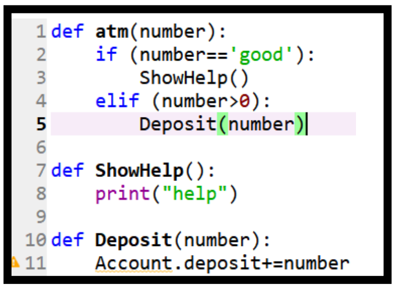
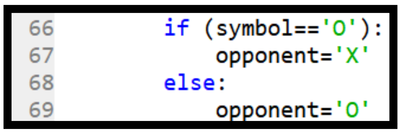
Name: Wong Tsz Yin CSCI3180 HW2 Task2

More generic code can be written. In other words, functions can be defined to apply on arguments of different types



```
1  def atm(number):
2      if (number=='good'):
3          ShowHelp()
4      elif (number>0):
5          Deposit(number)
6  
7  def ShowHelp():
8      print("help")
9  
10 def Deposit(number):
11     Account.deposit+=number
```

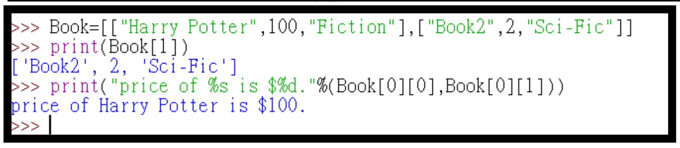
From the above code, it demonstrates that we can allows the same variable to perform action by different types, in line 2, it acts as a string. In line 4, it acts as int.



```
        if (symbol=='O'):
            opponent='X'
        else:
            opponent='O'
```

From the above segment, we don't have to define the type of the variables and can directly assign the desired type into it. Time can be saved.

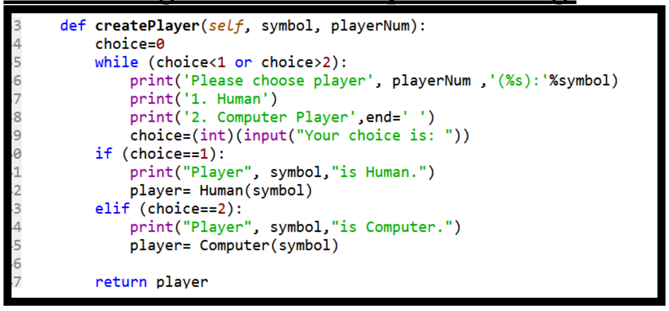
Possibilities of mixed type collection data structures



```
>>> Book=[["Harry Potter",100,"Fiction"],["Book2",2,"Sci-Fic"]]
>>> print(Book[1])
['Book2', 2, 'Sci-Fic']
>>> print("price of %s is $%d."%(Book[0][0],Book[0][1]))
price of Harry Potter is $100.
>>> 
```

We declare a list called book here, you can see the fields inside each sub-list are with different types. We can also print the code like the last line, but with static typing, we may have to declare one new structure to make this happen.

Disadvantage: Sometime requires casting.



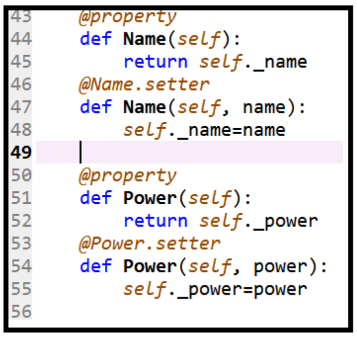
```
    def createPlayer(self, symbol, playerNum):
        choice=0
        while (choice<1 or choice>2):
            print('Please choose player', playerNum ,'(%s):'%symbol)
            print('1. Human')
            print('2. Computer Player',end=' ')
            choice=(int)(input("Your choice is: "))
        if (choice==1):
            print("Player", symbol,"is Human.")
            player= Human(symbol)
        elif (choice==2):
            print("Player", symbol,"is Computer.")
            player= Computer(symbol)

        return player
```

For variable "choice", we must cast it into integer. Otherwise, it will return error because we cannot compare str with int. But with static typing, we don't need casting.

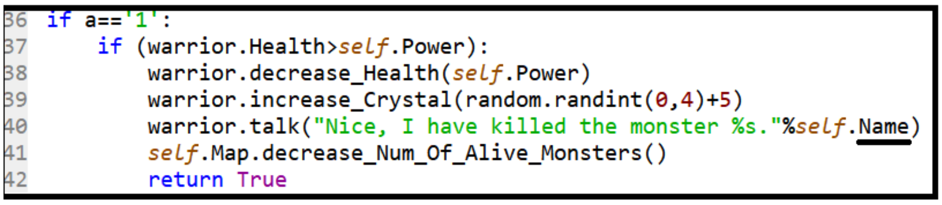
Two scenarios in which the Python implementation is better than the Java implementation

python

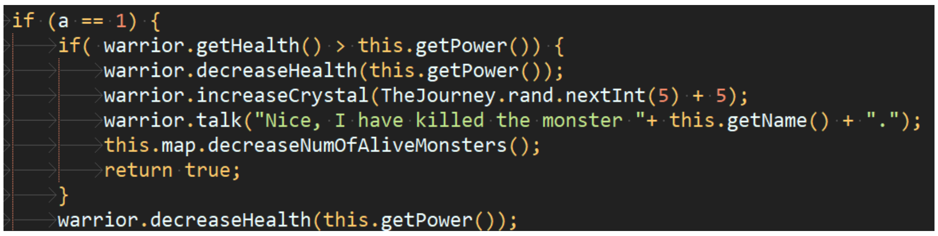
1. **Automatically call getter/setter**



```
    @property
    def Name(self):
        return self._name
    @Name.setter
    def Name(self, name):
        self._name=name

    @property
    def Power(self):
        return self._power
    @Power.setter
    def Power(self, power):
        self._power=power

```



```
if a=='1':
    if (warrior.Health>self.Power):
        warrior.decrease_Health(self.Power)
        warrior.increase_Crystal(random.randint(0,4)+5)
        warrior.talk("Nice, I have killed the monster %s."%self.Name)
        self.Map.decrease_Num_Of_Alive_Monsters()
        return True
```

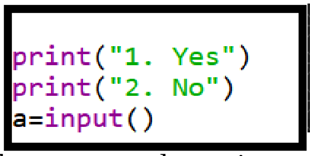


```
if (a == 1) {
    if( warrior.getHealth() > this.getPower()) {
        warrior.decreaseHealth(this.getPower());
        warrior.increaseCrystal(TheJourney.rand.nextInt(5) + 5);
        warrior.talk("Nice, I have killed the monster "+ this.getName() + ".");
        this.map.decreaseNumOfAliveMonsters();
        return true;
    }
    warrior.decreaseHealth(this.getPower());
```

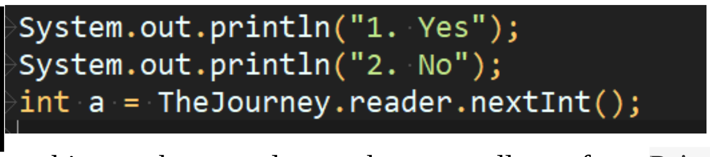
Java

Python will decide call getter or setter automatically by how we call the functions. (Line 40: it calls the getter). But in java, we have to state explicitly which one to call.

2.**<u>Console i/o is easy to use and neat</u>**



```
print("1. Yes")
print("2. No")
a=input()
```

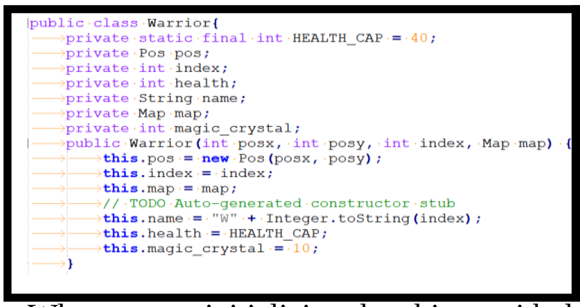


```
System.out.println("1. Yes");
System.out.println("2. No");
int a = TheJourney.reader.nextInt();
```

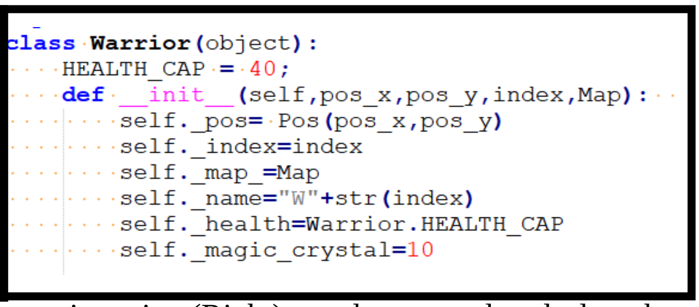
In java, when you need to print something to the console, you have to call .out from PrintStream . Similar with reading something from console. But in Python, the thing you need to type is shorter and you do not have to specify the type.

<u>The advantages of Dynamic Typing and Duck Typing</u>

<u>Dynamic Typing</u>



```
public class Warrior{
    private static final int HEALTH_CAP = 40;
    private Pos pos;
    private int index;
    private int health;
    private String name;
    private Map map;
    private int magic_crystal;
    public Warrior(int posx, int posy, int index, Map map) {
        this.pos = new Pos(posx, posy);
        this.index = index;
        this.map = map;
        // TODO Auto-generated constructor stub
        this.name = "W" + Integer.toString(index);
        this.health = HEALTH_CAP;
        this.magic_crystal = 10;
    }
```

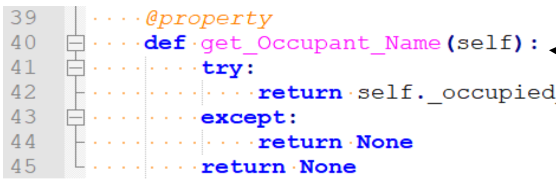


```
class Warrior(object):
    HEALTH_CAP = 40;
    def __init__(self,pos_x,pos_y,index,Map):
        self._pos= Pos(pos_x,pos_y)
        self._index=index
        self._map =Map
        self._name="W"+str(index)
        self._health=Warrior.HEALTH_CAP
        self._magic_crystal=10
```

Python

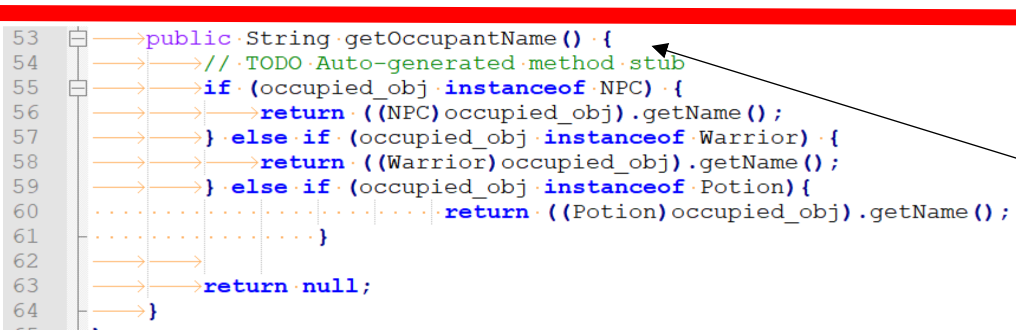
When we are initializing the object, with dynamic typing (Right), we do not need to declare the private variables and we can use them directly. When we are initializing the object with many fields, it saves us a lot of time. Just like the above example, the variable declaration part is equal to the initializing pat.

<u>Duck Typing</u>



```
39      @property
40      def get_Occupant_Name(self):
41          try:
42              return self._occupied_obj.get_Name
43          except:
44              return None
45          return None
```

python



```
53      public String getOccupantName() {
54          // TODO Auto-generated method stub
55          if (occupied_obj instanceof NPC) {
56              return ((NPC)occupied_obj).getName();
57          } else if (occupied_obj instanceof Warrior) {
58              return ((Warrior)occupied_obj).getName();
59          } else if (occupied_obj instanceof Potion){
60                              return ((Potion)occupied_obj).getName();
61                  }
62
63          return null;
64      }
```

Java

When we using duck typing (Top) , there is no need to care about the type of the object. We just need to try to call the function. Compared to Java which has no duck typing (Bottom), and we need to check specifically that calling getName() from which class. It is obvious that the writability of the code increases with duck typing.