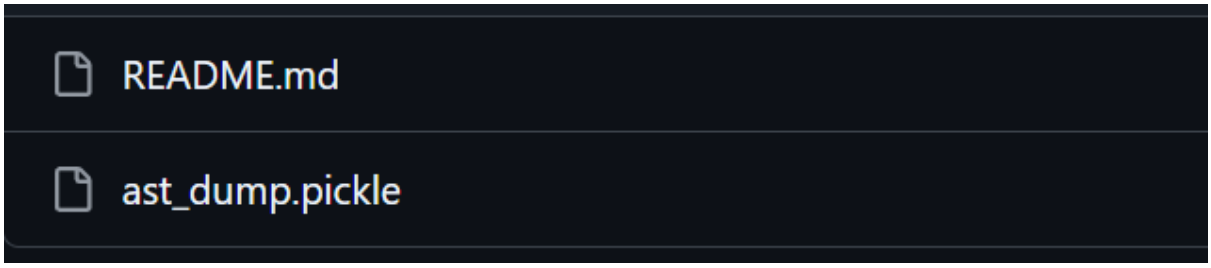

pickle 파일 하나가 주어진다.

pickle파일은 class, 함수, 변수 등 파이썬 객체를 직렬화한 파일이다.

pickle을 사용하는 이유는 시간절약과 코드 재사용성에 있다. pickle은 정확히 말하면 파이썬 객체(실행된 결과)를 저장한 파일이다. 그냥 함수를 호출해서 사용할 경우 처음부터 끝까지 코드를 시행하니 시간도 오래걸린다. 그런데 pickle 파일을 바로 load해서 사용하면 해당 실행 데이터를 바로 쓸 수 있어 시간절약을 할 수 있다.

pickle을 왜 사용하는지는 게임 세이브 버튼에 예시를 들어서 설명할 수 있다.

게임 저장 버튼을 누르면 다음과 같은 정보가 저장된다.

- 플레이어 위치
- 체력·마나
- 인벤토리 아이템
- 진행된 퀘스트
- 적 상태
- 맵 상태

해당 데이터는 실행된 결과이다.

해당 데이터를 함수로 호출해서 불러온다고 하면 시간도 오래걸릴 뿐더러 리소스 낭비가 된다. 그래서 pickle이 존재한다. 해당 데이터들을 pickle로 저장하면 load만 하면 바로 불러와서 사용할 수 있기 때문이다.

```python
import pickle
import ast

with open("ast_dump.pickle", "rb") as f:
    ast_obj = pickle.load(f)

    source = ast.unparse(ast_obj)

    with open("deced.py", "w") as df:
        df.write(source)
```

어쨌든 해당 파일을 역직렬화해서 코드로 표현한다.

```python
import random
import base64
import hashlib
from typing import List, Dict, Optional, Union, Any
class DataProcessor:
    def __init__(self):
        self.data = []
        self.cache = {}
        self.config = {'debug': False, 'verbose': True, 'timeout': 30, 'retries': 3}
    def process_data(self, input_data: Any) -> Optional[str]:
        if not input_data:
            return None
        transformed = self._transform(input_data)
        validated = self._validate(transformed)
        return self._encode(validated) if validated else None
    def _transform(self, data: Any) -> str:
        fake_secret = 'not_the_flag'
        decoy_1 = 'tjctf{h4ck3r_t1m3}'
        decoy_2 = 'tjctf{c0d3_br34k3r}'
        decoy_3 = 'tjctf{s3cur1ty_f41l}'
        decoy_4 = 'tjctf{3xpl01t_m0d3}'
        decoy_5 = 'tjctf{p4ssw0rd_cr4ck}'
        decoy_6 = 'tjctf{syst3m_h4ck}'
        decoy_7 = 'tjctf{d4t4_br34ch}'
        decoy_8 = 'tjctf{n3tw0rk_p3n3tr4t3}'
        decoy_9 = 'tjctf{cyb3r_w4rr10r}'
        decoy_10 = 'tjctf{c0mput3r_v1rus}'
        decoy_11 = 'tjctf{m4lw4r3_d3t3ct}'
        decoy_12 = 'tjctf{f1r3w4ll_byp4ss}'
        decoy_13 = 'tjctf{r00tk1t_1nst4ll}'
        decoy_14 = 'tjctf{k3yl0gg3r_r3c0rd}'
        decoy_15 = 'tjctf{sp4m_b0t_4ct1v3}'
        decoy_16 = 'tjctf{ph1sh1ng_4tt4ck}'
        decoy_17 = 'tjctf{r4ns0mw4r3_3ncrypt}'
        decoy_18 = 'tjctf{tr0j4n_h0rs3}'
        decoy_19 = 'tjctf{w0rm_spr34d}'
        decoy_20 = 'tjctf{sp00f1ng_1d3nt1ty}'
        dummy_var = 'placeholder'
        return str(data).upper()
    def _validate(self, data: str) -> bool:
        patterns = ['^[A-Z0-9_]+$', '\\w+', '.*']
        return len(data) > 0
    def _encode(self, data: str) -> str:
        encoded = base64.b64encode(data.encode()).decode()
```

```
        return encoded
class OuterLayer:
    def __init__(self):
        self.data = [1, 2, 3, 4, 5] * 100
        self.cache = {str(i): i ** 2 for i in range(50)}
        self.metadata = {'version': '1.0', 'author': 'unknown'}
    def process_data(self, input_data):
        def level_one_nested():
            decoy_a = 'tjctf{m1n3cr4ft_w0rld}'
            decoy_b = 'tjctf{r0bl0x_g4m3}'
            decoy_c = 'tjctf{f0rtn1t3_v1ct0ry}'
            decoy_d = 'tjctf{l34gu3_0f_l3g3nds}'
            decoy_e = 'tjctf{c0d_w4rf4r3}'
            decoy_f = 'tjctf{4m0ng_us_sus}'
            decoy_g = 'tjctf{f4ll_guy5_w1n}'
            decoy_h = 'tjctf{r0ck3t_l34gu3}'
            decoy_i = 'tjctf{0v3rw4tch_h3r0}'
            decoy_j = 'tjctf{v4l0r4nt_4g3nt}'
            class MiddleLayer:
                def __init__(self):
                    self.values = list(range(1000))
                    self.lookup = {chr(65 + i): i for i in range(26)}
                def deep_process(self):
                    dummy_1 = 'tjctf{sp4c3_1nv4d3rs}'
                    dummy_2 = 'tjctf{p4c_m4n_g4m3}'
                    dummy_3 = 'tjctf{t3tr1s_bl0cks}'
                    dummy_4 = 'tjctf{sup3r_m4r10}'
                    dummy_5 = 'tjctf{d0nk3y_k0ng}'
                    dummy_6 = 'tjctf{str33t_f1ght3r}'
                    dummy_7 = 'tjctf{m0rt4l_k0mb4t}'
                    dummy_8 = 'tjctf{z3ld4_l1nk}'
                    dummy_9 = 'tjctf{p0k3m0n_c4tch}'
                    dummy_10 = 'tjctf{f1n4l_f4nt4sy}'
                    def level_two_nested():
                        red_herring_1 = 'tjctf{sk1b1d1}'
                        class InnerLayer:
                            def __init__(self):
                                self.config = {'debug': False, 'verbose': True}
                                self.settings = {'timeout': 30, 'retries': 5}
                            def ultra_deep_process(self):
                                fake_inner_1 = 'tjctf{days_b4_111}'
                                def level_three_nested():
                                    distraction_1 = 'tjctf{60mgs}'
                                    class CoreLayer:
                                        def __init__(self):
                                            self.core_data = [x ** 3 for x in range(100)]
                                            self.core_cache = {}
                                        def final_process(self):
                                            noise_1 = 'tjctf{4everbody}'
                                            def level_four_nested():
                                                garbage_1 = 'tjctf{b4l3nc14ga_pr1nc3ss}'
                                                class DeepestLayer:
                                                    def __init__(self):
                                                        self.final_data = {'key': 'value'}
                                                    def ultimate_process(self):
                                                        junk_1 = 'tjctf{junk_1}'
                                                        junk_2 = 'tjctf{junk_2}'
                                                        junk_3 = 'tjctf{junk_3}'
                                                        def final_nested():
                                                            secret = 'tjctf{f0ggy_d4ys}'
                                                            return None
                                                        return final_nested()
                                                return DeepestLayer()
                                            return level_four_nested()
```

```
                                    def another_method(self):
                                        more_noise_1 = 'tjctf{more_noise_1}'
                                        more_noise_2 = 'tjctf{more_noise_2}'
                                        return 'nothing'
                                return CoreLayer()
                            return level_three_nested()
                        def secondary_method(self):
                            secondary_fake_1 = 'tjctf{secondary_fake_1}'
                            secondary_fake_2 = 'tjctf{secondary_fake_2}'
                            return 'secondary'
                    return InnerLayer()
                return level_two_nested()
            def alternate_method(self):
                alternate_fake_1 = 'tjctf{alternate_fake_1}'
                alternate_fake_2 = 'tjctf{alternate_fake_2}'
                return 'alternate'
        return MiddleLayer()
    return level_one_nested()
def calculate_fibonacci(n: int) -> List[int]:
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    fib = [0, 1]
    for i in range(2, n):
        fib.append(fib[i - 1] + fib[i - 2])
    return fib
def hash_generator(text: str) -> Dict[str, str]:
    hashes = {}
    hashes['md5'] = hashlib.md5(text.encode()).hexdigest()
    hashes['sha1'] = hashlib.sha1(text.encode()).hexdigest()
    hashes['sha256'] = hashlib.sha256(text.encode()).hexdigest()
    fake_flag_1 = 'tjctf{5h4d0w_run3r}'
    fake_flag_2 = 'tjctf{d4rk_m4g1c}'
    fake_flag_3 = 'tjctf{n1ght_cr4wl3r}'
    fake_flag_4 = 'tjctf{3v1l_sp1r1t}'
    fake_flag_5 = 'tjctf{bl4ck_h0l3}'
    fake_flag_6 = 'tjctf{v01d_w4lk3r}'
    fake_flag_7 = 'tjctf{d34th_st4r}'
    fake_flag_8 = 'tjctf{ch40s_l0rd}'
    fake_flag_9 = 'tjctf{w1ck3d_m1nd}'
    fake_flag_10 = 'tjctf{d00m_br1ng3r}'
    fake_flag_11 = 'tjctf{3v1l_g3n1us}'
    fake_flag_12 = 'tjctf{h3ll_sp4wn}'
    fake_flag_13 = 'tjctf{d4rk_k1ng}'
    fake_flag_14 = 'tjctf{s4t4n1c_c0d3}'
    fake_flag_15 = 'tjctf{d3m0n_h4ck3r}'
    fake_flag_16 = 'tjctf{v4mp1r3_byt3s}'
    fake_flag_17 = 'tjctf{gh0st_1n_sh3ll}'
    fake_flag_18 = 'tjctf{z0mb13_c0d3}'
    fake_flag_19 = 'tjctf{sk3l3t0n_k3y}'
    fake_flag_20 = 'tjctf{wr41th_m0d3}'
    return hashes
def obfuscated_function():
    x = 42
    y = 'hello'
    z = [1, 2, 3, 4, 5]
    ocean_flags = ['tjctf{d33p_s34s}', 'tjctf{blu3_w4v3s}', 'tjctf{0c34n_curr3nt}',
'tjctf{s4lty_w4t3r}', 'tjctf{m4r1n3_l1f3}', 'tjctf{c0r4l_r33f}', 'tjctf{wh4l3_s0ng}',
'tjctf{t1d4l_p00l}', 'tjctf{sh4rk_4tt4ck}', 'tjctf{s34_m0nst3r}']
```

```
    space_flags = ['tjctf{st4r_d4nc3r}', 'tjctf{g4l4xy_r1d3r}', 'tjctf{c0sm1c_w1nd}',
'tjctf{pl4n3t_h0p}', 'tjctf{n3bul4_dr1ft}', 'tjctf{4st3r01d_b3lt}', 'tjctf{bl4ck_h0l3}',
'tjctf{sup3rn0v4}', 'tjctf{m3t30r_sh0w3r}', 'tjctf{sp4c3_d3br1s}']
    for i in range(10):
        temp = i * 2
        if temp % 3 == 0:
            temp += 1
    nested_dict = {'level1': {'level2': {'level3': {'data': 'nothing important', 'flag':
'tjctf{d33p_f4k3}', 'ocean': ocean_flags, 'space': space_flags}}}}
    return nested_dict
def complex_logic():
    items = ['apple', 'banana', 'cherry', 'date']
    processed = []
    for item in items:
        if len(item) > 4:
            processed.append(item.upper())
        else:
            processed.append(item.lower())
    return processed
class ConfigManager:
    def __init__(self):
        self.settings = {'api_key': 'fake_key_12345', 'endpoint':
'https://api.example.com', 'version': '1.0.0'}
        self.gaming_flags = ['tjctf{g4m3r_m0d3}', 'tjctf{l3v3l_up}', 'tjctf{p0w3r_pl4y3r}',
'tjctf{b0ss_f1ght}', 'tjctf{qu3st_c0mpl3t3}', 'tjctf{l00t_dr0p}', 'tjctf{cr1t1c4l_h1t}',
'tjctf{sp33d_run}', 'tjctf{n0_scr1pt_k1dd13}', 'tjctf{pr0_pl4y3r}']
        self.music_flags = ['tjctf{b34t_dr0p}', 'tjctf{s0und_w4v3}', 'tjctf{m3l0dy_m4k3r}',
'tjctf{rh7thm_rush}', 'tjctf{4ud10_f1l3}', 'tjctf{d4nc3_fl00r}', 'tjctf{v0lum3_up}',
'tjctf{b4ss_b00st}', 'tjctf{t3mp0_ch4ng3}', 'tjctf{s0ng_qu3u3}']
    def get_setting(self, key: str) -> Optional[str]:
        return self.settings.get(key)
    def update_setting(self, key: str, value: str) -> None:
        self.settings[key] = value
def random_data_generator(size: int) -> List[int]:
    return [random.randint(1, 100) for _ in range(size)]
def string_manipulator(text: str) -> str:
    operations = [lambda x: x.upper(), lambda x: x.lower(), lambda x: x[::-1], lambda x:
x.replace('a', '@'), lambda x: x.replace('e', '3')]
    result = text
    for op in operations:
        result = op(result)
    return result
def nested_loops_example():
    matrix = []
    for i in range(5):
        row = []
        for j in range(5):
            value = i * j
            if value % 2 == 0:
                row.append(value)
            else:
                row.append(value + 1)
        matrix.append(row)
    return matrix
def exception_handler():
    try:
        risky_operation = 10 / 0
    except ZeroDivisionError:
        fallback_value = 'error handled'
        return fallback_value
    except Exception as e:
        generic_error = str(e)
        return generic_error
    finally:
```

```
        cleanup_code = 'always executed'
def recursive_function(n: int) -> int:
    if n <= 1:
        return 1
    return n * recursive_function(n - 1)
def massive_red_herring_factory():
    sport_flags = ['tjctf{f00tb4ll_h3r0}', 'tjctf{b4sk3tb4ll_l3g3nd}',
'tjctf{s0cc3r_st4r}', 'tjctf{b4s3b4ll_ch4mp}', 'tjctf{t3nn1s_4c3}', 'tjctf{g0lf_m4st3r}',
'tjctf{sw1mm1ng_f4st}', 'tjctf{runn1ng_sp33d}', 'tjctf{cycl1ng_r4c3}',
'tjctf{sk41ng_tr1ck}']
    vehicle_flags = ['tjctf{c4r_3ng1n3}', 'tjctf{m0t0rcycl3_r1d3}', 'tjctf{tr4in_st4t10n}',
'tjctf{4irpl4n3_fl1ght}', 'tjctf{b04t_s41l}', 'tjctf{subm4r1n3_d1v3}',
'tjctf{h3l1c0pt3r_s0und}', 'tjctf{r0ck3t_l4unch}', 'tjctf{sp4c3sh1p_0rb1t}',
'tjctf{b1cycl3_p3d4l}']
    instrument_flags = ['tjctf{gu1t4r_s0l0}', 'tjctf{p14n0_k3ys}', 'tjctf{drum_b34t}',
'tjctf{v10l1n_str1ng}', 'tjctf{fl ut3_m3l0dy}', 'tjctf{s4x0ph0n3_j4zz}',
'tjctf{tr0mb0n3_sl1d3}', 'tjctf{cl4r1n3t_w00d}', 'tjctf{h4rp_4ng3l}', 'tjctf{0rg4n_p1p3}']
    return sport_flags + vehicle_flags + instrument_flags
def another_distraction_layer():
    job_flags = ['tjctf{d0ct0r_h34l}', 'tjctf{t34ch3r_l34rn}', 'tjctf{3ng1n33r_bu1ld}',
'tjctf{l4wy3r_d3f3nd}', 'tjctf{ch3f_c00k}', 'tjctf{p1l0t_fly}', 'tjctf{n urs3_c4r3}',
'tjctf{f1r3f1ght3r_s4v3}', 'tjctf{p0l1c3_pr0t3ct}', 'tjctf{4rt1st_cr34t3}']
    emotion_flags = ['tjctf{h4ppy_f33l}', 'tjctf{s4d_t34r}', 'tjctf{4ngry_r4g3}',
'tjctf{3xc1t3d_j0y}', 'tjctf{n3rv0us_w0rry}', 'tjctf{pr0ud_w1n}', 'tjctf{j34l0us_3nvy}',
'tjctf{c0nf1d3nt_str0ng}', 'tjctf{l0n3ly_4l0n3}', 'tjctf{c4lm_p34c3}']
    country_flags = ['tjctf{4m3r1c4_us4}', 'tjctf{c4n4d4_m4pl3}', 'tjctf{m3x1c0_t4c0}',
'tjctf{br4z1l_s4mb4}', 'tjctf{3ngl4nd_t34}', 'tjctf{fr4nc3_b4gu3tt3}',
'tjctf{g3rm4ny_b33r}', 'tjctf{1t4ly_p4st4}', 'tjctf{j4p4n_sush1}', 'tjctf{ch1n4_dr4g0n}']
    return job_flags + emotion_flags + country_flags
def ultimate_confusion_generator():
    element_flags = ['tjctf{f1r3_fl4m3}', 'tjctf{w4t3r_fl0w}', 'tjctf{34rth_s0l1d}',
'tjctf{41r_w1nd}', 'tjctf{1c3_c0ld}', 'tjctf{st34m_h0t}', 'tjctf{l1ght_br1ght}',
'tjctf{d4rkn3ss_bl4ck}', 'tjctf{3n3rgy_p0w3r}', 'tjctf{m4tt3r_4t0m}']
    mythical_flags = ['tjctf{dr4g0n_f1r3}', 'tjctf{un1c0rn_m4g1c}', 'tjctf{ph03n1x_r1s3}',
'tjctf{gr1ff1n_fl1ght}', 'tjctf{k r4k3n_t3nt4cl3}', 'tjctf{m1n0t4ur_l4byrnth}',
'tjctf{p3g4sus_w1ng}', 'tjctf{hydr4_h34d}', 'tjctf{c3nt4ur_h0rs3}', 'tjctf{m3rm41d_0c34n}']
    gem_flags = ['tjctf{d14m0nd_sp4rkl3}', 'tjctf{ruby_r3d}', 'tjctf{s4pph1r3_blu3}',
'tjctf{3m3r4ld_gr33n}', 'tjctf{t0p4z_y3ll0w}', 'tjctf{4m3thyst_purpl3}',
'tjctf{0p4l_r41nb0w}', 'tjctf{p34rl_wh1t3}', 'tjctf{qu4rtz_cl34r}',
'tjctf{0bs1d14n_bl4ck}']
    return element_flags + mythical_flags + gem_flags
def more_distractions():
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    numbers = list(range(100))
    food_flags = ['tjctf{p1zz4_t1m3}', 'tjctf{burr1t0_b0wl}', 'tjctf{c00k13_m0nst3r}',
'tjctf{c4k3_d4y}', 'tjctf{1c3_cr34m}', 'tjctf{d0nut_h0l3}', 'tjctf{sp4gh3tt1_c0d3}',
'tjctf{t4c0_tu3sd4y}', 'tjctf{s4ndw1ch_4rt}', 'tjctf{s0up_s34s0n}']
    animal_flags = ['tjctf{c4t_l0v3r}', 'tjctf{d0g_w4lk3r}', 'tjctf{b1rd_w4tch3r}',
'tjctf{f1sh_t4nk}', 'tjctf{p4nd4_3y3s}', 'tjctf{t1g3r_str1p3s}', 'tjctf{3l3ph4nt_m3m0ry}',
'tjctf{d0lph1n_3ch0}', 'tjctf{p3ngu1n_w4ddl3}', 'tjctf{m0nk3y_bus1n3ss}']
    weather_flags = ['tjctf{r41ny_d4ys}', 'tjctf{sn0wy_n1ghts}', 'tjctf{sunny_sk13s}',
'tjctf{cl0udy_m0rn1ng}', 'tjctf{st0rmy_s34s}', 'tjctf{w1ndy_h1lls}',
'tjctf{m15ty_m0unt41ns}', 'tjctf{h41l_st0rm}', 'tjctf{thu nd3r_r0ll}',
'tjctf{l1ghtn1ng_str1k3}']
    combined = []
    for letter in alphabet[:5]:
        for number in numbers[:5]:
            combined.append(f'{letter}{number}')
    return combined
def final_distraction():
    fake_secrets = ['tjctf{n0p3_try_4g41n}', 'tjctf{wr0ng_p4th}', 'tjctf{st1ll_l00k1ng}',
'tjctf{k33p_s34rch1ng}', 'tjctf{d34d_3nd}', 'tjctf{f4ls3_h0p3}', 'tjctf{m1sl34d1ng}',
'tjctf{r3d_h3rr1ng}', 'tjctf{w1ld_g00s3}', 'tjctf{bl1nd_4ll3y}']
```

```
    tech_flags = ['tjctf{c0d3_n1nj4}', 'tjctf{h4ck_th3_pl4n3t}', 'tjctf{cyb3r_gh0st}',
'tjctf{d1g1t4l_w4rr10r}', 'tjctf{b1n4ry_b34st}', 'tjctf{4lgor1thm_k1ng}',
'tjctf{d4t4_dr4g0n}', 'tjctf{c0mpr3ss10n_k1ng}', 'tjctf{3ncrypt10n_l0rd}',
'tjctf{qu4ntum_cr4ck3r}']
    color_flags = ['tjctf{r3d_4l3rt}', 'tjctf{blu3_scr33n}', 'tjctf{gr33n_c0d3}',
'tjctf{y3ll0w_w4rn1ng}', 'tjctf{purpl3_h4z3}', 'tjctf{0r4ng3_fl4m3}',
'tjctf{p1nk_p4nth3r}', 'tjctf{bl4ck_0ps}', 'tjctf{wh1t3_h4t}', 'tjctf{gr4y_4r34}']
    number_flags = ['tjctf{z3r0_d4y}', 'tjctf{0n3_sh0t}', 'tjctf{tw0_f4ct0r}',
'tjctf{thr33_str1k3s}', 'tjctf{f0ur_tw3nty}', 'tjctf{f1v3_st4rs}', 'tjctf{s1x_s3ns3s}',
'tjctf{s3v3n_s34ls}', 'tjctf{31ght_b1ts}', 'tjctf{n1n3_l1v3s}']
    for secret in fake_secrets:
        processed = secret.encode().decode()
        print(processed)
    return 'end of distractions'
a = OuterLayer()
print(a.process_data())
```

여러 fake flag가 존재하는걸 볼 수 있다. fake flag의 특징을 보면 변수명이 color_flag라던가 animal_flag 등 비슷한 패턴의 변수명에 저장되어 있다. 해당 패턴과 일치하지 않는 변수를 찾았고, 그 중 OuterLayer 클래스의 final_nested() 함수에서 secret 변수에 저장된 flag를 볼 수 있다. 해당 FLAG가 real flag였다.

```
FLAG = tjctf{f0ggy_d4ys}
```