实验报告：生成随机数

一、基本思路：

生成(0,1)上均匀分布的随机数，然后利用此随机数生成其他常用分布的随机数

二、具体实现

每一种分布的随机数都抽象成一个类，类方法 `__init__` 是一些初始化操作，就是存一些参数等， `compute(self, k)` 是计算分布函数$f(k)$，`generate(self)` 就是随机数生成器。因为方法的类似性，我只选择三个说明，其他的实现和这三个类似。

1.（0，1）上的均匀分布

利用线性同余法：$x_i = (ax_{i-1} + c)\%m$

```
# (0,1)上的均匀分布
class UniformDistribution:
    def __init__(self, x0, a=630360016, c=7, m=2 ** 31 - 1):
        self.a = a
        self.m = m
        self.c = c
        self.x0 = x0

    # 随机数生成器
    def generate(self):
        tmp = (self.a * self.x0 + self.c) % self.m
        self.x0 = tmp
        return float(tmp) / self.m
```

2. 泊松分布

根据定理4.10，我们需要计算泊松分布的分布函数。

```
# 泊松分布
class PoissonDistribution:
    def __init__(self, randa):
        self.randa = randa
        self.unirnd = UniformDistribution(1)

    def compute(self, k):
        return (math.e ** (-self.randa)) * (self.randa ** k) / math.factorial(k)

    def generate(self):
        x = self.unirnd.generate()
        sum = 0
        k = 0
        while True:
            sum += self.compute(k)
            if x <= sum:
                return k
            k += 1
```

其他的离散型随机变量做法类似，就不具体说明了。

3. 正太分布

利用Box-Muller方法，先生成标准正态分布，然后利用标准正态分布生成一般正态分布。

```
# 正态分布
class NormalDistribution:
    def __init__(self, miu, kexi2):
        self.miu = miu
        self.kexi2 = kexi2
        self.unirnd1 = UniformDistribution(1)
        self.unirnd2 = UniformDistribution(2)

    # 这个是使用box-muller方法生成标准正太分布
    def generateN(self):
        x1 = self.unirnd1.generate()
        x2 = self.unirnd2.generate()
        return (-2 * math.log(x1)) ** 0.5 * math.cos(2 * math.pi * x2)

    def generate(self):
        x = self.generateN()
        return self.kexi2 ** 0.5 * x + self.miu
```

三、验证正确性

我是通过验证生成的伪随机数的期望和方差与真正的分布的期望与方差对比，来验证方法的有效性。对于每一个分布，我分别写了验证的函数，只需运行 `MyRandom.py` 然后输入相关的参数，就可以看到相应的结果了。

例如：

```
def validationExp():
    print("验证指数分布")
    tmp1, tmp2 = 0, 0
    p = eval(input("请输入指数分布的参数："))
    poi = ExponentsDistribution(p)
    # 真正的均值、方差
    avg, var = 1.0 / p, 1.0 / (p ** 2)
    for i in range(100000):
        tmp = poi.generate()
        print(tmp)
        tmp1 += tmp
        tmp2 += (tmp - avg) ** 2
    print("----------------")
    print("伪平均值：%f，伪方差：%f" % (tmp1 / 100000, tmp2 / 100000))
    print("真平均值：%f，真方差：%f" % (avg, var))
```