A Primer on Functional Programming

Sarah Withee

KC People:
What is your favorite experience at KCDC?

Other cities:
What's your favorite KC experience so far?

A Primer on Functional Programming

Sarah Withee

Monads

(jk)

Poll

Who has heard of functional programming?

Poll

Who has done functional programming?

Poll

Who IS a functional programmer?

Poll

Who has wanted to learn but never had time or good resources?

Intro

1. Functional Programming Concepts
2. Examples
3. Why Use Functional Programming?
4. Look at Programming Languages

Background

It’s not new

(Languages and ideas been around since the 1950s)

Background

Built on ideas of lambda calculus developed in the 1930s

(I promise, I won't cover this today)

Concepts – Pure Functions

Pure Functions:

Function that, given a certain input, ***always*** produces the same output.

Concepts – Pure Functions

Pure functions don’t have side effects

Concepts – Pure Functions

Side effects include:

Time, file access, database access, previous function calls, etc.

Concepts – Pure Functions

User input is never pure (duh)

Concepts – Pure Functions

Call by reference is never pure

Concepts – Pure Functions

Nearly impossible to write 100% pure function programs

Concepts – Pure Functions

Examples:

`sin(x)`

Returns the sine of angle x

Concepts – Pure Functions

Examples:

`str.length()`

Returns the length of the string str

Concepts – Pure Functions

Examples:

```
getAccountNumberFromDb(name)
```

… kidding… definitely not pure

Concepts – Pure Functions

- Pure functions, when given certain input, always produce certain output
- Pure functions don't have side effects
- User input is never pure (duh)
- Call by reference is never pure
- Nearly impossible to write 100% of a project in pure functions

Concepts - Referential Transparency

Referential transparency:

Any expression that can replace a function with its return value with no behavior changes

Concepts - Referential Transparency

Example:

If x = 3...

x + 5 = 8
3 + 5 = 8

Concepts - Referential Transparency

In mathematics, all functions are transparent

In programming, this is NOT the case

Concepts - Referential Transparency

Pure functions *always* have referential transparency

Concepts - Referential Transparency

Assignments are NOT transparent

```
x = x + 1
```

Concepts - Referential Transparency

```
def addOne(int x):
      return x + 1;
```

If x and y are the same, then

```
addOne(x) == addOne(y)
```

Concepts - Lambda functions

Lambda function:

A function without a name
(Also called an anonymous function)

Concepts - Lambda functions

Usually for higher level functions or to pass arguments to one

Usually used once to a few times

Concepts - Lambda functions

Can’t be recursive*

* otherwise they need a name or some way of maintaining state**

** which is possible but outside of this scope

Concepts - Lambda functions

Example:

```
f = lambda x:  x*x
print f(5)
```

Concepts - Lambda functions

Functions ARE values

Functions can be passed as values into functions

(Warning: mind blowing example ahead)

Concepts - Lambda functions

```
def divide(x, y):
    return x/y


def divisor(d):
    return lambda r: divide (r, d)

half = divisor(2)
print half(32)
```

```
lambda r: divide(r, 2)
```

```
divide(32, 2)
```

Concepts - Extra

Monads
Closures
Functors

Outside of the scope of today

Why Use Functional Languages?

Pure functions are simpler and faster to write

Why Use Functional Languages?

Pure functions that work correctly will *always* work correctly

Why Use Functional Languages?

Stack traces are a pain in OOP

Stack traces in FP simplify things

Why Use Functional Languages?

No side effects makes unit tests pass reliably

Why Use Functional Languages?

Global state of program isn’t affected by pure functions

Why Use Functional Languages?

Concurrency is WAY easier

Why Use Functional Languages?

Code ends up better as functions are designed better

Better small modules -> better large modules

Why Use Functional Languages?

- Pure functions are simpler and faster to write
- Pure functions that work correctly will *always* work correctly
- Stack traces in FP simplify things
- No side effects makes unit tests pass reliably
- Global state of program isn't affected by pure functions
- Concurrency is WAY easier
- Functions are designed better
- Better small modules -> better large modules

Activity 1

In a moment, everyone will stand up.

1. Start at the beginning of the room with 0
2. Take the previous number, add 1 to it
3. Say the number
4. Sit down
5. Last person reports the total

Activity 2

In a moment, everyone will stand up.

1. Find a neighbor
2. Take their total and add 1
3. Neighbor sits down
4. Return your result to person next to you
5. You sit down
6. Extra volunteer will count the array of results (end of each row), add them up
7. Volunteer will return the final result

Example of Functional Thinking

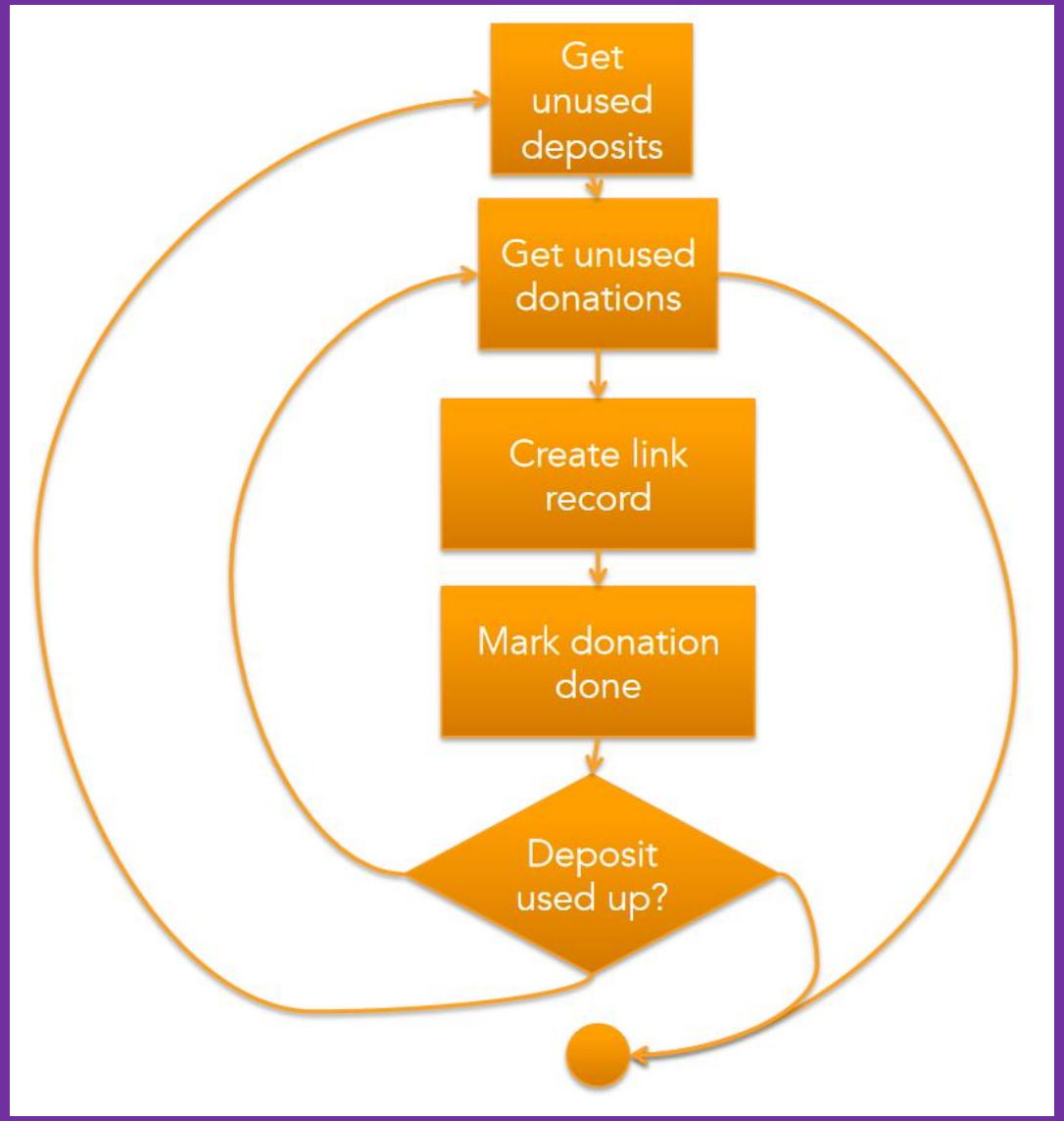
Activity 1 resembles a for or while loop

- `x = x + 1` type thought
- Took a long time
- n steps

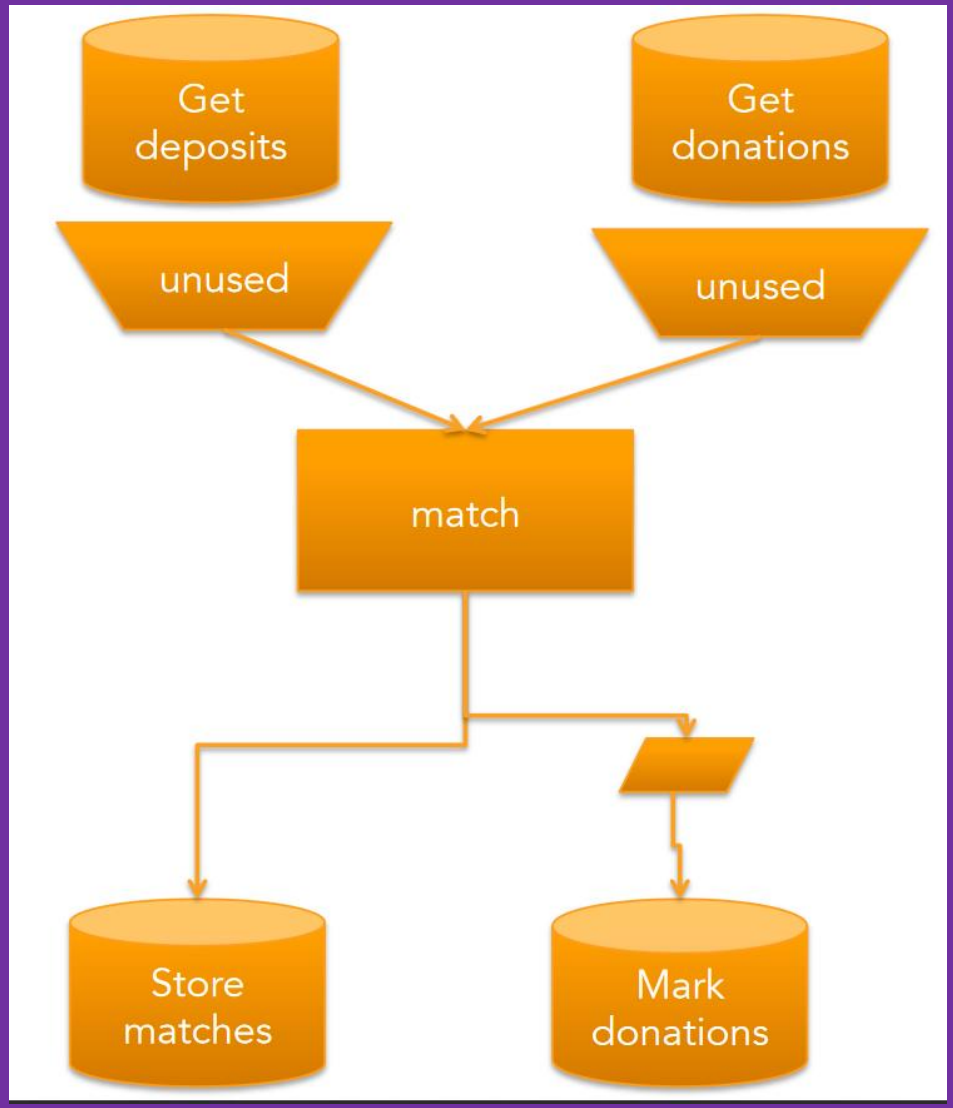
Example of Functional Thinking

Activity 2 resembles concurrent recursive function

- ```
  def countPerson (theirNum):
      return theirNum + 1
  def rows (arrayValues):
      return sum(arrayValues) + 1
  ```
- Multiple sets counted at the same time
- Much faster and fewer steps
  ```



Curtesy of Jessica Kerr (@jessitron)



Curtesy of Jessica Kerr (@jessitron)

List of Functional Languages (Pure)

Agda
Charity
Clean
Coq
Curry
Elm
Frege
Haskell
Hope
Joy
Mercury
Miranda
Idris
SequenceL

List of Functional Languages (Not Pure)

APL
ATS
CAL
C++ (since C++11)
C#
Ceylon
D
Dart
ECMAScript
- ActionScript
- ECMAScript for XML
- JavaScript
- Jscript

Erlang
- Elixir
- LFE

F#
FPr
Groovy
Hop
J
Java (since Java 8)
Julia
Lisp
- Clojure
- Common Lisp
- Dylan
- Emacs Lisp
- LFE
- Little b
- Logo
- Scheme
- Racket
- Tea

Mathematica
ML
- Standard ML
- Alice
- Ocaml

Nemerle
Opal
OPS5
Poplog
Python
Q
R
Ruby
REFAL
Rust
Scala
Spreadsheets

Languages - Elm

- Pure functional language
- Statically typed (primitive types, lists, tuples, records, unions)
- Immutable types (keeps data pure by making you create new variables)
- No runtime exceptions (compiler finds them first)
- Super friendly error messages
- Compiles to JavaScript for the browser

Languages - Haskell

Pure functional language
Statically typed, type inference
Lazy evaluation and pattern matching

Languages - LISP

- “LISt Processor”
- (Known as the language with all the parentheses)
- NOT a pure functional language
- Dynamically typed (mostly lists of any type)
- Solve on first item in list, recursively solve on rest of list

Languages - Clojure

Dialect of LISP

Dynamically typed

Runs on Java Virtual Machine (JVM)

Used by Amazon, Capital One, Cerner, Groupon, Spotify, many others

Languages – F#

Functional and Object Oriented (compiles into .Net)

Based on Ocaml and C#

Strongly typed, but inferred

Every statement returns a type

Parallelism is easily built into language

Great for data analysis

Conclusion

Functional programming is getting popular, but been around for decades

Functional principles makes your code simpler, smaller, and more reliable

There's many functional languages so you can find one that's similar to your favorite language

Thank You!

Sarah Withee
@geekygirlsarah
sarah@sarahwithee.com

I'd love to hear how you're using your new knowledge!