Singleton Distribution

In this lesson, we will explore the singleton distribution and Bernoulli distribution.

WE'LL COVER THE FOLLOWING

- Introduction to Trivial Distribution
- Bernoulli Distribution
- Implementation

In the previous lesson, we implemented our first discrete distribution, the *"choose an integer between min and max"* distribution. In this chapter, we will explore easier discrete distributions.

Introduction to Trivial Distribution

The easiest discrete distribution of all is the **trivial distribution**. Let's have a look at its code:

```
public sealed class Singleton<T> : IDiscreteDistribution<T>
{
  private readonly T t;
  public static Singleton<T> Distribution(T t) => new Singleton<T>(t);
  private Singleton(T t) => this.t = t;
  public T Sample() => t;
  public IEnumerable<T> Support()
  {
    yield return t;
  }

  public int Weight(T t) => EqualityComparer<T>.Default.Equals(this.t, t)
 ? 1 : 0;
  public override string ToString() => $"Singleton[{t}]";
}
```

That is the probability distribution where 100% of the time it returns the same value. You'll also sometimes see distributions like this called the "Dirac delta" or the "Kronecker delta", but we're going to call this the *singleton distribution* and be done with it.

> This is the first distribution we've seen that does not necessarily have some sort of number as its output. You might be thinking that maybe our distribution type should never have been generic in the first place if the only distributions we are going to come up with are either numeric-valued or trivial. In the next lessons, we'll see how we can naturally use more complex types in probability distributions.

Bernoulli Distribution

Let's look at another *integer-valued* distribution. The **Bernoulli distribution** considers the question "what if we flipped a possibly-unfair coin, and scored 1 for a head and 0 for a tail?" The parameter to the distribution is traditionally the probability of heads in our coin, between 0.0 and 1.0.

```
public sealed class Bernoulli : IDiscreteDistribution<int>
{
  public static IDiscreteDistribution<int> Distribution(int zero, int one)
  {
    if (zero < 0 || one < 0 || zero == 0 && one == 0)
      throw new ArgumentException();
    if (zero == 0) return Singleton<int>.Distribution(1);
    if (one == 0) return Singleton<int>.Distribution(0);
    return new Bernoulli(zero, one);
  }

  public int Zero { get; }
  public int One { get; }
  private Bernoulli(int zero, int one)
  {
    this.Zero = zero;
    this.One = one;
  }

  public int Sample() =>
    (SCU.Distribution.Sample() <=
      ((double)Zero) / (Zero + One)) ? 0 : 1;
```

```
  public IEnumerable<int> Support() => Enumerable.Range(0, 2);
  public int Weight(int x) => x == 0 ? Zero : x == 1 ? One : 0;
  public override string ToString() => $"Bernoulli[{this.Zero}, {this.One
}]";
}
```

And sure enough, if we graph it out:

```
Bernoulli.Distribution(1, 3).Histogram()
```

We get what you'd expect: three times as many heads as tails:

```
0|************
1|************************************
```

Implementation

The following code snippet sums up this lesson:

Program.cs

Bernoulli.cs

BetterRandom.cs

Distribution.cs

Episode05.cs

Extensions.cs

IDiscreteDistribution.cs

IDistribution.cs

Pseudorandom.cs

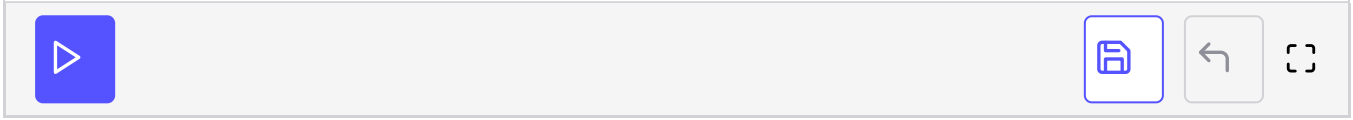
Singleton.cs

StandardCont.cs

```
using System;
namespace Probability
{
  static class Episode05
  {
    public static void DoIt()
```

```
    {
      Console.WriteLine("Episode 05");
      Console.WriteLine("Bernoulli 75% chance of 1");

      Console.WriteLine(Bernoulli.Distribution(1, 3).Histogram());
    }
  }
}
```



In the next lesson, we'll stop making new implementations of classic probability distributions and look at how to extend the ones we've got.