

Lecture #9. 게임 프레임웍

2D 게임 프로그래밍

이대현 교수

학습 내용

- python module
- 게임 상태
- 게임 프레임웍
- 로고 화면의 구현
- 타이틀 화면 상태의 구현
- 메인 게임 상태의 구현

Python Module

- 파이썬의 정의(definitions)와 문장(statements)을 담고 있는 파일
- 파일이름: 00000.py (확장자:py)

```
class Grass:
    pass

def update():
    global x
    x = x + 1

x = 0
update()
```

클래스 정의

함수 정의

문장

- 그 자체로도 실행 가능하며, 다른 모듈에서 импорт(import)해서 사용할 수도 있음. импорт되면, 그 자체가 하나의 객체가 됨.(싱글톤 객체가 됨)

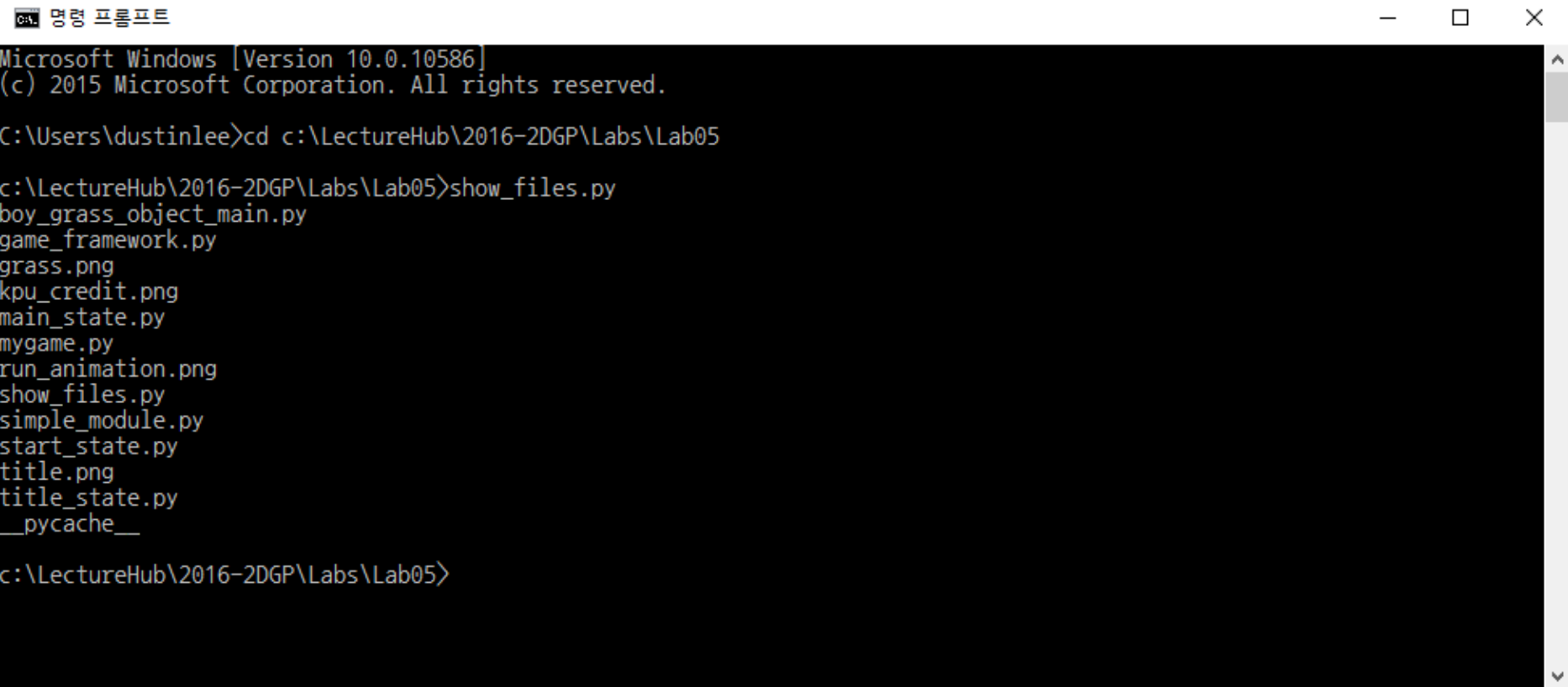
show_files.py

```
import os

file_name_list = os.listdir()
for name in file_name_list:
    print(name)
```

show_files.py 의 단독실행

- 코맨드창(cmd 실행) 실행 후, Lab 소스가 담긴 폴더로 이동한 후, 직접 show_files.py 를 치면, 실행이 됨. 아니면 python show_files.py 해도 실행이 됨.

A screenshot of a Windows Command Prompt window titled "명령 프롬프트". The window shows the following text: "Microsoft Windows [Version 10.0.10586] (c) 2015 Microsoft Corporation. All rights reserved. C:\Users\dustinlee>cd c:\LectureHub\2016-2DGP\Labs\Lab05 c:\LectureHub\2016-2DGP\Labs\Lab05>show_files.py boy_grass_object_main.py game_framework.py grass.png kpu_credit.png main_state.py mygame.py run_animation.png show_files.py simple_module.py start_state.py title.png title_state.py __pycache__ c:\LectureHub\2016-2DGP\Labs\Lab05>". The output lists various files and folders in the current directory.

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

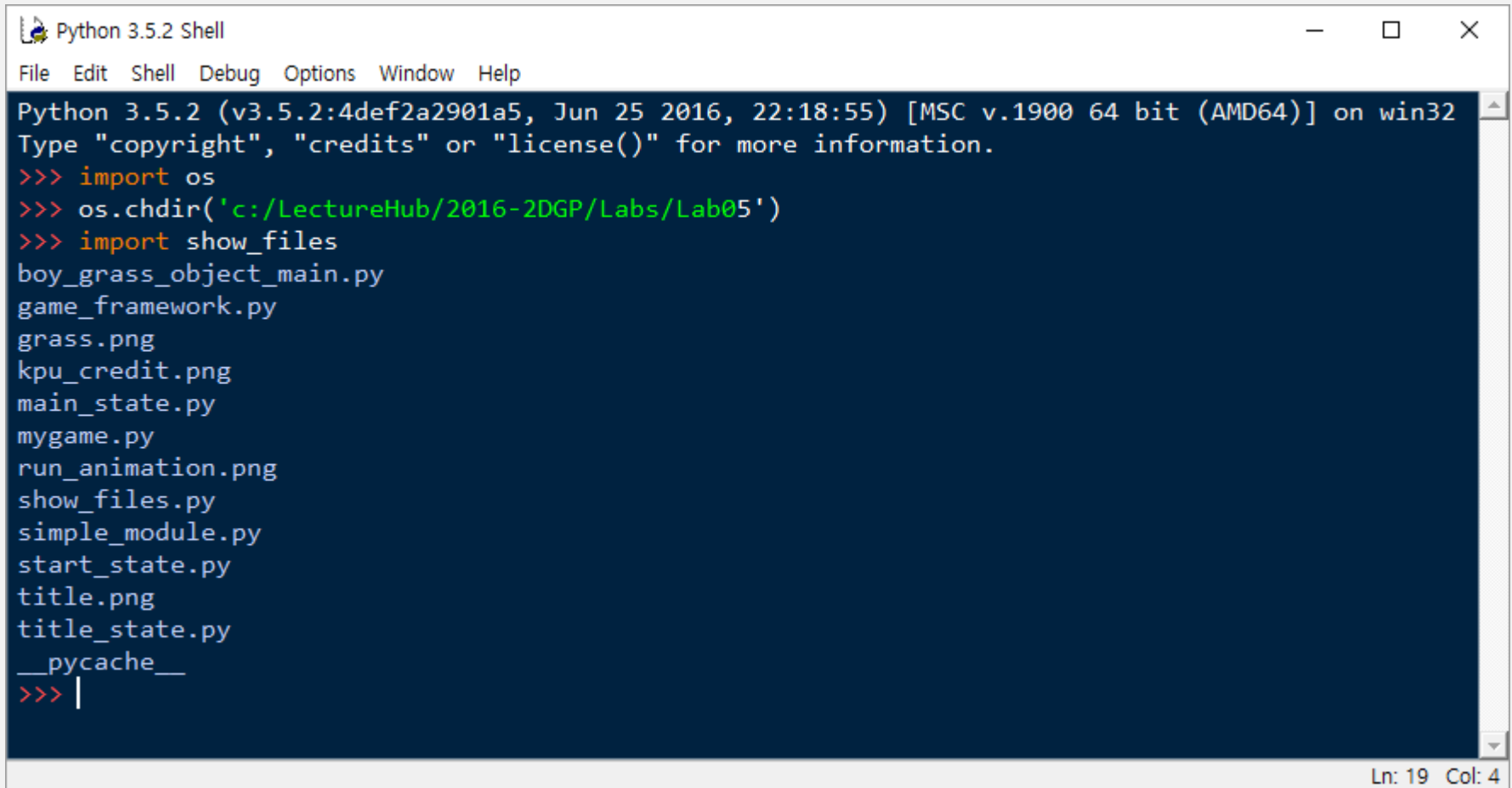
C:\Users\dustinlee>cd c:\LectureHub\2016-2DGP\Labs\Lab05

c:\LectureHub\2016-2DGP\Labs\Lab05>show_files.py
boy_grass_object_main.py
game_framework.py
grass.png
kpu_credit.png
main_state.py
mygame.py
run_animation.png
show_files.py
simple_module.py
start_state.py
title.png
title_state.py
__pycache__

c:\LectureHub\2016-2DGP\Labs\Lab05>
```

show_files.py 의 import

- 모듈을 임포트하는 경우, 모듈 내의 문장들이 실행됨.




```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.chdir('c:/LectureHub/2016-2DGP/Labs/Lab05')
>>> import show_files
boy_grass_object_main.py
game_framework.py
grass.png
kpu_credit.png
main_state.py
mygame.py
run_animation.png
show_files.py
simple_module.py
start_state.py
title.png
title_state.py
__pycache__
>>> |
```

Ln: 19 Col: 4

module 의 사용: 임포트한 후, 모듈이름.0000

game.py

```
class Grass:  
    pass  
  
def update():  
    global x  
    x = x + 1  
  
x = 0  
update()
```



```
import game  
  
grass = game.Grass()  
  
game.update()
```

__name__ 속성

- 파이썬 내부에서 사전에 정의된 속성.
- 일반적으로 그 값은 모듈의 이름임.
- 하지만, module 을 임포트하지 않고, 직접 실행하는 경우, "__main__" 이라는 문자열값을 갖게 되어, 현재 모듈이 단독적으로 실행되는 상황을 구분함.

simple_module.py

```
print("The value of __name__ is '" + __name__ + "'.")
```

import 하는 경우

실행하는 경우

```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>>
>>> import simple_module
The value of __name__ is 'simple_module'.
>>>
```

```
C:\Temp\TestPython>simple_module.py
The value of __name__ is '__main__'.
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
C:\Temp\TestPython>
```


__name__ 속성의 활용

- 어떤 모듈이 임포트되는 경우와 직접 실행이 되는 경우를 구분할 수 있음.
- 따라서 직접 실행되는 상황에서만 필요한 일들을 구분해서 처리할 수 있음.
- 메인함수를 흉내내는 경우에 많이 활용됨.

```
# 어쩌구저쩌구..  
# 어쩌구저쩌구..  
# 어쩌구저쩌구..  
# 어쩌구저쩌구..  
# 어쩌구저쩌구..
```

```
def identify():  
    print("이 함수는 단독으로 모듈을 실행할 경우만, 실행됩니다.")
```

```
if __name__ == '__main__':  
    identify()
```

리팩토링

```
open_canvas()

boy = Boy()
grass = Grass()

running = True
while running:
    handle_events()

    boy.update()

    clear_canvas()
    grass.draw()
    boy.draw()
    update_canvas()

    delay(0.05)

close_canvas()
```



```
boy = None
grass = None
running = True
```

```
def enter():
    global boy, grass
    open_canvas()
    boy = Boy()
    grass = Grass()
```

```
def exit():
    global boy, grass
    del(boy)
    del(grass)
    close_canvas()
```

```
def update():
    boy.update()
```

```
def draw():
    clear_canvas()
    grass.draw()
    boy.draw()
    update_canvas()
```

```
def main():
    enter()
    while running:
        handle_events()
        update()
        draw()
    exit()
```

```
if __name__ == '__main__':
    main()
```

게임 상태(Game State)의 이해 (1)

■ 게임 상태란?

- 게임 프로그램 실행 중의 어떤 특정 위치(또는 모드, 씬).
- 사용자 입력(키보드 또는 마우스 입력)에 대한 대응 방식은 게임의 상태에 따라 달라짐.



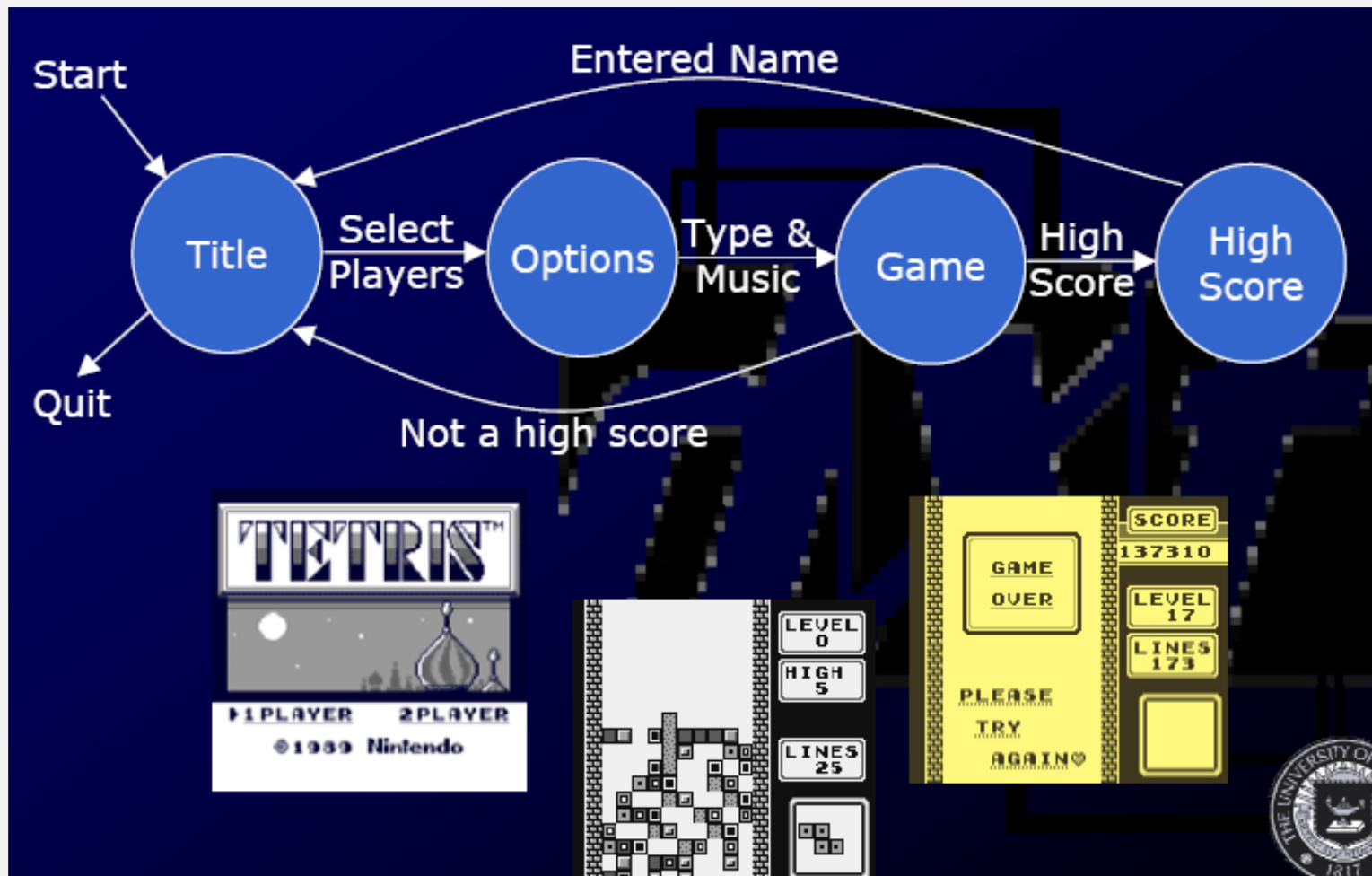
- 맵 선택 상태.
- 방향키는 맵 선택을 처리.

- 게임 메인 플레이 상태..
- 방향키는 캐릭터의 이동을 처리.

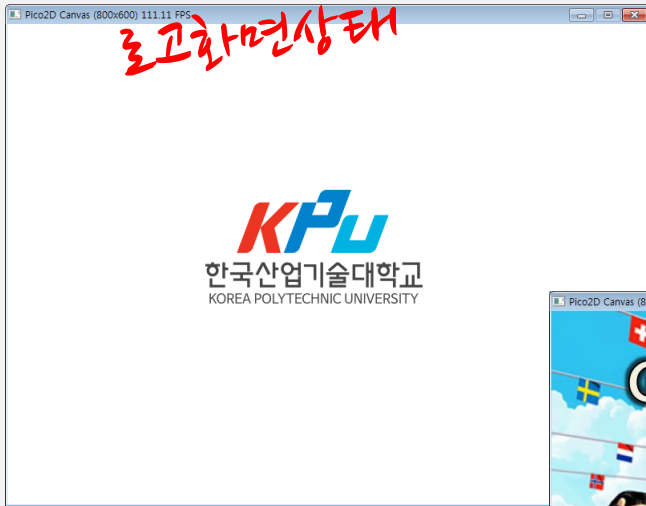
게임 상태(Game State)의 이해 (2)

■ 게임 프로그램은 게임 상태(모드, 씬)의 집합으로 구현됨.

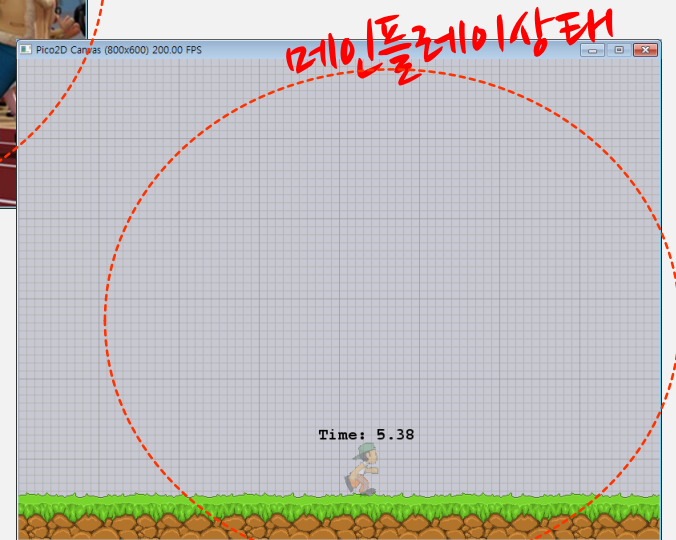
□ 예) 테트리스 게임



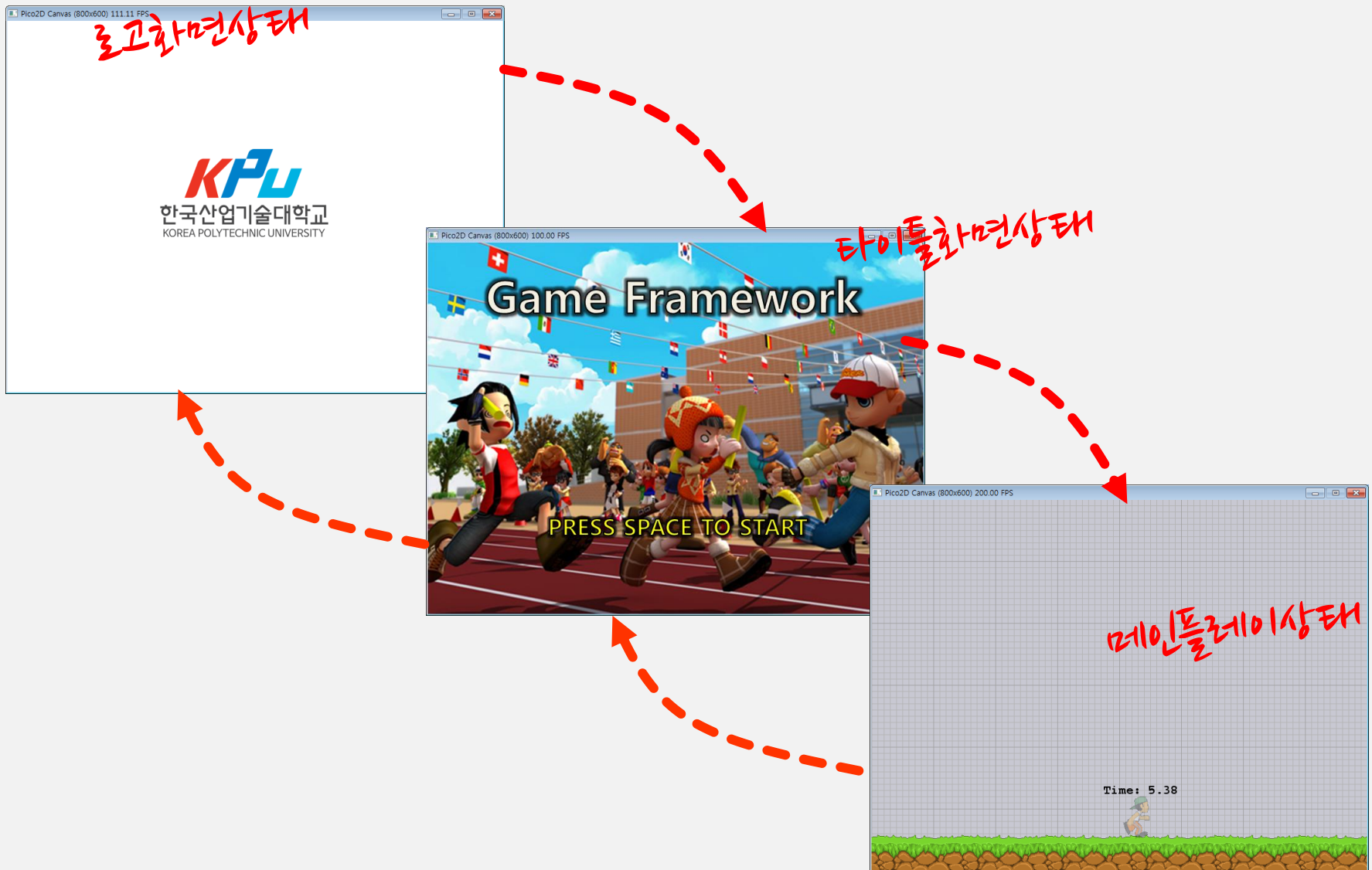
오늘 만들어 볼 것



#1. 각각의 상태를 구현해야 함.



#2. 상태간의 전환을 구현해야 함.



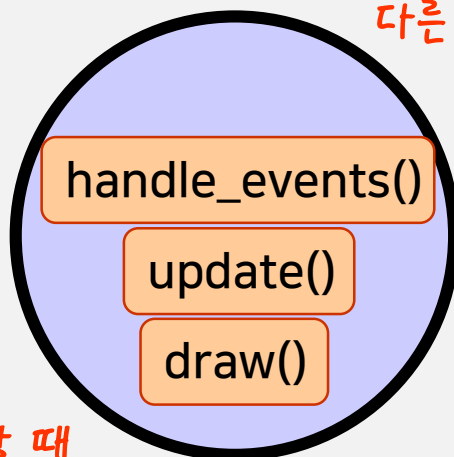
게임 상태의 구현

게임 상태에 들어올때 초기화

enter()

exit()

게임 상태에서 나갈 때
종료화

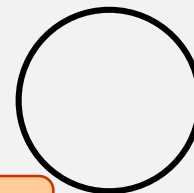


다른 상태로 잠깐 이동

pause()

resume()

현재 상태로 복귀



상태간의 전환: game_framework을 이용

run(state):

state를 시작 게임 상태로 하여, 게임 실행을 시작함.

change_state(state):

게임 상태를 state로 변화. 이전 게임 상태를 완전히 나눔.

push_state(state):

게임 상태를 state로 변화. 이전 게임 상태는 남아 있음.

pop_state(): 이전 게임 상태로 복귀

quit(): 게임을 중단

실습



로그 화면 구현

로고상태의 구현: start_state.py (1)



```
name = "StartState"
```

```
image = None
```

```
logo_time = 0.0
```

```
def enter():
```

```
    global image
```

```
    image = load_image('kpu_credit.png')
```

```
def exit():
```

```
    global image
```

```
    del(image)
```

로고상태의 구현: start_state.py (2)



```
def update():  
    global logo_time  
  
    if (logo_time > 1.0):  
        logo_time = 0  
        game_framework.quit()  
    delay(0.01)  
    logo_time += 0.01  
  
def draw():  
    global image  
    clear_canvas()  
    image.draw(400, 300)  
    update_canvas()
```

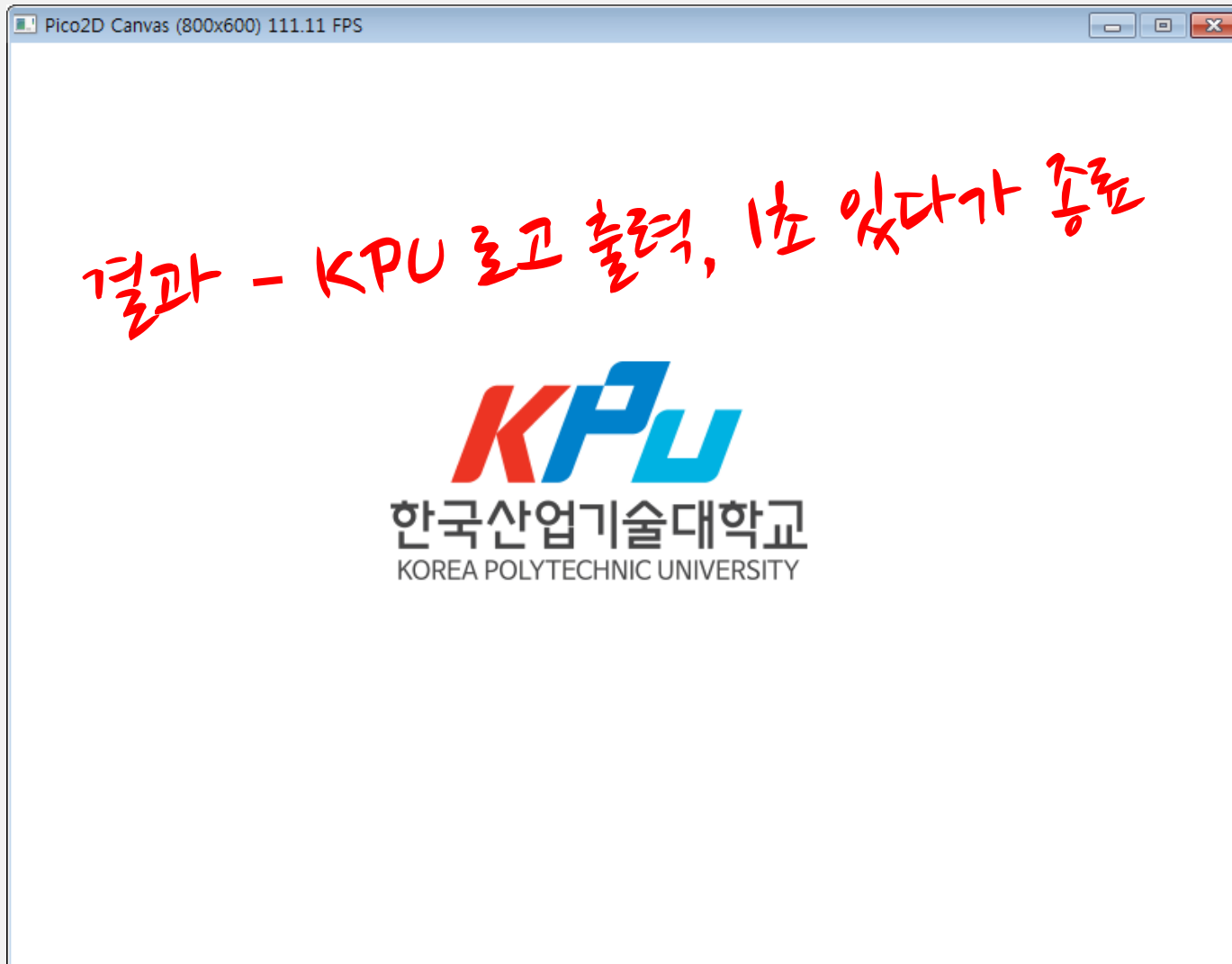


```
import game_framework
import pico2d
```

```
import start_state
```

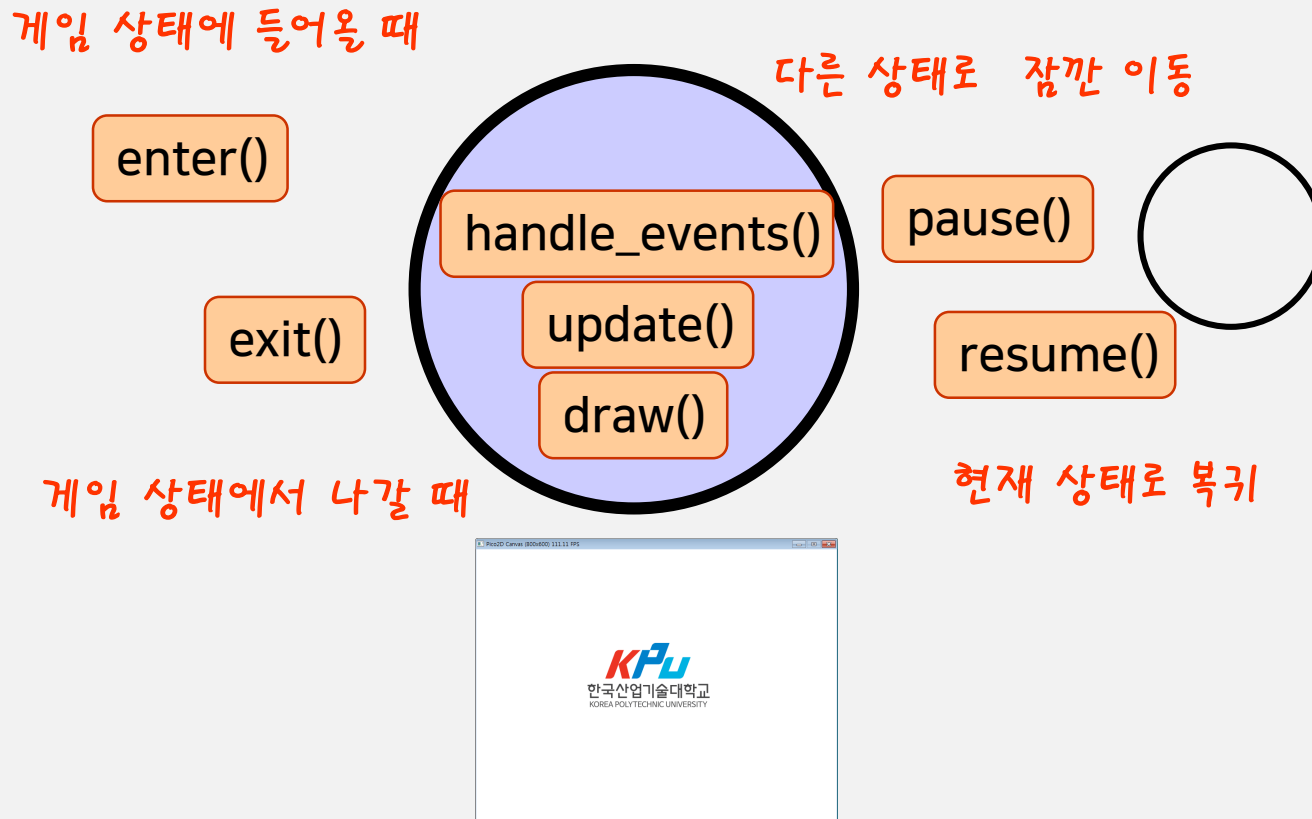
```
pico2d.open_canvas()
game_framework.run(start_state)
pico2d.close_canvas()
```

실행 - mygame.py 를 실행



start_state 의 구현과 활용

1. `start_state.py` 를 만든다
2. `start_state.py`의 내부 함수들을 작성한다.
3. 다른 소스에서 `import start_state` 를 해서 활용한다.



게임 상태의 뼈대

```
def enter(): pass
```

```
def exit(): pass
```

```
def update(): pass
```

```
def draw(): pass
```

```
def handle_events(): pass
```

```
def pause(): pass
```

```
def resume(): pass
```


enter()와 exit()의 구현

```
def enter():  
    global image  
    image = load_image('kpu_credit.png')
```

```
def exit():  
    global image  
    del(image)
```

update()

```
def update():  
    global logo_time  
  
    if (logo_time > 1.0):  
        logo_time = 0  
        game_framework.quit()  
    delay(0.01)  
    logo_time += 0.01
```

draw()

```
def draw():  
    global image  
    clear_canvas()  
    image.draw(400, 300)  
    update_canvas()
```

handle_events()

```
def handle_events():  
    events = get_events()
```

게임의 구성과 시작 – 게임프레임워크 활용

- `game_framework` 를 import 한다.
- 시작 게임 상태를 import 한다.
- 시작 게임 상태를 지정한 후, `game_framework` 를 시작한다.

```
import game_framework
import pico2d

import start_state

pico2d.open_canvas()
game_framework.run(start_state)
pico2d.close_canvas()
```

시스템



타이틀 상태 추가

로고 화면에 이어지는 타이틀 화면

start_state.py



title_state.py



title_state.py (1)

```
name = "TitleState"  
image = None
```

```
def enter():  
    global image  
    image = load_image('title.png')
```

```
def exit():  
    global image  
    del(image)
```



title_state.py (2)



```
def handle_events():
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            game_framework.quit()
        else:
            if (event.type, event.key) == (SDL_KEYDOWN, SDLK_ESCAPE):
                game_framework.quit()

def draw():
    clear_canvas()
    image.draw(400, 300)
    update_canvas()
```

start_state.py 의 수정



```
import title_state
```

```
def update():  
    global logo_time  
  
    if (logo_time > 1.0):  
        logo_time = 0  
        # game_framework.quit()  
        game_framework.change_state(title_state)  
    delay(0.01)  
    logo_time += 0.01
```

push_state를 쓸 것인가? change_state를 쓸 것인가?

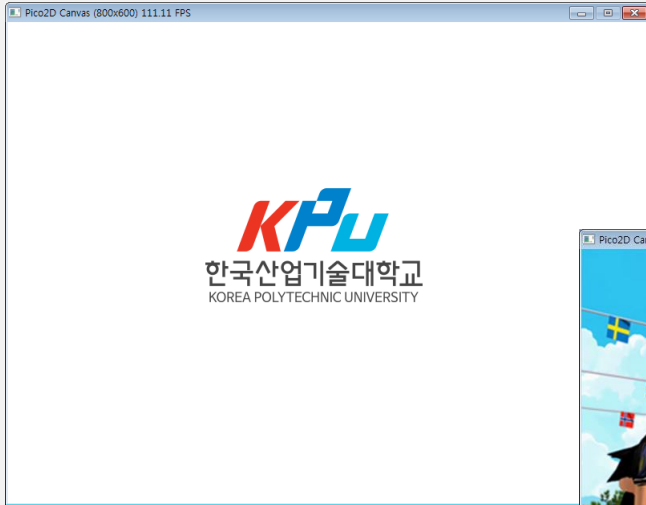
시작



게임 메인 상태 추가

로고 화면에 이어지는 타이틀 화면

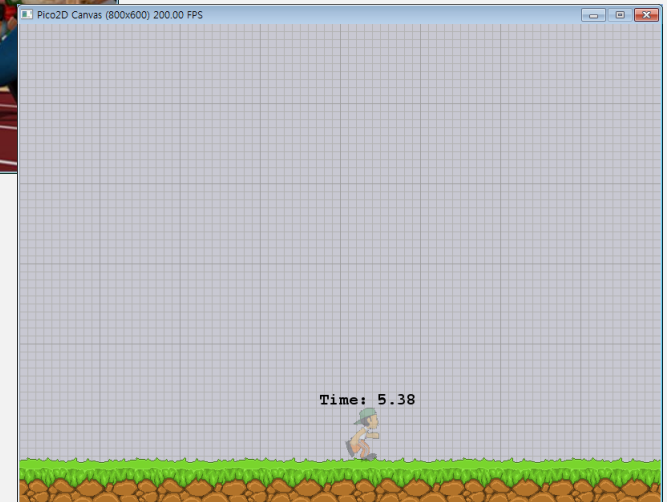
start_state.py



title_state.py



main_state.py



title_state.py 의 수정



```
import main_state
```

```
def handle_events():  
    events = get_events()  
    for event in events:  
        if event.type == SDL_QUIT:  
            game_framework.quit()  
        else:  
            if (event.type, event.key) == (SDL_KEYDOWN, SDLK_ESCAPE):  
                game_framework.quit()  
            elif (event.type, event.key) == (SDL_KEYDOWN, SDLK_SPACE):  
                game_framework.change_state(main_state)
```

main_state.py (1)

```
def enter():  
    global boy, grass  
    boy = Boy()  
    grass = Grass()
```

```
def exit():  
    global boy, grass  
    del(boy)  
    del(grass)
```



main_state.py (2)

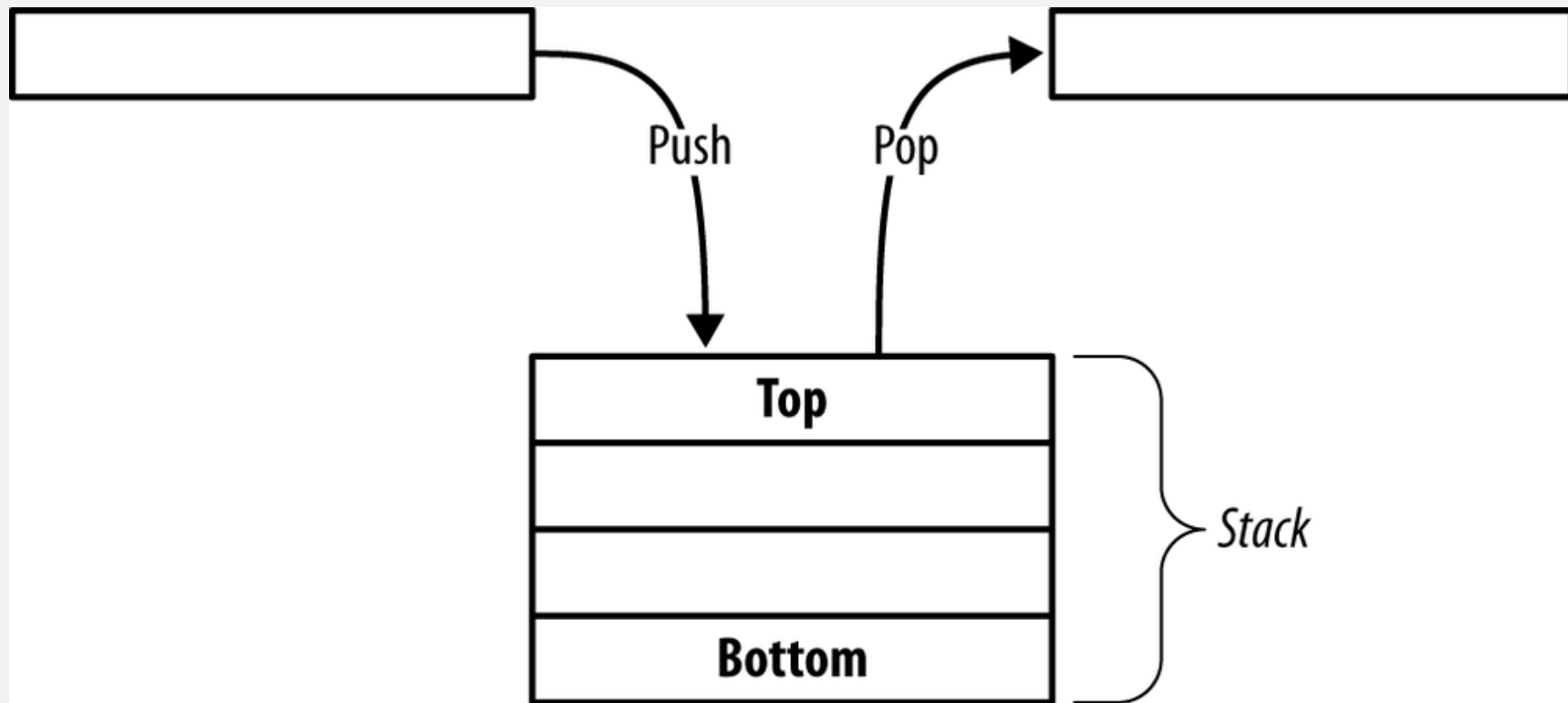


```
def handle_events():
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            game_framework.quit()
        elif event.type == SDL_KEYDOWN and event.key == SDLK_ESCAPE:
            game_framework.change_state(title_state)

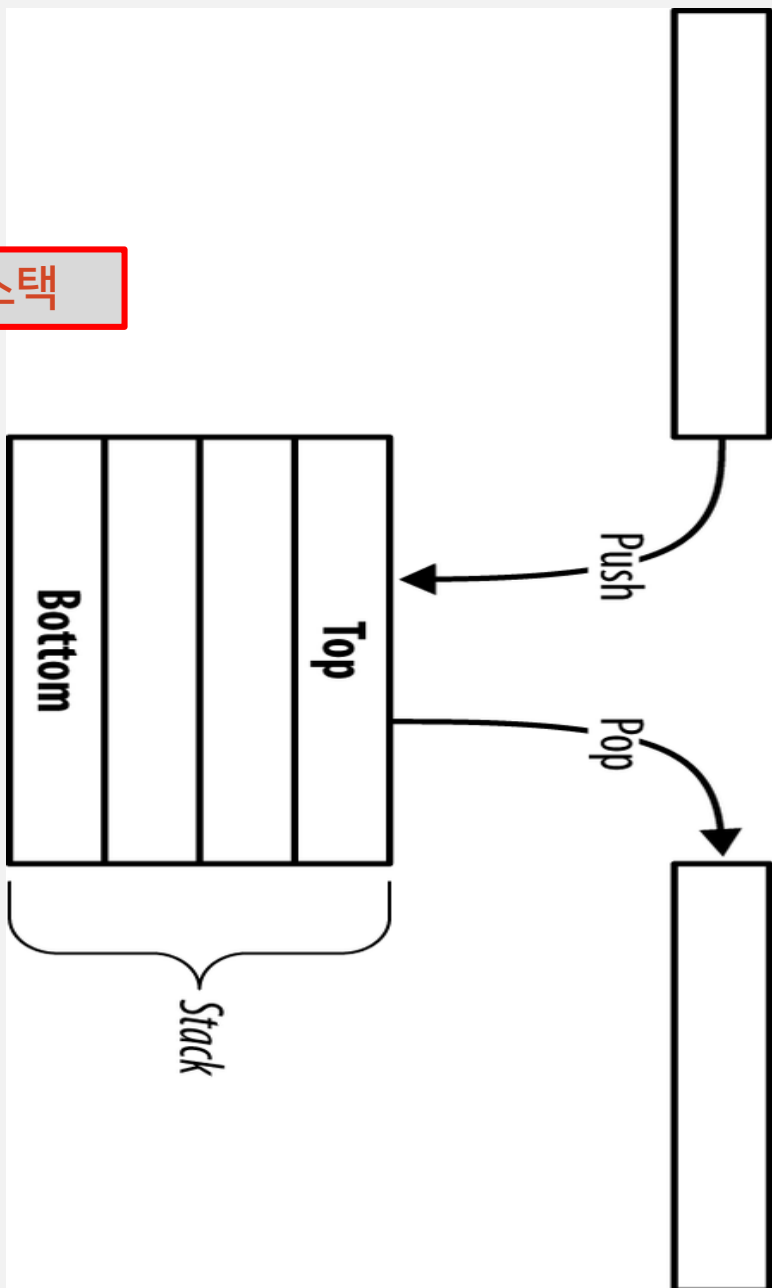
def update():
    boy.update()

def draw():
    clear_canvas()
    grass.draw()
    boy.draw()
    update_canvas()
```


Stack 자료 구조



list 를 이용한 스택



game_framework.py 분석(1)

```
def run(start_state):  
    global running, stack  
    running = True
```

```
    stack = [start_state]  
    start_state.enter()
```

start_state 를 담고 있는 스택을 생성

```
    while (running):  
        stack[-1].handle_events()  
        stack[-1].update()  
        stack[-1].draw()
```

현재 게임 상태(다시 말하면, stack top에 있는 게임 상태)에 대한 게임 루프를 진행

```
    # repeatedly delete the top of the stack  
    while (len(stack) > 0):  
        stack[-1].exit()  
        stack.pop()
```

스택에 남아있는 모든 게임 상태들을 차례로 제거

game_framework.py 분석 (2)

```
def change_state(state):  
    global stack  
    if (len(stack) > 0):  
        stack[-1].exit()  
        stack.pop()  
    stack.append(state)  
    state.enter()
```

현재 상태를 삭제한 후,
새로운 상태를 추가하고, enter로 들어간다.

```
def pop_state():  
    global stack  
    if (len(stack) > 0):  
        stack[-1].exit()  
        stack.pop()  
    if (len(stack) > 0):  
        stack[-1].resume()
```

Stack Top의 상태를 exit() 한 후, 상태를 제거.
이제 Stack Top에는 이전 상태가 있으므로, 이
내용을 다시 가져옴(resume)

game_framework.py 분석 (3)

```
def push_state(state):  
    global stack  
    if (len(stack) > 0):  
        stack[-1].pause()  
    stack.append(state)  
    state.enter()
```

현재 상태를 저장하고(Pause),
새로운 상태로 들어감.

```
def quit():  
    global running  
    running = False
```