- **Tikhon Guest Talk:**
  - How to implement software for applied math
  - Programming patterns don't necessarily show up in data science applications
  - **Abstractions in the code:**
    - Abstractions in distributions module:
      - Easiest way to do distributions is to implement functions that you call to get a single, random sample
        - ↳ But this does not give you a full distribution
      - Functions can be passed around to other functions
        - ↳ When functions return a value and you want to pass around the function, not value, use wrapper function:

```
def new_func: return old_func(arg)
```

          - ↳ This is the roll of lambda
      - Abstract class is a class that has non-implemented
        - ↳ Serves as an interface for other classes

```
from abc import ABC, abstractmethod
```

          - ↳ Abstract classes inherit ABC
      - Type annotations is a useful way to tell developers what types of variables are being passed around
      - Types of distributions inherit from the distribution abstract class.
      - EMacs has a code interpreter and enables split screen
      - help(function) gives documentation for function from terminal

- A concrete class cannot have abstract class
- Concrete classes inheriting from an abstract class can reimplement concrete methods in the abstract class
- dataclass gives better string representation of classes
- Always use dataclass
  - ↳ def __repr__(self): Specifies string representation manually. This, with other things, are represented in dataclass

* Vast majority of classes in practice are dataclasses
- Cannot have a mutable value as default argument in data class
  - # Rule of thumb: Never use mutable defaults in functions, since if you run the function with mutable default values, it can change the default value.
    - ↳ Very difficult to debug

- Mypy can check consistency of type annotations

- Abstractions:

  Express distributions which we do not know exactly how to execute
  - ↳ Use Callable[(args), return_type]

  In real world, we do not have analytical behavior or known distributions

  How many samples you take when Sampling mean of unknown distribution depends on the distribution.
  - ↳ Should be able to be specified by user - Design Spec
    - ↳ Hard to know how n relates to precision
    - ↳ Ideal interface shows convergence of mean and stops when mean is good enough.
      - ↳ Use generators (iterators) to do this.

↳ This allows the user to inspect process when calculating mean

- On-line Mean Calculation for Streaming Data

  ↳ Spark / Skala useful for Streaming data

```
while True:
    n += 1
    mean *= (n-1) / n        ← Downweight current mean
    mean += self.sample() / n
    yield mean
                ↑ returns a float
```

- Use sliding window to check convergence

- "_" skips everything but last element in python

- islice gives iterator a terminating point

- Iterators are composable: You can have multiple stopping conditions and combine them

- Use islice to cap all iterators with a max number of iterations

  ↳ Multiple ways to stop iterators

  ↳ Makes it simpler to not have to account for every possible usecase.

  ↳ Adaptable for new stopping logic

One iterator inside another

- In ML, lots of investment in visualization

- <u>Iterator vs. Iterable</u>:

  - Iterator is anything implements 'next' method

    ↳ Next mobilizes iterators

    ↳ Next simply accesses values in an iterable, while tracking indices.

      ↳ You can only go through list once

- But you can produce iterators from iterables as much as you want to loop through values multiples times

- Iterators are iterables, where iter() returns the original iterator

- Iterator is a special type of iterable

  Sequence is a special type of iterable with ordered looping

  iterator implements next

  iterable implements iter to create a fresh instance of the iterator