Functional Programming in Other Languages

David Letscher

Saint Louis University

Programming Languages

A Computational Framework

Map-Filter-Reduce

Many computational problems can be reduced to this model.

- Extremely expressive and clean logic

A Computational Framework

Map-Filter-Reduce

Many computational problems can be reduced to this model.

- Extremely expressive and clean logic
- Highly parallelizable: map and filter can be trivialy parallelized and run on many machines. Reduce sometimes can be parallelized.
    - MapReduce computational framework (Google), Hadoop (Amazon), ...
    - More general computational pipelines: Spark

A Computational Framework

Map-Filter-Reduce

Many computational problems can be reduced to this model.

- Extremely expressive and clean logic
- Highly parallelizable: map and filter can be trivialy parallelized and run on many machines. Reduce sometimes can be parallelized.
  - MapReduce computational framework (Google), Hadoop (Amazon), ...
  - More general computational pipelines: Spark
- Support in many languages
  - Python, C++, Java, JavaScript, ...

What is needed to incorporate functional programming?

Compiler support

Tail optimization to avoid issues with stack recursion depth limits.

Language features

- First-class functions
- Higher-order functions
- Currying and binding
- Immutable data
- Pure functions
- Lazy evaluation
- Functors, monads, ...

Python

Can we use a functional programming paradigm in Python?

Python

Can we use a functional programming paradigm in Python?

Yes!

Python

Can we use a functional programming paradigm in Python?

Yes!

Let's examine syntatical and library support for functional programming.

Python: Functions

First-class objects Everything (literally) in Python is an object, including functions, so they can be stored in variables, passed as parameters, etc.

Python: Functions

First-class objects Everything (literally) in Python is an object, including functions, so they can be stored in variables, passed as parameters, etc.

Pure functions Cannot have side effects...

Python: Functions

First-class objects
: Everything (literally) in Python is an object, including functions, so they can be stored in variables, passed as parameters, etc.

Pure functions
: Cannot have side effects...can do this by avoiding classes, globals, and do the use compromisses for I/O.

Lambda Functions

Tranditional function definition

```
def add(a, b):
  return a + b
```

Lambda Functions

Tranditional function definition

```
def add(a, b):
  return a + b
```

Using Lambda expressions

add = lambda a, b : a + b

Lambda Functions

Tranditional function definition

```
def add(a, b):
  return a + b
```

Using Lambda expressions

add = lambda a, b : a + b

Function can be made anonymous:

```
(lambda a, b: a + b)(3,4)
```

An Example

Key-based sorting

`sorted` takes an optional parameter key, which is a function to get a sort key for each object.

An Example

Key-based sorting

sorted takes an optional parameter key, which is a function to get a sort key for each object.

```
>>> unsorted = [('b', 6), ('a', 10), ('d', 0), ('c', 4
>>> print(sorted(unsorted, key=lambda x: x[1]))
[('d', 0), ('c', 4), ('b', 6), ('a', 10)]
```

An Example

Key-based sorting

sorted takes an optional parameter key, which is a function to get a sort key for each object.

```
>>> unsorted = [('b', 6), ('a', 10), ('d', 0), ('c', 4
>>> print(sorted(unsorted, key=lambda x: x[1]))
[('d', 0), ('c', 4), ('b', 6), ('a', 10)]

>>> words = ['Hello', 'how', 'are', 'you', 'today']
>>> print(sorted(words, key=lambda x: len(x))
['how', 'are', 'you', 'Hello', 'today']
```

Function Partials

What to specify some parameters (think about how functions with multiple parameters work in Haskell).

Function Partials

What to specify some parameters (think about how functions with multiple parameters work in Haskell).

```
from functools import partial, map, filter, reduce

def add(a, b):
  return a + b

add_two = partial(add, 2)

print(add_two(4)) # Prints 6
```

Map

Squaring a list

```
numbers = [1, 2, 3, 4, 5]

squared = []
for i in numbers:
  squared.append(i*i)
```

With list comprehension

```
numbers = [1, 2, 3, 4, 5]

squared = [i*i for i in numbers]
```

Map

Using map

```
numbers = [1, 2, 3, 4, 5]
square = lambda x: x*x

squared = map(square, numbers)
# An iterable object is returned
squared = list(map(square, numbers))
```

Filter

Only the odd numbers

```python
numbers = [1, 2, 3, 4, 5]

odd = []
for i in numbers:
  if i % 2 == 1:
    odd.append(i)
```

With list comprehension

```python
numbers = [1, 2, 3, 4, 5]

odd = [ i for i in numbers if i % 2 == 1]
```

Filter

Using filter

```
numbers = [1, 2, 3, 4, 5]
odd = lambda x: x % 2 == 1

squared = filter(odd, numbers)
# An iterable object is returned
squared = list(filter(odd, numbers))
```

Reduce

Sum the list

```
numbers = [1, 2, 3, 4, 5]

total = 0
for i in numbers:
  total += i
```

Reduce

Using reduce

```
numbers = [1, 2, 3, 4, 5]
add = lambda a, b: a+ b

total = reduce(add, numbers)
```

Putting it all togethers

Adding the squares of all of the odd numbers

```
numbers = [1, 2, 3, 4, 5]
add = lambda a, b: a+ b
square = lambda x: x*x
odd = lambda x: x % 2 == 1

reduce(add, map(square, filter(odd, numbers)))
```

Higher Order Functions

It's fairly easy to define higher order functions

Higher Order Functions

It's fairly easy to define higher order functions

Defining map

This is an implemention of map (that's not as good as the library version)

```
def map(function, data):
  return [ function(x) for x in data ]
```

More Complicated Example: Square Root Estimation

```
def average(x, y):
  return (x+y)/2

def improve(update, close, guess=1):
  while not close(guess):
    guess = update(guess)
  return guess

def approx_eq(x, y, tolerance=1e-5):
  return abs(x - y) < tolerance

def sqrt(a):
  def sqrt_update(x):
    return average(x, a/x)

  def sqrt_close(x):
    return approx_eq(x*x, a)

  return improve(sqrt_update, sqrt_close)
```

What About C++?

Not as clean of syntax but extremely powerful from a computational perspective.

What is needed to incorporate functional programming? (Revisited)

Compiler support

Tail optimization to avoid issues with stack recursion depth limits.

Language features

- First-class functions
- Higher-order functions
- Currying and binding
- Immutable data
- Pure functions
- Lazy evaluation
- Functors, monads, ...

First-class functions

Function pointers in C

```
double cm_to_inches(double cm) {
  return cm / 2.54;
}

double apply(double (*f)(double), double x) {
  return f(x);
}

int main(void) {
  double (*func1)(double) = cm_to_inches;
  double meter_in_inches = cm_to_inches(100);

  double meter_in_inches2 = apply(cm_to_inches, 100);
}
```

First-class functions in C++

Function objects

```
class square {
public:
  double operator()(double x) {
    return x*x;
  }
};
```

Can be passed as parameters to other functions, methods, ...

First-class functions in C++-11

Language enhancements

- Lambda functions
- `auto` keyword
- `std::function`
- `std::bind`

Lambda functions

```
[] (double x, double y) { return x + y; }
```

Return type is deduced by the compiler, if possible.

Lambda functions

```
[] (double x, double y) { return x + y; }
```

Return type is deduced by the compiler, if possible.

Return type specified

```
[] (double x, double y) -> double { return x + y; }
```

http://en.cppreference.com/w/cpp/language/lambda

What are function types?

```
auto add = [] (double x, double y)
    -> double { return x + y; }
add(2,3);
```

What are function types?

```
auto add = [] (double x, double y)
    -> double { return x + y; }
add(2,3);
```

What type is add?

```
std::function<int(int, int)>
```

http://en.cppreference.com/w/cpp/utility/functional/function

Example: performing arithmetic

```
map<const char, function<double(double, double)>>
    functionTable;

functionTable['+'] =
    [](double x, double y) { return x + y; }
functionTable['-'] =
    [](double x, double y) { return x - y; }
functionTable['*'] =
    [](double x, double y) { return x * y; }
functionTable['/'] =
    [](double x, double y) { return x / y; }

functionTable['^'] = std::pow;
```

Example: performing arithmetic

```
cout << functionTable['*'](3., 4.5) << endl;
cout << functionTable['^'](3., 4.5) << endl;
```

Imagine parsing a string, tokenizing it and using the function table to perform the calculations. Avoids lots of cases.

Higher-order functions

Three common patterns:

Map Apply a function to all elements of a container.
`map` in Haskell

Filter Remove elements of a container not meeting a condition.
`filter` in Haskell

Reduce Accumulate values from a container.
`foldl`, `foldr` in Haskell

Map in C++

Uses `std::transform`.

`http://en.cppreference.com/w/cpp/algorithm/transform`

Squaring all entries in a list

```
list<int> numbers = {0, 1, 2, 3, 4, 5};

auto square = [](int n) { return n*n; }
list<int> squaredNumbers;

transform(numbers.begin(), numbers.end(),
    squaredNumbers.begin(), square);
```

Result: `{0, 1, 4, 9, 16, 25}`

Filter in C++

Use `std::remove_if`.
http://en.cppreference.com/w/cpp/algorithm/remove

Keep the odd numbers

```
list<int> numbers = {0, 1, 2, 3, 4, 5};

remove_if(numbers.begin(), numbers.end(),
    [](int n) { return n \% 2 == 0; } );
```

Result: {0, 2, 4}.

Filter in C++: No Mutation

or `std::remove_copy`.
http://en.cppreference.com/w/cpp/algorithm/remove_copy

Without changing the original

```
list<int> numbers = {0, 1, 2, 3, 4, 5};

list<int> filteredNumbers;
remove_copy_if(numbers.begin(), numbers.end(),
    filteredNumbers.begin(),
    [](int n) { return n \% 2 == 0; } );
```

Reduce in C++

Uses `std::accumulate`.

http://en.cppreference.com/w/cpp/algorithm/accumulate

Sum a list of numbers

```
list<int> numbers = {0, 1, 2, 3, 4, 5};

int sum = accumulate(numbers.begin(), numbers.end(),
    0, [](int x, int y) { return x+y; });
```

Result: 15.

Function binding in C++

http://en.cppreference.com/w/cpp/utility/functional/bind

```
int foo(string s, int n, list<int> l);

auto f1 = std::bind(foo, "Hello", _1, _2);
auto f2 = std::bind(foo, _2, _3, _1);
```

Putting it Together: Mutating

Adding the squares of all of the odd numbers

```
list<int> numbers = {0, 1, 2, 3, 4, 5};

auto add = [](int x, int y) { return x+y; };
auto square = [](int x) { return x*x; };
auto even = [](int n) { return n \% 2 == 0; };

remove_copy_if(numbers.begin(), numbers.end(), even);
transform(numbers.begin(), numbers.end(),
    numbers.begin(), square);
int total = accumulate(numbers.begin(), numbers.end(),
    0, add);
```

Putting it Together: Non-mutating

Adding the squares of all of the odd numbers

```
list<int> numbers = {0, 1, 2, 3, 4, 5};
list<int> oddNumbers;
list<int> squaredNumbers;

auto add = [](int x, int y) { return x+y; };
auto square = [](int x) { return x*x; };
auto even = [](int n) { return n \% 2 == 1; };

remove_copy_if(numbers.begin(), numbers.end(),
    oddNumbers.begin(), even);
transform(oddNumbers.begin(), oddNumbers.end(),
    squaredNumbers.begin(), square);
int total = accumulate(squaredNumbers.begin(),
    squaredNumbers.end(), 0, add);
```

Pure functions, immutable data

Pure functions

- Like Python, avoid behaviors that could cause side effects
- Compile time tests (like those in clang) can detect non-pure functions

Immutable data

This can be done with care.

Lazy evaluation

In conditionals, etc. C++ already does this.

Compile time programming

Functional programming at compile time!