Hi there,
In today's lecture I tried to specialize a variadic template function.
There were two mistakes!

1) Red colored part is the missing part for template specialization.
2) However, function templates cannot be specialized! It's not gonna compile. Only class templates can be partially specialized.

So, below code without the **red part** (as illustrated in the class) presents two different templated function definitions of the printAll function. Overload resolution always prefers the first one since it is fitting a HEAD and a TAIL pack compared to fitting only a TAIL pack in the second version. Being able to fit two parameters is a more special case compared to being able to fit only one parameter…

```
void printAll()
{
}

template<typename HEAD, typename... TAIL>
void printAll(const HEAD& head, const TAIL&... tail)
{
    cout << head << endl;
    printAll(tail...);
}

template<typename... TAIL>
void printAll<std::string>(const std::string& s, const TAIL&... tail)
{
    cout << "'" << s << "'" << endl;
    printAll(tail...);
}
```

If you want to do different things when encountered with a string easy solution for C++17 is below:

```
void printAll()
{
}

template<typename HEAD, typename... TAIL>
void printAll(const HEAD& head, const TAIL&... tail)
{
    if constexpr(std::is_same_v<HEAD, std::string>)
        cout << "'" << head << "'" << endl;
    else
        cout << head << endl;
    printAll(tail...);
}
```

You can even do as below:

```
template<typename HEAD, typename... TAIL>
void printAll(const HEAD& head, const TAIL&... tail)
{
    if constexpr(std::is_same_v<HEAD, std::string>)
        cout << "'" << head << "'" << endl;
    else
        cout << head << endl;

    if constexpr(sizeof...(tail) > 0)
        printAll(tail...);
}
```

If all you want is a variadic number of strings only, here is the code for it (recursive version). Here you are going to learn a new thing called static_assert, which generates a compiler error conditionally:

```
template<typename HEAD, typename... TAIL>
void printAll(const HEAD& head, const TAIL&... tail)
{
    static_assert(is_same_v<HEAD, std::string>, "You can only use strings!");
    cout << "'" << head << "'" << endl;

    if constexpr(sizeof...(tail) > 0)
        printAll(tail...);
}
```

Here is another version with non-recursive tricky method (without using fold expressions)!

```
template<typename... Ts>
void printAll(const Ts&... args)
{
    auto print = [](const std::string& s) {
        cout << "'" << s << "'" << endl;
    };

    (void)std::initializer_list<int>{(print(args), 0)...};
}
```