CTF密码学中遇见的python 类知识

原创

M3ng@L 于 2022-04-10 19:55:32 发布 435 收藏

分类专栏： 密码学知识总结 文章标签： Crypto

本文链接：https://blog.csdn.net/qq_51999772/article/details/124084462

CTF密码学中遇见的python 类知识

keywords: 魔法方法

`PEP 8` 代码规范：类定义前后需要两行空行

Definition

定义类

```
class class_name(object):
    pass


real_object = class_name()
```

在类中定义的函数，称为**方法**

调用对象的方法

```
object.func()
```

类外部添加对象属性

```
object.attr = ...
```

类内部操作属性

通过`self`操作属性

`self`作为类中方法的第一个形参，在**通过对象调用**方法二栋时候，不需要手动的传递实参值（`self`只是一个形参的名字，那么可以写作其他字符串，但是一般不修改）

`python`解释器自动将使用该方法的对象传递给`self`，也即是`self`代表对象本身

魔法方法

在满足某个特定条件的情况下，会**自动调用**的函数

- `__init__`方法

调用时机：在**创建对象之后**会立即调用

一般应用：用来给对象添加属性，给对象属性一个初始值；

```
class class_name(object):
    def __init__(self):
        pass


class_name
```

如果`__init__`方法中，有除了`self`之外的形参，那么在创建对象时，需要有手动传参

- `__str__`方法

调用时机：`print(对象)`会自动调用`__str__`方法，打印输出结果是`__str__`方法的返回值；

`str(object)`类型转换，将自定义对象转换为字符串的时候，会自动调用

应用：在打印对象时，同时打印一些对象的属性

PS：该方法必须返回一**个字符串**，且只有`self`一个参数

- `__del__`方法（析构函数）

调用时机：对象在内存中被销毁删除时会自动调用`__del__`方法；

销毁情况：程序代码运行结束，在程序运行过程中，创建的所有对象和变量都会被删除销毁；是由`del 变量`，将这个对象的引用计数变为0；以上均会自动调用`__del__`方法

> 引用计数：`python`内存管理的一种机制，是指一块内存，有多少个变量在引用
>
> - 当一个变量，引用一块内存时，引用计数加1
> - 当删除一个变量或者这个变量不再引用这块内存，引用计数-1
> - 当内存的引用计数变为0时，这块内存被删除，内存的数据被销毁

Example

```
class PRNG:
    def __init__(self, seed = 0):
        self.__register = [0] * 624
        self.__state = 0
        self.__register[0] = seed
        for i in range(1, 624):
            prev = self.__register[i - 1]
            temp = 0x6c078965 * (prev ^ (prev >> 30)) + i
            self.__register[i] = temp & 0xffffffff

    def __call__(self):
        if self.__state == 0:
            for i in range(624):
                y = (self.__register[i] & 0x80000000) + (self.__register[(i + 1) % 624] & 0x7fffffff)
                self.__register[i] = self.__register[(i + 397) % 624] ^ (y >> 1)
                if y % 2:
                    self.__register[i] ^= 0x9908b0df
        y = self.__register[self.__state]
        y = y ^ (y >> 11)
        y = y ^ (y << 7) & 0x9d2c5680
        y = y ^ (y << 15) & 0xefc60000
        y = y ^ (y >> 18)
        self.__state = (self.__state + 1) % 624
        return y

    def load_register(self,register):
        self.__register = register
```

以上实例是一个`PRNG`生成器脚本，也就是伪随机数生成器脚本，这里使用的是`MT19937`算法

实际上`python3_random`内置的伪随机数生成算法就是`MT19937`

首先`__init__`方法在创建对象时被调用

```
    def __init__(self, seed = 0):
        self.__register = [0] * 624
        self.__state = 0
        self.__register[0] = seed
        for i in range(1, 624):
            prev = self.__register[i - 1]
            temp = 0x6c078965 * (prev ^ (prev >> 30)) + i
            self.__register[i] = temp & 0xffffffff
```

首先以`seed`为基础生成了由`624`个数字组成的列表，也就是`state`

```
    def __call__(self):
        if self.__state == 0:
            for i in range(624):
                y = (self.__register[i] & 0x80000000) + (self.__register[(i + 1) % 624] & 0x7fffffff)
                self.__register[i] = self.__register[(i + 397) % 624] ^ (y >> 1)
                if y % 2:
                    self.__register[i] ^= 0x9908b0df
        y = self.__register[self.__state]
        y = y ^ (y >> 11)
        y = y ^ (y << 7) & 0x9d2c5680
        y = y ^ (y << 15) & 0xefc60000
        y = y ^ (y >> 18)
        self.__state = (self.__state + 1) % 624
        return y
```

然后`__call__`方法的作用是当新建一个对象`object`之后，使用`object()`即会调用`__call__`方法里面的东西

连贯起来使用即是

```
mt = PRNG()
random_number = mt()
```

最后在类中定义了一个函数，相当于可以直接向对象`object`的属性进行赋值，而该属性即是`state`

```
    def load_register(self,register):
        self.__register = register
```