LECTURE 7

NOTEBOOKS

MCS 275 Spring 2023

Emily Dumas

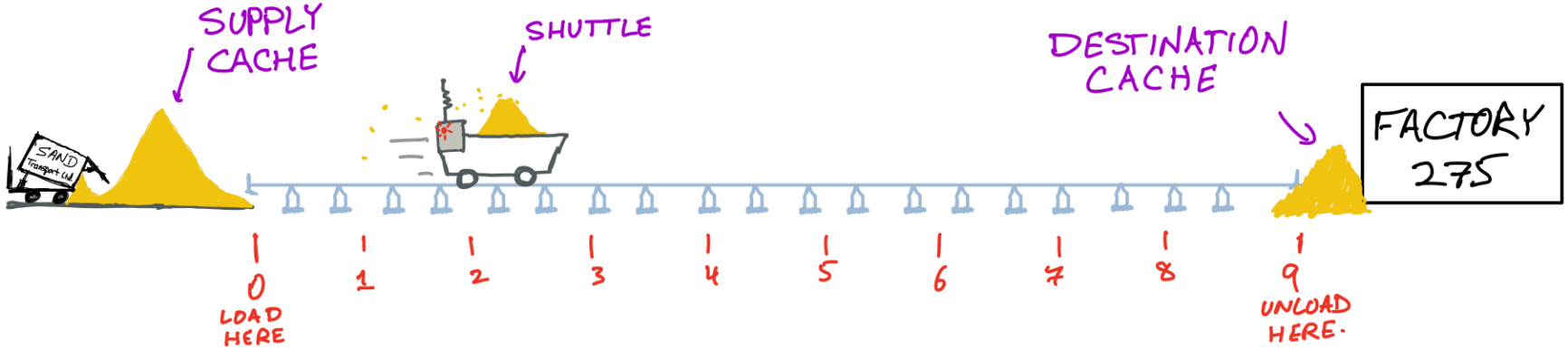
LECTURE 7: NOTEBOOKS

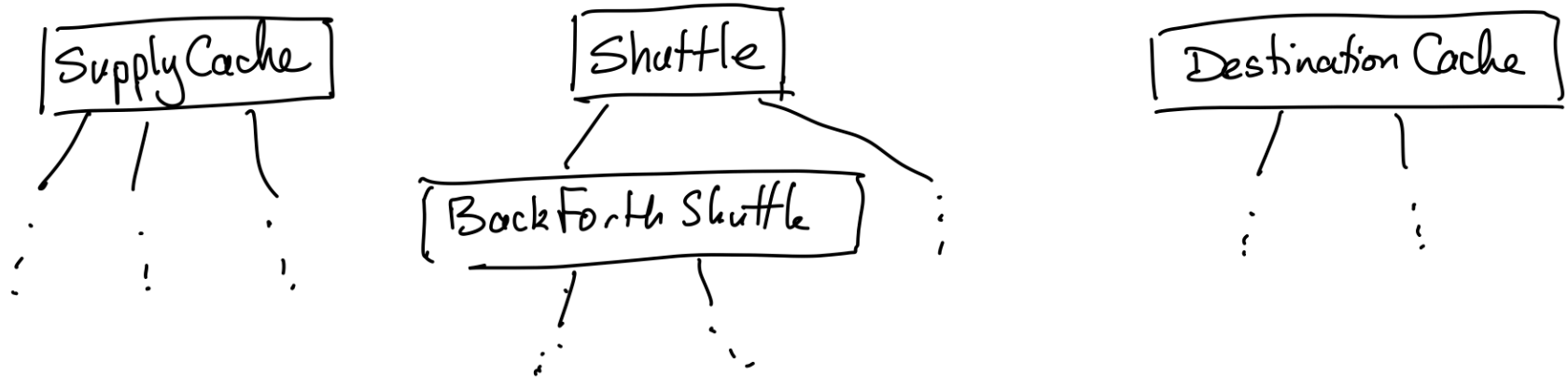
Reminders and announcements:

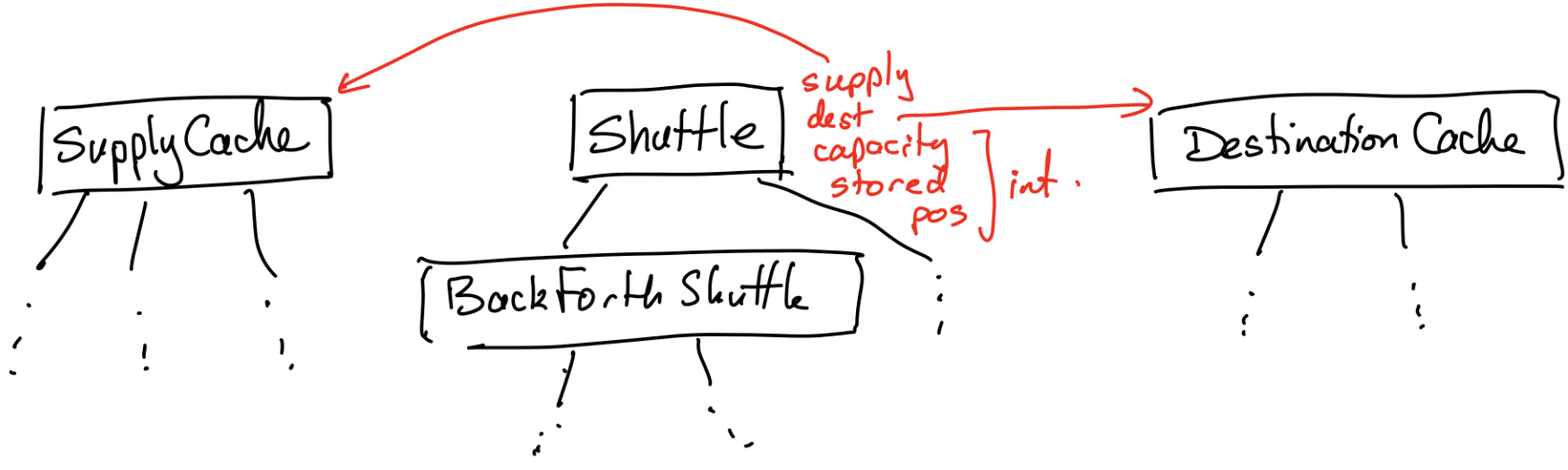
- Project 1 deadline 6pm central on Fri Feb 10.
- Project 1 autograder opens no later than Mon Feb 6.

PROJECT 1

- Major focus is **reading** existing code.
- Uses object-oriented programming.
- You must add a few subclasses of an existing class.


SUPPLY CACHE
SHUTTLE
DESTINATION CACHE
FACTORY 275
SAND
Transport Ltd
0
LOAD HERE
1
2
3
4
5
6
7
8
9
UNLOAD HERE.


SupplyCache
Shuttle
Destination Cache
BackForth Shuttle


SupplyCache
Shuttle
supply
dest
capacity
stored
pos
int.
Destination Cache
BackForth Shuttle

PROJECT 1 FILES

`shuttle.py`	**Edit and submit**. Add subclasses as requested.
`fixtures.py`	Read.
`simulation.py`	Read. Use/modify for testing.
`tui.py`	Module used by `simulation.py`. Don't need to read.

PROJECT 1

Now, a demo of a solution to the project in the REPL and using `simulation.py`.

PYTHON INTERFACES

- **REPL** — One command at a time. Result is printed.
- **Script mode** — Runs an entire file. Nothing printed unless explicitly requested (e.g. `print(...)`).

PYTHON INTERFACES

- **REPL** — One command at a time. Result is printed.
- **Notebook** — Make groups of commands (cells) to run when ready. Last result in a cell is printed.
- **Script mode** — Runs an entire file. Nothing printed unless explicitly requested (e.g. `print(...)`).

WHAT NOTEBOOKS LOOK LIKE

3. Cipher class hierarchy

Build a module `encoders` (in `encoders.py`) containing classes for simple ciphers (or codes; ways of obscuring the contents of a string that can be undone later by the intended recipient).

There should be a base class `BaseEncoder` that has two methods:

- `encode(self,text)` : Returns the string `text` unchanged. Subclasses will alter this behavior.
- `decode(self,text)` : Returns the string `text` unchanged. Subclasses will alter this behavior.

It should be the case that `obj.decode(obj.encode(s)) == s` is true for any string `s`, and for any object `obj` that is an instance of `BaseEncoder` or subclass thereof.

Then, build subclasses of `BaseEncoder` that implement encoding and decoding by different ciphers, including:

- `RotateEncoder` : Encoding rotates letters in the alphabet forward by a certain number of steps, e.g. so rotation by 5 turns "a" into "f" and "z" into "e" (because we wrap around when we reach the end of the alphabet). No transformation is applied to characters other than capital and lower case letters. Constructor accepts an integer, specifying the number of steps to rotate.
- `Rot13Encoder` : A subclass of `RotateEncoder` that fixes the steps at 13, so that encoding and decoding are the same operation.
- `SubstitutionEncoder` : The constructor accepts two arguments, `pre` and `post`. The string `pre` is a list of characters to be replaced when encoding, and string `post` indicates the things to replace them with. For example, using `pre="abcd"` and `post="1j4e"` would mean that "a" is supposed to be replaced by "1", "b" by "j", "c" by "4", and so on.
  - Be careful writing the encoder so that you don't replace things twice. For example `pre="abc"` and `post="bca"` should encode "banana" to "cbnbnb", and *not* "ccncnc".
  - You can assume that `pre` and `post` contain the same characters but in a different order. If that's not the case, then it would be impossible to ensure that decoding after encoding always gives the original text back again.

You can find some test code below. The test code assumes all of the classes are in the global scope.

In [ ]:
```
E = RotateEncoder(5)
s = E.encode("Hello world!") # Mjqqt btwqi!
print(s) # Mjqqt btwqi!
print(E.decode(s)) # Hello world!

F = SubstitutionEncoder("lmno","nolm")
s = F.encode("Hello everyone!")
print(s) # Hennm everymle!
print(F.decode(s)) # Hello everyone!
```

MCS 275 uses notebooks for homework, worksheets, and project descriptions, so you've seen these before.

But you usually see a version converted to HTML.

WHAT NOTEBOOKS LOOK LIKE

3. Cipher class hierarchy

Build a module `encoders` (in `encoders.py`) containing classes for simple ciphers (or codes; ways of obscuring the contents of a string that can be undone later by the intended recipient).

There should be a base class `BaseEncoder` that has two methods:

- `encode(self,text)` : Returns the string `text` unchanged. Subclasses will alter this behavior.
- `decode(self,text)` : Returns the string `text` unchanged. Subclasses will alter this behavior.

It should be the case that `obj.decode(obj.encode(s)) == s` is true for any string `s`, and for any object `obj` that is an instance of `BaseEncoder` or subclass thereof.

Then, build subclasses of `BaseEncoder` that implement encoding and decoding by different ciphers, including:

- `RotateEncoder` : Encoding rotates letters in the alphabet forward by a certain number of steps, e.g. so rotation by 5 turns "a" into "f" and "z" into "e" (because we wrap around when we reach the end of the alphabet). No transformation is applied to characters other than capital and lower case letters. Constructor accepts an integer, specifying the number of steps to rotate.
- `Rot13Encoder` : A subclass of `RotateEncoder` that fixes the steps at 13, so that encoding and decoding are the same operation.
- `SubstitutionEncoder` : The constructor accepts two arguments, `pre` and `post`. The string `pre` is a list of characters to be replaced when encoding, and string `post` indicates the things to replace them with. For example, using `pre="abcd"` and `post="1j4e"` would mean that "a" is supposed to be replaced by "1", "b" by "j", "c" by "4", and so on.
  - Be careful writing the encoder so that you don't replace things twice. For example `pre="abc"` and `post="bca"` should encode "banana" to "cbnbnb", and *not* "ccncnc".
  - You can assume that `pre` and `post` contain the same characters but in a different order. If that's not the case, then it would be impossible to ensure that decoding after encoding always gives the original text back again.

Cell of formatted text

You can find some test code below. The test code assumes all of the classes are in the global scope.

In [ ]:
```
E = RotateEncoder(5)
s = E.encode("Hello world!") # Mjqqt btwqi!
print(s) # Mjqqt btwqi!
print(E.decode(s)) # Hello world!

F = SubstitutionEncoder("lmno","nolm")
s = F.encode("Hello everyone!")
print(s) # Hennm everymle!
print(F.decode(s)) # Hello everyone!
```

MCS 275 uses notebooks for homework, worksheets, and project descriptions, so you've seen these before.

But you usually see a version converted to HTML.

WHAT NOTEBOOKS LOOK LIKE

3. Cipher class hierarchy

Build a module `encoders` (in `encoders.py`) containing classes for simple ciphers (or codes; ways of obscuring the contents of a string that can be undone later by the intended recipient).

There should be a base class `BaseEncoder` that has two methods:

- `encode(self,text)` : Returns the string `text` unchanged. Subclasses will alter this behavior.
- `decode(self,text)` : Returns the string `text` unchanged. Subclasses will alter this behavior.

It should be the case that `obj.decode(obj.encode(s)) == s` is true for any string `s`, and for any object `obj` that is an instance of `BaseEncoder` or subclass thereof.

Then, build subclasses of `BaseEncoder` that implement encoding and decoding by different ciphers, including:

- `RotateEncoder` : Encoding rotates letters in the alphabet forward by a certain number of steps, e.g. so rotation by 5 turns "a" into "f" and "z" into "e" (because we wrap around when we reach the end of the alphabet). No transformation is applied to characters other than capital and lower case letters. Constructor accepts an integer, specifying the number of steps to rotate.
- `Rot13Encoder` : A subclass of `RotateEncoder` that fixes the steps at 13, so that encoding and decoding are the same operation.
- `SubstitutionEncoder` : The constructor accepts two arguments, `pre` and `post`. The string `pre` is a list of characters to be replaced when encoding, and string `post` indicates the things to replace them with. For example, using `pre="abcd"` and `post="1j4e"` would mean that "a" is supposed to be replaced by "1", "b" by "j", "c" by "4", and so on.
  - Be careful writing the encoder so that you don't replace things twice. For example `pre="abc"` and `post="bca"` should encode "banana" to "cbnbnb", and *not* "ccncnc".
  - You can assume that `pre` and `post` contain the same characters but in a different order. If that's not the case, then it would be impossible to ensure that decoding after encoding always gives the original text back again.

You can find some test code below. The test code assumes all of the classes are in the global scope.

In [ ]:
```
E = RotateEncoder(5)
s = E.encode("Hello world!") # Mjqqt btwqi!
print(s) # Mjqqt btwqi!
print(E.decode(s)) # Hello world!

F = SubstitutionEncoder("lmno","nolm")
s = F.encode("Hello everyone!")
print(s) # Hennm everymle!
print(F.decode(s)) # Hello everyone!
```

Cell of code

MCS 275 uses notebooks for homework, worksheets, and project descriptions, so you've seen these before.

But you usually see a version converted to HTML.

HOW TO USE NOTEBOOKS

Several options:

- **Google Colab** — Web tool to create, edit, run notebooks. Need a Google account. Can save or download notebooks.
- Other online provider, e.g. Kaggle, CoCalc
- **Jupyter** — Software you install locally to create, edit, run notebooks. Browser shows the UI. Previously called IPython.
- **VS Code** — Has an extension for handling notebook files.

JUPYTER INSTALL INSTRUCTIONS

Most users can install Jupyter using pip:

```
python3 -m pip install notebook
```

Then run the interface with:

```
python3 -m notebook
```

Of course, you need to replace `python3` with your interpreter name.

USING COLAB / JUPYTER

A few of the many keyboard shortcuts:

- shift-enter — run the current cell
- escape — switch from cell editing to navigation
- a — in nav mode, add a new cell ABOVE this one
- b — in nav mode, add a new cell BELOW this one
- dd — in Jupyter, in nav mode, delete current cell (colab has a delete button, and a different shortcut)
- m — in Jupyter, in nav mode, make current cell a Markdown (text) cell

JUPYTER PITFALLS

The notebook interface is **stateful**: Behavior depends on the cells that have been evaluated so far.

If you open a previously saved notebook file, you may see old output. But you need to run all of the cells again if you want to use those values in the current session.

Cell execution order matters: Best to make a notebook that is meant to run top to bottom.

MARKDOWN

Text cells (Colab) or markdown cells (Jupyter) contain formatted text. When editing, formatting is specified with a language called Markdown.

```
# Heading level 1
## Heading level 2
### Heading level 3

* Bullet list item
* Another bullet list item

1. Numbered list item
1. Another numbered list item

Links: [text to display](https://example.com)
```

REFERENCES

- Google Colab offers notebook creation, editing, execution (can use netid@uic.edu google account).
- Some other online services allowing free use of Python notebooks: Kaggle, CoCalc
- A Markdown guide from GitHub.

REVISION HISTORY

- 2022-01-26 Last year's lecture on this topic finalized
- 2023-01-30 Updated version for spring 2023