Python面向对象语法精讲

嵩 天

Python应用基础 版权 by 嵩天

Python对象的引用

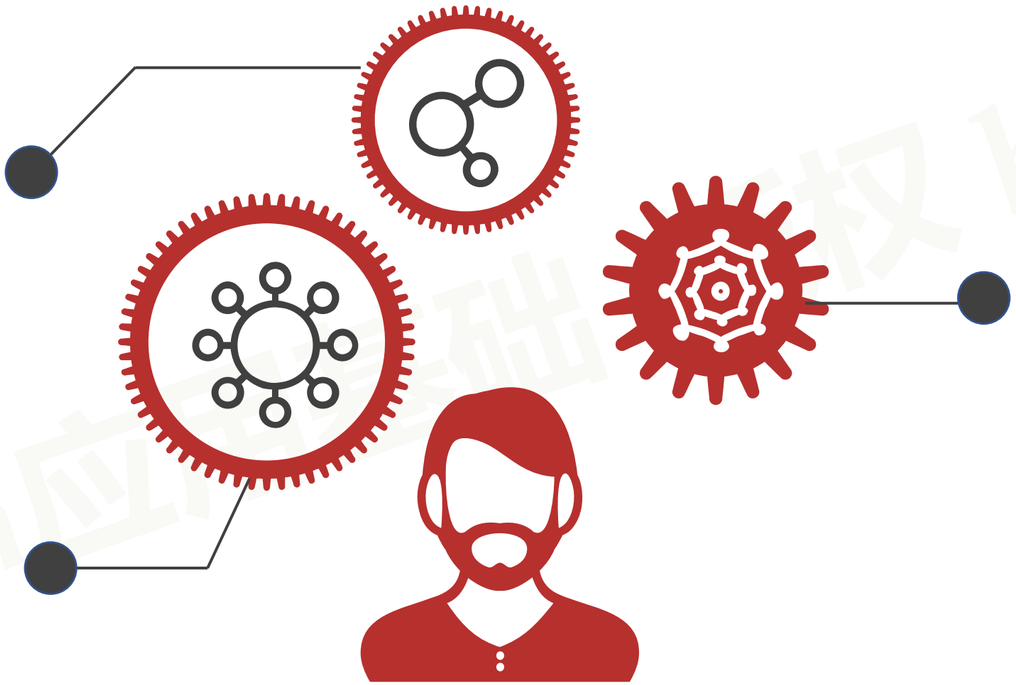
嵩天

Python对象的引用

单元开篇

单元开篇

(2) 浅拷贝和深拷贝



(3) 类的方法引用

(1) 引用的理解

Python应用基础 版权 by 嵩天

Python对象的引用

Python对象的引用

引用的理解

引用的理解

引用 Reference：对象的指针

- 引用是内存中真实对象的指针，表示为变量名或内存地址
- 每个对象存在至少1个引用，id()函数用于获得引用
- 在传递参数和赋值时，Python传递对象的引用，而不是复制对象

Python应用基础 版权 by 嵩天

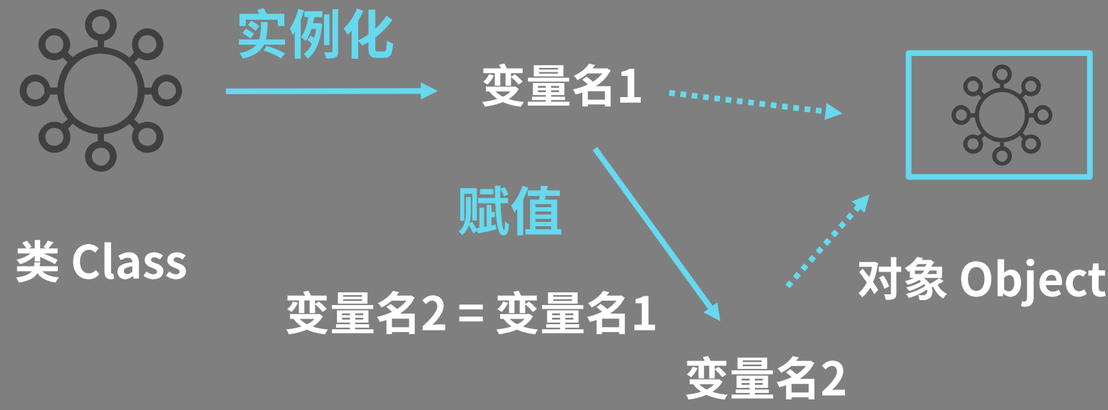
引用的理解

引用 Reference：对象的指针

```
ls = [1, 2, 3, 4, 5]
lt = ls
print(id(ls))
print(id(lt))
```

93463456

93463456



引用的理解

Python内部机制对引用的处理

- 不可变对象：immutable 解释器为相同值维护尽量少的内存区域
- 可变对象：mutable 解释器为每个对象维护不同内存区域

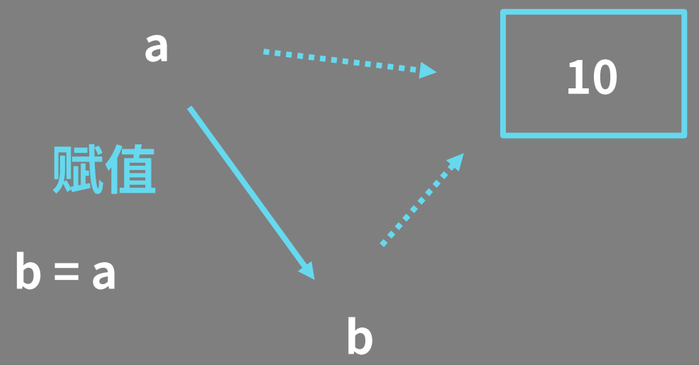
Python应用基础 版权 by 嵩天

对象引用详解

```
a = 10
b = a
c = 10
print(id(a))
print(id(b))
print(id(c))
```

不可变对象的引用：整数

```
1376965824
1376965824
1376965824
```



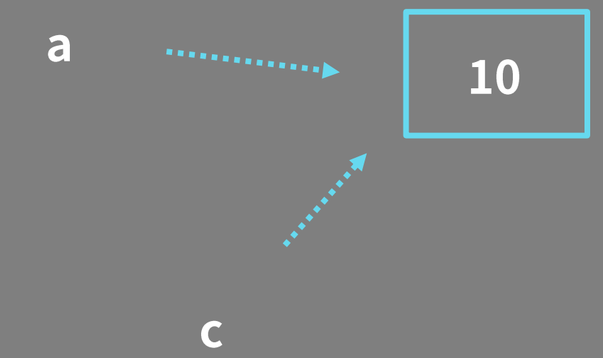
赋值仅实现了增加引用的功能

对象引用详解

```
a = 10
b = a
c = 10
print(id(a))
print(id(b))
print(id(c))
```

不可变对象的引用：整数

```
1376965824
1376965824
1376965824
```



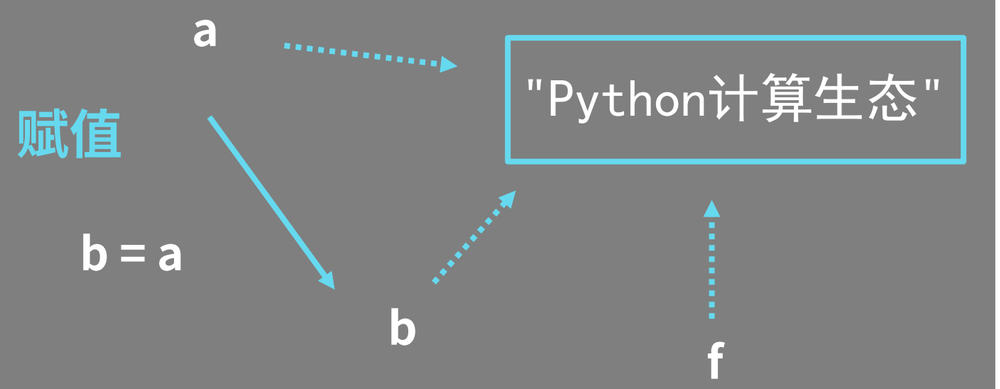
Python解释器内部对a和c维护1个对象

对象引用详解

```
a = "Python计算生态"
b = a
c = "Python"
d = "计算生态"
e = c + d
f = "Python计算生态"
print(id(a))
print(id(b))
print(id(c))
print(id(d))
print(id(e))
print(id(f))
```

不可变对象的引用：字符串

```
81794720
81794720
81222304
81757632
81794912
81794720
```



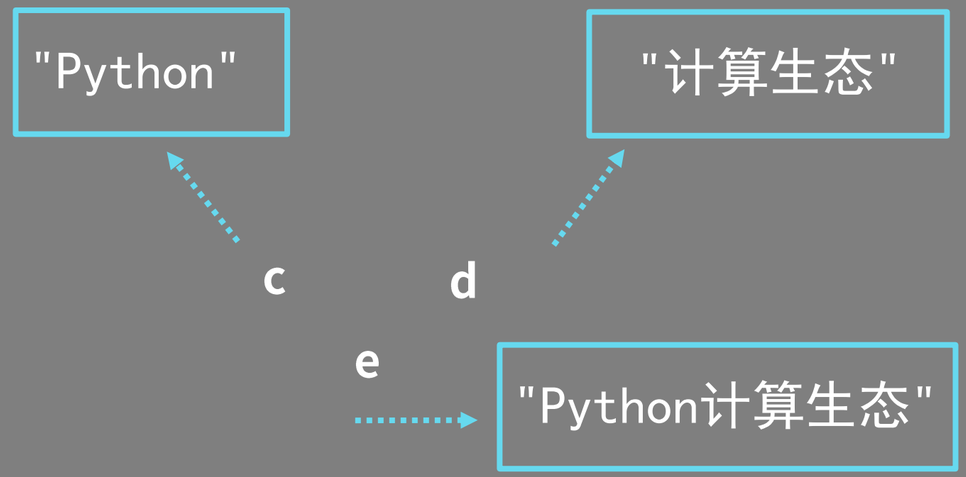
赋值仅实现了增加引用的功能

对象引用详解

```
a = "Python计算生态"
b = a
c = "Python"
d = "计算生态"
e = c + d
f = "Python计算生态"
print(id(a))
print(id(b))
print(id(c))
print(id(d))
print(id(e))
print(id(f))
```

不可变对象的引用：字符串

```
81794720
81794720
81222304
81757632
81794912
81794720
```



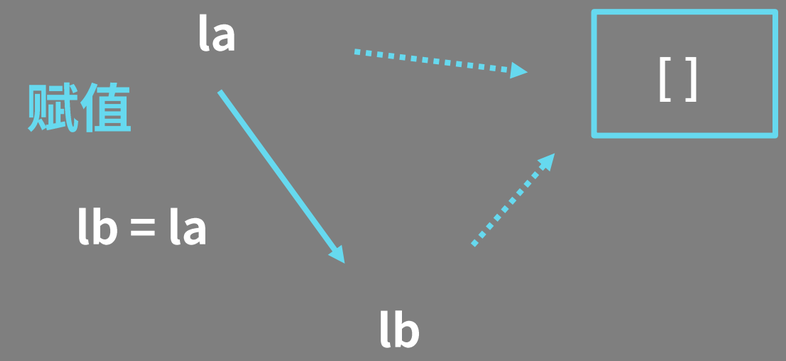
运算后产生的对象由解释器重新建立

对象引用详解

```
la = []
lb = la
lc = []
print(id(la))
print(id(lb))
print(id(lc))
```

可变对象的引用：列表

```
81732512
81732512
81021088
```



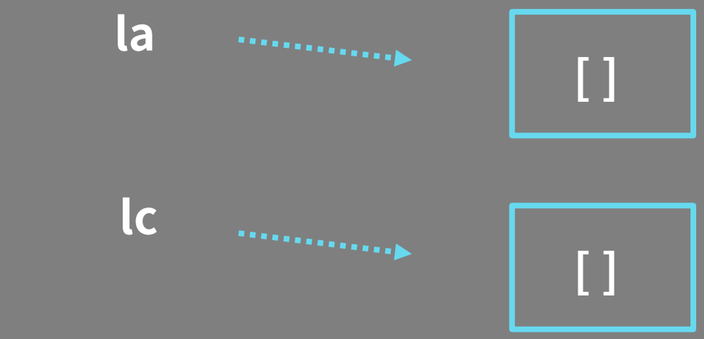
赋值新增了一个对象的引用

对象引用详解

```
la = []
lb = la
lc = []
print(id(la))
print(id(lb))
print(id(lc))
```

可变对象的引用：列表

```
81732512
81732512
81021088
```



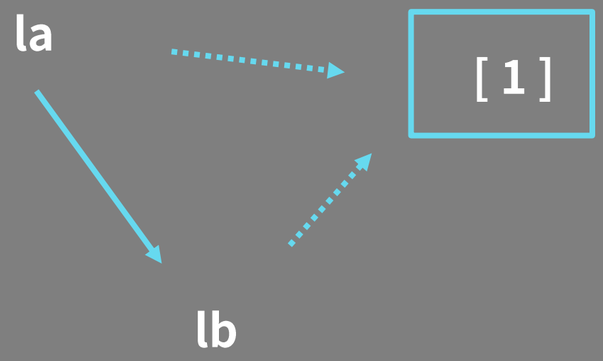
每个可变对象都由解释器重新创建，不复用内存

对象引用详解

```
la = []
lb = la
lb.append(1)
print(la, id(la))
print(lb, id(lb))
```

可变对象的引用：列表

```
[1] 92021664
[1] 92021664
```



对象修改后，所有引用的值都被修改

引用的理解

导致引用+1的情况

- 对象被创建： d = DemoClass()
- 对象被引用： a = d
- 对象被作为函数或方法的参数：sys.getrefcount(d)
- 对象被作为一个容器中的元素：ls = [d]

Python应用基础 版权 by 嵩天

引用的理解

导致引用-1的情况

- 对象被删除： del d
- 对象的名字被赋予新的对象： d = 123
- 对象离开作用域：foo()函数的局部变量count
- 对象所在容器被删除：del ls

Python应用基础 版权 by 嵩天

小结：引用的理解

引用 Reference：对象的指针

- 引用是内存中真实对象的指针，表示为变量名或内存地址
- 在传递参数和赋值时，Python传递对象的引用，而不是复制对象
- 不可变对象与可变对象的内存管理略有不同

Python应用基础 版权 by 嵩天

Python对象的引用

浅拷贝和深拷贝

Python123

对象的拷贝

浅拷贝和深拷贝

- 拷贝：复制一个对象为新的对象，内存空间有“变化”
- 浅拷贝：仅复制最顶层对象的拷贝方式，默认拷贝方式
- 深拷贝：迭代复制所有对象的拷贝方式

Python应用基础 版权 by 嵩天

浅拷贝详解

```
ls = ["Python", [1, 2, 3]]
la = ls.copy()
lb = ls[:]
lc = list(ls)
print("ls", id(ls), ls)
print("la", id(la), la)
print("lb", id(lb), lb)
print("lc", id(lc), lc)
```

```
ls 83314848 ['Python', [1, 2, 3]]
la 84026072 ['Python', [1, 2, 3]]
lb 83968168 ['Python', [1, 2, 3]]
lc 84027152 ['Python', [1, 2, 3]]
```

浅拷贝详解

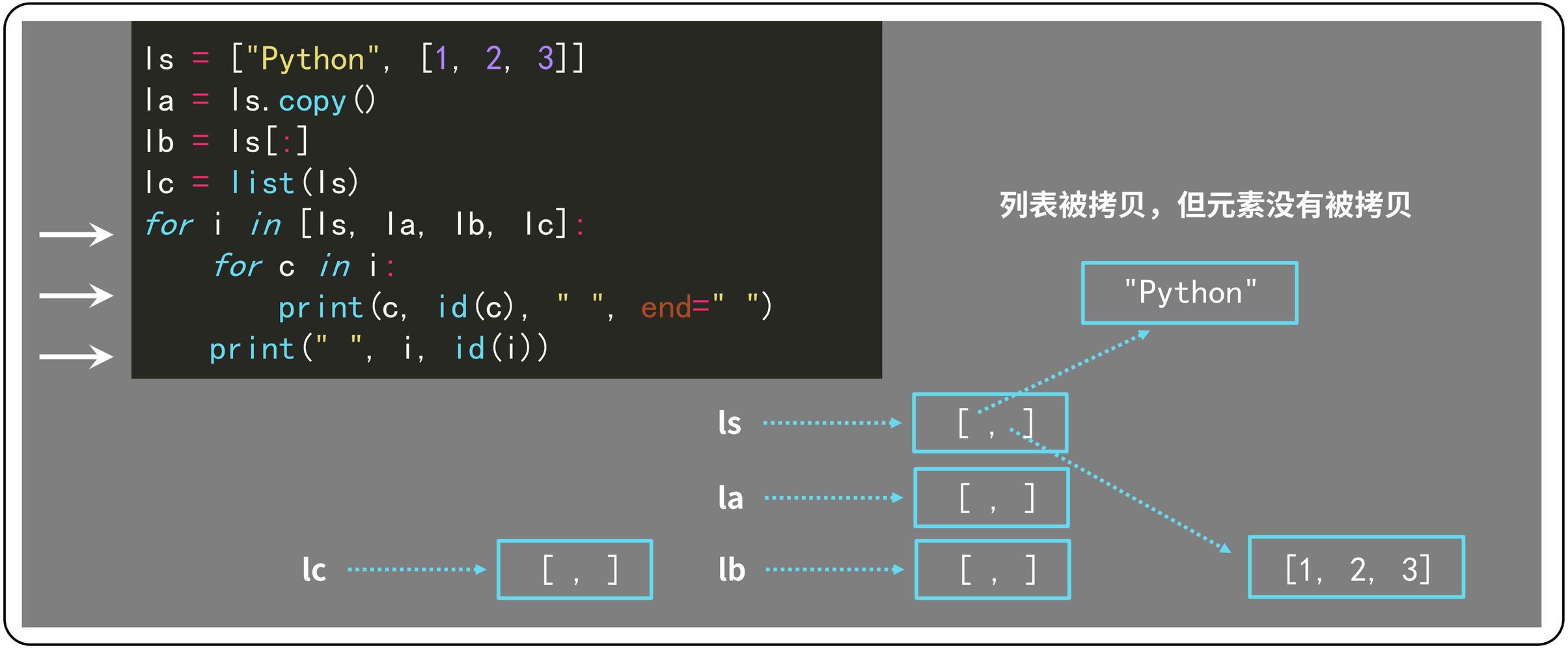
```
ls = ["Python", [1, 2, 3]]
la = ls.copy()
lb = ls[:]
lc = list(ls)
for i in [ls, la, lb, lc]:
    for c in i:
        print(c, id(c), " ", end=" ")
    print(" ", i, id(i))
```

列表被拷贝，但元素没有被拷贝

```
Python 91314848 [1, 2, 3] 91825056 ['Python', [1, 2, 3]] 91113632
Python 91314848 [1, 2, 3] 91825056 ['Python', [1, 2, 3]] 91824856
Python 91314848 [1, 2, 3] 91825056 ['Python', [1, 2, 3]] 91766952
Python 91314848 [1, 2, 3] 91825056 ['Python', [1, 2, 3]] 91825976
```

浅拷贝详解

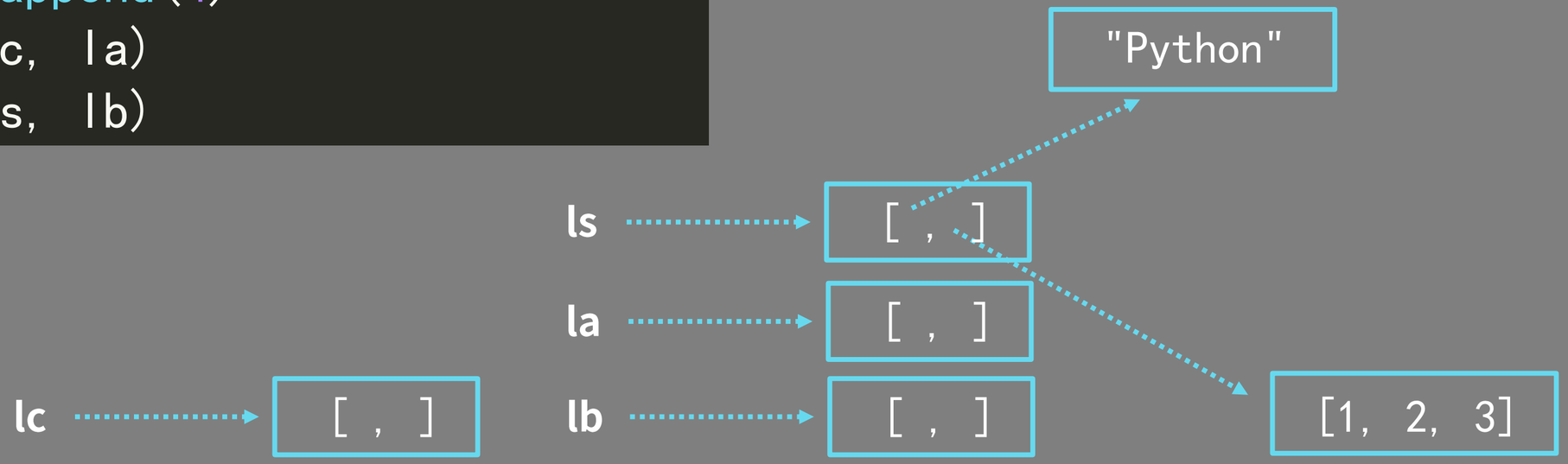
```python
ls = ["Python", [1, 2, 3]]
la = ls.copy()
lb = ls[:]
lc = list(ls)
for i in [ls, la, lb, lc]:
    for c in i:
        print(c, id(c), " ", end=" ")
    print(" ", i, id(i))
```



浅拷贝详解

```
ls = ["Python", [1, 2, 3]]
la = ls.copy()
lb = ls[:]
lc = list(ls)
lc[-1].append(4)
print(lc, la)
print(ls, lb)
```

```
['Python', [1, 2, 3, 4]] ['Python', [1, 2, 3, 4]]
['Python', [1, 2, 3, 4]] ['Python', [1, 2, 3, 4]]
```



深拷贝

完全拷贝对象内容

- 采用copy库的deepcopy()方法
- 迭代拷贝对象内各层次对象，完全新开辟内存建立对象
- 深拷贝仅针对可变类型，不可变类型无需创建新对象

Python应用基础 版权 by 嵩天

深拷贝详解

```
import copy
ls = ["Python", [1, 2, 3]]
lt = copy.deepcopy(ls)
for i in [ls, lt]:
    for c in i:
        print(c, id(c), " ", end=" ")
    print(" ", i, id(i))
```

列表内部各可变对象元素都被拷贝

```
Python 80173728 [1, 2, 3] 81038152 ['Python', [1, 2, 3]] 81038392
Python 80173728 [1, 2, 3] 81039112 ['Python', [1, 2, 3]] 81039152
```

小结：对象的拷贝

浅拷贝和深拷贝

- 浅拷贝：仅复制最顶层对象的拷贝方式，默认拷贝方式
- 深拷贝：迭代复制所有对象的拷贝方式，采用copy库的deepcopy()
- 一般深拷贝都与可变类型关联

Python应用基础 版权 by 嵩天

Python对象的引用

类的方法引用

Python123

类的方法引用

再看类的实例方法

- 定义方式：def <实例方法名>(self, <参数列表>)
- 实例方法名也是一种引用，即对方法本身的引用
- 当方法被引用时，方法(即函数)将产生一个对象：方法对象

Python应用基础 版权 by 嵩天

类的方法引用

```
class DemoClass:
    def __init__(self, name):
        self.name = name
    def lucky(self, salt = 0):
        s = 0
        for c in self.name:
            s += (ord(c) + id(salt)) % 100
        return s

dc1 = DemoClass("老李")
lucky = dc1.lucky
print(DemoClass.lucky(dc1, 10))
print(dc1.lucky(10))
print(lucky(10))
```

<对象名>.<方法>(方法参数)

等价于

<类名>.<方法名>(<对象名>, 方法参数)

类的方法引用

```
class DemoClass:
    def __init__(self, name):
        self.name = name
    def lucky(self, salt = 0):
        s = 0
        for c in self.name:
            s += (ord(c) + id(salt)) % 100
        return s

dc1 = DemoClass("老李")
lucky = dc1.lucky
print(DemoClass.lucky(dc1, 10))
print(dc1.lucky(10))
print(lucky(10))
```

<对象名>.<方法>

是一个对方法的引用

类的方法引用

```
class DemoClass:
    def __init__(self, name):
        self.name = name
    def lucky(self, salt = 0):
        s = 0
        for c in self.name:
            s += (ord(c) + id(salt)) % 100
        return s

dc1 = DemoClass("老李")
lucky = dc1.lucky
print(DemoClass.lucky(dc1, 10))
print(dc1.lucky(10))
print(lucky(10))
```

```
108
108
108
```

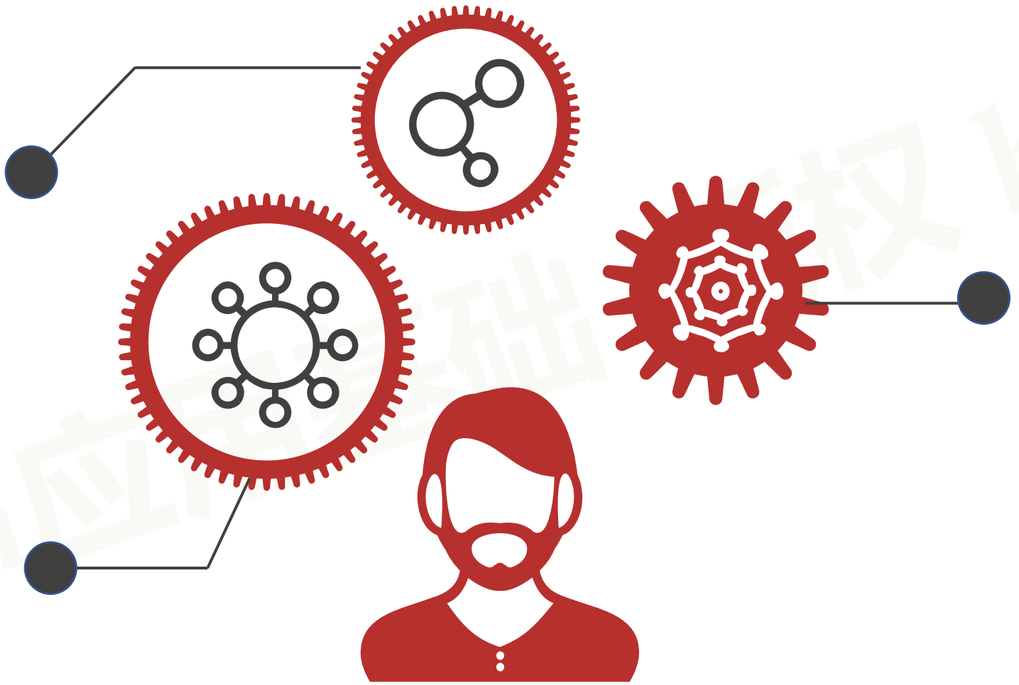
Python对象的引用

单元小结

单元小结

(2) 浅拷贝和深拷贝

一层拷贝、迭代拷贝



(3) 类的方法引用

方法亦引用

(1) 引用的理解

可变类型与不可变类型的引用

Python对象的引用

Python应用基础 版权 by 嵩天


Python 123
Thank you