**LEC01 : 파이썬 기초 (1)**

// 몫 %나머지 /나눗셈 \*\* 제곱

input 함수 : 사용자가 입력한 정보가 “문자열”로 되어 넘어옴

**Default Data Type - 기본 자료형**

int, float, str, bool

**Slice(슬라이스)**

title[0:3] ⇒ 인덱스 마지막은 안들어감 (3은 안써짐)

리스트&튜플에서 슬라이스 적용이 된다.

문자열의 일부분을 잘라내는 기법 - name[start : stop : step]

List에서 Slice가 적용됨

===============================

Dictionary

Tuple : 여러 개의 값을 동시에 관리, 리스트와 유사, 기본적으로 바꿀 수 없음

Set : 집합 자료형, 리스트와 달리, 중복을 허용하지 않고, 순서가 없음.

**list**

* append : 마지막 원소 추가
* sort : 정렬 , 리스트만 있는 함수
* remove(‘momo’) : 해당 요소 삭제
* del list : 리스트 전체 삭제
* 리스트 + 리스트 가능
* 리스트 - 리스트 불가능

**dictionary**

* 딕셔너리에선 키 값이 인덱스로 사용
* score.keys() : 딕셔너리 키 값을 아는 법 // score = {‘zwi’ : “hihi”, ….}
* score.values() : 딕셔너리 벨류값을 아는 법
* ‘zwi’ in score : 딕셔너리에 해당 키가 있나 확인하는 법
* score.clear() : 딕셔너리 요소 다 삭제하는 방법

**tuple**

* 슬라이스 가능  
  리스트랑 유사하다 다만, 값을 바꿀 순 없다. 프로그램 중 변경 되지 않는 값들의 모음이 필요할 때 사용
* 이미 있는 요소 자리에 값을 변경하거나 삭제는 안된다. 요소 추가는 가능하다.
* tuple + tuple 가능
* tuple - tuple 불가능

**set**

* 슬라이스 x
* 중복허용 x 순서 x
* set - set 가능
* set + set 불가능
* 원소 추가는 add
* 원소 제거 remove함수

===============================

**Complex Data type - 복합 자료형**

List – list **[] 순서 0 중복 0 변경 0**

* 순서가 있는, 중복을 허용하는 데이타들의 집합.
* 원하는 데이터를 찾기 위해, 순서 index 를 이용.

Dictionary-dict {}

* 검색을 위한 키를 갖는 데이타들의 집합
* key-value 쌍들의 집합

Tuple-tuple ( ) **순서0 중복 0 변경 x**

* 순서가 있는, 중복을 허용하는 데이타들의 집합
* 다만, 데이타값을 변경하는 것은 불가

Set-set **{ }순서 x 중복 x**

* 중복을 허용하지 않는, 순서에 상관없는 데이타들의 집합

**LEC03 : 파이썬 기초 (3)**

다중 대입 : a, b = b, a

random.randint(시작, 끝) – 끝 들어감!!

수학에서는 함수는 어떤 수식을 정의한 것.

**프로그래밍에서 함수란?**

어떤 특정한 일을 처리하는 기능을 모아놓은 것, 수학적인 함수도 구현 가능.

함수를 실행하려면 함수 호출을 해야하고 함수를 호출하려면 함수 정의가 먼저 되어야함

**LEC04 : 2D렌더링**

**게임이란?**   
가상 월드에 존재하는 여러 객체들의 상호작용을 시뮬레이션하고 그 결과를 보여주는(렌더링) 것

**게임의 기본 구성 요소**

1. 배경

2. 캐릭터, 오브젝트

3. UI – GUI, 입력

4. AI

5. 사운드

**2D게임**⇒현재 진행중인 게임 가상 월드의 내용을 화면에 2D 그림으로 보여주는 것  
⇒ 배경, 캐릭터(오브젝트)의 표현(렌더링)을 2D 이미지들의 조합으로 구성한다.

**캠버스의 좌표계**  
⇒open\_canvas(800,600)  
⇒왼쪽 맨 밑이 0,0  
⇒왼쪽 맨 위 (0,599)  
⇒오른쪽 맨 아래 (799,0)

⇒오른쪽 맨 위 (799,599)

**LEC05 : 2D렌더링**

**화면 플리커링** : 화면 깜빡 거리는 현상

**더블 버퍼링, 페이지 플리핑**

1. draw하면 그래픽스는 back 버퍼에 들어가고

2. update\_canvas()를 호출하면 screen에 back 버퍼 그래픽이 그려진다.

페이지 플리핑 ⇒ update\_canvas()  
화면 지우기 ⇒ clear\_canvas()// update\_canvas()

**스프라이트란 ?**

1. 게임 장면안에서 보여지는 이미지, 또는 애니메이션 되는 오브젝트

2. 2d게임에서는 게임의 모든 캐릭터들과 이동하는 물체들을 표현하는데 사용

3. 3d 게임에서는 2D로 표현될 수 있는 각종 오브젝트에 사용 된다.

**애니메이션이란?**

1. 여러 개의 이미지를 일정한 시간 간격을 통해서 화면에 뿌림으로써, 물체가 움직이는 효과를 주는 것
2. 스프라이트는 여러 개의 action으로 구성 됨
3. action: 달리기, 걷기 , 제자리 동작 등과 같이 캐릭터의 움직임을 나타낸다.
4. action은 여러 개의 Frame으로 구성된다. Frame은 한 개의 이미지를 말한다.

range(start,stop,step) : stop은 출력 되지 않음

clip\_draw(left, bottom, width, height, x, y)

character.clip\_draw(frame \* 100, 0, 100, 100, x, 130, 200, 200) : 확대

clip\_composite\_draw(self, left, bottom, width, height, rad, flip, x, y, w=None, h=None) : 좌우반전

**LEC06 : 입력 처리**

**키보드 및 마우스 입력 처리 과정**

1. 입력 이벤트들을 폴링한다. (get\_events())

2. 이벤트의 종류를 구분한다.(event.type을 이용)

3. 실제 입력 값을 구한다. (event. Key 또는 event.x ,event.y 등 )

**Global 키워드**함수 내에서 값이 결정되는 변수는 지역 변수로 간주된다. 따라서 변수를 전역이라는 점을 알리려면 반드시 global로 지정할 것

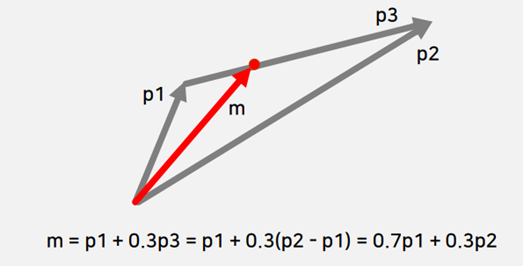
**마우스 좌표의 획득과 변환**

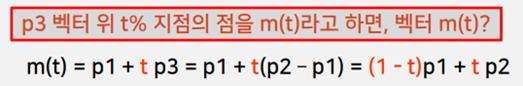
* 마우스가 이동하면, SDL\_MOUSEMOTION 이벤트가 발생 event.x 및 y는, 윈도우 API 의 좌표계를 따름. pico2d 좌표계 변환 필요
* x=event.x
* y=TUK\_HEIGHT-1-event.y

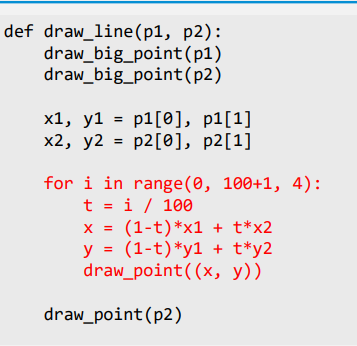
**LEC07 : 직선이동  
Bresenhem Line Algorithm**⇒컴퓨터 화면 상에 직선을 그리는 알고리즘  
⇒덧셈과 뺄셈을 이용함으로써, 고속으로 직선을 그릴 수 있음

⇒ y축과 평행인 직선 (x=c)을 그릴 수 없음

**Parametric Representation**

****

****

****

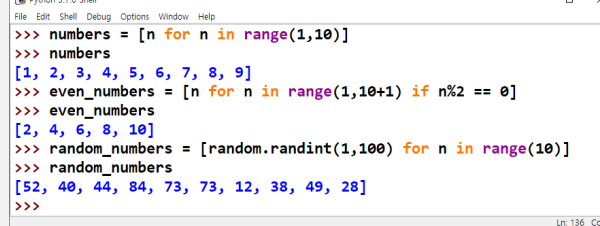
* 직선, 또는 곡선의 (x,y) 좌표를 공통적인 파라미터를 이용하여 표현하는 방법.
* 일반적인 수학적 표현에 비해, 컴퓨터를 이용하여 그리기가 편리함.
* 동일한 곡선에 대해, 파라미터 표현법은 여러 개 있음.

⇒ m(t) = (1 - t)\*p1 + t \* p2,

t의 범위: 0 <= t <=1

**Python List Comprehension**

* 리스트를 빠르게 만들기 위한 독특한 문법 구조
* 리스트 안에 있는 데이타들을 일정한 규칙을 가지고 생성해냄.



**LEC09 : 게임 오브젝트**

**추상화란?**   
꼭 필요로 하는 객체들만, 꼭 보여줘야 할 내용만, 꼭 보여줘야할 움직임만

**속성?**   
객체의 상태/ 상태를 나타내는 값 &게임 객체의 현재 상태   
**행위?(행동)**   
객체의 상태를 바꿔주는 것 x=x+2 & 시간에 따라, 혹은 이벤트에 반응해서 상태가 변하는 방식  
**속성 + 행위 ⇒객체**

**게임 객체** : 게임 월드를 구성하는 모든 요소들을 지칭  
객체 : 문제 영역의 실세계에 존재하는 구체적인 대상을 모델링한 것  
  
**클래스란 ?**

1. 유사한 여러 객체들에게 공통적으로 필요로 하는 데이터와 데이터 위에서 수행되는 함수들을 정의하는 소프트웨어 단위
2. 객체를 찍어내는 도장

**객체 생성 하라면 ?**⇒ 클래스라는 틀을 이용하여 붕어빵 찍어내듯이 객체를 생성하게 됨  
⇒ 찍어내는 과정을 object instantiation 이라고 함

**인스턴스 ?**

⇒ 생성된 각각의 객체

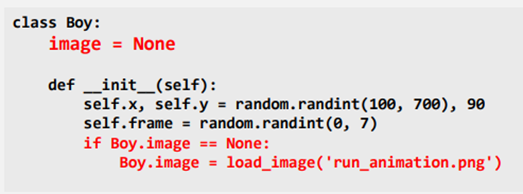
⇒ 모든 객체는 어떤 클래스로부터 생성된 인스턴스

**게임 기본 구조**

****

**LEC10 : 캐릭터 컨트롤러 (1)**

객체마다 로드 이미지를 한번만 하자 ! 속도를 빠르게 해줌

****

클래스 자체에 할당되는 변수. 객체들은 공유하는 동일한 변수를 갖게 된다.

단 한번의 이미지 로딩만 수행하게 된다.

**캐릭터 컨트롤러**

⇒ 게임 주인공의 행동을 구현한 것이다.

* 키 입력에 따른 액션
* 주변 객체와의 인터렉션

**상태 다이어그램**1**.** 시스템의 변화를 모델링하는 다이어그램  
2. 사건이나 시간에 따라 시스템 내의 객체들이 자신의 상태를 바꾸는 과정을 모델링 함

**상태(state)**

어떤 조건을 만족하는 동안 머무르면서, 정해진 일을 수행하고 이벤트를 기다리는 “상황”

1. entry action : 특정한 상태로 들어갈 때마다 발생하는 일
2. exit action : 특정한 상태에서 나갈 때 마다 발생하는 일
3. do activity : 특정 상태에 머무르는 동안 수행하는 일(반복될 수 있다)

**이벤트 : 상태 변화를 일으키는 원인이 되는 일**  
외부 이벤트 : 키보드 입력  
내부 이벤트 : 타이머

경우에 따라서는 이벤트 없이도 상태 변화가 될 수 있다

**Python module**⇒파이썬의 정의와 문장을 담고 있는 파일  
⇒그 자체로도 실행 가능하고, 다른 모듈에서 임포트해서 사용할 수 있다. 임포트가 되면 그 자체가 하나의 객체가 된다.(싱글톤 객체가 된다)

**모듈의 사용**

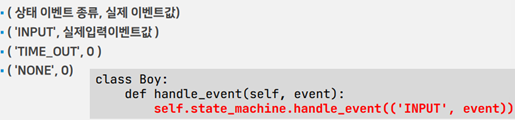
Import 파일명 으로 한 경우 파일명 안에 클래스를 사용하고 싶다?  
⇒Grass=game.Grass() 이렇게 모듈이름.클래스() !

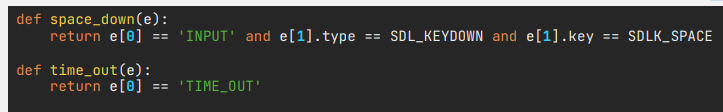
From 파일명 import 클래스명 이렇게 한 경우  
⇒그냥 파일명 안쓰고 바로 클래스 인스턴스 생성 가능하다.

**staticmethod**여기서 class 의 역할은 특정함수를 모아서 그루핑하는 역할. 객체 생성이 아님!

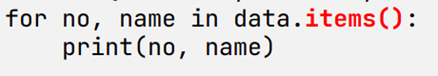
**LEC11 : 캐릭터 컨트롤러 (2) - 코드 보고 들어가기**

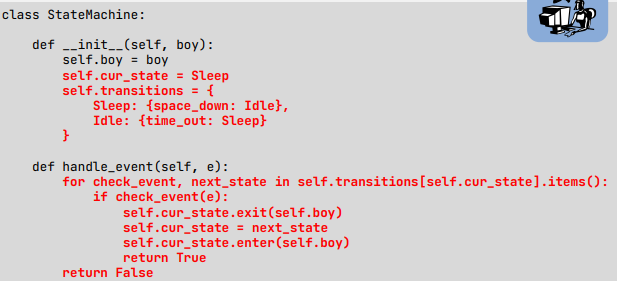
**튜플**을 이용해서 **상태 이벤트**를 나타내도록 함

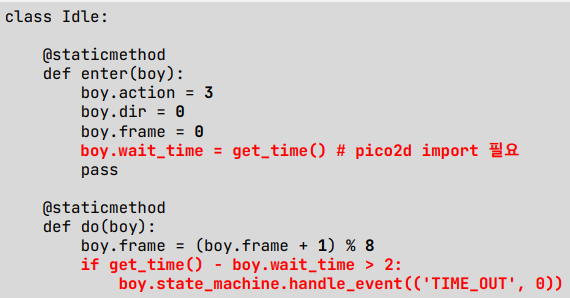


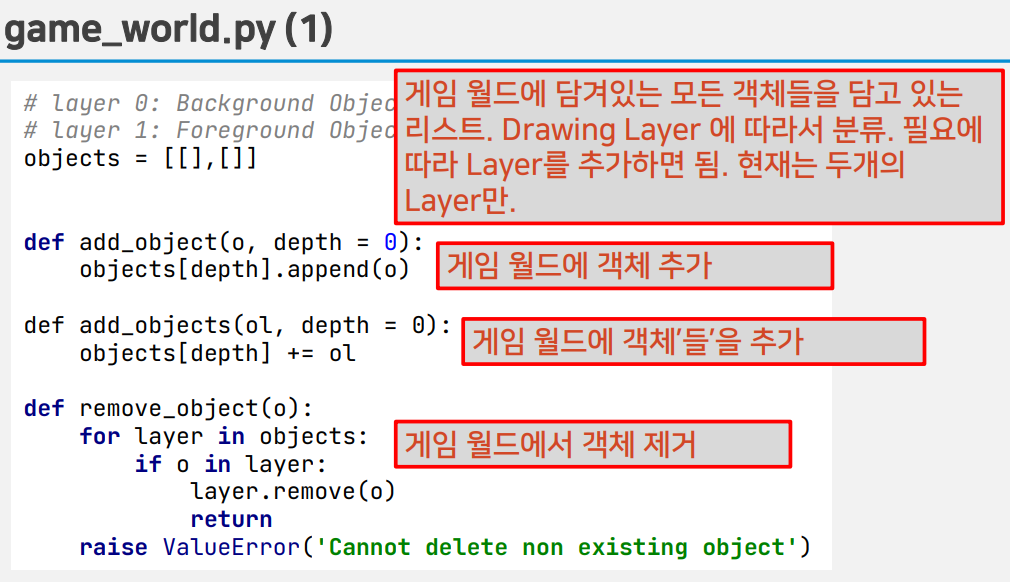
이벤트 체크 함수에서 튜플을 인자로 받아서 어떤 이벤트인지 판단함  


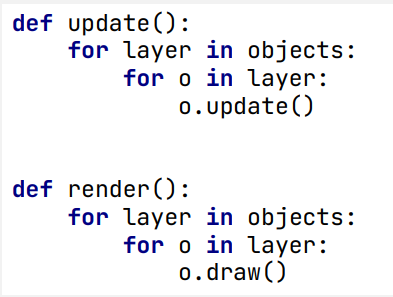
**Dictionary 아이템 꺼내기**







**LEC12 : 게임월드**

****

* depth : 그리는 순서를 지정 (숫자가 클수록 나중에 그리므로 앞쪽에 표시됨)
* 객체를 game\_world 리스트에 넣을 때 몇 번째 레이어에 넣을 것인지 인자를 넣어 줄 것

**LEC13 : 게임월드  
게임 모드란?**

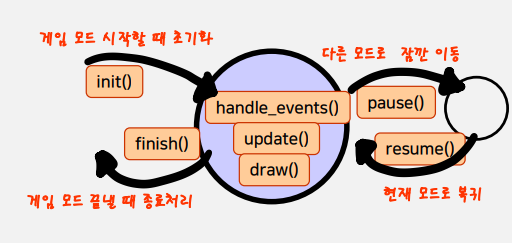
1. 게임 프로그램 실행 중에 지속적으로 머물러 있는 **특정 상황**, 씬, ....
2. 사용자 입력(키보드 또는 마우스 입력)에 대한 대응 방식은 게임 모드에 따라 달라짐.
3. 작은 **게임 루프**로 볼 수 있음.

게임 프로그램은 여러 개의 게임 모드들의 연결로 구현됨.

**게임 프레임워크**

1. **게임 모드**들을 효과적으로 연결하는 **소프트웨어 구조.**
2. 일종의 Task Switching System

**게임 모드의 구현**

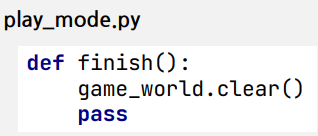


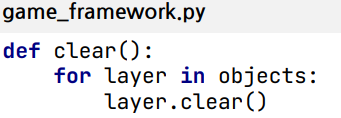
push\_mode 와 pop\_mode 를 호출하면, pause와 resume 이 call back

되므로 실제 내용은 없더라도 뼈대는 만들어줘야 함.

**모드 이동 : game\_framework을 이용**

* run(mode): mode를 시작 게임 모드로 하여, 게임 실행을 시작함.
* quit(): 게임을 중단
* change\_mode(mode): 게임 모드를 mode로 이동. 이전 게임 모드를 완전히 나옴.
* push\_mode(mode): 게임 모드를 mode로 이동. 단, 이전 게임 모드 데이타는 남아 있음.
* pop\_mode(): 이전 게임 모드로 복귀

**모드에서 나갈 때 clear가 필요하다!!  
**

****

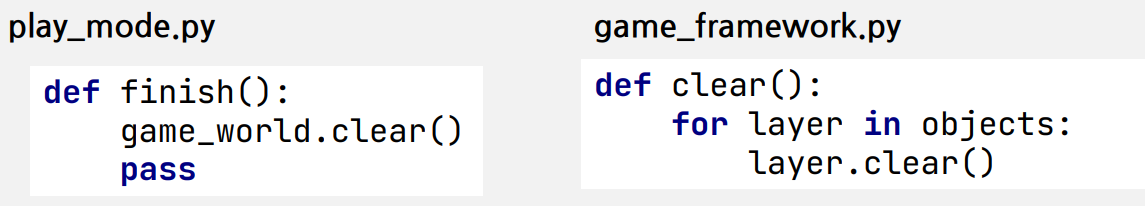
**objects 리스트의 리스트들의 내용을 싹 클리어!!**

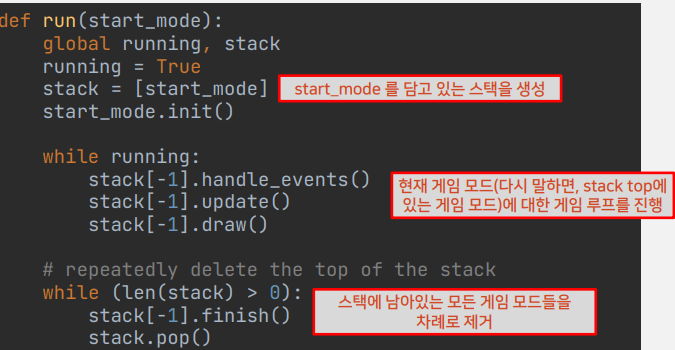
**game\_framework.py 분석 <사진 보면서 읽기>**

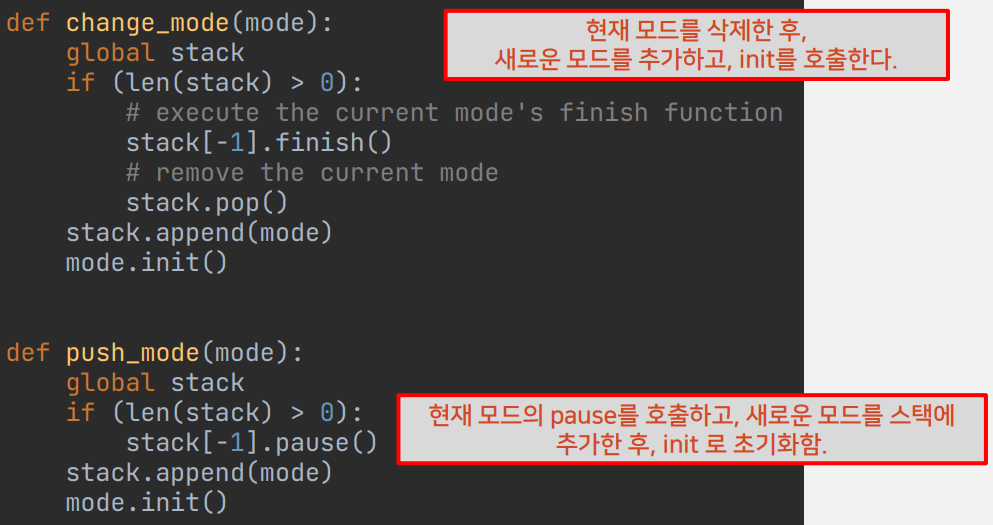
* start\_mode 를 담고 있는 스택을 생성
* 현재 게임 모드(다시 말하면, stack top에 있는 게임 모드)에 대한 게임 루프를 진행
* 스택에 남아있는 모든 게임 모드들을 차례로 제거
* 현재 모드를 삭제한 후, 새로운 모드를 추가하고, init를 호출한다.
* 현재 모드의 pause를 호출하고, 새로운 모드를 스택에 추가한 후, init 로 초기화함
* 현재 모드를 finish 한 후, 현재 모드를 제거함. 이제 Stack Top에는 이전 모드가 있으므로, 이전 모드에 대해서 resume 을 호출함

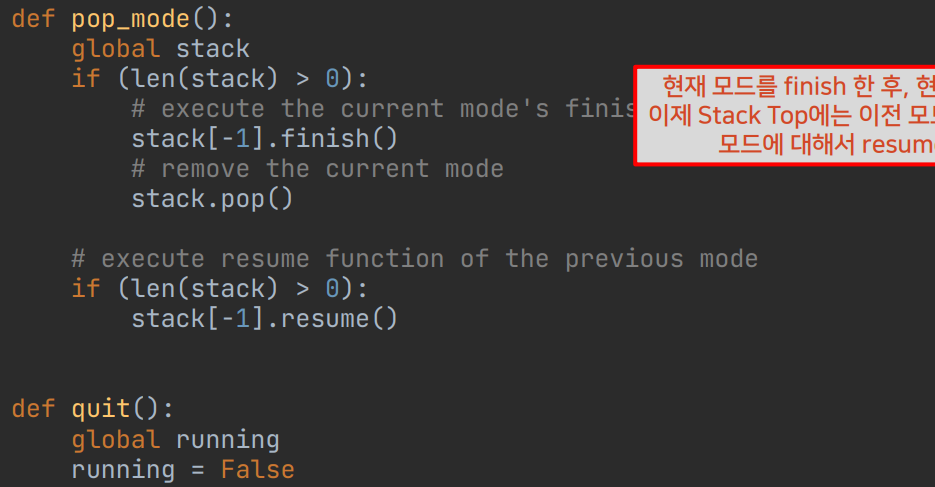
**게임 모드의 뼈대**

1. def init(): pass
2. def finish(): pass
3. def update(): pass
4. def draw(): pass
5. def handle\_events(): pass
6. def pause(): pass
7. def resume(): pass

**game world clear  
**

****

****

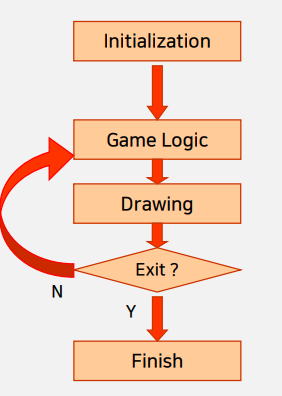
****

**LEC14 : 시간**

**시간의 개념이 없는 코드의 문제점?**

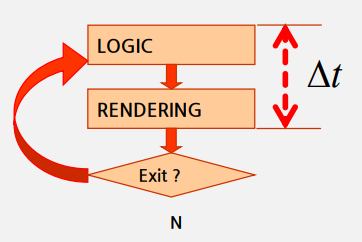
1. **CPU 성능에** 따라, 물체의 움직이는 **속도**가 달라짐.

**프레임(Frame)**

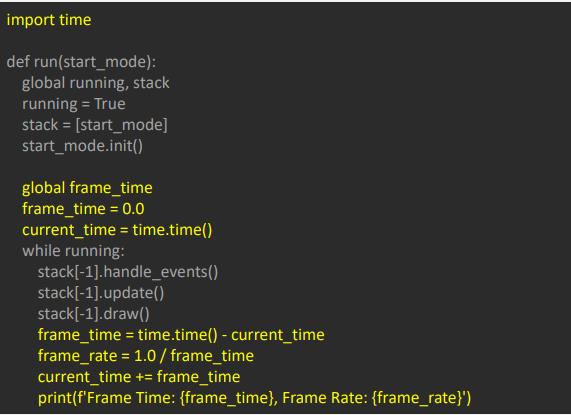


1. 특정 시점에서 **씬(장면)**을 **화면**에 그린 한장의 **그림.**
2. 드로잉(렌더링)의 결과물 → 드로잉(렌더링)이 끝나는 시점에 만들어짐.
3. 스크린샷

**프레임 시간(Frame Time)**

****

1. 한장의 **프레임**을 만들어내는데 걸리는 **시간.**

**  
<game\_framework.py>**

**프레임 속도(Frame Rate)**

**프레임 속도란**?

1. 얼마나 빨리 프레임(일반적으로 하나의 완성된 화면)을 만들어 낼 수 있는지를 나타내는 척도
2. 일반적으로 **초당 프레임 출력 횟수**를 많이 사용한다.
3. **FPS**(Frame Per Sec)
4. 컴퓨터 게임에서는 일반적으로 최소 **25~30 fps** 이상이 기준이며,   
   최근엔 **60, 120fps**

프레임 시간과 프레임 속도의 관계   
⇒ Frame per sec = **1 / Frame time**

**프레임 시간은 균일하지 않다..**

* 씬이 복잡하거나, 처리해야 할 **계산**이 많으면 시간이 많이 걸림
* 동일한 씬이라도, **컴퓨터의 성능에** 따라서도 차이가 남.

**문제점은?**

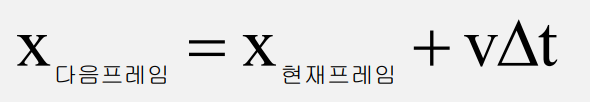
* 게임의 **실행속도가 컴퓨터마다**,,,, 또는 게임 내의 **씬의 복잡도**에 따라 달라지므로, **게임 밸런싱에** 큰 문 제를 야기함. → ex. 캐릭터의 이동속도가 달라짐.

**해결 방법은? 아예 고정하기…**

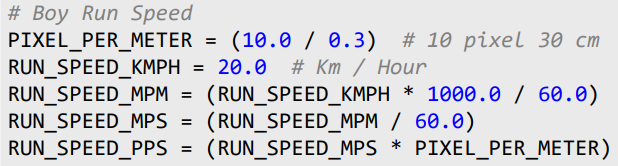
* 그래픽 라이브러리 자체에서 싱크를 조정하도록…
* open\_canvas(1600, 600, sync=True)
* 60fps 로 고정할 수 있음

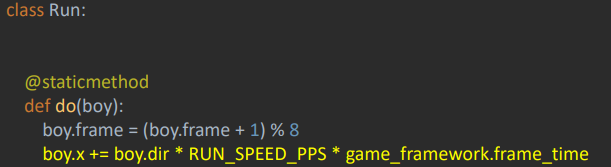
문제점? → **그림을 그리는 시간이 있어서 항상 60**이 나오는 것이 아니다.

**아주 아주 아주 근사한 방법 →** 게임 객체들의 운동에 “**시간**”의 개념을 도입   
**거리 = 경과시간(frame\_time) \* 객체의 속도**   
**위치 = 초기 위치 + 거리**

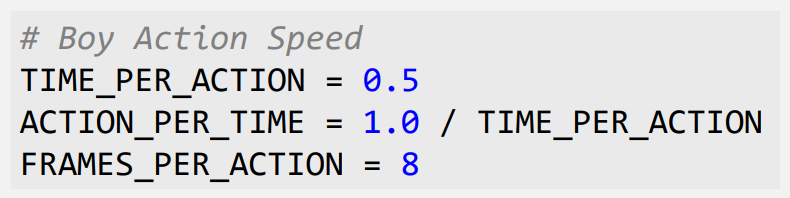


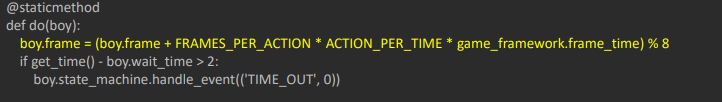
**속력 계산 (boy.x를 cpu 고려해서 계산하기 위함)**





**frame 증가량 계산 (frame도 cpu 고려)**





**LEC15 : 충돌처리**

**충돌 검사(Collision Detection)**

충돌 검사 → 게임 상의 **오브젝트** 간에 충돌이 발생했는지를 검사하는 것.

모든 게임에서 가장 기본적인 물리 계산.

기본적으로 시간이 많이 소요되기 때문에, 게임의 오브젝트의 특성에 따라 각종 방법을 통해 최적화해야 함. ⇒ **O(N2) 알고리즘 , nC2 = n(n-1)/1\*2**

**픽셀 단위의 충돌 검사 → 두 개의 오브젝트들의 모든 점들을 일일이 비교.**   
각 오브젝트들의 픽셀수를 곱한 만큼의 계산 시간이 소요됨.  
**캐릭터 픽셀 수 \* 공 픽셀 수**

점과 원: |P-C|^<= r^2

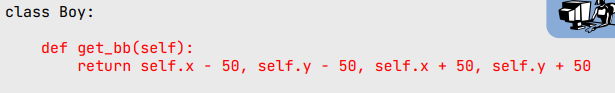
원과 원 : |C1-C2|^2 <= (r1+r2)^2

점과 사각형: min\_x <= Px < = max\_x and min\_y <= Py <= max\_y

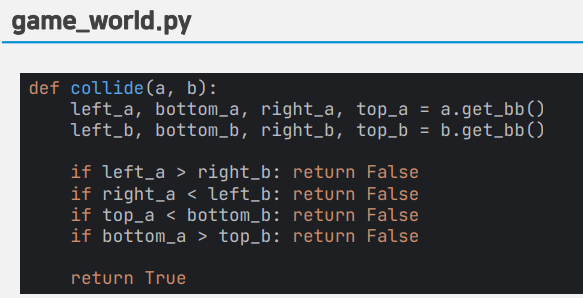
사각형과 사각형 : left A > right B : false , right A < left B: false, Bottom A > top B : false , Top A < Bottom B: false

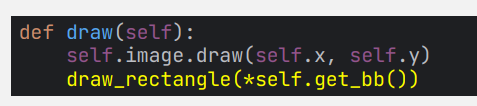
**바운딩 박스(Bounding Box)를 이용한 충돌 검사**

1. 오브젝트를 감싸는 사각형(바운딩 박스)의 충돌을 비교.
2. 사각형의 두개의 교차 여부만 결정하면 되 므로 매우 빠름
3. 오브젝트의 형태가 복잡하면, 충돌 검사 결 과가 매우 부정확해짐.



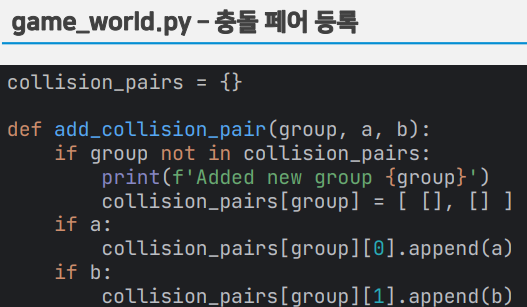
<충돌 영역 설정> - 왼쪽 하단 x,y 좌표, 오른쪽 상단 x,y좌표



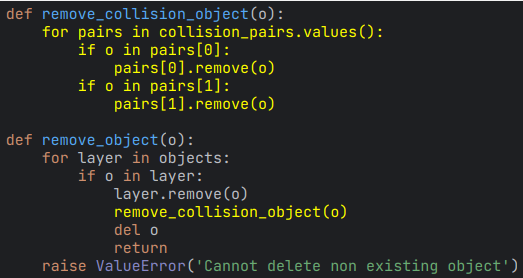


**좀 더 객체 지향적인 방법은??**

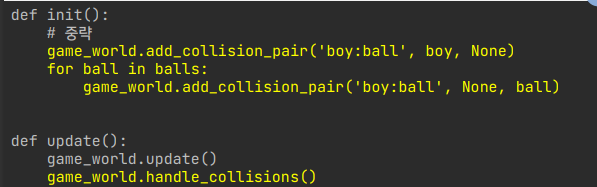
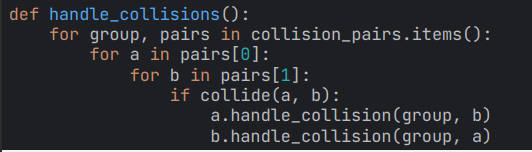
1. 충돌 처리가 필요로 하는 두개의 객체 A, B 를 먼저 일일이 등록하고, 나중에 등록된 페어 전체 에 대해서 충돌 검사를 수행한 후, 충돌이 있으면 A, B 에게 충돌 처리를 할 수 있도록 지시.
2. 충돌 처리해야할 대상들을 그룹화해서 처리



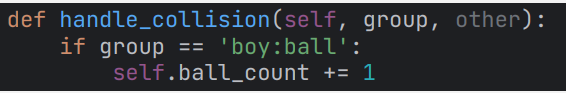
<충돌 페어 등록>



<충돌 객체의 제거>

  
<충돌 페어 추가 및 충돌 처리 업데이트>  


<game\_world.py – 충돌 감지에 따른 충돌 처리>



**충돌 검사 및 처리 기본 절차**

1. 충돌 처리가 필요한 객체에 대해서 충돌 영역 정의
2. 디버그를 위해서 충돌 영역을 시각화할수 있도록 설정
3. 공간에 이미 존재하는 여러 객체 들 중 충돌 처리가 필요한 두개의 객체 A, B 를 골라서 등록
4. 만약 게임 실행 중에 생성된 객체에 대한 충돌 처리가 필요하면, 그때마다 실시간으로 A, B 를 등록. 만약 한쪽이 이미 등록된 상태라면, None 으로 처리.
5. 등록된 모든 충돌 페어에 대해서 매 프레임마다 충돌 검사를 실시.
6. 충돌이 발생한 경우, A,B 각각에 대해서 충돌 처리하도록 지시

**트리거** : 특정 ( 위치 ) 에 캐릭터가 들어갈 경우, ( 이벤트 ) 을 발생

**LEC16 : 인공지능**

**게임 인공 지능**

* 게임 객체는 **주변의 상황을 인식(Sens)**
* **인식된 결과를 바탕으로 행동을 결정(Think)**
* 실제로 행동을 **수행함(Act)**

**FSM – 가장 전통적인 게임 AI 구현 방식**

▪시스템의 변화를 모델링하는 다이어그램.   
▪사건이나 시간에 따라 시스템 내의 객체들이 자신의 상태(state)를 바꾸는 과정을 모델링함.

**Behavior Tree**

1. 객체의 인공지능행동을 트리 구조로 구현한 것.
2. FSM 방식 – 상태와 이벤트에 따라서, 다음 상태를 결정
3. BT 방식 – **Goal 을 달성하기 위한 Task들을 구성**. 재사용이 쉽고 직관적임.

**Behavior Tree 기본 구조**

▪트리 구조   
⇒ 말 그대로, 객체의 행위들을 tree 구조로 연결하여 나타냄.

▪매 프레임마다 tree 구조가 실행됨.  
⇒ Root node 부터 시작해서, 아래로 실행되어 나감.

▪node는 상태값을 반환함.   
⇒ SUCCESS, FAIL, RUNNING   
▪Node가 자식 노드가 있으면, 자식 노드들을 실행하고, 그 결과를 종합하여 노드의 최종 상태 값을 결정함.

**Leaf Node → 단위 작업을** 수행하는 노드로써, **Action** 또는 **Condition** 처리.

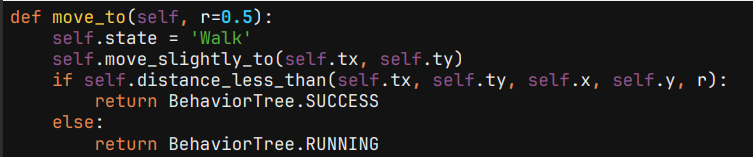
**Sequence Node**

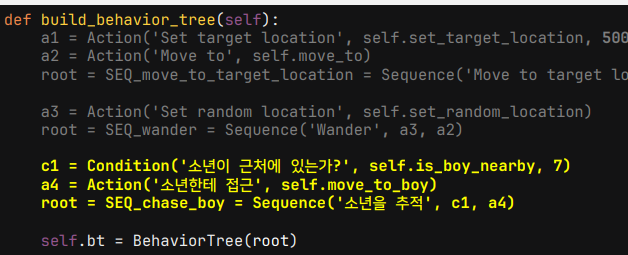
1. 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하며서 실행됨.
2. 모든 자식 노드가 다 SUCCESS 되면, 노드도 성공
3. 여러 개의 작업이 모두 다 차근 차근 진행되어야 하는 경우 – AND 조건
4. 하나라도 FAIL 되면, 실행 중단. Sequence Node 도 FAIL
5. 실행 결과, 처음으로 RUNNING이 나오면, 자식 노드의 위치를 기록함. 결과는 RUNNING임.
6. 어떤 목표를 달성하기 위해 수행해야 하는 Task 들을 차례로 모두 완수해야 하는 경우에 사용 됨.

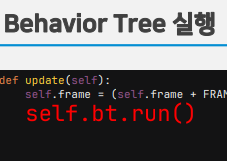
**Selector Node**

1. 자식 노드 중, 하나만 성공하면 성공
2. 여러 개의 작업 중, 하나를 선택하는 개념 – OR
3. 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하면서 실행됨.
4. 실행 결과 처음으로 SUCCESS, 또는 RUNNING이 나오면 더 이상 진행되지 않으며, **노드의 결과는 SUCCESS 또는 RUNNING 이 됨.**
5. 모든 자식 노드가 다 FAIL이면, 노드의 결과도 FAIL임.
6. 작업에 우선 순위를 부여할 때 사용됨. 즉, 왼쪽에 있는 노드가, 오른쪽에 있는 노드보다 우선 순위가 높음.

PDF 한 번 볼 것(코드)



****

****

**LEC17 : 스크롤링**

**스크롤링(Scrolling)**그림이나 이미지의 일부분을 디스플레이 화면 위에서 **상하좌우**로 움직이면서 나타내는 기법.

**실제 좌표와 화면 좌표를 분리 처리**

실제 공간 좌표 – 객체의 실제 좌표 계산할 때,   
화면 좌표 – 화면 상에 그릴 때

**스크린 윈도우를 이용한 스크롤링**

1. 맵 상의 플레이어의 실제 좌표 계산

2. 플레이어를 가운데에 놓고, 맵 상의 **윈도우 좌표를 계산**

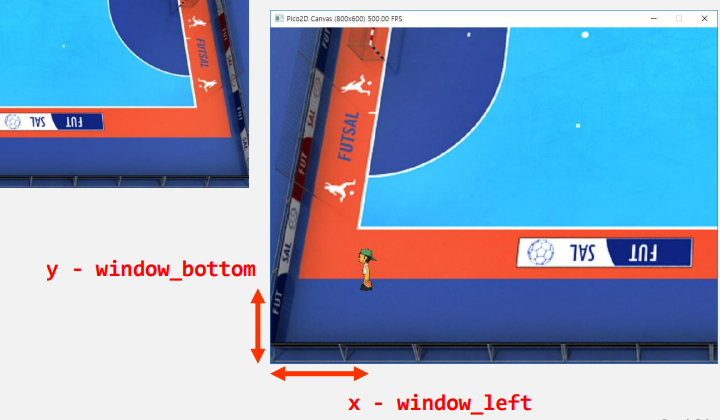
[맵을 클리핑할 위치 계산 (왼쪽 하단 좌표)]

window\_left = x-canvas\_width//2 window\_bottom = y-canvas\_height//2

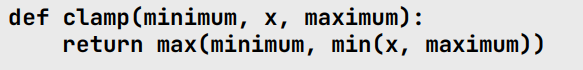
x, y는 플레이어의 화면상의 좌표

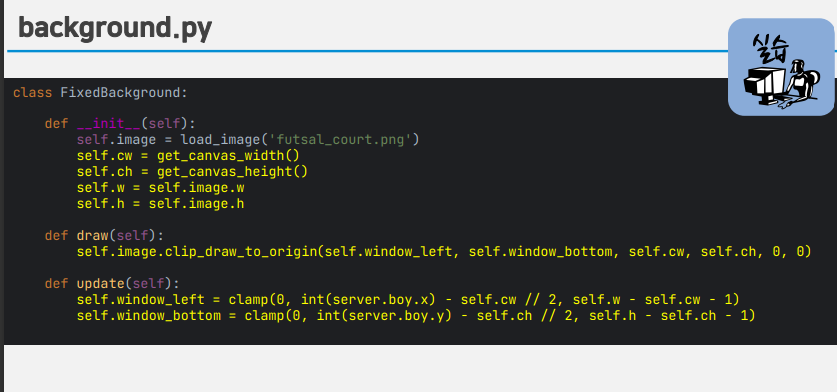
3. 윈도우 영역을 클리핑하여, 캔버스에 그림.   
**clip\_draw\_to\_origin :** 피봇 맨왼쪽 맨아래쪽 기준

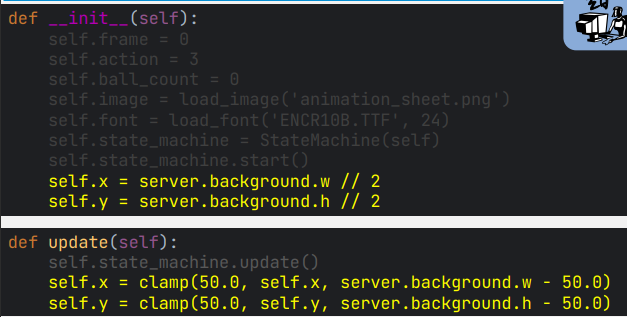
4. 플레이어를 캔버스에 그림.

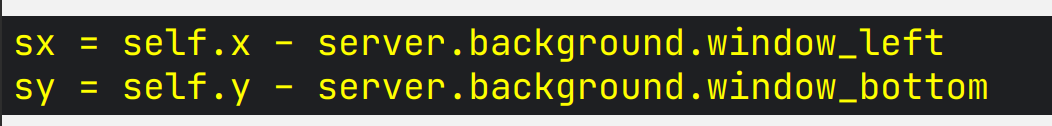


**clamp함수**





****

****

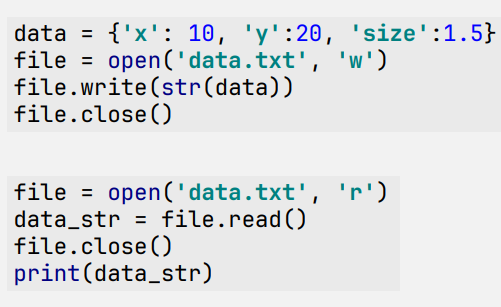
****

**LEC18 : 게임데이터**

**직렬화(Serialization)**▪프로그램 내의 **객체 데이터를 외부에 저장 또는 내보내는 행위.**▪나중에 다시 **복구**(de-serialization)할 수 있어야 함.

**파이썬 파일 입출력**

* C와 매우 유사
* open()으로 열고, close()로 닫음.
* write(), read()는 각각 str 을 쓰고 읽을 수 있음.(text mode 일 경우)



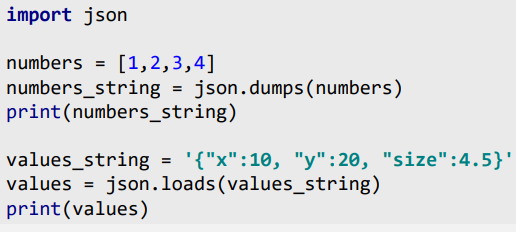
(string으로 쓰고 string으로 읽어온다)

**JSON(Java Script Object Notation)**

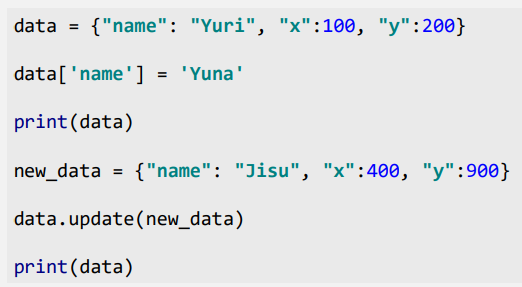
1. File write/read 의 경우, 문자열을 **실제 데이터로 변환하는 작업**이 필수적임.   
   ⇒ 까다롭고 복잡하고, 에러 발생의 소지가 많음.
2. 객체를 **교환(저장 및 전송 등)**하기 위한 **텍스트 형식 표준**
3. 파이썬의 **리스트와 딕셔너리**와 거의 동일
4. json 형식에서는 **문자열 표시를 큰따옴표**만 허용!

**Python JSON Module의 마법**

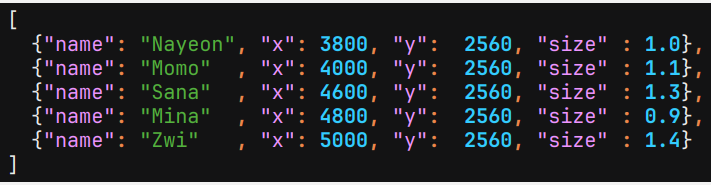
* import json   
  s = json.dumps(o) # 객체 obj 를 문자열 s로 변환
* o = json.loads(s) # 문자열 s 를 객체 o로 변환
* json.dump(o, f) # 객체 o를 파일 f로 저장
* o = json.load(f) # 파일 f를 로드해서 객체 o로 변환

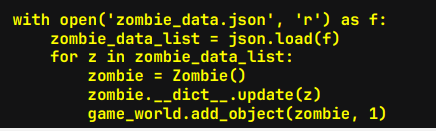


마지막 values의 타입은 딕셔너리가 된다. 문자열은 loads로 딕셔너리 객체로 바꿀 수 있음 .



dictiony update -> 인자로 받은 딕셔너리에 기존의 키와 중복되는 키이 있으면 업데이트를 해준다.

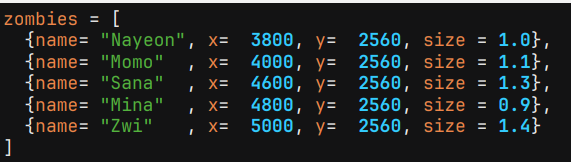


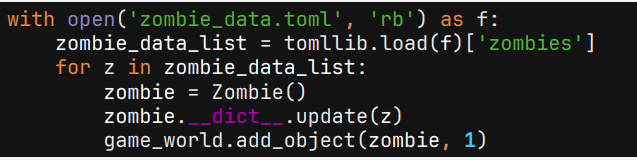


(json + 객체.\_\_dict\_\_.update()를 이용해서 객체에 값을 넣어주기)

**\_\_dict\_\_**   
▪클래스를 이용해서 생성한 객체는 모든 **속성(멤버 변수)을 dictionary 형태로 내부적으로 저 장하여 사용함**.  
▪obj.\_\_dict\_\_ 라는 내부변수가 바로 그것임.  
 ▪이것을 통해, 객체의 속성을 쉽게 바꿀 수 있음. – 다른 dictionary 데이터를 이용해서.

**TOML**

****

****

설정 데이터를 좀 더 읽기 쉽게 저장하기 위한 텍스트 파일 형식

**게임 Save 와 Load**

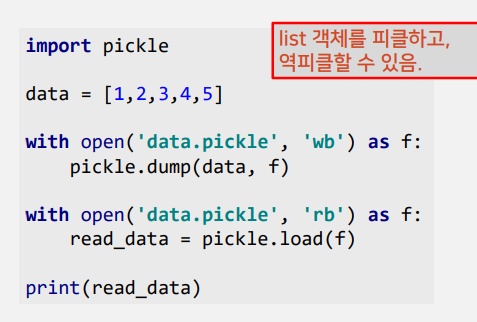
* 게임 플레이 중, **특정 상황을 저장**하고, **나중에 다시 불러와서 쓰는 기능**.
* 가장 쉽게 구현하는 방법은 게임 월드의 **모든 객체의 상태를 저장**하고, 나중에 불러오는 것.
* JSON 이용?

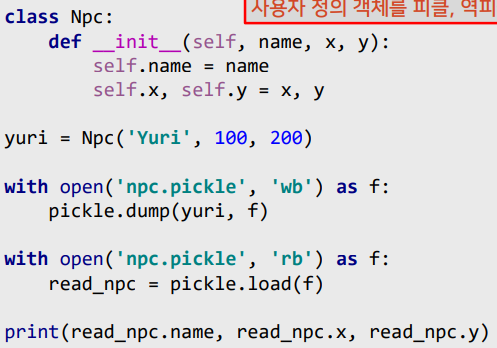
간단히 객체들의 저장은 되지만, JSON의 가장 **기본적인 data type** 만을 지원하기 때문에, 게임 내 객체 들의 다양한 데이터를 저장하는 것은 번거로울 수 있음.

* 가장 확실한 해법은? → Python Pickle !!!!

**Pickle(https://docs.python.org/ko/3/library/pickle.html )**

* 파이썬이 제공하는 객체 직렬화 모듈
* **대부분의 파이썬** **내부 데이타**들을 **직렬화**할 수 있음.





**Pickle 로 안 되는 것들..**

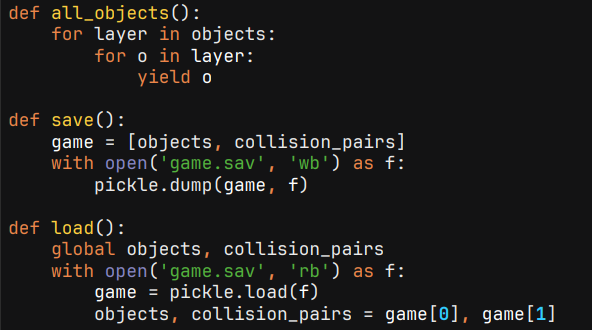
* **클래스 변수**
* 순수 파이썬이 아닌, **외부 라이브러리를 통해서 획득한 데이터**

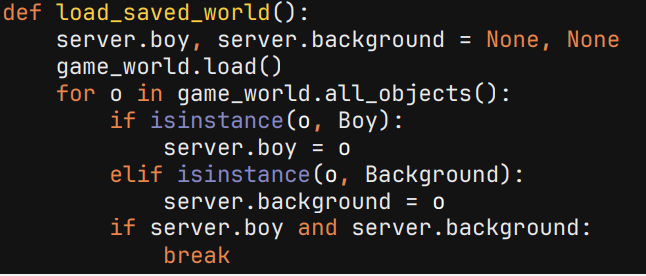
**Python Generator**

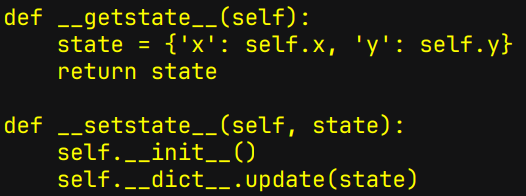
* 객체들을 하나씩 만들어서(발전) 넘겨주는 기능
* for 문 등에서 효과적으로 사용.

**객체의 저장과 복구 (코드 문제 나옴)**

* 객체의 멤버 변수들을 모두 저장할 필요가 없는 경우가 대부분.
* 저장이 필요한 내용만 선택적으로 골라서 저장할 수 있음.
* 피클링이 필요한 멤버 변수만 저장하고 복구함 \_\_getstate\_\_ & \_\_setstate\_\_







<boy.py>

**\_\_getstate\_\_** : 객체를 저장할 때, 즉 피클링할 때 필요한 내용을 결정. 피클 모듈은 객체의 \_\_getstate\_\_ 함수를 이용해서, 저장할 내용을 가져옴

\_\_setstate\_\_ : 객체를 복구할 때, 즉, 역피클링할 때, 복구할 내용을 결정

pickle.load 를 이용해서 객체를 만들 경우, \_\_init\_\_() 가 자동 호출되지 않으므로, 수동으로 호출함. 저장된 내용 이외의 값들을 채우기 위함

**LEC19 : 사운드 및 패키징**

**사운드란 무엇인가?**

▪ 음원으로부터의 **기계적 진동이 공기** 등의 **매체를 통해 전파되는** 것.   
▪ 전파되는데 시간이 걸린다.

**진폭과 주파수**

▪**진폭**(Amplitude) 🡺 **파형의 크기**   
▪**주파수**(Frequency) **🡺 초당 특정 파형이 반복되는 횟수**. **단위는 Hz**

**디지털 사운드와 합성 사운드**

**▪디지털 사운드**(Digital Sound)   
소리의 직접적인 **녹음**   
**효과음**등에 사용(폭발, …)   
▪**합성 사운드**알고리즘과 톤 발생기에 의하여 **합성된 소리.**  
주로 **음악의 재생에** 사용

**디지털 사운드의 샘플링 주파수와 데이터 비트수  
▪샘플링 속도**   
▪ 디지털 사운드를 기록할 때, 초당 몇번의 샘플링을 하는가? 🡺 샘플링되는 사운드 주파수보다 2배 이상으로 샘플링을 해야함.   
  
▪**샘플 당 비트수(bits ber samples)**   
🡺 **8 비트 샘플**: 256개의 진폭 크기**. 게임 효과음** 등에는 충분.   
🡺**16 비트 샘플**: 65536개의 진폭 크기. **음악** 등에 사용

**미디 (MIDI: Musical Instruments Digital Interface)**

🡺음악 재생을 기술하는 일종의 언어.

**청크**   
🡺일정한 크기를 가지는 사운드 데이터.

**채널**   
🡺소리를 전달하는 통로.

**Music 관련 function 들**

* 음악 파일(mp3, ogg) 의 로딩
* load\_music(파일이름 )

**음악 연주 기능**

* repeat\_play() - 반복재생
* play(n) – n 번 재생
* set\_volume(v) – 소리 크기 0~128
* get\_volume() – 현재 소리 크기
* stop() – 중단
* pause() – 일시정지
* resume() - 재개

**Wav 관련 function 들**

* 이펙트 파일(wav)의 로딩
* load\_wav(파일이름)

**이펙트 재생 기능**

* repeat\_play() - 반복재생
* play(n) – n 번 재생
* set\_volume(v) – 소리 크기 0~128
* get\_volume() – 현재 소리 크기

