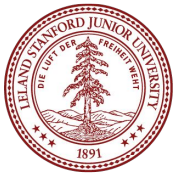
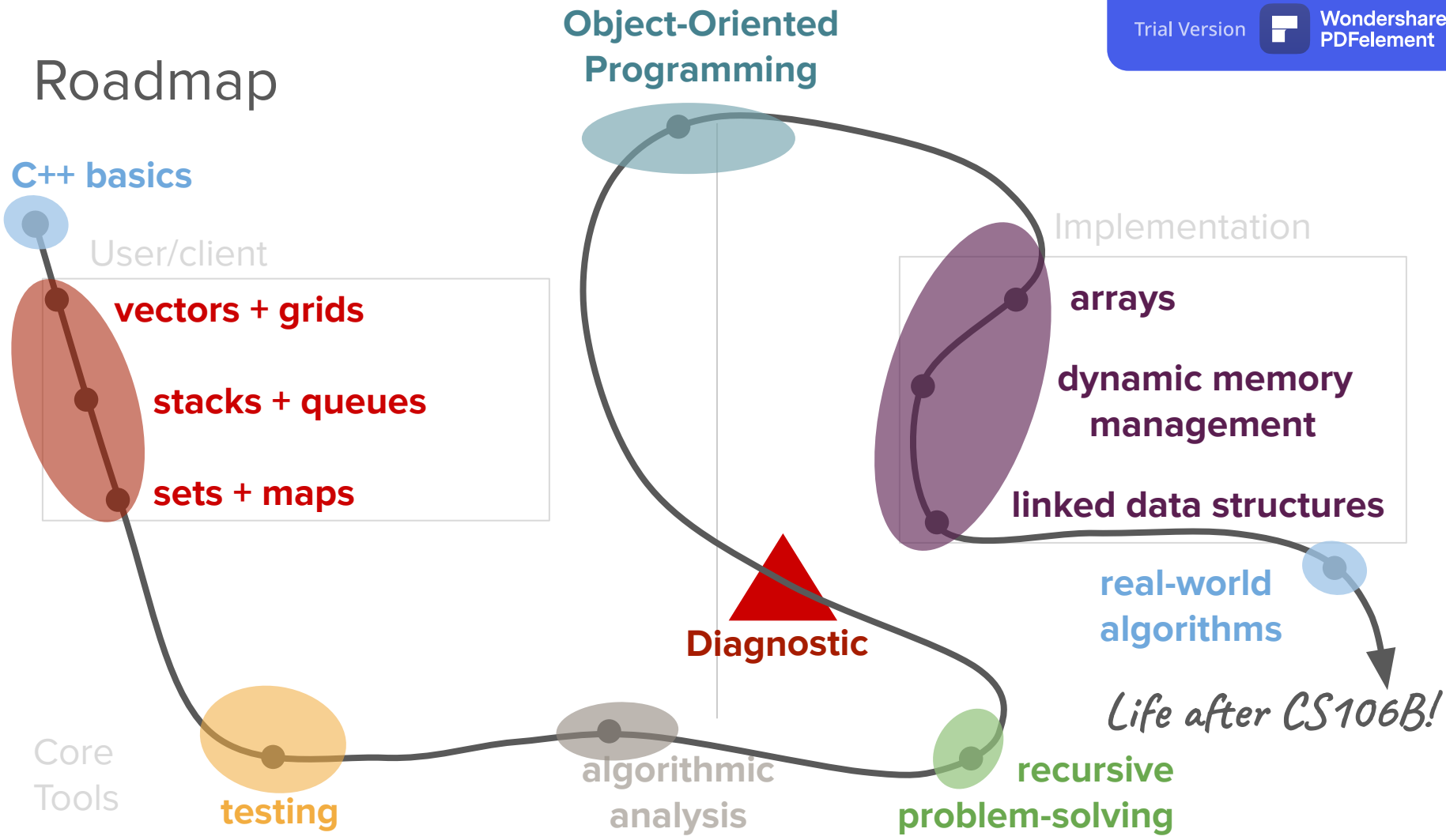
Strings and Testing

What's something that's *become more difficult* for you during online learning?

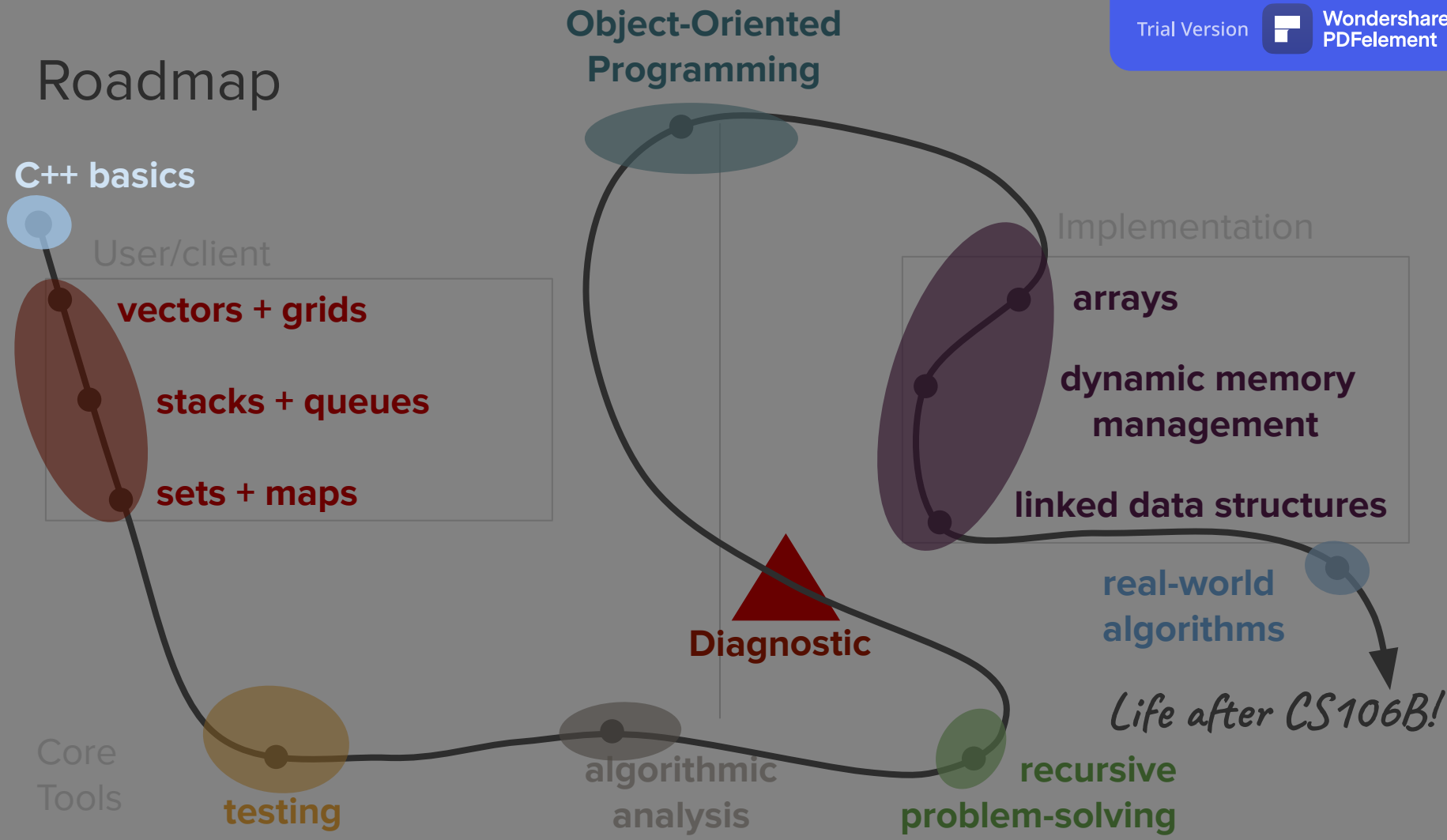
(put your answers the chat)



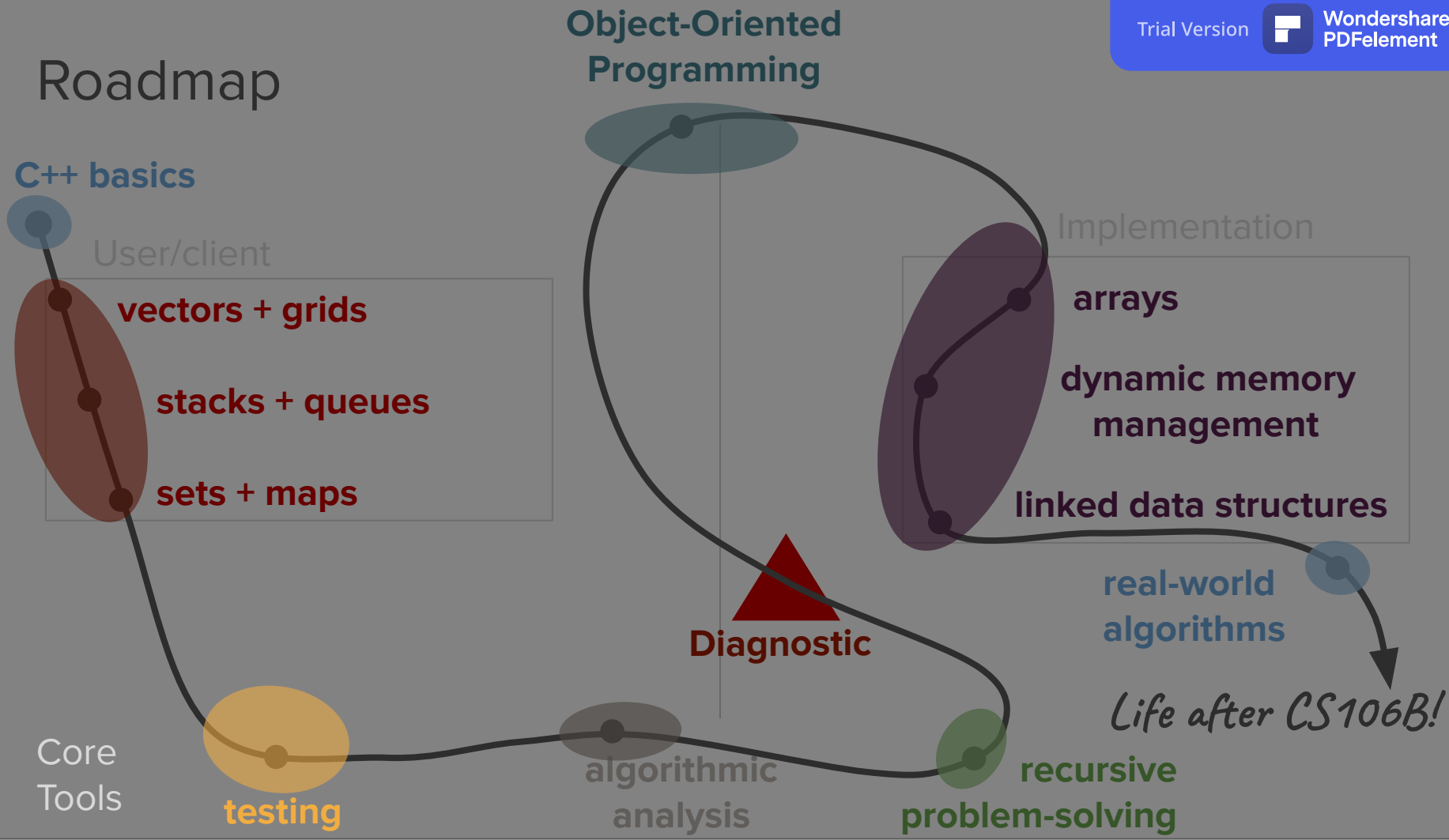
Roadmap



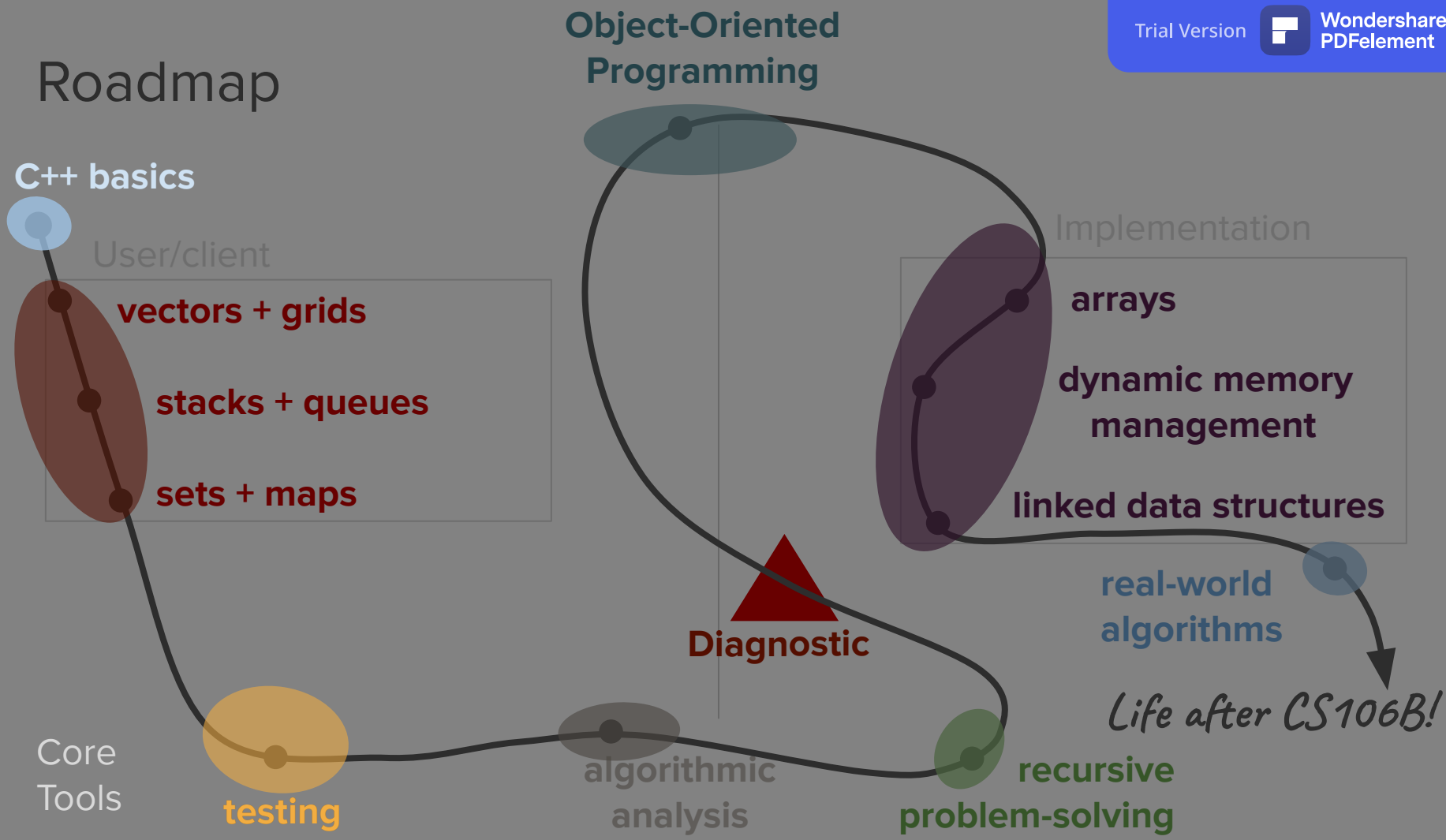
Roadmap



Roadmap



Roadmap



Today's questions

How can we improve the online learning environment?

[Review with **`spaceship.cpp`**]

What's special about strings in C++?

How do we test code in CS106B?

What's next?

How can we improve the online learning environment?

Zoom chat is distracting.

The enthusiasm and curiosity is inspiring!

But we also realized that the chat can be very overwhelming for some students during lecture (especially if you're trying to view from a phone).

I found it hard to figure out whose questions had been answered, and students who asked questions wouldn't always see the answers.

New norm: Anyone can ask questions, but only course staff will answer questions in the chat. (We'll try this, and we can always adjust later if this isn't working.)

Thank you for the early feedback!

We can better center questions around learning.

Thinking about your own learning (metacognition) is important!

Sometimes asking a question immediately and waiting for an answer can distract from the learning experience (and the question will often get answered in a slide or two).

There are two (vastly oversimplified) types of questions:

1. Questions that will enable you to understand the rest of the topic/lecture.

Strategy: Ask immediately by raising your hand (or putting it in the chat if you're more comfortable with that). If you found something confusing, someone else probably did, too. And remember, celebrate struggle!

We can better center questions around learning.

Thinking about your own learning (metacognition) is important!

Sometimes asking a question immediately and waiting for an answer can distract from the learning experience (and the question will often get answered in a slide or two).

There are two (vastly oversimplified) types of questions:

2. Questions will expand your depth of knowledge but that your immediate understanding does not depend upon.

Strategy: Write down your question and ask when it's clear we're transitioning to a new topic. We'll also often stop for questions then.

We can better center questions around learning.

Thinking about your own learning (metacognition) is important!

Sometimes asking a question immediately and waiting for an answer can distract from the learning experience (and the question will often get answered in a slide or two).

There are two (vastly oversimplified) types of questions:

2. Questions will expand your depth of knowledge but that your immediate understanding does not depend upon.

Strategy: If you can answer the question yourself by writing a small piece of code to test your question, we encourage you to do that, too!

We can better center questions around learning.

Thinking about your own learning (metacognition) is important!

Sometimes asking a question immediately and waiting for an answer can distract from the learning experience (and the question will often get answered in a slide or two).

There are two (vastly oversimplified) types of questions:

1. Questions that will enable you to understand the rest of the topic/lecture.
2. Questions will expand your depth of knowledge but that your immediate understanding does not depend upon.

Think about how to use questions to maximize your concentration and learning!

We can better center questions around inclusivity.

One of the most difficult things about teaching CS is catering to an audience of diverse backgrounds and prior programming experience.

Curiosity is wonderful, and we're happy to talk about advanced CS topics with you during office hours.

But we also don't want to send the message that you need to know about these things when entering CS106B.

- In particular, we don't expect students in this class to have prior C++ knowledge or knowledge of the topics that we explicitly introduce from scratch. So please keep this mind when you're asking questions!

We can better center questions around inclusivity.

One of the most difficult things about teaching CS is catering to an audience of diverse backgrounds and prior programming experience.

Curiosity is wonderful, and we're happy to talk about advanced CS topics with you during office hours.

But we also don't want to send the message that you need to know about these things when entering CS106B.

It also benefits your learning to approach these concepts with a beginner's mindset – you might notice and learn things that you didn't before.

Consider if lecture or individual office hours is the right venue for your question.

Review

(using `spaceship.cpp`!)

Functions, variables, for loops, console programs

I wanted to follow up on a couple of the questions asked during class yesterday…

- *Can you declare two variables with the same name but within different scopes of the same function (e.g. one inside a for loop and one before the loop)?*
  ➔ Yes, but this gets confusing quickly so avoid variables with the same names!

- *Can you create two functions with the same name but different return types?*
  ➔ No, not if the return type is the only aspect that differs between the function prototypes. But you are allowed to create two functions that have the same name if they take different parameters (number or type). In this case, the return types don't matter. Again, this gets confusing quickly so try to avoid it!

Write a program that prints out the calls for a spaceship that is about to launch. Countdown the numbers from 10 to 1 and then print “Liftoff.”

```
10
9
8
7
6
5
4
3
2
1
Liftoff
```

```python
def main():
    for i in range(10, 0, -1):
        print(i)
    print("Liftoff")

if __name__ == "__main__":
    main()
```

Python

```cpp
#include <iostream>
using namespace std;

int main() {
    /* TODO: Your code goes here! */

    return 0;
}
```

C++

Ed Example

(workspace)

Notes about Ed workspaces

- To immediately preserve work done during lecture, we're going to set the "Breakout Room X" workspaces to view-only right after lecture.

- We'll be reusing the same breakout room workspaces every lecture, which means we'll also wipe the contents before the next day's class (likely in the morning before lecture).

- Feel free to download or fork the workspace after class to continue working on the code independently or with others.

What's special about **strings** in C++?

Characters can be letters, digits, symbols (& !, ~), etc.

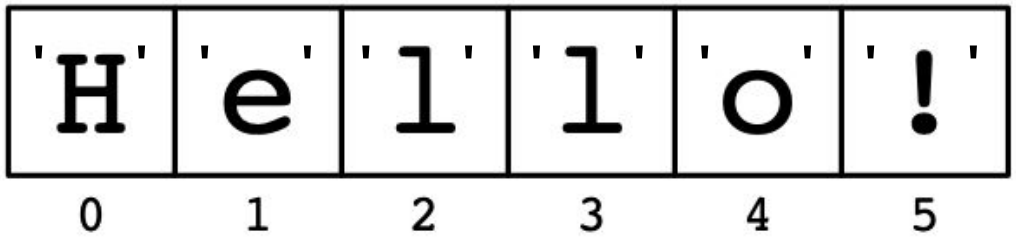
Definition

string

A data type that represents a sequence of characters

Strings review

Strings are made up of characters of type char, and the characters of a string can be accessed by the index in the string (this should be familiar):



string activity

[demo + poll]

What are the key characteristics of strings in C++?

- Strings are mutable in C++ (可变的)
  - Unlike in Python and Java
  - But you must assign string indices to a character:

YES: `word[1] = ‘a’;`

NO: `word[1] = “a”;`

character use the single quote not the double quotes.

What are the key characteristics of strings in C++?

- Strings are mutable in C++
- You can add characters to strings and strings to strings using += and +
  - Strings must use double quotes (“”) while characters use single (‘’).
  - There is a caveat (警告) you’ll see shortly
- You can use logical operators to compare strings (and characters)

string and char conventions

[demo]

string utility functions

Three categories of functions

- Built-in C++ char functions (<cctype> library)
- Built-in C++ string methods
- Stanford string library functions

<cctype> library

- **#include <cctype>**
- This library provides functions that check a single **char** for a property (e..g, if it is a digit), or return a **char** converted in some way (e.g., to uppercase)
  - **isalnum**: checks if a character is alphanumeric
  - **isalpha**: checks if a character is alphabetic
  - **islower**: checks if a character is lowercase
  - **isupper**: checks if a character is an uppercase character
  - **isdigit**: checks if a character is a digit
  - **isxdigit**: checks if a character is a hexadecimal character
  - **iscntrl**: checks if a character is a control character
  - **isgraph**: checks if a character is a graphical character
  - **isspace**: checks if a character is a space character
  - **isblank**: checks if a character is a blank character
  - **isprint**: checks if a character is a printing character
  - **ispunct**: checks if a character is a punctuation character
  - **tolower**: converts a character to lowercase
  - **toupper**: converts a character to uppercase

16进制

```
char letter = 'L';
islower(letter);
//returns false
```

string methods

```
#include <string>
```

- **`s.append(str)`**: add text **`str`** to the end of a string
- **`s.compare(str)`**: return **`-1`**, **`0`**, or **`1`** depending on relative ordering
- **`s.erase(index, length)`**: delete text from a string starting at given **`index`**
- **`s.find(str)`**: return first index where the start of **`str`** appears in this string (returns **`string::npos`** if not found)
- **`s.rfind(str)`**: return last index where the start of **`str`** appears in this string (returns **`string::npos`** if not found)
- **`s.insert(index, str)`**: add text **`str`** into a string at a given **`index`**
- **`s.length()`** or **`s.size()`**: number of characters in this string
- **`s.replace(index, len, str)`**: replaces **`len`** chars at **`index`** with text **`str`**
- **`s.substr(start, length)`** or **`s.substr(start)`**: the next length characters beginning at **`start`** (inclusive); if length omitted, grabs till end of string

Stanford string library functions

```
#include "strlib.h"
```

- **`endsWith(str, suffix)`**
  `startsWith(str, prefix)`: returns **`true`** if the given string begins or ends with the given **`suffix`**/**`prefix`** text
- **`integerToString(int)`**
  `realToString(double)`
  `stringToInteger(str)`
  `stringToReal(str)`: returns a conversion between numbers and strings
- **`equalsIgnoreCase(s1, s2)`**: **`true`** if **`s1`** and **`s2`** have same **`chars`**, ignoring casing
- **`toLowerCase(str)`**: returns a lowercase version of a string
- **`toUpperCase(str)`**: returns an uppercase version of a string
- **`trim(str)`**: returns string with surrounding whitespace removed

Two types of C++ strings

[poll]

Poll: What will happen with the following line of code?

```
string hiThere = "hi" + "there";
```

You would get...

- An error ✓
- The string “hithere” stored in `hiThere`
- A garbage value stored in `hiThere`

Poll: What will happen with the following line of code?

```
string hiThere = "hi" + '?'
```

You would get...

- An error
- The string “hi?” stored in `hiThere`
- A garbage value stored in `hiThere` ✓

C strings vs. C++ strings summary

- C strings have no methods
  - This is why you can't do something like `"hi".length()` in C++

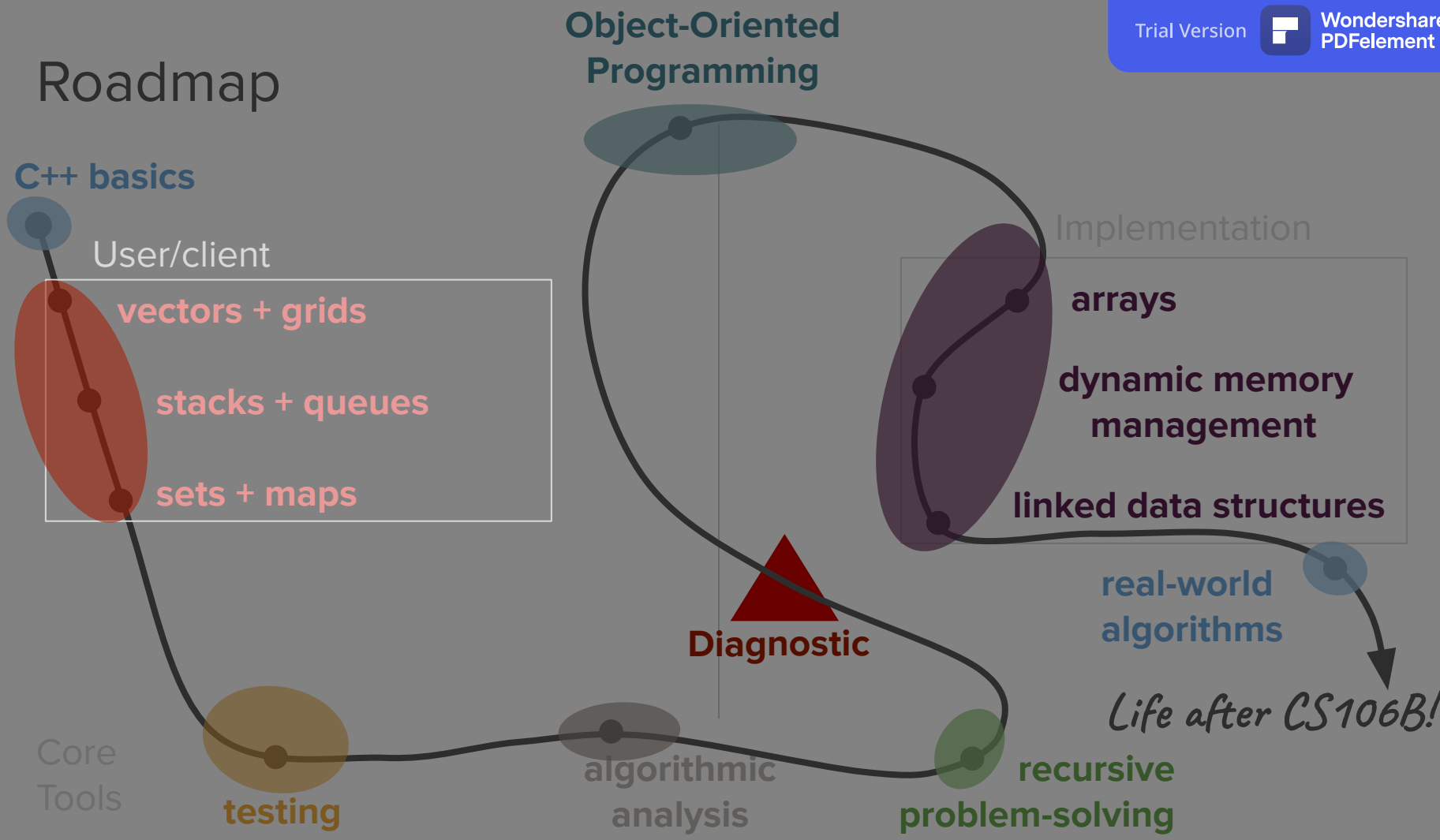
- Conversion fixes
  - Store the C string in a variable first to convert it to a C++ string
  - Use a conversion function
    - `string("text");` converts the C string literal into a C++ string
    - `string.c_str()` returns a C string from a C++ string

- **Takeaway**: Beware the C string!

Announcements

What’s next?

Roadmap



Vectors and Grids