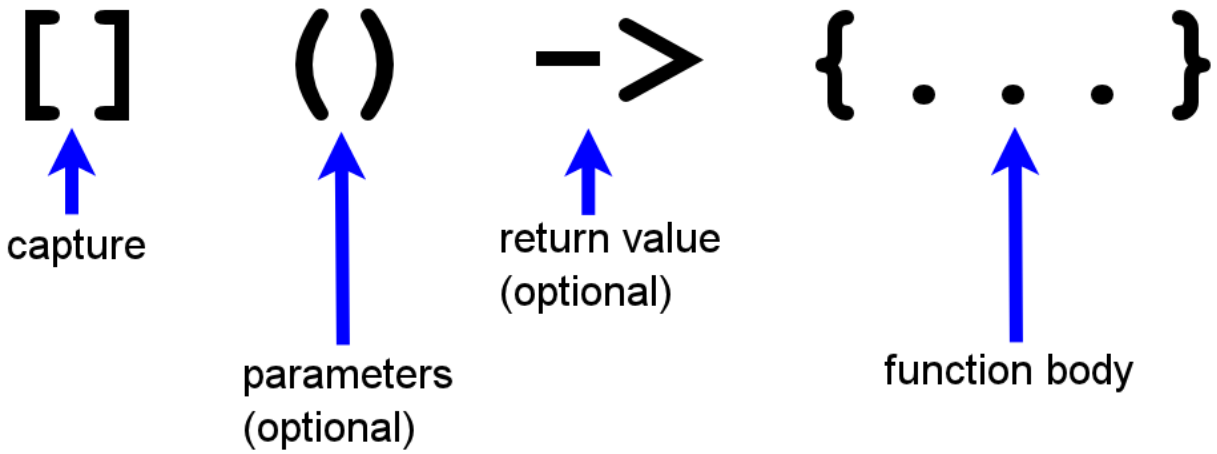
Lambda Functions

- Lambda functions
  - are functions without a name.
  - define their functionality just in place.
  - could be copied like data.
  - are able to store the invocation environment.
- Lambda functions should be
  - short and concise.
  - self-explenatory.

Lambda Functions



- `[]`: Captures the used variables
- `()`: Necessary for parameters
- `->`: Necessary for complex lambda functions
- `{}`: Function body , per default `const`

  `[]() mutable -> {...}` has a non-constant function body.

lambdaFunction.cpp

Lambda Functions

Lambda functions can bind their context. ➡ Closure

Binding	Description
`[]`	no binding
`[a]`	`a` per copy
`[&a]`	`a` per reference
`[=]`	all used variables per copy
`[&]`	all used variables per reference
`[=,&a]`	per default per copy; `a` per reference
`[&,a]`	per default per reference; a per copy
`[this]`	data and member of the enclosing class per copy
`[l= std::move(lock)]`	moves `lock` (C++14)

lambdaFunctionClosure.cpp
lambdaFunctionCapture.cpp

Lambda Functions

Lambda functions can be stored in variables and can be returned from functions.

- First class function

```
auto addTwoNumber= [](int a, int b){ return a + b; };

std::thread th( [](){std::cout <<  "Hello from thread" << std::endl;} );

std::function<int(int,int)> makeAdd(){
  return [](int a, int b){ return a + b; };
}

std::function<int(int, int)> myAdd= makeAdd();
myAdd(2000, 11);   // 2011
```

Generic Lambda Functions

A C++14 lambda functions can deduce their argument type.

```
auto add11=[ ](int i, int i2){ return i + i2; };
auto add14= [ ](auto i, auto i2){ return i + i2; };

std::vector<int> myVec{1, 2, 3, 4, 5};
auto res11= std::accumulate(myVec.begin(), myVec.end(), 0, add11);
auto res14= std::accumulate(myVec.begin(), myVec.end(), 0, add14);
  // res11 == res14 == 15;

std::vector<std::string> myVecStr{"Hello"s, " World"s};
auto st= std::accumulate(myVecStr.begin(), myVecStr.end(), ""s, add14);
std::cout << st << std::endl;   // Hello World
```

lambdaFunctionGeneric.cpp