What is Functional Programming
## Elements of Functional Programming

<table>
<thead>
<tr>
<th>Pure Functions</th>
<th>Currying</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Class Functions</td>
<td>Memoization</td>
</tr>
<tr>
<td>Higher-Order Functions</td>
<td>Destructuring</td>
</tr>
<tr>
<td>Immutability</td>
<td>Collection Pipelines</td>
</tr>
<tr>
<td>Lazy Evaluation</td>
<td>List Compressions</td>
</tr>
<tr>
<td>Recursion</td>
<td>Raw Data + functions</td>
</tr>
</tbody>
</table>
Raw Data + functions

class Person {
    private String firstName;
    private String lastName;
    private int age;

    {:first-name "Roger"
     :last-name "Whitney"
     :age 21 }

    filter (select), remove
    map (fold)
    reduce
    transducers (Clojure 7)
# Cycles required to Fetch Data

<table>
<thead>
<tr>
<th></th>
<th>~40 per core, sort of</th>
<th>0 cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1</td>
<td>32KB per core</td>
<td>64B line</td>
</tr>
<tr>
<td>L2</td>
<td>256KB per core</td>
<td>64B line</td>
</tr>
<tr>
<td>L3</td>
<td>6MB</td>
<td>64B line</td>
</tr>
<tr>
<td>Main memory</td>
<td>8GB</td>
<td>4KB page</td>
</tr>
</tbody>
</table>

Locality of data helps keep data in same cache
Array of integers

Array of objects

Tuesday, September 29, 15
Pure Functions

Functions with no side-effects

Only depend on arguments

Don't change state

Why important

Easier to debug
test
understand program

class Foo {
    int bar

    public int notPure(int y) {
        return bar + y
    }

    public void alsoNotPure(int y) {
        bar = y
    }

    // OO makes code understandable by encapsulating moving parts.
    // FP makes code understandable by minimizing moving parts.

    Michael Feathers
First Class Functions

Functions can be

- Assigned to variables
- Passed as arguments
- Returned from functions

Anonymous functions

- Lambdas
- Closures

Why important

- Flexibility
- Generality
Java Lambda Expression

Anonymous Function

```
(Integer a, Integer b) -> a + b
```

```
(Integer start, Integer stop) -> {
    for (int k = start; k < stop; k++)
        System.out.println(k);
}
```
Short Version of Java Lambda Syntax

(String text) -> text.length();

(Integer a, Integer b) -> a + b

text -> text.length();

(a, b) -> a + b
Using Java Lambdas

```java
Function<String, Integer> length = text -> text.length();
int nameLength = length.apply("Roger Whitney");

BiFunction<Integer, Integer, Integer> adder = (a, b) -> a + b;
int sum = adder.apply(1, 2);
```
onClickExample

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View source) {
        makeToast();
    }
});

button.setOnClickListener( event -> makeToast());
Higher-Order Functions

Functions that operate on functions

Why important

map
reduce
filter
comp
partial
complement

Fewer details/
higher level logic

Concurrency
Java Stream methods

count()
distinct
filter
findAny
findFirst
flatMap
forEach
forEachOrdered
limit
map
max
min
nonMatch
reduce
sorted
Immutability

Data structures can not be modified

Like Java's Strings

Why important

Concurrency

No need for private data

Easier to
debug
test
understand program

OO makes code understandable by encapsulating moving parts.
FP makes code understandable by minimizing moving parts.

Michael Feathers
Java Immutability

Strings

Collections.unmodifiableList(List<? extends T> list)
Collections.unmodifiableMap(Map<? extends K, ? extends V> m)
Collections.unmodifiableSet(Set<? extends T> s)
Collections.unmodifiableSortedMap(SortedMap<K, ? extends V> m)
Collections.unmodifiableSortedSet(SortedSet<T> s)
Lazy Evaluation

Operations & functions evaluated
When used
Not when called

Why important
Simplifies logic
Java Lazy Evaluation

String[] words = {"a", "ab", "abc", "abcd", "bat"};
List<String> wordList = Arrays.asList(words);
List<String> longWords = wordList.stream()
    .filter( s -> s.length() > 2)
    .filter( s -> s.charAt(0) == 'a')
    .map( s -> s.toUpperCase())
    .collect(Collectors.toList());

System.out.println(longWords);

Only One pass of List to do all operations
Recursion

function factorial(n) {
    if n = 1 return 1
    return n * factorial(n-1)
}

Tail recursion/Tail Call Optimization

When last statement is just the recursion
Compiler can convert recursion into loop

Why important
Powerful tool
Currying

function add(int x, int y) {
    return x + y;
}

addTen = add(10);

addTen(3) //returns 13
Memoization

Cache value of functions

memoize(factorial)

factorial(1000) // 1000 recursive calls
factorial(1001) // 1 recursive call

Why important

Performance
Collection Pipelines

String[] words = {"a", "ab", "abc", "abcd", "bat"};
List<String> wordList = Arrays.asList(words);
List<String> longWords;
longWords = wordList.stream()
  .filter( s -> s.length() > 2)
  .filter( s -> s.charAt(0) == 'a')
  .map( s -> s.toUpperCase())
  .collect(Collectors.toList());

(->> ["a", "ab", "abc", "abcd", "bat"]
  (filter #(< 2 (count %)))
  (filter #(= \a (first %)))
  (map clojure.string/upper-case))
Some Java
## Accessing Static Methods & Fields

<table>
<thead>
<tr>
<th>Static Fields</th>
<th>Static Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class/fieldName</td>
<td>(Class/methodName arg1 arg2 …)</td>
</tr>
<tr>
<td>Math/PI</td>
<td>(Double/parseDouble &quot;3.14159&quot;)</td>
</tr>
<tr>
<td>Float/MAX_VALUE</td>
<td>(Integer/toBinaryString 3)</td>
</tr>
</tbody>
</table>
Accessing Java instance methods

(.instanceMethod object arg1 ...)

(.toUpperCase "cat")

(.isEmpty [1 2 3])

(.size [1 2 3])

(.get [1 2 3] 1)
Examples

(defn decimal-to-hex [x]
  (-> x
      Integer/parseInt
      (Integer/toString 16)
      .toUpperCase))

(def iterator (.iterator [1 2 3]))
(while (.hasNext iterator)
  (println (.next iterator)))
Exceptions

(defn as-int
 [s]
 (try
  (Integer/parseInt s)
  (catch NumberFormatException e
   (.printStackTrace e))
  (finally
   (println "Attempted to parse as integer: " s))))
Raising an Exception

(throw (IllegalStateException. "I don't know what to do!"))
Common Exceptions

java.lang.IllegalArgumentException

java.lang.UnsupportedOperationException

java.lang.IllegalStateException

java.io.IOException

java.io.IOException

Text claims that these handle 90% of cases where you need exceptions
When to Use Exceptions?

Googles answer:

Exceptions should be used for situation where a certain method or function could not execute normally.

Does this mean nil nodes in a tree?
Multimethods
Example

```
(defmulti even-odd even?)
(defmethod even-odd true [n] (str n " is even"))
(defmethod even-odd false [n] (str n " is odd"))
```

```
(even-odd 5)  5 is odd  
(even-odd 4)  4 is even
```
Example

(defmulti even-odd even?)

(defmethod even-odd true [n] (str n " is even"))

(defmethod even-odd false [n] (str n " is odd"))
Default values

(defmulti fibonacci identity)

(defmethod fibonacci 0 [n] 0)

(defmethod fibonacci 1 [n] 1)

(defmethod fibonacci :default [n] (+ (fibonacci (dec n)) (fibonacci (- n 2))))

(fibonacci 1) 1
(fibonacci 10) 55
Dispatch Function can be any function

(defmulti types class)
(defmethod types java.lang.String [x] "it is a string")

(defmethod types java.lang.Long [x] "it is a Long")

(defmethod types :default [x] "Don't know")
(defmulti by-size (fn [a b] (size a)))

(defmethod by-size :small
  [x y]
  "small")

(defmethod by-size :medium
  [x y]
  "medium")

(defmethod by-size :default
  [x y]
  "large & other")

(defn size [x]
  (cond
    (< x 5) :small
    (< x 20) :medium
    (< x 100) :large))

(by-size 2 20)   "small"
(by-size 10 20)  "medium"
Vectors as Match

(defmulti by-size (fn [a b] [(size a) (size b)]))

(by-size 2 90) "small-large"
(by-size 10 20) "other"

(defmethod by-size [:small :small]
  [x y]
  "small-small")

(defmethod by-size [:small :large]
  [x y]
  "small-large")

(defmethod by-size [:medium :medium]
  [x y]
  "medium-medium")

(defmethod by-size :default
  [x y]
  "other")
Warning about defmulti

defmulti is define once

If you need to modify your defmulti need to remove it from the bindings

In previous example used

(ns-unmap *ns* 'by-size)
One Last Example

(defmulti by-children (fn [[a c b]] [(nil? b) (nil? c)]))

(defmethod by-children [true true] [x] "no children")

(defmethod by-children [true false] [x] "right child")

(defmethod by-children [false true] [x] "left children")

(defmethod by-children [false false] [x] "both children")

(by-children [1 4 nil]) "right child"  
(by-children [1 nil nil]) "no children"
Open-Closed Principle

"software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification"

Wikipedia