

2장 Manim 맛보기

2.1. Manim으로 첫 번째 Scence (원과 정사각형 그리기)

Manim을 처음 실행하여 보자.

클래스와 메소드의 개념이 필요는 하지만 우선 실행을 하여 보도록 하자. 첫 번째 매개 변수로 클래스의 인스턴스가 넘어오게 된다. 이 첫 번째 매개 변수의 이름은 보통 관행적으로 self라고 하며, 인스턴스 메서드는 이 self를 통해 인스턴스 속성(attribute)에 접근하거나 다른 인스턴스 메서드를 호출할 수 있다. 뿐만 아니라 self를 통해, 클래스 속성에 접근하거나 클래스 메서드를 호출할 수도 있다.

우선 원 하나를 그려보려고 한다. 소스 코드를 보자.

manim 패키지의 모든 모듈을 가져온다.

```
└─ from manim import *
```

CreateCircle란 이름의 클래스를 정의하고 Scene에 상속한다. 왜 상속을 해야 할까? 보통 상속은 기존 클래스를 변경하지 않고 기능을 추가하거나 기존 기능을 변경하려고 할 때 사용한다. "클래스에 기능을 추가하고 싶으면 기존 클래스를 수정하면 되는데 왜 굳이 상속을 받아서 처리해야 하지?" 라는 의문이 들 수도 있다. 하지만 기존 클래스가 라이브러리 형태로 제공되거나 수정이 허용되지 않는 상황이라면 상속을 사용해야 한다.

```
└─ class CreateCircle(Scene):
```

클래스 CreateCircle 안에 메소드 construct를 def 키워드를 사용하여 정의한다. 파이썬은 클래스의 메소드를 정의할 때 self를 명시한다. 메소드를 불러올 때 self는 자동으로 전달된다. self를 사용함으로써 클래스 안에 정의한 멤버에 접근할 수 있게 된다. 또한 파이썬에서는 중괄호 { }을 사용하여 메소드를 묶지 않고 들여쓰기를 하여 이를 구분한다. 코드 블록을 구성하기 위해 if, for, class, def 등등을 작성하면서 나오는 : 다음 아랫줄은 반드시 들여쓰기를 해야 한다. 들여쓰기의 방법은 스페이스바로 네 칸 또는 한 번의 탭 방식을 사용한다. 중요한 것은 같은 블록 내에서는 들여쓰기 칸 수가 같아야 한다는 것이다. 위반시에는 "IndentationError: unexpected indent"라는 에러 메시지가 출력된다.

┆ def construct(self):

이제 manim 패키지에서 불러들인 Circle() 모듈을 이용하여 원을 그리려고 한다. 아래와 같이 두 탭 들여쓰기를 하여 작성하자.

┆ self.add(Circle())

Circle()의 () 안에 아무것도 코딩을 하지 않으면 기본값인 중심이 원점이고 반지름 1이며 테두리 색이 빨간색인 원을 그린다. 나중에 매개변수, 메소드, 속성 등에 대하여 논의하자. 지금은 그냥 넘어가자.

전체 코딩은 아래와 같다.

```
1 from manim import * #manim 패키지의 모든 모듈 가져오기.
2
3 class CreateCircle(Scene): #클래스 만들기
4     def construct(self):
5         self.add(Circle())
```

그림 1

그리고 아래에 있는 터미널 창에 ①과 같이

manim -p lecture_1.py

을 입력하고 엔터를 쳐 실행을 하자. 그러면 아래와 같이 실행되고 원이 그려진 창이 새롭게 실행된다.

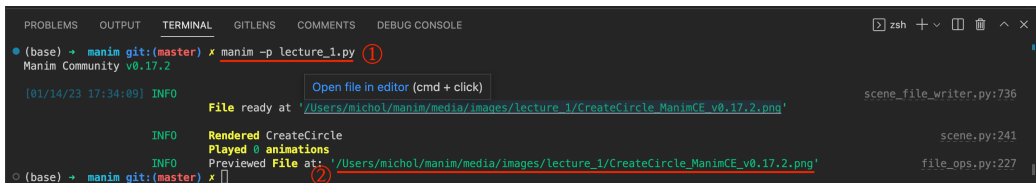


그림 2

②을 윈도우는 'Ctrl + 클릭', 맥은 'command + 클릭'을 하면 실행이 된다. 터미널 창에 이미 옵션 -p를 넣어 실행시켰으므로 자동 실행된다. 파일 위치도 확인할 수 있다. 위의 manim 예는 애니메이션이 없는 단지 원만 추가한 것이라 그림 파일인 png로 결과가 저장된다.

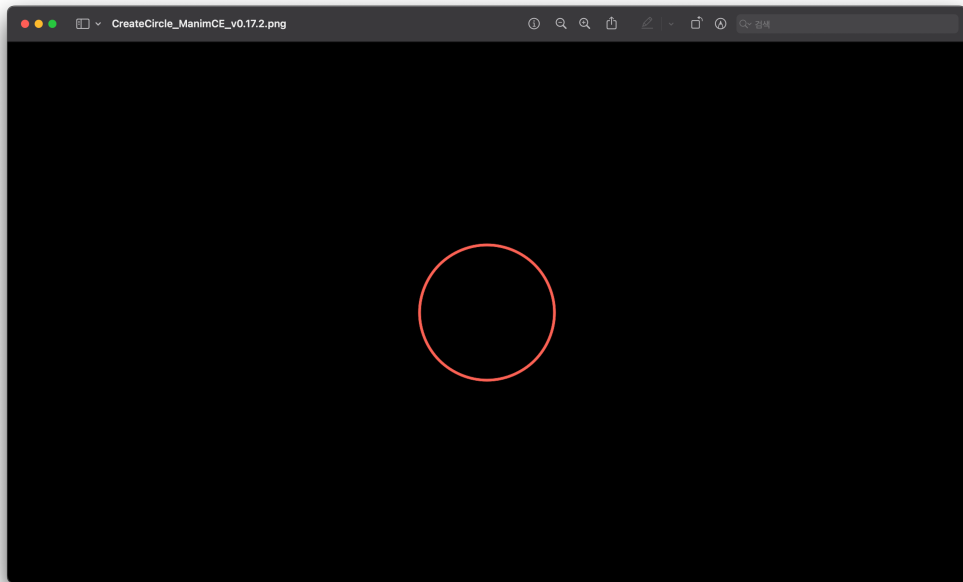


그림 3

컴퓨터 프로그래머들은 이러한 코딩을 그리 좋아하지 않는다. cir이란 객체 (instance)에 Circle() 모듈을 만든다. 여러 개의 같은 객체를 만들어 사용할 수도 있다.

```

|         cir = Circle()
|         self.add(cir)

```

전체 코드는 아래와 같다.

```

1  from manim import * #manim 패키지의 모든 모듈 가져오기.
2
3  class CreateCircle(Scene): #클래스 만들기
4      def construct(self):
5          cir = Circle()
6          self.add(cir)

```

그림 4

[그림 4]와 같이 코딩을 하고 터미널 창에서 실행을 시키면 같은 실행 결과를 얻는다.

이제 lecture_1.py에 또 다른 클래스 정사각형 클래스를 만들어 보자.

```

| class CreateSquare(Scene):

```

```

1 |     def construct(self):
2 |         sqr = Square()
3 |         self.add(sqr)

```

전체 코드는 다음과 같다.

```

1 | from manim import * #manim 패키지의 모든 모듈 가져오기.
2 |
3 | class CreateCircle(Scene): #클래스 만들기
4 |     def construct(self):
5 |         cir = Circle()
6 |         self.add(cir)
7 |
8 | class CreateSquare(Scene):
9 |     def construct(self):
10 |         sqr = Square()
11 |         self.add(sqr)

```

그림 5

그리고 아래의 터미널에서 실행을 하여 보자.

```
manim -p lecture_1.py
```

그러면 아래와 같이 선택의 메시지가 나온다.



그림 6

정사각형을 실행하려면 2번을 입력하고 엔터를 누른다.

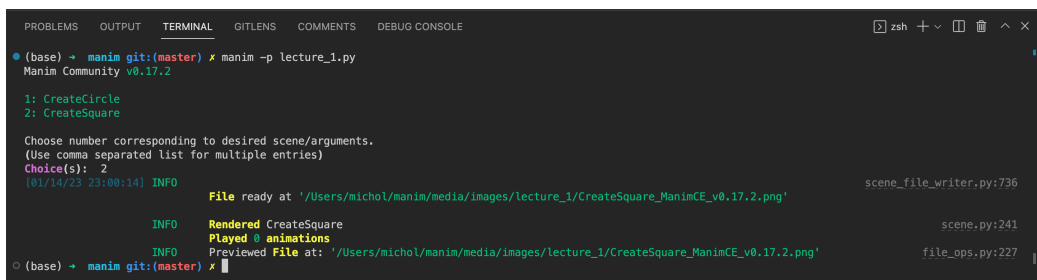


그림 7

[그림 8]과 같이 정사각형의 모양이 보인다.

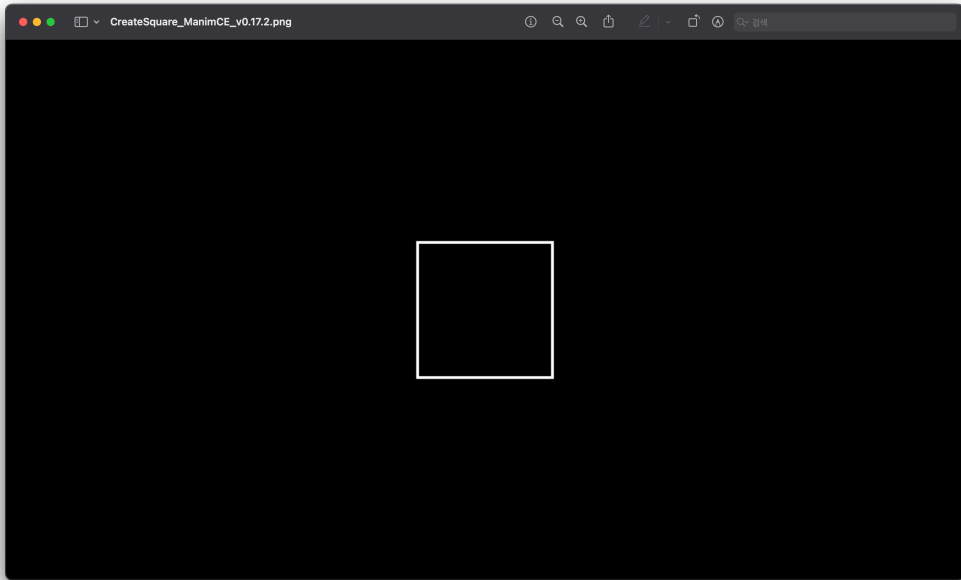


그림 8

둘 중 선택을 하지 않고 CreateCircle 클래스를 실행하려면 다음과 같이 입력하고 실행을 한다.

```
manim -p lecture_1.py CreateCircle
```

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  COMMENTS  DEBUG CONSOLE
(base) → manim git:(master) x manim -p lecture_1.py CreateCircle
Manim Community v0.17.2
[01/14/23 23:22:17] INFO File ready at '/Users/michol/manim/media/images/lecture_1/CreateCircle_ManimCE_v0.17.2.png' scene_file_writer.py:736
INFO Rendered CreateCircle scene.py:241
INFO Played 0 animations
INFO Previewed File at: '/Users/michol/manim/media/images/lecture_1/CreateCircle_ManimCE_v0.17.2.png' file_ops.py:227
(base) → manim git:(master) x
```

그림 9

CreateSquare 클래스를 실행하려면 다음과 같이 입력을 하고 실행을 한다.

```
manim -p lecture_1.py CreateSquare
```

```
PROBLEMS OUTPUT TERMINAL GITLENS COMMENTS DEBUG CONSOLE
(base) → manim git:(master) x manim -p lecture_1.py CreateSquare
Manim Community v0.17.2

[01/14/23 23:22:46] INFO File ready at '/Users/michol/manim/media/images/lecture_1/CreateSquare_ManimCE_v0.17.2.png'
scene_file_writer.py:736

INFO Rendered CreateSquare
scene.py:241
INFO Played 0 animations
file_ops.py:227
INFO Previewed File at: '/Users/michol/manim/media/images/lecture_1/CreateSquare_ManimCE_v0.17.2.png'
file_ops.py:227
(base) → manim git:(master) x
```

그림 10

2.2. 애니메이션 구현하기

이번에는 그림이 아니라 애니메이션을 만들어 보자. 원을 한 점에서 커져서 원이 되는 애니메이션을 제작하여 보자.

위의 예제에서 다른 것은 그대로 하고 `self.add(cir)` 대신 아래와 같이 수정을 하자. 가운데에서 점점커지게 하려면 `GrowFromCenter()` 명령어를 사용하여야 한다.

```
self.play(GrowFromCenter(cir))
```

그리고 `manim -p lecture_1.py`를 터미널 창에 입력을 하고 실행을 시키자.

```
(base) → manim git:(master) x manim -p lecture_1.py
Manim Community v0.17.2

[01/15/23 18:31:40] INFO Animation 0 : Using cached data (hash : 1413466013_1741245124_223132457)
cairo_renderer.py:78
[01/15/23 18:31:41] INFO Combining to Movie file.
scene_file_writer.py:617
INFO File ready at '/Users/michol/manim/media/videos/lecture_1/1080p60/CreateCircle.mp4'
scene_file_writer.py:736

INFO Rendered CreateCircle
scene.py:241
INFO Played 1 animations
file_ops.py:227
INFO Previewed File at: '/Users/michol/manim/media/videos/lecture_1/1080p60/CreateCircle.mp4'
file_ops.py:227
(base) → manim git:(master) x
```

그림 11

그러면 다음과 같은 애니메이션이 구현된다. 렌더링이 아무 오류 없이 실행되며 mp4 동영상의 위치를 볼 수 있다.

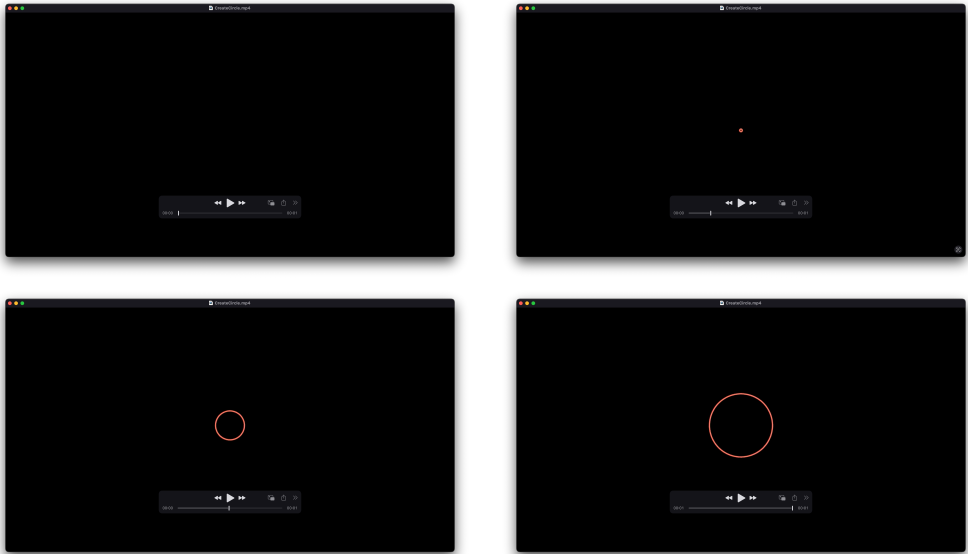


그림 12

Manim으로 수학 애니메이션을 구현하는 기초적인 방법을 익혔다. 물론 `Tex`과 `sox`의 나레이션 부분은 필요한 부분에서 다시 설명을 하겠다.

2.2. Manim 출력

Manim으로 개발을 하면 ./manim/에서 하위 폴더 project 폴더를 만들어 그 폴더 안에 scene.py를 만들고 애니메이션을 개발을 한다.

project/

└─scene.py

scene.py 안에 SquareToCircle이라는 scene을 컴파일 시키자.

```
manim -pql scene.py SquareToCircle
```

그러면 -l 옵션은 가로:세로 비가 16:9 비로 세로가 480 픽셀인 60프레임으로 영상을 만들라는 것이다.그러면 아래와 같은 폴더들이 생성이 된다.

project/

├─scene.py

└─media

├─videos

| └─scene

| └─480p15

| └─SquareToCircle.mp4

| └─partial_movie_files

├─text

└─Tex

다시 아래와 같이 컴파일 시키자.

```
manim -pqh scene.py SquareToCircle
```

그러면 -h 옵션은 가로:세로 비가 16:9 비로 세로가 1080pix인 60프레임으로 영상을 만들라는 것이다. 그래서 1080p60폴더가 생성이 된다.

project/

├─scene.py

└─media

├─videos

| └─scene

| └─480p15

| └─└─SquareToCircle.mp4


```

|   |   └─partial_movie_files
|   └─1080p60
|       └─SquareToCircle.mp4
|       └─partial_movie_files
└─text
  └─Tex

```

애니메이션이 없다면 이미지 파일로만 컴파일이 되었다면 아래와 같은 images 폴더가 생성이 된다.

```

project/
├─scene.py
└─media
  ├─images
  │   └─scene
  │       └─SquareToCircle.png
  ├─videos
  │   └─scene
  │       └─480p15
  │           └─SquareToCircle.mp4
  │           └─partial_movie_files
  │       └─1080p60
  │           └─SquareToCircle.mp4
  │           └─partial_movie_files
  └─text
    └─Tex

```

만약 세션별로 파일을 저장하고 싶다면 아래와 같이 명령어를 실행시키면 된다.

```
manim --save_sections scene.py
```

영상 출력 파일 외에도 섹션을 사용할 수 있다. 각 섹션은 자체 출력 영상을 생성한다. 두 섹션 사이의 컷은 다음과 같이 설정할 수 있다.

```
def construct(self):
    # 첫 번째 애니메이션 재생
    # 섹션이 자동으로 생성되므로 처음에 섹션이 필요하지 않다.
    self.next_section()
    # 다음 애니메이션 재생
    self.next_section("고유한 이름일 필요는 선택적인 이름이다.")
    # 더 많은 애니메이션 재생
    self.next_section("재생할 애니메이션이 없으면 이것은 제거한다.")
```

이 두 컷 사이의 모든 애니메이션은 단일 출력 비디오 파일로 연결됩니다. 각 섹션에 적어도 하나의 애니메이션이 필요하다는 점에 유의하십시오. 예를 들어, 이것은 출력 비디오를 만들지 않을 것이다.

```
def construct(self):
    self.next_section()
    # 이 섹션에는 애니메이션이 없으므로 제거한다.
    # 그러나 있어도 오류는 발생하지 않는다.
    # 만약 당신이 원한다면 자유롭게 빈 섹션을 컴파일 해 보아라.
    self.add(Circle())
    self.next_section()
```

이것을 고치는 한 가지 방법은 조금 기다리는 것이다.

```
def construct(self):
    self.next_section()
    self.add(Circle())
    # 1초를 기다리며 다음 섹션의 애니메이션이 구현된다.
    self.wait()
    self.next_section()
```

각 섹션에 대해 비디오를 만들려면 다음과 같이 Manim 명령어에 `--save_sections` 플래그를 추가해야 한다.

```
manim --save_sections scene.py
```

그러면 다음과 같이 파일들이 저장된다.

```
media
├── images
│   └── simple_scenes
├── videos
│   └── simple_scenes
│       └── 480p15
│           ├── ElaborateSceneWithSections.mp4
│           ├── partial_movie_files
│           │   └── ElaborateSceneWithSections
│           │       ├── 2201830969_104169243_1331664314.mp4
│           │       ├── 2201830969_398514950_125983425.mp4
│           │       ├── 2201830969_398514950_3447021159.mp4
│           │       ├── 2201830969_398514950_4144009089.mp4
│           │       ├── 2201830969_4218360830_1789939690.mp4
│           │       ├── 3163782288_524160878_1793580042.mp4
│           │       └── partial_movie_file_list.txt
│           └── sections
│               ├── ElaborateSceneWithSections_0000.mp4
│               ├── ElaborateSceneWithSections_0001.mp4
│               ├── ElaborateSceneWithSections_0002.mp4
│               └── ElaborateSceneWithSections.json
```