先说结论：

1. 对于原子（不可变）类型的数据（字符串、数字、元组等），没有拷贝这一说法。原子类型的数据发生改变时，相当新建了对象。
2. 浅拷贝： 1. 拷贝的对象是可变类型，则开辟新的空间去存储，拷贝后的地址跟原地址不一样。 2. 拷贝的对象是不可变类型，则只拷贝其地址的引用，拷贝后的地址跟原地址一样。 3. 拷贝的对象有嵌套的子对象，不会拷贝所有的子对象。

   浅拷贝是拷贝了源对象的引用，并创建了一个新的内存空间地址。但是引用的对象的子对象的地址仍然是源对象的，所以当源对象的子对象发生改变时，拷贝对象内的子对象同时也跟着改变。
3. 深拷贝： 1. 拷贝的对象是可变类型，则开辟新的空间去存储，拷贝后的地址跟原地址不一样。 2. 拷贝的对象是不可变类型，则只拷贝其地址的引用，拷贝后的地址跟原地址一样。 3. 拷贝的对象有嵌套的子对象，会对所有的子对象进行拷贝，拷贝方法如1,2点 4. 拷贝的对象值有修改，不会影响到深拷贝后的对象。

   深拷贝就是彻底的拷贝，完全的拷贝了父对象及子对象，同时指向一个新的内存空间地址。源对象与拷贝对象之间的修改互不影响。

两者的优缺点对比：

（1）深拷贝拷贝程度高，将原数据复制到新的内存空间中。改变拷贝后的内容不影响原数据内容。但是深拷贝耗时长，且占用内存空间。

（2）浅拷贝拷贝程度低，只复制原数据的地址。其实是将副本的地址指向原数据地址。修改副本内容，是通过当前地址指向原数据地址，去修改。所以修改副本内容会影响到原数据内容。但是浅拷贝耗时短，占用内存空间少。

1.Python的浅拷贝

```
import copy
obj1 = ["a", 111, ["c", "php", "python"]]
obj2 = copy.copy(obj1) # 浅拷贝
print(obj1)
print(f"id of obj1 is: {id(obj1)}")
print([id(i) for i in obj1])
print("-"*10)
print(obj2)
print(f"id of obj2 is: {id(obj2)}")
print([id(i) for i in obj2])
print("改变obj1的值后")
obj1[0] = "b"
obj1[2].append("java")
print(obj1)
print(f"id of obj1 is: {id(obj1)}")
```

```
print([id(i) for i in obj1])
print("-"*10)
print(obj2)
print(f"id of obj2 is: {id(obj2)}")
print([id(i) for i in obj2])
```

输出：

结论：对于python的浅拷贝，如果被拷贝的对象包含不可变的子对象，比如例题中的“a”，111，拷贝时会开辟新的空间去存储该子对象，可变类型的子对象拷贝时只是拷贝的对象的引用，如代码中["c", "php", "python"]。

2.Python的深拷贝

```
import copy
obj1 = ["a", 111, ["c", "php", "python"]]
obj2 = copy.deepcopy(obj1) # 深拷贝
print(obj1)
print(f"id of obj1 is: {id(obj1)}")
print([id(i) for i in obj1])
print("-"*10)
print(obj2)
print(f"id of obj2 is: {id(obj2)}")
print([id(i) for i in obj2])
print("改变obj1的值后")
obj1[0] = "b"
obj1[2].append("java")
print(obj1)
print(f"id of obj1 is: {id(obj1)}")
print([id(i) for i in obj1])
print("-"*10)
print(obj2)
print(f"id of obj2 is: {id(obj2)}")
print([id(i) for i in obj2])
```

输出

结论：对于python的深拷贝，如果对象中有不可变类型的子对象，则拷贝子对象的引用，否则就是创建了一个与之前对象完全独立的对象。

如果对象只包含原子类型（不可变类型的对象），深拷贝就不会重新生成一个对象，这其实是python解释器内部的一种优化，对于只包含原子类型对象的元组，如果他们的值相等，就在内存中保存一份，类似的还有小整数从-5~256.在内存中只保留一份，可节省内存，提高访问速度。