

Lecture 11: Elements in Python

Xiao-Xin Li

Zhejiang University of Technology

Revision: 2023/05/27, 2023/05/30

1. Basic Concepts

- docstring vs. comment
- parameter (形参) vs. argument (实参)
- `class` vs. `object`
 - A class is a template showing what something can look like in memory, like a type with methods defining what can be done to this type.
 - An object is an instance in memory following this template, which we can refer to and operate on.
- `function` vs. `method`
 - A function is a piece of code that is called by name. It can be passed with data to operate on (i.e. the parameters) and can optionally return data (the return value). All data that is passed to a function is explicitly passed.
 - A method is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:
 - i. A method is implicitly passed by the object on which it was called.
 - ii. A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).
- `function definition` vs. `function call`
 - A **function definition** is code that describes what a function should do when it is executed (but it does not execute that code). It starts with the keyword `def` and a function **header**.
 - A **function call** is an expression that instructs Python to execute the function.

2. Calculation

```
1 print(2 * 5)
2 print(2 * 5.0)
3 print(11 % 3)
4 print(-3.1 // 1)
```

2. Boolean Expression

2.1 in

```
1 a_dict = {'key1': 'val1', 'key2': 'val2', 'key3': 'val3'}
2
3 print('key2' in a_dict)
4 print('val2' in a_dict)
```


2.2 等价判断: is, ==, id

参考: <https://www.runoob.com/note/33502>

- `==` 用于判断两个变量的值是否相等;
- `is` 用于判断两个变量引用对象是否为同一个(或者说,内存地址是否相同);
 - `a is b` 相当于 `id(a) == id(b)`, `id()` 能够获取对象的内存地址。

那么,除了使用 `id()`,如何判断两个对象的内存地址是否相同呢?

- 通常,如果使用赋值语句 `b = a`,则 `b` 和 `a` 必然有相同的内存地址;
- Python 中的一种比较特殊的情况是“不可变变量”或“不可变对象”(详见[这里](#)),如 `int`, `float`, `string`, `tuple`, 它们都是不可变对象。不可变对象的值和内存地址是绑定的,值相同则内存地址也相同,值不同则内存地址也不同,这是 Python 出于性能优化进行设置的结果。
- 可变变量的内容是可以改变的,如: `list`、`dict`、`set`、自定义的 `class`, 如果 `b` 和 `a` 是内容相同的两个变量,则 `b == a` 为 `True` (但是自定义的 `class` 除外,可以通过自定义魔法函数 `def __eq__(self, other)` 来修改 `==` 的含义)。

 为了加深理解上面的结论,可以进行测试。注意,如果是在 Python 控制台下,以“交互式”的方式进行测试,所得到的结果可能与通过脚本进行执行所得到的结果不同,但须以后者为准。

2.2.1 不可变变量

```
1 a = 10
2 b = a
3 print('b is a = ', b is a, '; b == a =', b == a)
4 print('id(a) =', id(a))
5 print('id(b) =', id(b))
6
7 a = 10
8 b = 10
9 print('b is a = ', b is a, '; b == a =', b == a)
10 print('id(a) =', id(a))
11 print('id(b) =', id(b))
12
13 # 下面代码，以交互式方式在 Python 控制台下运行，与把它们当做“脚本”执行，其结果是不同的：
14 a = 10.0
15 b = 10.0
16 print('b is a = ', b is a, '; b == a =', b == a)
17 print('id(a) =', id(a))
18 print('id(b) =', id(b))
19
20 a = 1111
21 b = 1111
22 print('b is a = ', b is a, '; b == a =', b == a)
```

```
1 a = 'Hi!'
2 b = 'Hi!'
3
4 print()
5 print('b is a =', b is a, '; b == a =', b == a)
6 print('id(a) =', id(a))
7 print('id(b) =', id(b))
```

```
1 a = (1, 2, 3)
2 b = (1, 2, 3)
3 print('b is a = ', b is a, '; b == a =', b == a)
4 print('id(a) =', id(a))
5 print('id(b) =', id(b))
```

2.2.1 可变变量

```

1 a = [1, 2, 3]
2
3 b = a
4 # b 和 a 的内存地址相同, b is a 和 b == a 的结果都为 True
5 print('b is a =', b is a, '; b == a =', b == a)
6
7 c = a[:]
8 # c 和 a 的内存地址不同, 但内容相同, c is a 为 False, c == a 为 True
9 print('b is a =', c is a, '; c == a =', c == a)
10

```

```

1 a = dict(k1=1, k2=2, k3=3)
2 b = a
3 c = {'k1': 1, 'k2': 2, 'k3': 3}
4 d = a.copy()
5
6 print('b is a =', b is a, '; b == a =', b == a)
7
8 # c/d 和 a 的内存地址不同, 但内容相同, c/d is a 为 False, c/d == a 为 True
9 print('c is a =', c is a, '; c == a =', c == a)
10 print('d is a =', d is a, '; d == a =', d == a)
11

```

```

1 a = {1, 2, 3}
2 b = {1, 2, 3}
3 print('b is a =', b is a, '; b == a =', b == a)
4 print('id(a) =', id(a))
5 print('id(b) =', id(b))

```

2.2.3 自定义的类以及深拷贝和浅拷贝

```

1 class A:
2     pass
3
4
5 def test_is():
6     a = A()
7     b = A()
8     c = a # c is a, as c and a has the same id
9
10    print()

```

```
11     print('b is a =', b is a, b == a)
12     print('b is A =', b is A, b == A)
13     print('c is a =', c is a, c == a)
14     print('id(a) =', id(a))
15     print('id(A) =', id(A))
16     print('id(b) =', id(b))
17     print('id(c) =', id(c))
```

```
1  import copy
2  import pytest as pytest
3
4
5  class Item:
6      def __init__(self):
7          self.s = 0
8
9
10 class A:
11     def __init__(self):
12         self.x = 1
13         self.y = 2
14         self.z = Item()
15
16     def __eq__(self, other):
17         return self.x == other.x and self.y == other.y
18
19
20 class TestA:
21     a = None
22
23     @pytest.fixture
24     def set_a(self):
25         self.a = A()
26
27     def test_assign(self, set_a):
28         b = self.a
29         b.x = 0
30         assert b.x == self.a.x
31
32     def test_copy_1(self, set_a):
33         b = copy.copy(self.a)
34         b.x = 0
35         assert b.x != self.a.x
36
37     def test_copy_2(self, set_a):
```

```
38         b = copy.copy(self.a)
39         b.z.s = -1
40         assert b.z.s == self.a.z.s
41
42     def test_deep_copy(self, set_a):
43         b = copy.deepcopy(self.a)
44         b.z.s = -1
45         assert b.z.s != self.a.z.s
```

2.3 not, None

```
1 class A:
2     pass
3
4
5 def test_not():
6     print()
7     if not False:
8         print('not False is True')
9
10    if not None:
11        print('not None is True')
12
13    if not []:
14        print('type of []:', type([]))
15        print('not [] is True')
16
17    if not [0]:
18        print('not [0] is True')
19
20    if not ():
21        print('type of ():', type(()))
22        print('not () is True')
23
24    if not {}:
25        print('type of {}:', type({}))
26        print('not {} is True')
27
28    if not set():
29        print('not set() is True')
30
31    if not dict():
32        print('not dict() is True')
33
34    if not A():
```

```
35         print('not A() is True')
36     else:
37         print('not A() is False')
38
39     if not 0:
40         print('not 0 is True')
41
42     if not 1:
43         print('not 1 is True')
```

2.4 Lazy Operator

- `and`, `or` is lazy
- `&`, `|` is not lazy

```
1 class TestLazy:
2     def fun1(self):
3         print('fun1() is called')
4         return False
5
6     def fun2(self):
7         print('fun2() is called')
8         return False
9
10    def test(self):
11        print('')
12        a = self.fun1() and self.fun2()
13        b = self.fun1() & self.fun2()
```

3. List

关于copy和删除，Dict 与 List 只有细微的差别（如 Dict 中并没有 `remove` 方法），不再针对 Dict 进行专门的讨论。

```
1 class TestList:
2     a = []
3
4     @pytest.fixture
5     def set_a(self):
6         self.a = [1, 2, 3, 4, 5]
```

3.1 copy

How to create a copy of a list?

```
1 class TestListCopy(TestList):
2     def test_copy_by_assignment(self, set_a):
3         b = self.a
4
5         b[0] = -1
6         print()
7         print('a[0] =', self.a[0])
8         print('b[0] =', b[0])
9
10    def test_copy_by_loop_all(self, set_a):
11        b = [_ for _ in self.a]
12        b[0] = -1
13        print()
14        print('a[0] =', self.a[0])
15        print('b[0] =', b[0])
16
17    def test_copy_by_indexing_all(self, set_a):
18        b = self.a[:]
19        b[0] = -1
20        print()
21        print('a[0] =', self.a[0])
22        print('b[0] =', b[0])
23
24    def test_copy_by_method(self, set_a):
25        b = self.a.copy()
26        b[0] = -1
27        print()
28        print('a[0] =', self.a[0])
29        print('b[0] =', b[0])
```

3.2 remove: 删除 List 中元素的四种方法

```
1 # 1) 根据索引值删除元素: del
2 # del 是 Python 中的关键字, 专门用来执行删除操作,
3 # 它不仅删除整个列表, 还可以删除列表中的某些元素
4 del list_name[index]
5 del list_name[start : end]
6
7 # 2) 根据索引值删除元素: pop
8 list_name.pop(index)
```



```
9
10 # 3) 根据元素值进行删除: remove
11 list_name.remove(list_name[index])
12
13 # 4) 删除列表所有元素: clear
14 list_name.clear()
```

```
1 class TestListRemove(TestList):
2     def test_clear(self, set_a):
3         # print('test clear a')
4         print()
5         self.a.clear()
6         print(self.a)
7
8     def test_del(self, set_a):
9         # print('test clear a')
10        print()
11        del self.a
12        print(self.a)
13
14    def test_pop(self, set_a):
15        # print('test a.pop(0)')
16        print()
17        self.a.pop(0)
18        print(self.a)
19
20    def test_remove(self, set_a):
21        # print('test a.remove(0)')
22        print()
23        self.a.remove(0)
24        print(self.a)
25
26    def test_del_items(self, set_a):
27        # print('test a.remove(0)')
28        print()
29        del self.a[2:4]
30        print(self.a)
```