id(object)

- 返回 object 的唯一标识符（内存地址）

赋值

- Python中, 变量定义和赋值必须是同时进行的，比如当执行程序 a = 999 是先在内存中开辟一块空间来存放数据999，然后定义变量名a来指向这块空间的内存地址，方便我们使用变量值，所以变量名和值其实是一种引用的关联关系
- 严格来说，变量名本身没有类型，通常我们所说的变量类型指的是值的数据类型

```
var1 = 999
print(type(var1))  # <class 'int'>
var1 = "999"
print(type(var1))  # <class 'str'>
```

- 通过内置函数id()获取变量地址时，实际是返回了值的地址

```
var1 = 256
print(id(256))
print(id(var1))  # 同上

var2 = -5
print(id(-5))
```

```
print(id(var2))  # 同上

var3 = 257
print(id(257))
print(id(var3))  # 不一样

var4 = -6
print(id(-6))
print(id(var4))  # 不一样
```

- 为了节省内存空间，解释器做了优化，如果直接执行程序文件，相同值的变量会引用自同一个对象，而在交互式窗口运行则可以看到区别
- 小整数对象池：Python 为了优化速度，避免整数频繁申请和销毁内存空间，把范围在 [-5, 256] 之间的数字放在提前建立好的小整数对象池里面，不会被垃圾回收，在这范围内的数值如果相等，地址也就相同，因为使用的都是同一个对象

```
a = 257
b = 257
print(id(a))
print(id(b))

a = 256
b = 256
print(id(a))
print(id(b))

a = -5
b = -5
print(id(a))
print(id(b))
```

```
a = -6
b = -6
print(id(a))
print(id(b))
```

- 一个值可以被多个变量名引用，当变量值的引用计数（即值被关联的次数）为0时，可认定为该值不可用，那么Python的垃圾回收机制会收回所占用的内存空间

```
a = 999  # a = [1, 2, 3]
b = 999  # b = [1, 2, 3]
c = a

# 变量a、c都指向第一个999，引用计数为2
print(id(a))
print(id(c))
# 变量b指向第二个999，引用计数为1
print(id(b))

del a  # 解除a的引用，第一个999的引用计数为-1，为1
print(id(c))  # c还是指向第一个999

c = b  # 解除c的引用，重新指向第二个999，则第一个999的引用计数再-1，为0，被回收
# b、c都指向第二个999，引用计数+1，为2
print(id(b))
print(id(c))
```

- 当一个可变类型的数据被多个变量名引用时，如果对该原数据进行修改，那么它所有引用都会改变

```
a = [1, 2, 3]
b = [1, 2, 3]
```

```
c = a

# 变量a、c都指向第一个[1, 2, 3], 引用计数为2
print(id(a))
print(id(c))

# 变量b指向第二个[1, 2, 3], 引用计数为1
print(id(b))

a.append(4)
print(a)
print(c)
print(b)
```

浅拷贝 & 深拷贝

Python 中的赋值语句不复制对象，只是建立引用关联，对于可变数据，有时我们不希望直接对它进行修改，因为这可能会导致一些意外的情况发生，所以我们就可以把它copy一份，对它的副本进行操作

这种copy操作又分为浅层copy和深层copy，我们所学的 list.copy()、dict.copy()、set.copy() 和切片都属于浅层copy。在copy模块中，提供了通用的浅层和深层copy操作

浅拷贝

- 如果浅拷贝对象是不可变数据类型（复合类型数据只要最外侧的类型不可变即可），那么和赋值语句等效（没有拷贝的意义）

```
import copy


tup1 = (991, "abc")
tup2 = copy.copy(tup1)  # 浅拷贝
# tup2 = tup1
print(tup1)
print(tup2)
print(id(tup1))
print(id(tup2))


tup3 = (991, "abc", [])
tup4 = copy.copy(tup3)  # 浅拷贝
print(tup3)
print(tup4)
print(id(tup3))
print(id(tup4))
print(id(tup3[-1]))
print(id(tup4[-1]))
```

- 如果浅拷贝对象是可变数据类型（复合类型数据只要最外侧的类型可变即可），那么浅拷贝会把该对象复制一份，但是该对象中的其他所有元素（包括复合类型数据中的元素）仍为引用关系

```
import copy
```

```
lis1 = [991, "abc", (9, 993), [994, 995], [888,
887], {"name": "Tom"}, (996, [997, 998]), (888,
(886, 886))]
lis2 = copy.copy(lis1)  # 浅拷贝
print(id(lis1))
print(id(lis2))

lis1.append(9)
print(lis1)
print(lis2)



print("索引0对应的id", id(lis1[0]))
print("索引0对应的id", id(lis2[0]))
print("索引1对应的id", id(lis1[1]))
print("索引1对应的id", id(lis2[1]))
print("索引2对应的id", id(lis1[2]))
print("索引2对应的id", id(lis2[2]))
print("索引3对应的id", id(lis1[3]))
print("索引3对应的id", id(lis2[3]))
print("索引4对应的id", id(lis1[4]))
print("索引4对应的id", id(lis2[4]))
print("索引5对应的id", id(lis1[5]))
print("索引5对应的id", id(lis2[5]))
print("索引6对应的id", id(lis1[6]))
print("索引6对应的id", id(lis2[6]))
print("索引7对应的id", id(lis1[7]))
print("索引7对应的id", id(lis2[7]))
print(id(lis1[6][-1]))
print(id(lis2[6][-1]))
print(id(lis1[7][-1]))
print(id(lis2[7][-1]))

lis1[3] = [994]
```

```
print(lis1)
print(lis2)

lis1[3].append(999)
print(lis1)
print(lis2)

lis1[4].append(886)
print(lis1)
print(lis2)

lis1[5].update(age=18)
print(lis1)
print(lis2)

lis1[6][-1].pop()
print(lis1)
print(lis2)
```

深拷贝

- 如果深拷贝对象是不可变数据类型（复合类型数据还需确保其中的所有元素不可变），那么和赋值语句等效（没有拷贝的意义）
- 如果深拷贝对象是可变数据类型或者其中的复合类型数据中有可变的元素，那么深拷贝会把该对象复制一份
- 对于其中的元素分别递归的去适用前两点规则

```
import copy


tup1 = (991, "abc")
```

```
tup2 = copy.deepcopy(tup1)  # 深拷贝
# tup2 = tup1
print(tup1)
print(tup2)
print(id(tup1))
print(id(tup2))


tup3 = (991, "abc", [])
tup4 = copy.deepcopy(tup3)  # 深拷贝
print(tup3)
print(tup4)
print(id(tup3))
print(id(tup4))
print(id(tup3[0]))
print(id(tup4[0]))
print(id(tup3[1]))
print(id(tup4[1]))
print(id(tup3[-1]))
print(id(tup4[-1]))
```

```
import copy


lis1 = [991, "abc", (9, 993), [994, 995], [888,
887], {"name": "Tom"}, (996, [997, 998]), (888,
(886, 886))]
lis2 = copy.deepcopy(lis1)  # 深拷贝

print(id(lis1))
print(id(lis2))

lis1.append(9)
print(lis1)
```

```python
print(lis2)


print("索引0对应的id", id(lis1[0]))
print("索引0对应的id", id(lis2[0]))
print("索引1对应的id", id(lis1[1]))
print("索引1对应的id", id(lis2[1]))
print("索引2对应的id", id(lis1[2]))
print("索引2对应的id", id(lis2[2]))
print("索引3对应的id", id(lis1[3]))
print("索引3对应的id", id(lis2[3]))
print("索引4对应的id", id(lis1[4]))
print("索引4对应的id", id(lis2[4]))
print("索引5对应的id", id(lis1[5]))
print("索引5对应的id", id(lis2[5]))
print("索引6对应的id", id(lis1[6]))
print("索引6对应的id", id(lis2[6]))
print("索引7对应的id", id(lis1[7]))
print("索引7对应的id", id(lis2[7]))
print(id(lis1[6][-1]))
print(id(lis2[6][-1]))
print(id(lis1[7][-1]))
print(id(lis2[7][-1]))

lis1[3].append(999)
print(lis1)
print(lis2)

lis1[5].update(age=18)
print(lis1)
print(lis2)

lis1[6][-1].pop()
print(lis1)
```

```
print(lis2)
```