1. Providing example code and necessary elaborations for demonstrating advantages and disadvantages of Dynamic Typing as specified in Task 2.

   - Advantages
     - More generic code can be written. In other words, functions can be defined to apply on arguments of different types.
       For example, I have a `findLarger` function, which return the larger integer of the two provided to the user.

```java
public int findLarger(int num1, int num2){
    return (num1 > num2)? num1 : num2 ;
}
```

Listing 1: Java implementation

```python
def find_larger(self, num1, num2):
    num1 if (num1 > num2) else num2
```

Listing 2: Python implementation

       In Listing 1, I can only pass in `int` to the method, if I need to pass a `float` or `String`, I have to create a new method to accept those types. But in Listing 2, the function can accept `int`, `float` or `String` to the argument `num1` and `num2`, and the function will still works.
     - Possibilities of mixed type collection data structures.
       In task 1, we are able to declare the player as a human player or a computer without explicily declaring the type.

```python
def start_game(self):
(...)
    if x == '1':
        self.players.append(Human(1, self.board))
        print('Player 1 is a human.')
    elif x == '2':
    self.players.append(Computer(1, self.board))
    print('Player 1 is a computer.')
(...)
    if x == '1':
        self.players.append(Human(2, self.board))
        print('Player 2 is a human.')
    elif x == '2':
        self.players.append(Computer(2, self.board))
        print('Player 2 is a computer.')
```

Listing 3: Code snippet from SixMensMorris.py

   - Disadvantage
     - Loss of type checking at compile time.
       As shown in Listing 1 and 2, Java implementation of `findLarger` can ensure that I can only compare integer-typed variable, if I misused the method, it will give a warning. But in Python implementation, a `int` and a `float` is comparable, and it will return a result. However, it will defeat my original purpose as I only want to return the larger integer of the two provided.

2. Using codes for Task 3, give two scenarios in which the Python implementation is better than the Java implementation. Given reasons.

   – When using `actionOnSoldier`, in Java implementation, we have to care about the type of the object, but not in Python.

```java
(...)
if (occupiedObject instanceof Monster) {
    ((Monster)occupiedObject).actionOnSoldier(this.soldier);
} else if (occupiedObject instanceof Spring) {
    ((Spring)occupiedObject).actionOnSoldier(this.soldier);
} else {
(...)
```

```python
(...)
if occupied_object != None:
    occupied_object.action_on_soldier(self._soldier)
(...)
```

   – There are no access control in Python, which is more convenient than Java as we need do create additional method just to give usage to other classes.

```java
private int monsterID;
private int healthCapacity;
private int health;
private Pos pos;
private ArrayList<Integer> dropItemList;
private ArrayList<Integer> hintList;
(...)
public int getMonsterID() {
    return this.monsterID;
}
(...)
```

3. Using codes for Task 4, illustrate further advantages of Duck Typing.
   As in duck typing, we only consider the method that the class have, but not the type of the class. This can ignore the object type as long as we have the same function.

```python
def action_on_soldier(self, soldier):
    self._choice = input("Do you want to buy something? (1. Elixir, 2. Shield) Input: ")
    (...)
```

Listing 4: action_on_soldier() in Merchant

```python
def action_on_soldier(self, soldier):
    if(self._health <= 0):
    (...)
```

Listing 5: action_on_soldier() in Monster

```python
if(self._map.check_move(self._new_row, self._new_column)):
    occupied_object = self._map.get_occupied_object(self._new_row, self._new_column)
if occupied_object != None:
    occupied_object.action_on_soldier(self._soldier)
```

Listing 6: Usage of action_on_soldier() in Python

```java
if (occupiedObject instanceof Task4Monster) {
    ((Task4Monster)occupiedObject).actionOnSoldier(this.soldier);
} else if (occupiedObject instanceof Spring) {
    ((Spring)occupiedObject).actionOnSoldier(this.soldier);
} else if (occupiedObject instanceof Task4Merchant) {
    ((Task4Merchant)occupiedObject).actionOnSoldier(this.soldier);
}
```

Listing 7: Usage of actionOnSoldier() in Java