CSCI3180 Asgn 2
Choi Man Kin
1155077469

Task 2

1. Functions can be applied on arguments of different types.

`class A:`
 	
`    def b(self, i='default'):`
`        print('Nothing')`
`        print(i)`
 	
`a = A()`
`a.b(2)`
`a.b()`
	Nothing
	2
	Nothing
	default

Python	stdout

We can use default value for arguments. This is flexiable and convenient to design method. Even though we want to put an integer instead of string, it works in this example. The type will bind at run-time.

2. Possibilities of mixed type collection data structures

```
class Human(object):
    def nextMove(self):
        return None
    def eat(self):
        return None

class Ghost(object):
    def nextMove(self):
        return None
    def kill(self):
        return None

class Computer(object):
    def nextMove(self):
        return None
    def calculate(self):
        return None

# the following collection contains different types of object
objects = [Human(), Ghost(), Computer()]
moves = []
for obj in objects:
    # they all have nextMove(), so I can call it
    moves.append(obj.nextMove())
```

As you can see, Human, Ghost, Computer are three different types of object. They can be put into one single list, and we can still call their nextMove() without casting or specifying that they have nextMove().

3. No need for type declaring

I don't need to declare type for every variable in Python.

Task 3

1. Scenarios 1 – we don't need to do so many type castings when using the object.

`for (Object obj : teleportObjects) {`
`  if (obj instanceof Human)`
`  ((Human) obj).teleport();`
`  else if (obj instanceof Chark)`
`  ((Chark) obj).teleport();`
`  else if (obj instanceof Obstacle)`
`  ((Obstacle) obj).teleport();`
`}`
	`for teleportObject in self.__teleportObjects:`
	`  teleportObject.teleport()`

JAVA	Python

We can see JAVA version is longer. If we have more races in the future, we need more casting and the code is longer and longer. However, what it does is just teleporting, not adding new features. We can conclude that Python is more flexible and convenient.

However, I think this may not be an absolute advantage. If somebody append some objects without teleport(), it will result run-time errors. Instead, JAVA can achieve the similar thing by introducing a new interface but without run-time errors. Anyway, Python is shorter.

2. Scenarios 2 – we can use dynamically length array

`teleportObjects = new Object[n + 0];`	`self.__teleportObjects = []`
JAVA	Python

In JAVA, we need to declare teleportObjects' size. In Python, we don't need, because it is a list. I can just use append in Python instead of array assignment in JAVA. I don't need to calculate the array index of the inserting object. This is more convenient.

Task 4

Using Scenarios 1 example, if an object can teleport(), I don't need to cast it to an object which contains a method teleport() to invoke it. I just need to invoke it deservedly, because the method is defined on it. Therefore, it cannot check during compile time. When the method is called, the existence of the method will be checked on run-time.

Nowadays, we always use interactive Python notebook to do experiment. Dynamic typing is more suitable for interactivity. We can prototype rapidly.