len(raindrops):
    raindrops[i].move()
    raindrops[i].draw()
    if raindrops[i].off_screen():
        del raindrops[i]
        i -= 1
    i += 1
```

13. 빗방울 지우기 Step2


- 음영 1번째 줄에 if 문이 있습니다.
- if 문은 조건이 참인지 거짓인지를 판단합니다.
- 이 대답은 빗방울 클래스에 있는 off_screen() 함수가 알려 줍니다.
- off_screen() 함수는 빗방울이 스크린 바닥에 떨어졌는지 아닌지를 감지합니다.
- 1번 빗방울에 대해 참을 돌려주면, del이라는 키워드를 사용해 이 빗방울을 삭제하고, i에서 1을 뺍니다.
- 항목을 하나 지웠기 때문에 리스트의 길이도 1만큼 줄어들어야 합니다.
- 만약 8개의 항목이 있는 리스트에서, 4번 항목을 지운다고 가정해 봅시다.
- 그럼 5번 항목이 새로운 4번 항목이 될 것입니다.
- 다음 루프에서는 새로운 4번째 항목에 대해 함수를 실행시켜야 하고, 따라서 i를 3으로 내려야합니다. 다음 줄에서 i에 1을 더해서 4로 바꿉니다.
- 그러면 루프는 새로운 4번 항목을 움직이고 그리고 off_screen()함수가 참을 돌려주면(스크린 밖으로 나가면) 지웁니다.

14. 랜덤 플레이 빗방울

- 지금은 모든 빗방울이 같은 속도로 떨어집니다.
- 빗방울이 떨어지는 속도를 각각 다르게 하여 비내리는 모습이 좀 더 현실적으로 보이도록 할 수도 있습니다.

```
def __init__(self):  
    self.x = random.randint(0,1000)  
    self.y = -5  
    self.speed = random.randint(5,18)
```

```
def move(self):  
    self.y += 7 * self.speed
```



14. 랜덤 플레이 빗방울

- `randint()` 함수를 사용해 빗방울이 서로 다른 속도로 떨어지도록 해 봅시다.
- `randint()`의 인자는 양 끝의 수를 포함합니다.
- `self.speed`라는 변수를 만들고, 5부터 18 사이의 수 중 하나를 임의로 고릅니다.
- `randint()` 함수를 `__init__()` 함수 안에 쓰는 이유는 이 일이 특정 빗방울이 만들어질 때 1번만 하면 되는 일이기 때문입니다.
- 일단 만들어진 빗방울은 멀어질 때까지 같은 속도를 유지합니다.
- 만약 `self.speed`를 `move()` 함수 안에 넣으면 `self.speed`는 빗방울이 움직일 때마다 변할 것입니다.
- `move()` 함수가 불려오면, `self.speed`의 값을 `self.y`에 더합니다.
- `move()` 함수는 각 빗방울에 대해 게임 루프 1번당 1번씩 불려옵니다.
- 즉, 루프를 돌 때마다 빗방울의 속도는 바뀌겠지요.



Thank You
