Lamda functions

Automatic functions

Lambda functions

- Create a function on the fly
- Why?
  - Sometimes a function makes the most sense as an argument
  - Do not want programming overhead of creating a simple function

Basics

```
[capture](parameters) -> return_type { function_body }
```

Basics

```
[capture](parameters) -> return_type { function_body }
```

What parameters you want to capture from your function

Basics

```
[capture](parameters) -> return_type { function_body }
```

What parameters you want to capture from your function

Parameters for the function

Basics

```
[capture](parameters) -> return_type { function_body }
```

What parameters you want to capture from your function

Return type for function

Parameters for the function

Basics

```
[capture](parameters) -> return_type { function_body }
```

What parameters you want to capture from your function

Return type for function

Parameters for the function

Code for function

Capture function

```
[]          //no variables defined. Attempting to use
any external variables in the lambda is an error.
[x, &y]     //x is captured by value, y is captured
by reference
[&]         //any external variable is implicitly
captured by reference if used
[=]         //any external variable is implicitly
captured by value if used
[&, x]      //x is explicitly captured by value.
Other variables will be captured by reference
[=, &z]     //z is explicitly captured by reference.
Other variables will be captured by value
```

Example 1

```
std::vector<int> some_list{ 1, 2, 3, 4, 5 };
int total = 0;
std::for_each(begin(some_list), end(some_list),
[&total](int x) {
  total += x;
});
```

Example 1

```
std::vector<int> some_list{ 1, 2, 3, 4, 5 };
int total = 0;
std::for_each(begin(some_list), end(some_list),
[&total](int x) {
  total += x;
});
```

Note: total captured by referenced, the function will see the updated value

Example 2

```
std::vector<int> some_list{ 1, 2, 3, 4, 5 };
int total = 0;
int value = 5;
std::for_each(begin(some_list), end(some_list),
 [&, value, this](int x) {
  total += x * value * this->some_func();
});
```

Example 2

```
std::vector<int> some_list{ 1, 2, 3, 4, 5 };
int total = 0;
int value = 5;
std::for_each(begin(some_list), end(some_list),
 [&, value, this](int x) {
  total += x * value * this->some_func();
});
```

Note: Captured total by reference, value and this by value.