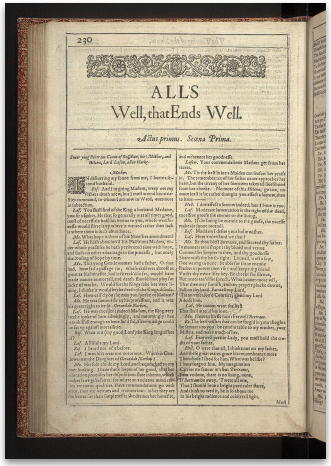
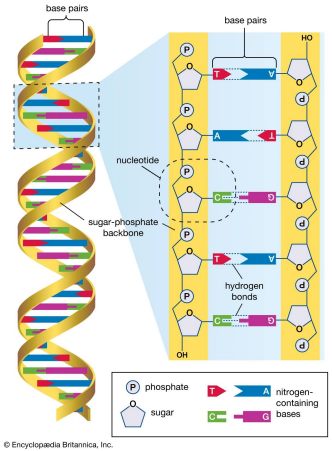




YEAH Hours: Assign5

Python Programs

CS106AX Autumn 2025

cs106ax.stanford.edu

Stanford | ENGINEERING
Computer Science

Welcome to Week 6 of CS106AX!

Autumn Week 6 **Winter**

So … how's life everyone

🐍 Lecture Recap!

JavaScript → Python Translation

⏳ ~10 minutes

Recap on the classroom whiteboard; feel free to ask any questions!

If you have questions on 🪄 *PyCharm, lmk also!*

Jerry Picture of the Week 💖

Part 1: Random Sentence Generator

Part 2: Reassemble

📚 Part 1: Random Sentence Generator

📂 Reading context-free grammar from a file

A context-free grammar – for the sake of this assignment – is a series of definitions, each comprised of a **placeholder** (called a **non-terminal**) and **replacement options** (called **expansions**).


```
<start>
1
The <object> <verb> tonight.

<object>
3
waves
big yellow flowers
slugs

<verb>
3
sigh <adverb>
portend like <object>
die <adverb>

<adverb>
2
warily
grumpily
```

```
grammar = {
 "<start>": ["The <object> <verb> tonight"],
 "<object">: ["waves", "big yellow flowers", "slugs"],
 "<verb">: ["sigh <adverb>", "portend like <object>",
            "die <adverb>"],
 "<adverb>": ["warily"]
}
```

🎯 Our goal is to **read a grammar file into a dictionary** of this form, mapping **non-terminals** to a **list of expansions!**

Part 1: Random Sentence Generator

To **start generating a random sentence** from this grammar dictionary, like Monopoly game, we always start with `"<start>"` :)

"<start>"

Part 1: Random Sentence Generator

To **start generating a random sentence** from this grammar dictionary, like Monopoly game, we always start with `"<start>"` :)

Grammar Non-Terminals: “<start>”, “<object>”, “<verb”, “<adverb>”

“<start>”

Find first non-terminal in the string

Part 1: Random Sentence Generator

To **start generating a random sentence** from this grammar dictionary, like Monopoly game, we always start with `"<start>"` :)

"<start>"

Let's expand it!

Part 1: Random Sentence Generator

To **start generating a random sentence** from this grammar dictionary, like Monopoly game, we always start with `"<start>"` :)

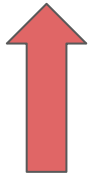
"<start>"

grammar["<start>"]	"The <object> <verb> tonight" 🟢

One possible expansion, so just choose that one!

Part 1: Random Sentence Generator

“The <object> <verb> tonight”



Expanded!

Part 1: Random Sentence Generator

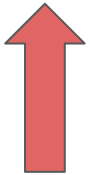
Repeat until no more non-terminals in string, in which we case we terminate with a full, randomly generated sentence!

“The <object> <verb> tonight”

Part 1: Random Sentence Generator

Grammar Non-Terminals: “<start>”, “<object>”, “<verb”, “<adverb>”

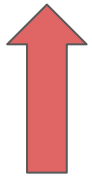
“The <object> <verb> tonight”



Against, find first non-terminal

📚 Part 1: Random Sentence Generator

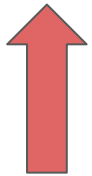
“The <object> <verb> tonight”



Let’s expand it!

📚 Part 1: Random Sentence Generator

“The <object> <verb> tonight”



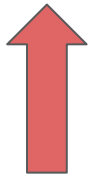
Let’s expand it!

grammar[“<object>”]	“waves”	“big yellow flowers”	“Slugs”

3 possible expansions: choose a random one! 🎰

📚 Part 1: Random Sentence Generator

“The <object> <verb> tonight”



Let’s expand it!

grammar[“<object>”]	“waves” 🟢	“big yellow flowers”	“slugs”

Three possible expansions: **have chosen a random one!**

📚 Part 1: Random Sentence Generator

“The waves <verb> tonight”

expanded!

Part 1: Random Sentence Generator

Repeat until no more non-terminals in string, in which we case we terminate with a full, randomly generated sentence!

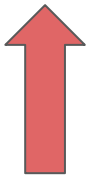
“The waves <verb> tonight”

Part 1: Random Sentence Generator

Repeat until no more non-terminals in string, in which we case we terminate with a full, randomly generated sentence!

Grammar Non-Terminals: “<start>”, “<object>”, “<verb”, “<adverb>”

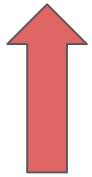
“The waves <verb> tonight”



Against, find first non-terminal

Part 1: Random Sentence Generator

“The waves <verb> tonight”



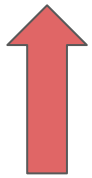
Let’s expand it!

grammar[“<verb>”]	“sigh <adverb>”	“portend like <object>”
	“die <adverb>”	

3 possible expansions: choose a random one!

📚 Part 1: Random Sentence Generator

“The waves <verb> tonight”



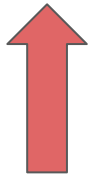
Let’s expand it!

grammar[“<verb>”]	“sigh <adverb>”	“portend like <object>” 🟢
	“die <adverb>”	

Three possible expansions: **have chosen a random one!**

📚 Part 1: Random Sentence Generator

“The waves portend like <object> tonight”



Expanded!

📚 Part 1: Random Sentence Generator

Repeat until no more non-terminals in string, in which we case we terminate with a full, randomly generated sentence!

“The waves portend like <object> tonight”

Part 1: Random Sentence Generator

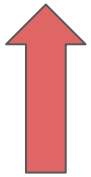
Grammar Non-Terminals: “<start>”, “<object>”, “<verb”, “<adverb>”

“The waves portend like <object> tonight”

Against, find first non-terminal

📚 Part 1: Random Sentence Generator

“The waves portend like <object> tonight”



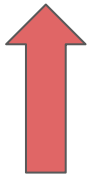
Let’s expand it!

grammar[“<object>”]	“waves”	“big yellow flowers”	“slugs”

3 possible expansions: choose a random one! 🎰

📚 Part 1: Random Sentence Generator

“The waves portend like <object> tonight”



Let’s expand it!

grammar[“<object>”]	“waves”	“big yellow flowers” 🟢	“slugs”

3 possible expansions: **have chosen a random one!**

📚 Part 1: Random Sentence Generator

“The waves portend like big yellow flowers tonight”

Expanded!

📚 Part 1: Random Sentence Generator

With no more nonterminals, the sentence generation has concluded!

“The waves portend like big yellow flowers tonight”

We have our random sentence yay! 🎉

It’s poetry (not really at all)

🖌️ Interactive examples on the classroom board!

Random Sentence Generation

🧬 Part 2: Reassemble

📂 Reading fragments from a file

Our goal is to reconstruct the original full text from a list of fragments, which we have to first read from a text file.

allswell_

```
{all is well}
{ell that en}
{hat end}
{t ends well}
```

fragments = [“all is well”, “ell that en”,
“hat end”, “t ends well”]

🎯 Our next goal is to assemble them together! In the case above, the result should be *“all is well that ends well”*

Part 2: Reassemble

Let's put these **four fragments back together**!

➔ First, we'll want to identify the **pair of fragments with the most overlap** when the back of one fragment is combined with the front of another

all is well

ell that en

hat end

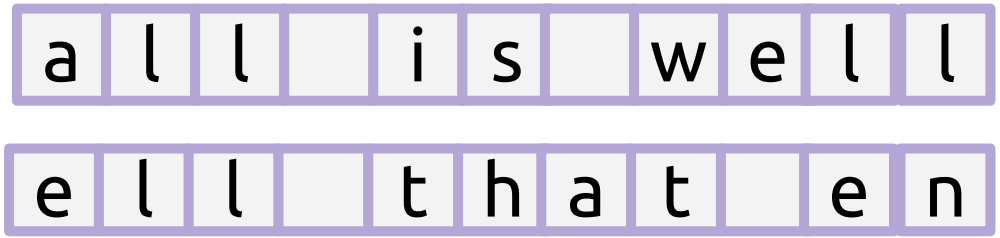
t ends well

Part 2: Reassemble

There's 6 possible pairs among 4 fragments. For each pair:

➔ We identify the **largest possible overlap** when combining them

➔ In the case of the two strings **"all is well"** and **"ell that en"**,

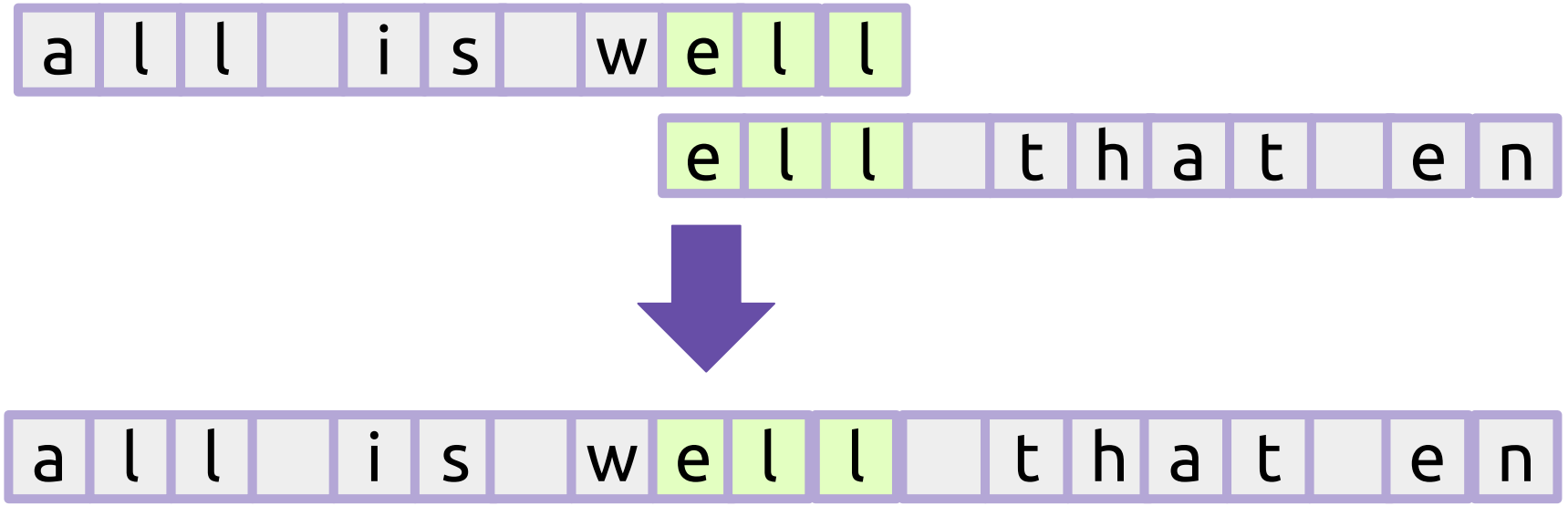


Part 2: Reassemble

There's 6 possible pairs among 4 fragments. For each pair:

➔ We identify the **largest possible overlap** when combining them

➔ In the case of the two strings **"all is well"** and **"ell that en"**, the max overlap is 3 characters, with **"all is well"** going first



Part 2: Reassemble

➔ Let’s check the max overlap for all possible pairs among our fragments!

Part 2: Reassemble

➔ Let's check the **max overlap** among all possible pairs of fragments!

Pair of Fragments	Longest Overlap Result	Length
"all is well" + "ell that en"	"all is well that end"	3
"all is well" + "hat end"	no overlap	0
"all is well" + "t ends well"	no overlap	0
"ell than en" + "hat end"	"ell than end"	6
"ell that en" + "t ends well"	ell that ends well	3
"hat end" + "t ends well"	t ends well	5

🧬 Part 2: Reassemble

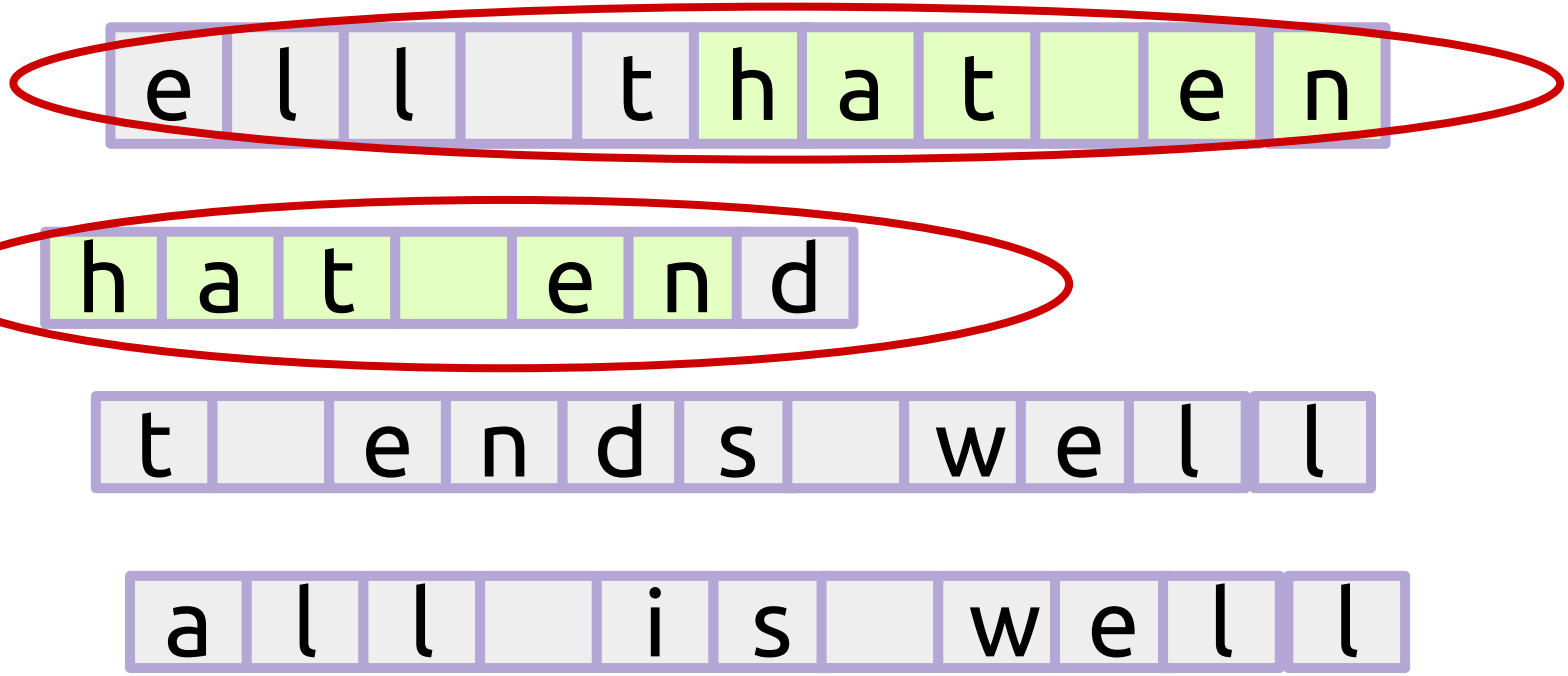
➔ We have a winner! 🏆

Pair of Fragments	Longest Overlap Result	Length
“all is well” + “ell that en”	“all is well that end”	3
“all is well” + “hat end”	no overlap	0
“all is well” + “t ends well”	no overlap	0
“ell than en” + “hat end”	“ell than end”	6
“ell that en” + “t ends well”	ell that ends well	3
“hat end” + “t ends well”	t ends well	5

✅

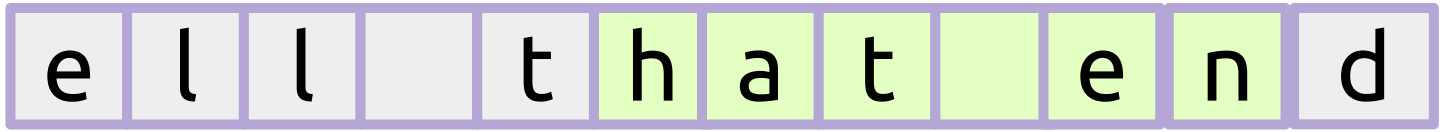
Part 2: Reassemble

This pair has highest overlap (6), so let's combine!



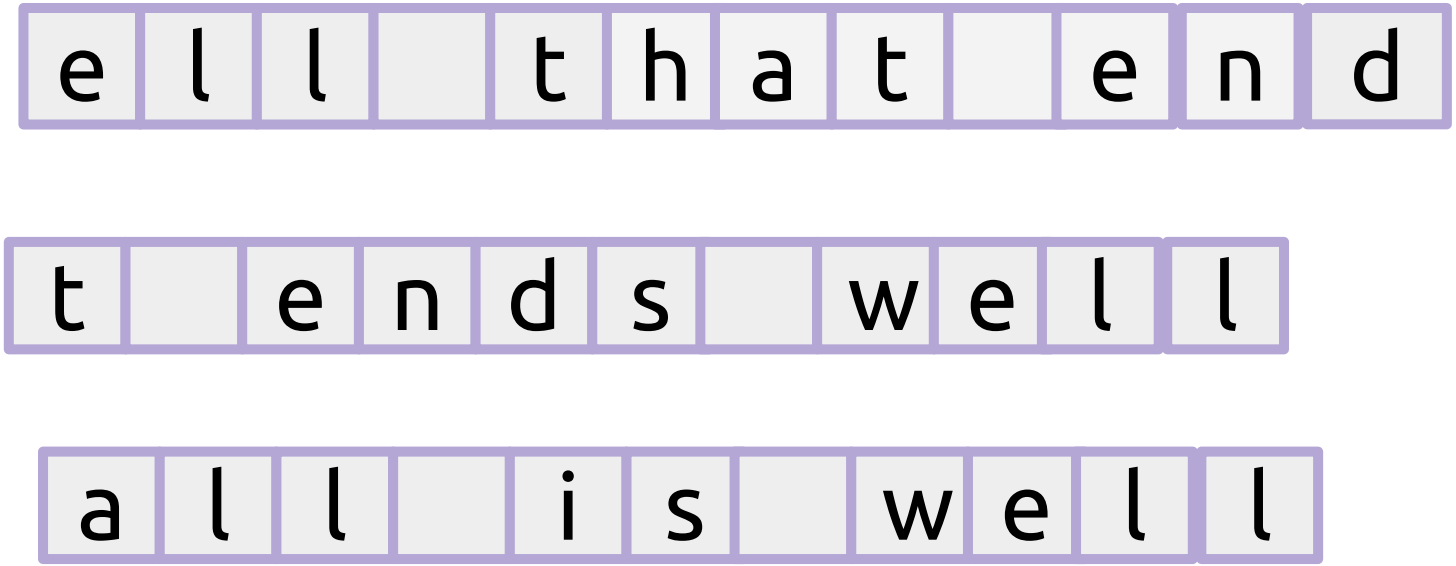
Part 2: Reassemble

We’ve combined them!



Part 2: Reassemble

➔ Let's repeat the same thing as before: check the **max overlap** among all possible pairs of fragments! (three in total)



Part 2: Reassemble

➔ Let's repeat the same thing as before: check the **max overlap** among all possible pairs of fragments! (3 possible pairs in total)

Pair of Fragments	Longest Overlap Result	Length
"ell that end" + "t ends well"	"ell that ends well"	5
"all is well" + "ell that end"	"all is well that end"	3
"t ends well" + "all is well"	no overlap	0

🧬 Part 2: Reassemble

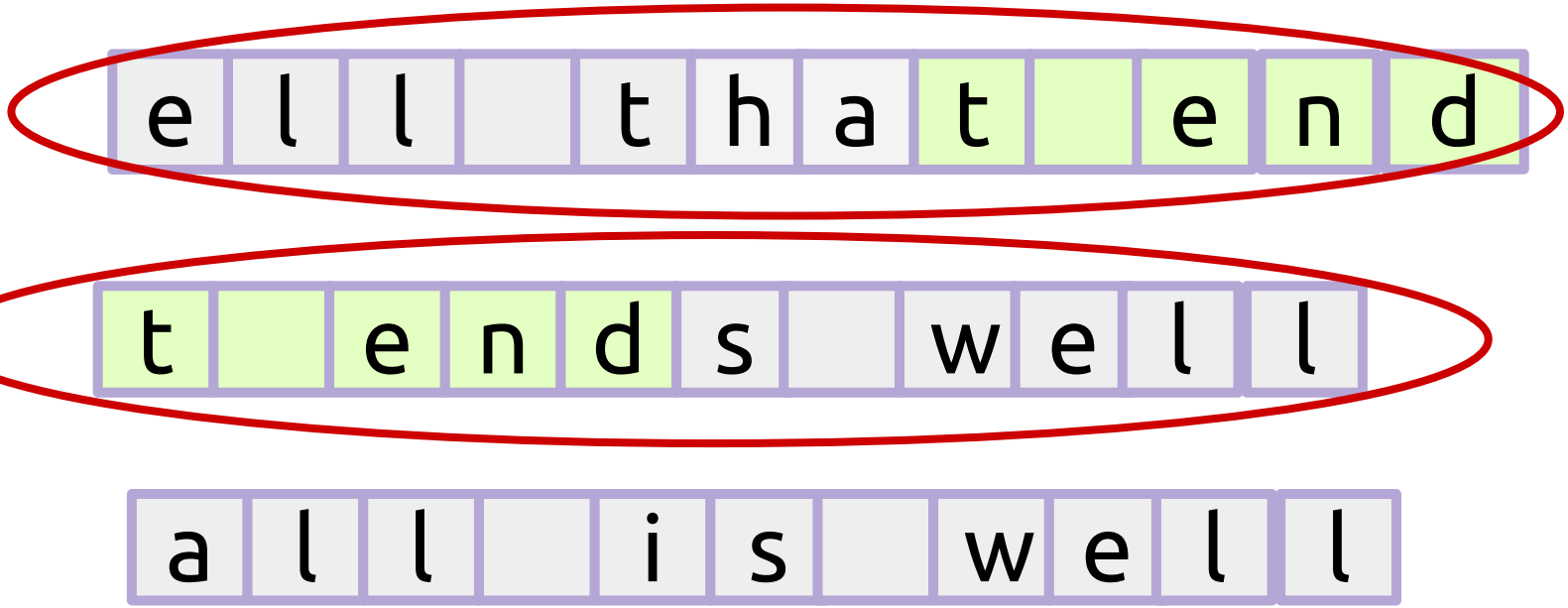
➔ We have a winner! 🏆

Pair of Fragments	Longest Overlap Result	Length
“ell that end” + “t ends well”	“ell that ends well”	5
“all is well” + “ell that end”	“all is well that end”	3
“t ends well” + “all is well”	no overlap	0

✅

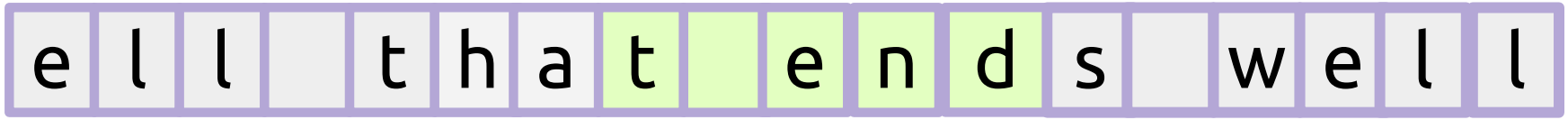
Part 2: Reassemble

This pair has highest overlap (5), so let's combine!



Part 2: Reassemble

This pair has highest overlap (5), so let's combine!



Part 2: Reassemble

➔ Let's repeat the same thing **one last time**: check the **max overlap** among the last two remaining fragments

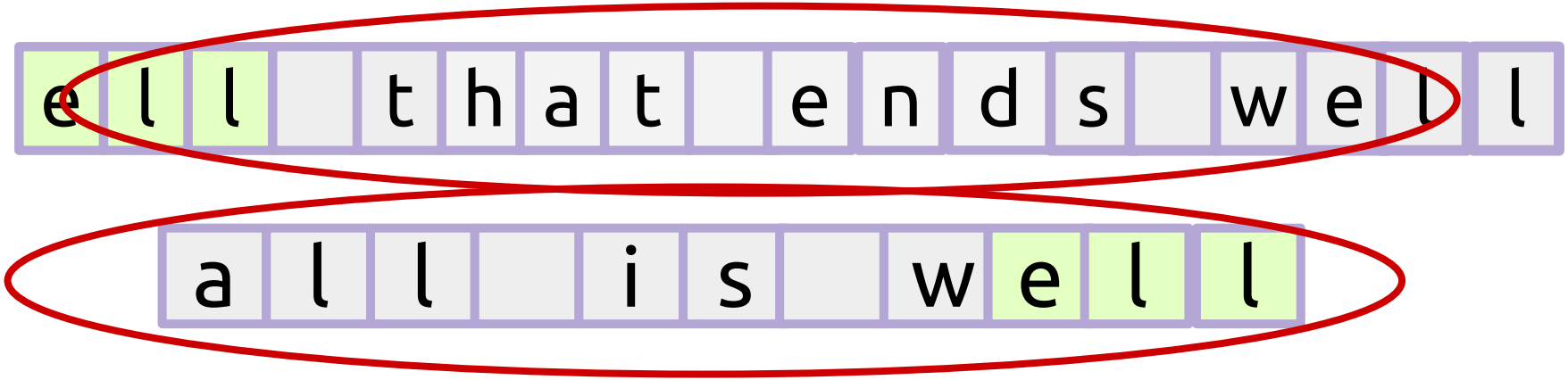
Part 2: Reassemble

Pair of Fragments	Result	Length
“all is well” + “ell that ends well”	“all is well that ends well”	3

✅

Part 2: Reassemble

Let’s combine along their overlapping portion!



Part 2: Reassemble

And we can get the final reassembly, tada! 🎉

“all is well that ends well”

🖌️ Interactive examples on the classroom board!

Reassemble

Keep up the amazing work! :)

Please feel free to reach out if you have any questions, e.g., in the office hours following the overview portion of the YEAH session.

Make sure to take a break also after the midterm – you deserve it! :)