Quiz - Python Programming (Year 2024)

National Software Contest: NSC เขตภาคกลาง

โครงการการแข่งขันพัฒนาโปรแกรมคอมพิวเตอร์แห่งประเทศไทย

1. Arithmetic Operations (an answer provided)

Write a Python program that takes the health points (HP) of two characters from the user and performs the following operations:

1. Add their HP.
2. Subtract the second character's HP from the first.
3. Multiply their HP values.
4. Divide the first character's HP by the second.

Expected Input:

```
Character 1 HP: 100
Character 2 HP: 50
```

Expected Output:

```
Total HP: 150
HP Difference: 50
HP Product: 5000
HP Ratio: 2.0
```

Answer:

```
# Input health points for two characters
char1_hp = int(input("Character 1 HP: "))
char2_hp = int(input("Character 2 HP: "))

# Perform arithmetic operations
total_hp = char1_hp + char2_hp
hp_difference = char1_hp - char2_hp
hp_product = char1_hp * char2_hp
hp_ratio = char1_hp / char2_hp if char2_hp != 0 else "undefined (division by
zero)"

# Display results
print(f"Total HP: {total_hp}")
print(f"HP Difference: {hp_difference}")
print(f"HP Product: {hp_product}")
print(f"HP Ratio: {hp_ratio}")
```

2. Data Types and Variables (an answer provided)

Write a program to assign a character name (string), attack power (integer), and energy level (float) to three variables and print them with their types using `type()`.

Expected Output:

```
Character Name: Naruto (Type: <class 'str'>)
Attack Power: 95 (Type: <class 'int'>)
Energy Level: 89.5 (Type: <class 'float'>)
```

Answer:

```
# Assigning values to variables
character_name = "Naruto"  # string
attack_power = 95          # integer
energy_level = 89.5        # float

# Printing the variables and their types
print(f"Character Name: {character_name} (Type: {type(character_name)})")
print(f"Attack Power: {attack_power} (Type: {type(attack_power)})")
print(f"Energy Level: {energy_level} (Type: {type(energy_level)})")
```

3. Lists

Create a Python program that initializes a list of 5 game weapons. Perform the following operations:

1. Append a new weapon (e.g. “Axe”) to the list.
2. Remove the second weapon.
3. Sort the list alphabetically.

Expected Operations:

```
weapons = ["Sword", "Spear", "Bow", "Axe", "Dagger"]
```

Expected Output:

```
["Axe", "Bow", "Dagger", "Staff", "Sword"]
```

Answer:

```
weapon_lst = ["Sword", "Spear", "Bow", "Staff", "Dagger"]


def weapon_add(weapon: str) -> None:
    weapon_lst.append(weapon)


def main() -> None:
    weapon_add("Axe")
    weapon_lst.pop(1)
    print(sorted(weapon_lst))


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

4. Dictionaries

Create a dictionary with three key-value pairs representing anime characters and their power levels. Add a new character's power level and update an existing character's power level.

Expected Operations:

```
characters = {"Goku": 9000, "Luffy": 5000, "Naruto": 7000}
```

Expected Output:

```
{"Goku": 9000, "Luffy": 10000, "Naruto": 7000, "Saitama": 6000}
```

Answer:

```
characters_power = {
    "Goku": 9000,
    "Luffy": 5000,
    "Naruto": 7000,
}


def power_update(characters: str, power: int) -> None:
    characters_power[characters] = power


def add_member(characters: str, power: int) -> None:
    characters_power.update({characters: power})


def main() -> None:
    add_member("Saitama", 6000)

    power_update("Goku", 9000)
    power_update("Luffy", 10000)
    power_update("Naruto", 7000)

    print(characters_power)


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

5. Conditional Statements

Write a Python program that asks the user to input a character's element type (e.g., Fire, Water, Earth, Air). Based on the input, print out the character's strength.

- Fire: Strong against Earth
- Water: Strong against Fire
- Earth: Strong against Air
- Air: Strong against Water
- Any other input: Unknown element

Expected Input:

```
Fire
```

Expected Output:

```
Fire is strong against Earth.
```

Answer:

```
while True:
    try:
        element = input("Enter a element: ")
        if type(float(element)) == float:
            print("number is not supported, please enter a string")
    except ValueError:
        break


match element.strip():
    case "Fire":
        print("Fire: Strong against Earth")
    case "Water":
        print("Water: Strong against Fire")
    case "Earth":
        print("Earth: Strong against Air")
    case "Air":
        print("Air: Strong against Water")
    case _:
        print("Unknown element")
```

6. Loops

Write a Python program that prints the first 10 levels of experience points (XP) required to level up using a for-loop. Assume the XP required at each level increases by 100.

Expected Output:

```
100
200
300
400
500
600
700
800
900
1000
```

Answer:

```
max_level = 10

for i in range(1, max_level + 1, 1):
    print(f"You need {i * 100} (XP) to level up to {i}")
```

7. Functions

Create a function `is_ultimate_attack_ready(chakra)` that checks if **a ninja** has enough chakra to perform an ultimate attack (requires at least 100 chakra). Use this function to check chakra levels for three characters.

Expected Input:

```
Character 1's chakra levels: 90
Character 2's chakra levels: 150
Character 3's chakra levels: 120
```

Expected Output:

```
Character 2 can perform the ultimate attack.
Character 3 can perform the ultimate attack.
```

Answer:

```
def is_ultimate_attack_ready(chakra: int) -> None:
    if chakra >= 100:
        return True
    else:
        return False


def main() -> None:

    chakra_1 = float(input("Character 1's chakra levels: "))
    chakra_2 = float(input("Character 2's chakra levels: "))
    chakra_3 = float(input("Character 3's chakra levels: "))
    print()

    ninja = {
        "Character 1": chakra_1,
        "Character 2": chakra_2,
        "Character 3": chakra_3,
    }

    for character in ninja:
        if is_ultimate_attack_ready(ninja[character]) == True:
            print(f"{character} can perform the ultimate attack")


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

8. Object-Oriented Programming

Create a Python class `Hero` that has attributes `name`, `health`, and `attack`. Include methods to display the hero's stats and to attack another hero, reducing their health. Demonstrate the usage of this class by creating two hero objects.

Expected Input:

```
Hero1
Name: "Link", Health: 100, Attack: 30

Hero2
Name: "Zelda", Health: 120, Attack: 25
```

Expected Output:

```
Link attacks Zelda! Zelda's health drops to 90.

Hero1
Name: "Link", Health: 100, Attack: 30

Hero2
Name: "Zelda", Health: 90, Attack: 25
```

Answer:

```
class Hero:
    def __init__(self, name, health, attack):
        self.name = name
        self.health = health
        self.attack = attack

    def display_status(self):
        print(f"Name: {self.name}, Health: {self.health}, Attack:
{self.attack}")

    def is_alive(self):
        return self.health > 0

    def attack_hero(self, other_hero):

        if not self.is_alive():
            print(f"{self.name} cannot attack because they have been
defeated.")
            return
        if not other_hero.is_alive():
            print(f"{other_hero.name} has already been defeated.")
            return

        damage = self.attack
        other_hero.health -= damage

        if other_hero.health < 0:
            other_hero.health = 0
```

```
        print(
            f"{self.name} attacks {other_hero.name} for {damage} damage!
{other_hero.name} health drops to {other_hero.health}"
        )

        if not other_hero.is_alive():
            print(f"{other_hero.name} has been defeated!")


def main() -> None:
    hero1 = Hero("Link", 230, 50)
    hero2 = Hero("Zelda", 100, 12)

    hero1.display_status()
    hero2.display_status()
    print()

    hero1.attack_hero(hero2)
    hero2.display_status()
    print()

    hero2.attack_hero(hero1)
    hero1.display_status()


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

9. File Handling

Monster Encyclopedia Create a Python program that reads from a file containing details of monsters in a fantasy game (name, type, and HP). The program should list all the monsters of a specified type, which the user will input.

```
Dragon,Fire,1500
Goblin,Earth,500
Phoenix,Fire,1200
Ogre,Earth,1000
```

Expected Input:

```
Fire
```

Expected Output:

```
Fire Monsters
Dragon: 1500 HP
```

```
Phoenix: 1200 HP
```

Answer:

```
def file_write() -> None:
    with open("monsters.txt", "w") as file:
        file.write("Dragon,Fire,1500\n")
        file.write("Goblin,Earth,500\n")
        file.write("Phoenix,Fire,1200\n")
        file.write("Ogre,Earth,1000\n")


def file_read(lst: list) -> list:
    with open("monsters.txt", "r") as file:
        for line in file:
            name, monster_type, hp = line.strip().split(",")
            lst.append({"name": name, "type": monster_type, "hp": int(hp)})
    return lst


def main() -> None:
    file_write()

    monsters = []
    search_type = (input("Enter a type of monster: ")).strip()

    filtered_monsters = [
        monster for monster in file_read(monsters) if monster["type"] ==
search_type
    ]

    if filtered_monsters:
        print(f"{search_type} Monsters")
        for monster in filtered_monsters:
            print(f"{monster['name']}: {monster['hp']} HP")
    else:
        print(f"No monsters of type '{search_type}' found.")


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