Dead Ship - Python Programming Student Guide

Table of Contents

Introduction

Welcome to the **Dead Ship** programming project! This guide will teach you Python programming through creating a text-based adventure game. You'll learn fundamental programming concepts while building something fun and interactive.

What You'll Learn

- **Functions** - Breaking code into reusable pieces
- **Lists** - Storing and managing data
- **Variables** - Tracking game state
- **Loops** - Creating the main game cycle
- **Conditionals** - Making decisions in code
- **User Input** - Interacting with players
- **File Organization** - Splitting code across multiple files

Prerequisites

- Basic understanding of Python syntax
- Ability to run Python programs
- Text editor or IDE (like VS Code)

Game Overview

The Story

Your spaceship has crashed on an alien planet! The energy generator has broken into 6 pieces scattered across 10 different locations. You must find all the pieces before your energy runs out, or you'll be stranded forever!

Game Mechanics

- **Energy System**: Start with 50 energy, lose 3 per move, gain 10 per item found
- **Time Pressure**: Energy decays 1 point every 3 turns
- **Exploration**: 10 locations arranged in a grid pattern
- **Victory**: Collect all 6 generator pieces
- **Defeat**: Run out of energy

Visual Map

```
[0]────[1]────[2]      Crash Site ──── Rocky Outcrop ──── Crystal Cave
 │      │      │                 │                 │                │
[3]────[4]────[5]    Dense Forest ──── Ancient Ruins ──── Frozen Lake
 │      │      │                 │                 │                │
[6]────[7]────[8]   Volcanic Vents ─── Metal Wreckage ─── Strange Monolith
       │                         │
      [9]              Energy Crater
```

Project Structure

Our game consists of three main files:

```
SimpleVersion/
├── main.py          # Main game loop and user interface
├── functions.py     # Game functions and logic
├── data.py         # Game data (locations, items, descriptions)
└── show_map.py     # Visual map display
```

Why Split Code Into Files?

- **Organization**: Easier to find and modify specific parts
- **Reusability**: Functions can be used in multiple places
- **Collaboration**: Different people can work on different files
- **Maintenance**: Easier to fix bugs and add features

Programming Concepts

1. Functions

Functions are reusable blocks of code that perform specific tasks.

Why use functions?

- Avoid repeating code
- Make code easier to read
- Break complex problems into smaller pieces
- Allow for easier testing and debugging

Example:

```
def show_location(location_number, inventory):
    """Display the current location description"""
    print(f"--- {LOCATION_NAMES[location_number]} ---")
    print(LOCATION_DESCRIPTIONS[location_number])
```

2. Lists

Lists store multiple items in a single variable.

Why use lists?

- Store related data together
- Access items by position (index)
- Easily add or remove items
- Loop through all items

Example:

```
LOCATION_NAMES = [
    "Crash Site",       # Index 0
    "Rocky Outcrop",    # Index 1
    "Crystal Cave"      # Index 2
]
```

3. Variables

Variables store data that can change during the program.

Game State Variables:

```
player_location = 0     # Current location index
player_energy = 50      # Current energy level
inventory = []          # List of found items
turn_count = 0          # Number of turns taken
```

4. Loops

Loops repeat code until a condition is met.

Main Game Loop:

```
while True:  # Run forever until break
    # Show current state
    # Get user input
    # Process command
    # Check win/lose conditions
```

5. Conditionals

Conditionals make decisions based on conditions.

Command Processing:

```
if command == "quit":
    break
elif command == "look":
    continue
elif command.startswith("go "):
    # Handle movement
```

Code Walkthrough

File 1: data.py - Game Data

This file contains all the game's static data using simple lists.

Location Names

```
LOCATION_NAMES = [
    "Crash Site",           # 0
    "Rocky Outcrop",        # 1
    "Crystal Cave",         # 2
    "Dense Forest",         # 3
    "Ancient Ruins",        # 4
    "Frozen Lake",          # 5
    "Volcanic Vents",       # 6
    "Metal Wreckage",       # 7
    "Strange Monolith",     # 8
    "Energy Crater"         # 9
]
```

Key Concept: Using list indexes (0-9) to reference locations makes movement calculations easier.

Location Descriptions

```
LOCATION_DESCRIPTIONS = [
    "The twisted remains of your ship lie scattered here. Smoke still rises from
the wreckage.",
    "Jagged rocks jut from the ground like broken teeth. The wind howls between
them.",
    # ... more descriptions
]
```

Key Concept: Parallel lists - the description at index 1 describes the location at index 1.

Items at Locations

```
LOCATION_ITEMS = [
    None,                      # 0 - Crash Site (no item)
    "Power Core Fragment",     # 1 - Rocky Outcrop
    "Energy Crystal",          # 2 - Crystal Cave
    # ... more items
]
```

Key Concept: Using None to represent "no item" - this is a common Python pattern.

File 2: functions.py - Game Logic

This file contains all the game's functions.

Displaying Locations

```
def show_location(location_number, inventory):
    """Display the current location description and available items"""
    print(f"\n--- {LOCATION_NAMES[location_number]} ---")
    print(LOCATION_DESCRIPTIONS[location_number])

    # Show exits based on grid position
    exits = []
    if location_number not in [0, 1, 2]:  # Not in top row
        exits.append("south")
    if location_number not in [7, 8, 9]:  # Not in bottom row
        exits.append("north")
    # ... more exit logic

    if exits:
        print(f"You can go: {', '.join(exits)}")
```

Key Concepts:

- **Function parameters**: `location_number` and `inventory` are inputs
- **List operations**: Using `not in` to check if location is in a list
- **String formatting**: Using f-strings to insert variables into text
- **List methods**: Using `join()` to combine list items into a string

Movement Logic

```
def move_player(current_location, direction):
    """Move the player in the given direction, return new location"""
    if direction == "north":
        if current_location >= 3:  # Can move north if not in top row
            return current_location - 3
        else:
            return current_location  # Can't move, stay in place

    elif direction == "south":
        if current_location <= 6:  # Can move south if not in bottom row
            return current_location + 3
        else:
            return current_location
    # ... more directions
```

Key Concepts:

- **Grid mathematics**: Moving north subtracts 3, south adds 3, east adds 1, west subtracts 1
- **Boundary checking**: Preventing invalid moves
- **Return values**: Functions can send data back to the caller

Game State Management

```
def check_game_over(energy, inventory):
    """Check if the game is over (win or lose)"""
    if len(inventory) >= 6:
        print("🎉 CONGRATULATIONS! 🎉")
        print("You found all 6 generator pieces!")
        return True

    if energy <= 0:
        print("💀 GAME OVER 💀")
        print("You ran out of energy!")
        return True
        return False
```

Key Concepts:

- **Boolean return values**: Functions can return `True` or `False`
- **Multiple conditions**: Checking both win and lose conditions
- **List length**: Using `len()` to count items

File 3: main.py - Game Loop

This file contains the main game logic and user interface.

Game Initialization

```
def main():
    """Main game loop"""
    # Game introduction
    print("=" * 50)
    print("           DEAD SHIP")
    print("=" * 50)

    # Initialize game state
    player_location = 0  # Start at crash site
    player_energy = 50   # Starting energy
    inventory = []       # Empty inventory
    turn_count = 0       # Track turns
```

Key Concepts:

- **Initialization**: Setting up starting values
- **Comments**: Explaining what variables represent
- **String multiplication**: Using `"=" * 50` to create decorative lines

Main Game Loop

```
while True:
    # Show current location
    show_location(player_location, inventory)

    # Show player status
    print(f"\nEnergy: {player_energy}")
    print(f"Generator pieces found: {len(inventory)}/6")
    # Check if game is over
    if check_game_over(player_energy, inventory):
        break

    # Get player input
    command = input("\nWhat do you do? ").strip().lower()

    # Process commands...
```

Key Concepts:

- **Infinite loop**: `while True` runs until `break`
- **User input**: `input()` gets text from the player
- **String methods**: `strip()` removes spaces, `lower()` converts to lowercase
- **Function calls**: Using functions we defined in other files

Command Processing

```
if command == "quit":
    print("Thanks for playing!")
    break

elif command == "look":
    continue  # Just redisplay location

elif command == "inventory":
    show_inventory(inventory)
    continue

elif command.startswith("go "):
    direction = command[3:]  # Get direction after "go "
    new_location = move_player(player_location, direction)

    if new_location != player_location:
        player_location = new_location
        player_energy -= 3  # Moving costs energy

        # Check for items
        found_item = check_for_item(player_location)
        if found_item and found_item not in inventory:
            print(f"\nYou found a {found_item}!")
            add_to_inventory(inventory, found_item)
            player_energy += 10  # Finding item restores energy
```

Key Concepts:

- **String slicing**: `command[3:]` gets everything after position 3

- **String methods**: `startswith()` checks if string begins with text
- **State changes**: Updating variables based on player actions
- **Conditional logic**: Different outcomes based on player input

---

Exercises and Extensions

Beginner Exercises

1. **Add More Locations**

   - Add 2 new locations to the map
   - Create descriptions for them
   - Update the movement logic

2. **New Commands**

   - Add a "health" command to show current energy
   - Add a "score" command to show progress

3. **Better Messages**

   - Add more descriptive messages when moving
   - Create different messages for finding each item

Intermediate Exercises

1. **Random Events**

   - Add random encounters that can help or harm the player
   - Create a 10% chance of finding extra energy

2. **Inventory Management**

   - Limit inventory to 3 items at a time
   - Add a "drop" command to remove items

3. **Difficulty Levels**

   - Easy: Start with 75 energy

- Hard: Start with 25 energy
- Expert: Energy decays faster

Advanced Extensions

1. **Save/Load System**
    - Save game state to a file
    - Load saved games
2. **Multiple Endings**
    - Different endings based on how quickly you finish
    - Special ending if you visit all locations
3. **Combat System**
    - Add enemies at some locations
    - Simple combat mechanics

Complete Code Listing

data.py

```
"""
Dead Ship - Simple Text Adventure Game
Game data using lists
"""

# Location names (10 locations in a 3x3 grid plus crash site)
LOCATION_NAMES = [
    "Crash Site",           # 0
    "Rocky Outcrop",        # 1
    "Crystal Cave",         # 2
    "Dense Forest",         # 3
    "Ancient Ruins",        # 4
    "Frozen Lake",          # 5
    "Volcanic Vents",       # 6
    "Metal Wreckage",       # 7
    "Strange Monolith",     # 8
    "Energy Crater"         # 9
]
```

```python
# Location descriptions
LOCATION_DESCRIPTIONS = [
    "The twisted remains of your ship lie scattered here. Smoke still rises from the wreckage.",  # 0
    "Jagged rocks jut from the ground like broken teeth. The wind howls between them.",           # 1
    "Sparkling crystals line the walls of this cave, casting rainbow reflections everywhere.",    # 2
    "Towering alien trees block most of the light. Strange sounds echo in the darkness.",         # 3
    "Crumbling stone structures hint at an ancient civilization. Vines cover everything.",        # 4
    "A lake of perfectly still, frozen water reflects the alien sky like a mirror.",              # 5
    "Steam and heat rise from cracks in the ground. The air shimmers with thermal energy.",       # 6
    "Pieces of unknown technology are scattered among the rocks. Some still spark with power.",   # 7
    "A tall, black stone pillar hums with mysterious energy. Strange symbols glow on its surface.", # 8
    "A massive crater pulses with blue energy. The ground here feels warm to the touch."          # 9
]

# Items at each location (None means no item)
LOCATION_ITEMS = [
    None,                       # 0 - Crash Site (no item)
    "Power Core Fragment",      # 1 - Rocky Outcrop
    "Energy Crystal",           # 2 - Crystal Cave
    "Bio-Fuel Cell",            # 3 - Dense Forest
    "Ancient Battery",          # 4 - Ancient Ruins
    "Cooling Unit",             # 5 - Frozen Lake
    "Heat Exchanger",           # 6 - Volcanic Vents
    None,                       # 7 - Metal Wreckage (no item)
    None,                       # 8 - Strange Monolith (no item)
    None                        # 9 - Energy Crater (no item)
]

# Visual Map Layout:
#  [0]--[1]--[2]     Crash Site -- Rocky Outcrop -- Crystal Cave
#   |    |    |              |             |              |
#  [3]--[4]--[5]   Dense Forest -- Ancient Ruins -- Frozen Lake
#   |    |    |              |             |              |
#  [6]--[7]--[8]  Volcanic Vents -- Metal Wreckage -- Strange Monolith
#       |                        |
#      [9]                 Energy Crater

def print_game_map():
    """Print a static visual representation of the game map"""
    print("\n" + "=" * 70)
    print("                        DEAD SHIP - PLANET MAP")
    print("=" * 70)
    print()
    print("  [0]───[1]───[2]      Crash Site ─── Rocky Outcrop ─── Crystal Cave")
    print("   │      │      │              │               │                │")
    print("   |      |      |              |               |                |")
    print("  [3]────[4]────[5]    Dense Forest ──── Ancient Ruins ──── Frozen
Lake")
    print("   |      |      |              |               |                |")
    print("   |      |      |              |               |                |")
    print("  [6]────[7]────[8]   Volcanic Vents ─── Metal Wreckage ─── Strange
Monolith")
    print("         |                      |")
    print("         |                      |")
    print("        [9]              Energy Crater")
    print()
    print("GENERATOR PIECES LOCATIONS:")
    print("  [1] Rocky Outcrop     - Power Core Fragment")
    print("  [2] Crystal Cave      - Energy Crystal")
    print("  [3] Dense Forest      - Bio-Fuel Cell")
    print("  [4] Ancient Ruins     - Ancient Battery")
    print("  [5] Frozen Lake       - Cooling Unit")
    print("  [6] Volcanic Vents    - Heat Exchanger")
    print()
    print("EMPTY LOCATIONS (no generator pieces):")
    print("  [0] Crash Site        - Starting location")
    print("  [7] Metal Wreckage    - Exploration area")
    print("  [8] Strange Monolith - Exploration area")
    print("  [9] Energy Crater     - Exploration area")
    print("=" * 70)
```

functions.py

```
"""
Dead Ship - Simple Text Adventure Game
Game functions
"""

from data import LOCATION_NAMES, LOCATION_DESCRIPTIONS, LOCATION_ITEMS

def show_location(location_number, inventory):
    """Display the current location description and available items"""
    print(f"\n--- {LOCATION_NAMES[location_number]} ---")
    print(LOCATION_DESCRIPTIONS[location_number])

    # Show exits based on grid position
    exits = []
    if location_number not in [0, 1, 2]:  # Not in top row
        exits.append("south")
    if location_number not in [7, 8, 9]:  # Not in bottom row
        exits.append("north")
    if location_number not in [0, 3, 6]:  # Not in left column
        exits.append("east")
    if location_number not in [2, 5, 8]:  # Not in right column
        exits.append("west")

    if exits:
        print(f"You can go: {', '.join(exits)}")

    # Show if there's an item here (and player hasn't found it yet)
    if LOCATION_ITEMS[location_number] and LOCATION_ITEMS[location_number] not in
inventory:
        print(f"You see something glinting: {LOCATION_ITEMS[location_number]}")

def move_player(current_location, direction):
    """Move the player in the given direction, return new location"""
    if direction == "north":
        if current_location >= 3:  # Can move north if not in top row
            return current_location - 3
        else:
            return current_location  # Can't move north from top row

    elif direction == "south":
        if current_location <= 6:  # Can move south if not in bottom row
            return current_location + 3
        else:
            return current_location  # Can't move south from bottom row

    elif direction == "east":
        if current_location not in [2, 5, 8]:  # Can move east if not in right
column
            return current_location + 1
        else:
            return current_location  # Can't move east from right column

    elif direction == "west":
        if current_location not in [0, 3, 6]:  # Can move west if not in left
column
            return current_location - 1
        else:
            return current_location  # Can't move west from left column

    else:
        return current_location  # Invalid direction

def check_for_item(location_number):
    """Check if there's an item at the current location"""
    return LOCATION_ITEMS[location_number]

def add_to_inventory(inventory, item):
    """Add an item to the player's inventory"""
    if item and item not in inventory:
        inventory.append(item)

def show_inventory(inventory):
    """Display the player's inventory"""
    print("\n--- INVENTORY ---")
    if inventory:
        print("Generator pieces found:")
        for i, item in enumerate(inventory, 1):
            print(f"  {i}. {item}")
    else:
        print("No generator pieces found yet.")
    print(f"Total pieces: {len(inventory)}/6")
```

```python
def check_game_over(energy, inventory):
    """Check if the game is over (win or lose)"""
    # Check if player has won (found all 6 pieces)
    if len(inventory) >= 6:
        print("\n" + "=" * 50)
        print("🎉 CONGRATULATIONS! 🎉")
        print("You found all 6 generator pieces!")
        print("Your ship's energy generator is now complete!")
        print("You can escape this planet and return home!")
        print("=" * 50)
        return True

    # Check if player has lost (ran out of energy)
    if energy <= 0:
        print("\n" + "=" * 50)
        print("💀 GAME OVER 💀")
        print("You ran out of energy and collapsed!")
        print(f"You found {len(inventory)}/6 generator pieces.")
        print("Your ship remains stranded on this alien world...")
        print("=" * 50)
        return True

    # Game continues
    return False

def show_help():
    """Display help information"""
    print("\n--- HELP ---")
    print("Commands:")
    print("  go north/south/east/west - Move around the planet")
    print("  look - Look around your current location")
    print("  inventory - Check what pieces you've found")
    print("  map - Show the planet map")
    print("  quit - Quit the game")
    print("\nGoal: Find all 6 generator pieces before your energy runs out!")
    print("Moving costs 3 energy, finding pieces gives you 10 energy.")

def show_map(current_location, inventory):
    """Display a visual map of the planet"""
    print("\n" + "=" * 60)
    print("                    PLANET MAP")
    print("=" * 60)

    # Map symbols
    symbols = []
    for i in range(10):
        if i == current_location:
            symbols.append("[@]")  # Player location
        elif LOCATION_ITEMS[i] and LOCATION_ITEMS[i] in inventory:
            symbols.append("[✓]")  # Found item
        elif LOCATION_ITEMS[i]:
            symbols.append("[?]")  # Unknown item location
        else:
            symbols.append("[ ]")  # Empty location

    # Print the map grid
    print(f"  {symbols[0]}--{symbols[1]}--{symbols[2]}")
    print("   |    |    |")
    print(f"  {symbols[3]}--{symbols[4]}--{symbols[5]}")
    print("   |    |    |")
    print(f"  {symbols[6]}--{symbols[7]}--{symbols[8]}")
    print("        |")
    print(f"       {symbols[9]}")

    print("\nLEGEND:")
    print("[@] = Your current location")
    print("[✓] = Generator piece found")
    print("[?] = Unexplored area (may contain pieces)")
    print("[ ] = Empty area")

    print("\nLOCATIONS:")
    print("0: Crash Site       1: Rocky Outcrop    2: Crystal Cave")
    print("3: Dense Forest     4: Ancient Ruins    5: Frozen Lake")
    print("6: Volcanic Vents   7: Metal Wreckage   8: Strange Monolith")
    print("9: Energy Crater")
    print("=" * 60)
```

main.py

```
#!/usr/bin/env python3
"""
Dead Ship - Simple Text Adventure Game
Main game loop
"""

from functions import show_location, move_player, check_for_item, add_to_inventory,
show_inventory, check_game_over, show_help, show_map
from data import *

def main():
    """Main game loop"""
    # Game introduction
    print("=" * 50)
    print("           DEAD SHIP")
    print("=" * 50)
    print("\nYour ship has crashed on an alien planet!")
    print("The energy generator is broken into 6 pieces.")
    print("You must find all 6 pieces before your energy runs out!")
    print("Moving around costs energy, but finding pieces restores some.")
    print("\nCommands: go north, go south, go east, go west, look, inventory, map,
help, quit")
    print("=" * 50)

    # Initialize game state
    player_location = 0  # Start at location 0 (crash site)
    player_energy = 50   # Starting energy
    inventory = []       # Empty inventory
    turn_count = 0       # Track turns for energy decay
```

```python
# Main game loop
while True:
    # Show current location
    show_location(player_location, inventory)

    # Show player status
    print(f"\nEnergy: {player_energy}")
    print(f"Generator pieces found: {len(inventory)}/6")

    # Check if game is over
    if check_game_over(player_energy, inventory):
        break

    # Get player input
    command = input("\nWhat do you do? ").strip().lower()

    # Handle quit command
    if command == "quit":
        print("Thanks for playing!")
        break

    # Handle look command
    elif command == "look":
        continue  # Just redisplay location

    # Handle inventory command
    elif command == "inventory":
        show_inventory(inventory)
        continue

    # Handle help command
    elif command == "help":
        show_help()
        continue

    # Handle map command
    elif command == "map":
        show_map(player_location, inventory)
        continue

    # Handle movement commands
    elif command.startswith("go "):
        direction = command[3:]  # Get direction after "go "
        new_location = move_player(player_location, direction)

        if new_location != player_location:
            player_location = new_location
            player_energy -= 3  # Moving costs energy
            turn_count += 1

            # Check for items at new location
            found_item = check_for_item(player_location)
            if found_item and found_item not in inventory:
                print(f"\nYou found a {found_item}!")
                add_to_inventory(inventory, found_item)
                player_energy += 10  # Finding item restores energy
            else:
                print("You can't go that way!")

        # Handle unknown commands
        else:
            print("Unknown command. Try: go north/south/east/west, look, inventory, quit")

        # Energy decays over time
        if turn_count > 0 and turn_count % 3 == 0:
            player_energy -= 1
            if player_energy <= 0:
                player_energy = 0

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

Conclusion

Congratulations! You've now learned how to create a complete text adventure game using Python. This project demonstrates many fundamental programming concepts:

- **Problem solving** - Breaking down a complex game into smaller parts
- **Code organization** - Using multiple files and functions
- **Data management** - Using lists to store game information
- **User interaction** - Processing commands and providing feedback
- **Game logic** - Implementing rules and win/lose conditions

Next Steps

1. **Experiment** - Try the exercises and extensions
2. **Customize** - Change the story, locations, or mechanics
3. **Share** - Show your game to friends and family
4. **Learn more** - Explore advanced Python concepts like classes and file I/O

Remember: Programming is about solving problems creatively. Don't be afraid to experiment and make mistakes - that's how you learn!

Happy coding! 🚀