std::vector

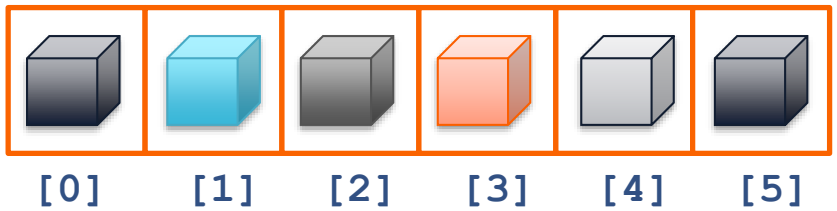
Prof. Wade Fagen-Ulmschneider

A **template type** is a special type that can take on different types when the type is initialized. **std::vector** uses a template type:

`std::vector<char>`

`std::vector<int>`

`std::vector<uiuc::Cube>`



std::vector

std::vector standard library class that provides the functionality of a dynamically growing array with a "templated" type. Key ideas:

Defined in:	`#include <vector>`
Initialization:	`std::vector<T> v;`
Add to (back) of array:	`::push_back(T);`
Access specific element:	`::operator[](unsigned pos);`
Number of elements:	`::size()`

Template Type

When initializing a “templated” type, the template type goes inside of `< >` at the end of the type name:

```
std::vector<char> v1;
std::vector<int> v2;
std::vector<uiuc::Cube> v3;
```

cpp-vector/ex1/main.cpp

```cpp
#include <vector>
#include <iostream>

int main() {
  std::vector<int> v;
  v.push_back( 2 );
  v.push_back( 3 );
  v.push_back( 5 );

  std::cout << v[0] << std::endl;
  std::cout << v[1] << std::endl;
  std::cout << v[2] << std::endl;

  return 0;
}
```

cpp-vector/ex2/main.cpp

```cpp
#include <vector>
#include <iostream>

int main() {
  std::vector<int> v;
  for (int i = 0; i < 100; i++) {
    v.push_back( i * i );
  }

  std::cout << v[12] << std::endl;

  return 0;
}
```