Minggu 7: Fungsi & Prosedur

Tujuan minggu ini adalah menulis kode yang rapi, terorganisir, dan dapat digunakan kembali. Ini adalah langkah besar menuju penulisan program yang lebih kompleks dan mudah dikelola.

Konsep Utama & Kegiatan Lab

1. **Fungsi vs. Prosedur**
    - **Mengapa:** Jelaskan bahwa membedakan antara "melakukan sesuatu" dan "mengembalikan nilai" adalah konsep inti pemrograman. Sebuah **prosedur** melakukan tugas tanpa mengembalikan nilai (seperti `print()`), sementara sebuah **fungsi** menghitung atau memproses sesuatu dan **mengembalikan hasilnya** (seperti `len()` atau `max()`).
    - **Bagaimana:** Tunjukkan kedua konsep secara berdampingan.
        - **Prosedur:** Buat kode yang hanya mencetak pesan, tanpa `return`.
        - **Fungsi:** Buat kode yang menjumlahkan dua angka dan **mengembalikan** hasilnya menggunakan `return`.
2. **Parameter dan Argumen**
    - **Mengapa:** Fungsi dan prosedur membutuhkan informasi untuk bekerja. Informasi ini disebut **parameter**.
    - **Bagaimana:** Jelaskan bahwa parameter adalah *input* yang diberikan saat fungsi dipanggil. Tunjukkan bagaimana membuat fungsi yang menerima parameter.
3. **Menerapkan Fungsi**

Cheatsheet

Sintaks Fungsi/Prosedur:

- `def nama_fungsi(parameter):`
- Gunakan `return` untuk mengembalikan nilai.

Contoh Praktis:

- **Prosedur:** `sapa_pengguna("Budi")` → hanya mencetak di layar.
- **Fungsi:** `luas_kamar = hitung_luas_persegi(5)` → menyimpan nilai yang dikembalikan.

Kode Monolit: `Dungeon_Plus_Battle.py`

Perhatikan bagaimana `main` loop sekarang menjadi sangat 'kotor' karena ada `while` loop (battle) di dalam `if` (update dungeon) di dalam `while` loop (main game).

```
import os
import time
import random

# --- (STATE DUNIA) DEKLARASI PETA ---
# '#' = Tembok (Obstacle)
# ' ' = Jalan (Free Space)
# 'P' = Player (Posisi awal)
# 'G' = Goal (Tujuan)
# 'E' = Enemy (Musuh)
MAP_DATA = [
    "####################",
    "#P #        #     #",
    "#  # ##### # #### #",
    "#  # #   # # #    #",
    "#  # # E # # # ####", # Kita letakkan musuh 'E' di sini
    "#      # #   #    #",
    "###### # ######## #",
    "#              # G#",
    "####################"
]

def clear_screen():
    """Membersihkan layar terminal."""
    os.system('clear' if os.name == 'posix' else 'cls')


def main():
    """Fungsi utama yang menjalankan game loop."""

    # --- INISIALISASI ---
    map_grid = [list(row) for row in MAP_DATA]

    # --- INISIALISASI PLAYER & MAP ---
    initial_pos = None
    for y, row in enumerate(map_grid):
        for x, char in enumerate(row):
            if char == 'P':
                initial_pos = y, x
```

```
if initial_pos is None:
    print("Error: Player 'P' tidak ditemukan di peta!")
    return

player_y, player_x = initial_pos

# Menambahkan State Baru: HP Player
player_hp = 100
player_hp_max = 100

# Konfigurasi Field of View
VIEW_RADIUS = 3 # Jarak pandang
map_height = len(map_grid)
map_width = len(map_grid[0])

# --- GAME LOOP / CONTROL LOOP ---
is_running = True
while is_running:

    # ============================================================
    # 1. RENDER (Menggambar state saat ini) DENGAN FIELD OF VIEW
    # ============================================================

    clear_screen()
    print("--- SELAMAT DATANG DI DUNGEON! ---")
    print(f"HP Player: {player_hp}/{player_hp_max}\n")

    # Iterasi setiap sel di grid
    for y in range(map_height):
        row_output = []
        for x in range(map_width):
            distance_y = abs(y - player_y)
            distance_x = abs(x - player_x)
            distance = max(distance_y, distance_x)

            if distance <= VIEW_RADIUS:
                row_output.append(map_grid[y][x])
            else:
                row_output.append('?') # Kabut

        print("".join(row_output))

    print("\nKontrol: [w/a/s/d] untuk gerak | [x] untuk keluar")

    # ============================================================
    # 2. GET INPUT (Menerima sinyal kontrol)
```

```
# ============================================================
key = input("Gerak (w/a/s/d/x): ")

# ============================================================
# 3. UPDATE STATE (Menghitung state berikutnya)
# ============================================================

old_y, old_x = player_y, player_x
new_y, new_x = player_y, player_x

if key == 'w':
    new_y -= 1
elif key == 's':
    new_y += 1
elif key == 'a':
    new_x -= 1
elif key == 'd':
    new_x += 1
elif key == 'x':
    is_running = False
    continue

# --- VALIDASI GERAKAN ---
if (new_y < 0 or new_y >= map_height or
    new_x < 0 or new_x >= map_width):
    continue # Abaikan jika di luar peta

target_cell = map_grid[new_y][new_x]

# --- Logika Menabrak Tembok ---
if target_cell == '#':
    continue # Gerakan tidak valid, kembali ke awal loop

# --- Logika Menemukan Goal ---
if target_cell == 'G':
    # ... (Logika kemenangan seperti kode sebelumnya) ...
    clear_screen()
    print("--- KAMU BERHASIL! ---")
    is_running = False
    continue

# --- LOGIKA BERTEMU MUSUH ('E') ---
if target_cell == 'E':
    print("\n!!! Kamu bertemu Goblin Penjaga !!!")
    time.sleep(1.5)
```

```
# --- INISIALISASI BATTLE ---
enemy_name = "Goblin Penjaga"
enemy_hp = 30  # HP-nya dikit, sesuai permintaan
enemy_hp_max = 30

battle_is_running = True

# ===== AWAL DARI BATTLE LOOP (Loop di dalam Loop) =====
while battle_is_running:
    clear_screen()
    print(f"--- PERTARUNGAN MELAWAN {enemy_name} ---")

    # Render Battle HUD
    print(f"PLAYER:   HP {player_hp}/{player_hp_max}")
    print(f"{enemy_name.upper()}: HP {enemy_hp}/{enemy_hp_max}")
    print("====================================")
    print("1. Serang")
    print("2. Kabur")

    # Get Battle Input
    battle_choice = input("Aksi (1-2): ")

    # Update Battle State
    if battle_choice == '1':
        # Player Menyerang
        player_damage = random.randint(10, 18)
        enemy_hp -= player_damage
        print(f"Kamu menyerang {enemy_name} sebesar {player_damage} HP!")
        time.sleep(1)

        # Cek Musuh Mati
        if enemy_hp <= 0:
            print(f"Kamu telah mengalahkan {enemy_name}!")

            # UPDATE DUNGEON: Musuh hilang, Player maju
            map_grid[old_y][old_x] = ' ' # Kosongkan posisi lama player
            map_grid[new_y][new_x] = 'P' # Player pindah ke tempat musuh
            player_y, player_x = new_y, new_x # Update posisi player

            battle_is_running = False # Hentikan battle loop
            time.sleep(2)
            continue # Lanjut ke iterasi dungeon loop berikutnya

        # Musuh Menyerang
        print(f"{enemy_name} menyerang balik...")
        time.sleep(1)
```

```
                    enemy_damage = random.randint(5, 12)
                    player_hp -= enemy_damage
                    print(f"Kamu menerima {enemy_damage} HP kerusakan!")

                    # Cek Player Mati
                    if player_hp <= 0:
                        print("Kamu telah dikalahkan... GAME OVER.")
                        battle_is_running = False # Hentikan battle loop
                        is_running = False # Hentikan game loop utama

                    time.sleep(1.5)

                elif battle_choice == '2':
                    print("Kamu berhasil kabur!")
                    battle_is_running = False # Hentikan battle loop
                    # Player tidak jadi bergerak, posisi tetap di old_y, old_x
                    time.sleep(1.5)

                else:
                    print("Pilihan tidak valid!")
                    time.sleep(1)

            # Setelah battle selesai (menang/kabur/kalah),
            # kita `continue` agar tidak menjalankan gerakan normal di bawah
            continue
            # ===== AKHIR DARI BATTLE LOOP =====


        # --- GERAKAN VALID (ke ' ') ---
        map_grid[old_y][old_x] = ' '
        map_grid[new_y][new_x] = 'P'
        player_y, player_x = new_y, new_x

    print("Game Selesai.")

# --- Entry Point (Titik Mulai) Program ---
if __name__ == "__main__":
    main()
```

Misi Refactoring menjadi procedure/ function

GOAL: bikin minimal 3 dari antara ini semua (atau klo mau hardcore, sekalian semuanya)

nilai jika berhasil menyelesaikan function / procedure sebanyak: - 3 = 60 - 5 =

80 - 7 = 100

Level: Mudah

Fungsi-fungsi ini biasanya pendek, memiliki satu tanggung jawab yang jelas, dan tidak mengelola *state* yang kompleks.

Nama Fungsi / Prosedur	Argumen yang Dibutuhkan	Nilai Kembali (Return Value)	Deskripsi
`clear_screen`	`(Tidak ada)`	`Tidak ada (Prosedur)`	Membersihkan layar terminal (`clear` atau `cls`).
`get_player_input`	`(Tidak ada)`	`string`	Menunggu user menekan tombol dan mengembalikan input tersebut (misal: 'w', 'a', 's', 'd', 'x').
`initialize_map`	`map_data` (list of string)	`list` (Grid peta)	Mengonversi data peta statis (`MAP_DATA`) menjadi grid 2D (list of list) yang bisa dimodifikasi.
`calculate_new_pos`	`player_y` (int) `player_x` (int) `key` (string)	`tuple` (new_y, new_x)	Menghitung *potensi* posisi baru berdasarkan posisi lama dan input tombol. Tidak memvalidasi gerakan.
`move_player`	`map_grid` (list of list) `old_y` (int) `old_x` (int) `new_y` (int) `new_x` (int)	`Tidak ada (Prosedur)`	Memperbarui `map_grid` dengan menghapus 'P' dari posisi lama dan meletakkannya di posisi baru.

Level: Sedang

Fungsi-fungsi ini melibatkan perulangan (loop) pada struktur data (seperti grid peta) untuk mencari atau menampilkan sesuatu.

Nama Fungsi / Prosedur	Argumen yang Dibutuhkan	Nilai Kembali (Return Value)	Deskripsi
`find_start_position`	`map_grid` (list of list)`char_to_find` (string)	`tuple` (y, x) atau `None`	Mencari koordinat (y, x) dari karakter pertama yang ditemukan (misal 'P') di dalam grid.
`render_game`	`map_grid` (list of list)`player_y` (int)`player_x` (int)`player_hp` (int)`player_hp_max` (int)`view_radius` (int)	`Tidak ada (Prosedur)`	Menggambar seluruh tampilan game, termasuk HUD (HP) dan peta dengan *Field of View* (kabut '?').

Level: Sulit / Kompleks

Fungsi-fungsi ini mengelola *state* yang kompleks dan/atau memiliki *loop* (perulangan) sendiri.

Nama Fungsi / Prosedur	Argumen yang Dibutuhkan	Nilai Kembali (Return Value)	Deskripsi
`start_battle`	`player_hp` (int)`player_hp_max` (int)	`tuple` (string, int)	**Fungsi Kunci.** Menjalankan *loop pertarungan* terpisah. Mengembalikan **hasil** (`"WIN"`, `"LOSE"`, `"FLEE"`) dan **HP player terbaru** setelah pertarungan.

Nama Fungsi / Prosedur	Argumen yang Dibutuhkan	Nilai Kembali (Return Value)	Deskripsi
`main`	`(Tidak ada)`	`Tidak ada (Prosedur)`	**Fungsi Orkestrator.** Menginisialisasi game dan menjalankan *game loop* utama, mengoordinasikan semua fungsi/prosedur lain dan mengelola *state* utama (posisi player, HP, dll).