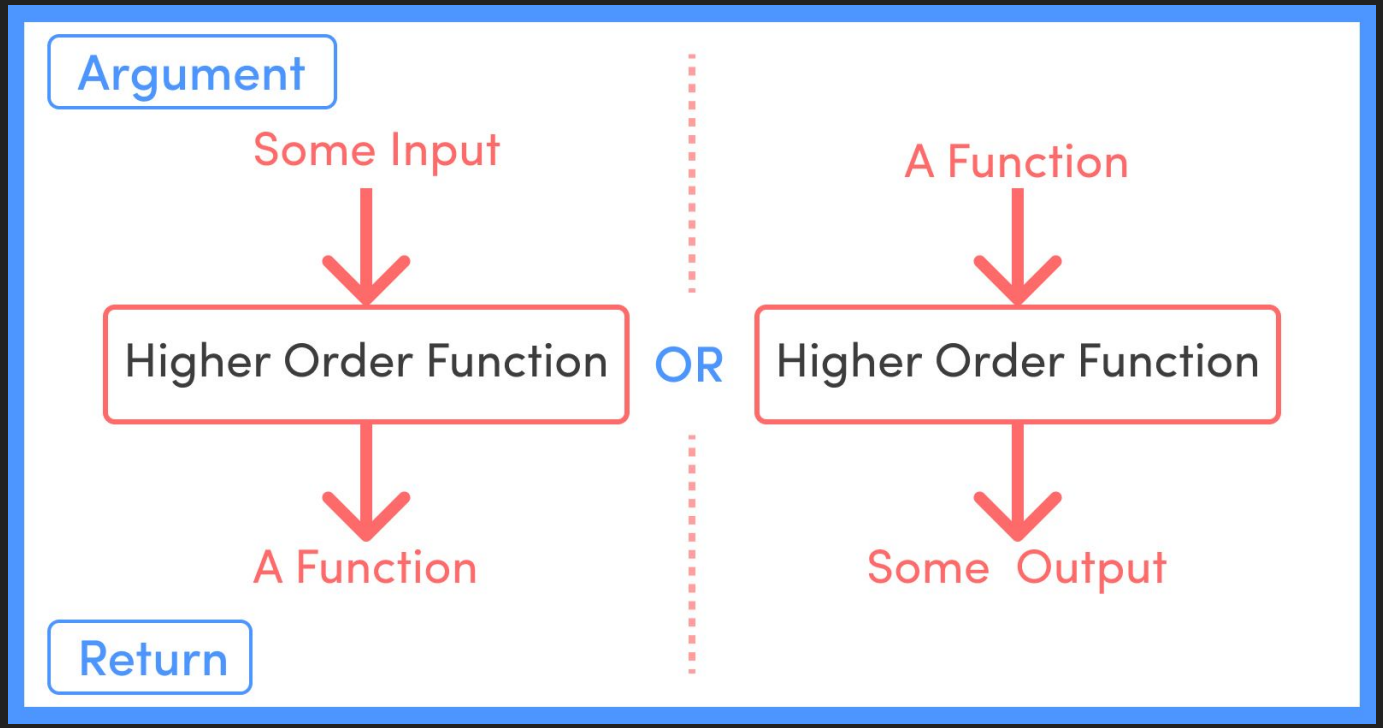
Higher-Order Functions

C++ Edition

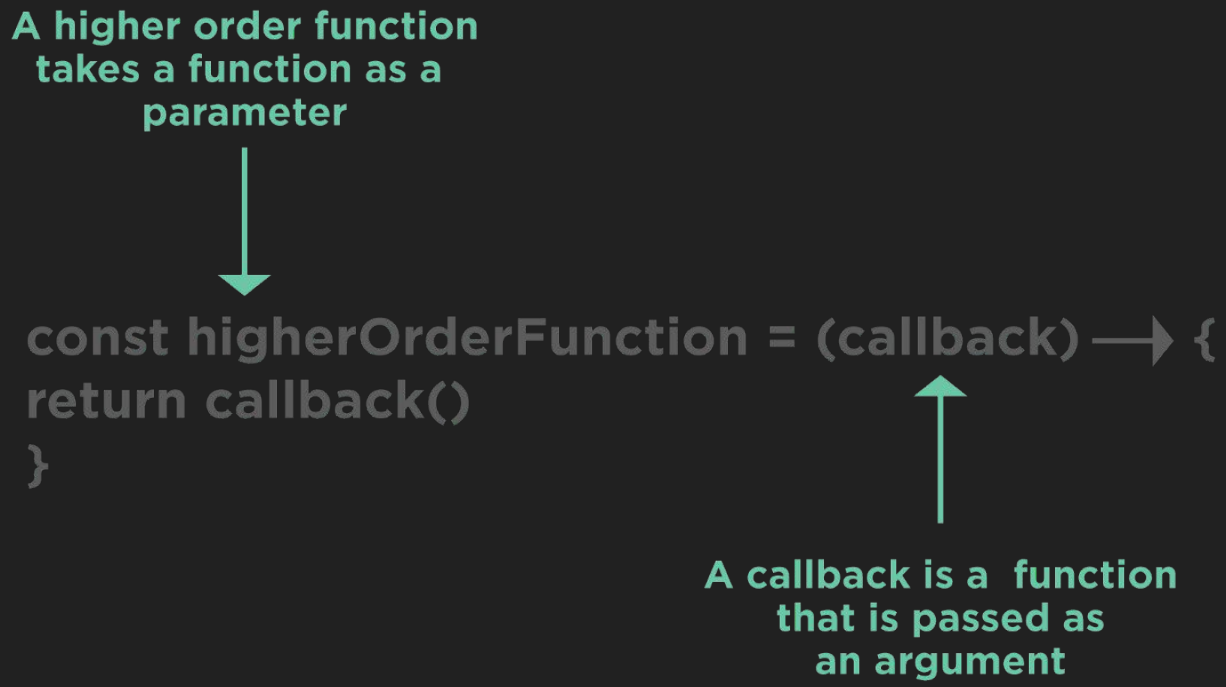
by Tommy Hoang and Jake Davis

What are Higher-order Functions?



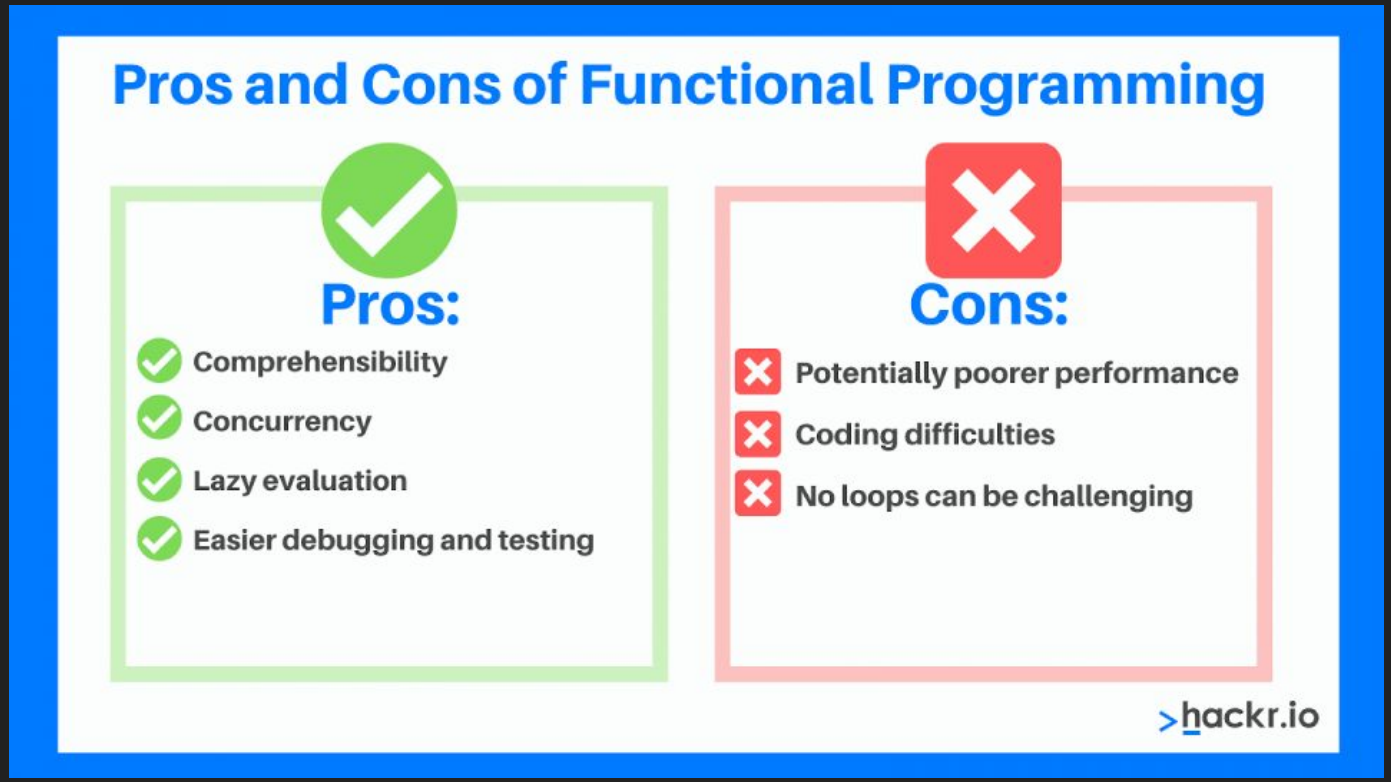
https://www.freecodecamp.org/news/higher-order-functions-in-javascript/

Callback Functions



https://medium.com/@gaathapaul/what-is-a-callback-function-bd6d893f4c59

Functional Programming



https://hackr.io/blog/functional-programming

FUNCTIONS ARE VALUES!!!

Examples From CSCI 3155

```
def foldLeftAndThen[A,B](t: Tree)(z: A)(f: (A,Int) => A)(sc: A => B): B = {
  def loop(acc: A, t: Tree)(sc: A => B): B = t match {
      case Node(l, d, r) => loop(acc, l)((acc) => loop(f(acc, d), r)(sc))
      case Empty => sc(acc)
  }
  loop(z, t)(sc)
}
```

Higher-order Functions in C++?

Workaround

Higher-order Function ->

Lambda ->

```
void Foo()
{
  vector<int> v;
  v.push_back(1);
  v.push_back(2);
  v.push_back(3);
  for_each(begin(v), end(v), [](int i) {
    cout << i << " ";
  });
}
// Outputs:
// 1 2 3
```

https://learn.microsoft.com/en-us/archive/msdn-magazine/2012/august/c-functional-style-programming-in-c

transform()

```
int square_plus_one(int n){
    return n*n+1;
}
```

```
46      vector<int> nums{1,2,3,4,5,6,7,8,9};
47
48      cout<<"Before any functions:";
49      for(vector<int>::iterator i=nums.begin();i!=nums.end();i++){
50          cout<<" "<<*i<<" ";
51      }
52      cout<<endl;
53
54      //applies square_plus_one to nums and sets it to transform
55      transform(nums.begin(),nums.end(),nums.begin(),square_plus_one);
56
57      //prints [2,5,10,17,26,37,50,65,82]
58      cout<<"After transform:";
59      for(vector<int>::iterator i=nums.begin();i!=nums.end();i++){
60          cout<<" "<<*i<<" ";
61      }
62      cout<<endl;
```

remove_if()

```
bool is_odd(int n){
    return n%2==1;
}
```

```
//vector to array for second example
int nums_arr[9];
copy(nums.begin(),nums.end(),nums_arr);
int* first=nums_arr;
int* last=nums_arr+sizeof(nums_arr)/sizeof(int);


//applies is_odd to num_arr and returns the last location that is_odd returns false
last=remove_if(first,last,is_odd);

//prints [2,10,26,50,82]
cout<<"After remove_if:";
for(int* i=first;i!=last;i++){
    cout<<" "<<*i<<" ";
}
cout<<endl;
```

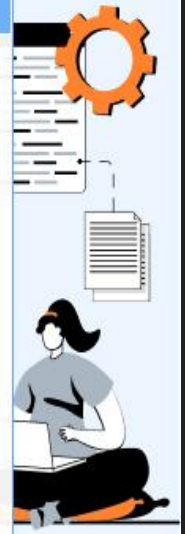
Classes and Polymorphism

```
class Calculator{
    public:
        virtual int operator()(int a, int b) const{
            return 0;
        }
};

class Add: public Calculator{
    public:
    int operator()(int a, int b) const{
        return a+b;
    }
};

class Multiply: public Calculator{
    public:
    int operator()(int a, int b) const{
        return a*b;
    }
};

int operate(const Calculator& calc, int a, int b){
    return calc(a,b);
```

```
//classes to overload Calculator's original function
Add add;
Multiply mult;

cout<<"Addition of 2 and 7: "<<operate(add,2,7)<<endl;
cout<<"Multiplication of 2 and 7: "<<operate(mult,2,7)<<endl;
```

Benefits of Using Higher-order Functions

Object-oriented programming	Functional programming
• It is used for executing fewer operations with common behavior and different variants. • It has a stateful programming model. • It focusses on how are doing. • It supports abstraction over data only. • Conditional statements can be used like switch and if-else statements. • A state exists. • An object is the primary manipulation unit. • Methods can have side effects and may impact processors.	• It is used for executing many different operations for which the data is fixed. • It has a stateless programming model. • It focusses on what we are doing. • It supports abstraction over data and behavior. • It doesn't support conditional statements. • A state doesn't exists. • Function is the primary manipulation unit. • Functions don't affect the code.



Thank you for watching!