No Quiz Today
Milestone 4 Feedback

Did you run into any major obstacles with the assignment?

A. My group had problems identifying entity types.
B. My group had problems decomposing large tables.
C. A and B.
D. My group did not face any major obstacles.
Beam/Dataflow Setup

Beam/Dataflow Setup Outcome

Did you successfully complete your setup?

A. Yes, the Wordcount jobs ran without errors.
B. No, I got stuck during the setup and need help.
C. I’m still setting things up and need more time to finish.
Dataflow Concepts

- A system for processing arbitrary computations on large amounts of data
- Can process batch data and streaming data using the same code
- Uses Apache Beam, an open-source programming model
- Designed to be very scalable, millions of QPS
Apache Beam Concepts

- A model for describing data and data processing operations:
  - **Pipeline**: a data processing task from start to finish
  - **PCollection**: a collection of data elements
  - **Transform**: a data transformation operation

- **SDKs** for Java, Python and Go
- Executed in the cloud on Dataflow, Spark, Flink, etc.
- Executed locally with Direct Runner for dev/testing
Beam Pipeline

- **Pipeline** = A directed acyclic graph where the nodes are the Transforms and the edges are the PCollections

- **General Structure of a Pipeline:**
  - Reads one or more data sources as input PCollections
  - Applies one or more Transforms on the PCollections
  - Outputs resulting PCollection as one or more data sinks

- Executed as a single unit
- Run in batch or streaming mode
PCollection

- $\text{PCollection} = \text{A collection of data elements}$
- Elements can be of any type (String, Int, Array, etc.)
- $\text{PCollections are distributed across machines}$
- $\text{PCollections are immutable}$
- Created from a data source or a $\text{Transform}$
- Written to a data sink or passed to another $\text{Transform}$
Transform Types

- Element-wise:
  - maps 1 input to (1, 0, many) outputs
  - **Examples:** ParDo, Map, FlatMap

- Aggregation:
  - reduces many inputs to (1, fewer) outputs
  - **Examples:** GroupByKey, CoGroupByKey

- Composite: combines element-wise and aggregation
  - GroupByKey -> ParDo
Transform Properties

- Serializable
- Parallelizable
- Idempotent
ParDo

- ParDo = “Parallel Do”
- Maps 1 input to (1, 0, many) outputs
- Takes as input a PCollection
- Applies the user-defined ParDo to the input PCollection
- Outputs results as a new PCollection
- Typical usage: filtering, formatting, extracting parts of data, performing computations on data elements
import apache_beam as beam
from apache_beam.io import ReadFromText
from apache_beam.io import WriteToText

# DoFn to perform on each element in the input PCollection.
class ComputeWordLengthFn(beam.DoFn):
    def process(self, element):
        words = element.strip().split(' ')
        result_list = []
        for word in words:
            result_list.append((word, len(word)))
        return result_list

# Create a Pipeline using a local runner for execution.
with beam.Pipeline('DirectRunner') as p:

    # create a PCollection from the file contents.
in_pcoll = p | 'Read' >> ReadFromText('input.txt')

    # apply a ParDo to the PCollection
out_pcoll = in_pcoll | beam.ParDo(ComputeWordLengthFn())

    # write PCollection to a file
out_pcoll | 'Write' >> WriteToText('output.txt')
import apache_beam as beam
from apache_beam.io import ReadFromText
from apache_beam.io import WriteToText

# DoFn to perform on each element in the input PCollection.
class ComputeWordLengthFn(beam.DoFn):
    def process(self, element):
        words = element.strip().split(' ')
        result_list = []
        for word in words:
            result_list.append((len(word), word))
        return result_list

# Create a Pipeline using a local runner for execution.
with beam.Pipeline('DirectRunner') as p:

    # create a PCollection from the file contents.
in_pcoll = p | 'Read' >> ReadFromText('input.txt')

    # apply a ParDo to the PCollection
word_pcoll = in_pcoll | 'ParDo' >> beam.ParDo(ComputeWordLengthFn())

    # apply GroupByKey to the PCollection
out_pcoll = word_pcoll | 'GroupByKey' >> beam.GroupByKey()

    # write PCollection to a file
out_pcoll | 'Write' >> WriteToText('output.txt')
Hands-on Exercises

git clone https://github.com/cs327e-spring2019/snippets.git
**Best Practices:**

1. Know basic UNIX commands (e.g. `ls`, `cp`, `mv`, `rm`, etc.)
2. Start with some initial working code. See [snippets repo](#) for samples.
3. Test and debug **each** new line of code.
4. Write temporary and final PCollections to log files.
5. Test and debug **end-to-end** pipeline locally before running on Dataflow.
6. If you get stuck, go to OHs. If you can’t make OHs, make an appointment with one of the TAs.
7. Start assignments **early**. The Beam Python documentation is sparse and learning Beam requires *patience, perseverance, and experimentation*. 
Milestone 5

1) Requirements: assignment sheet

2) Design issues: sign-up sheet

3) Beam setup problems: sign-up sheet