A Primer on Functional Programming

Sarah Withee
@geekygirlsarah

Slides: geekygirlsarah.com/primer-fp

Who has heard of functional programming?

Who has done some form of functional programming?

Who IS a functional programmer?

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

Intro

1. Functional Programming Concepts
2. Why Use Functional Programming?
3. Brief Glance at Functional Programming Languages

Hello!

I am Sarah Withee

I'm a software engineer at Arcadia
I'm on social media as @geekygirlsarah

(Yes, you can tweet all the things!)

1. Functional Programming Concepts

Background

It's not new!

(Languages and ideas have been around since 1950s)

Background

Built on ideas of lambda calculus developed in the 1930s

(I promise, we're not discussing this today)

Pure Functions:

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

Functional Programming Concepts

Pure functions don't have side effects

Functional Programming Concepts

Side effects include:

- Time
- File access
- Database access
- Network access
- Previous function calls

Functional Programming Concepts

User input is never pure

Duh.

Functional Programming Concepts

Call by reference is never pure

Functional Programming Concepts

Basically impossible to write software using 100% pure functions

Functional Programming Concepts

```
sin(x)
abs(x)
sqrt(x)
```

Always returns same values for x

All math functions do this

Functional Programming Concepts

```
str.length()
str.isEmpty()
str.concat(str2)
```

Always returns same values for the string str (and str2)

Generally functions based on simple data types will be like this

Functional Programming Concepts

```
getAccountNumberFromDb(acctOwner)
```

Kidding... definitely not pure.

Why?

- What if someone changes their name?
- What if they close an account?
- What if the number changes (like debit card stolen)?
- What if database goes down?

Review – Pure Functions

- Pure functions, given certain input, always produce same output
- No side effects
- User input is never pure
- Call by reference is never pure
- Impossible to write software in 100% pure functions

Referential Transparency

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

Functional Programming Concepts

Example:

If x = 3...

$$x + 5 = 8$$

$$3 + 5 = 8$$

Functional Programming Concepts

(note: sin 30° = 0.5, cos 60° = 0.5)

```
x = 30
y = sin(x) + cos(x*2)  # y=1
y =  0.5   +    0.5    # y=1
```

Functional Programming Concepts

Pure functions *always* have referential transparency

Functional Programming Concepts

In mathematics, all functions are transparent

In programming, this is NOT true

Functional Programming Concepts

Assignments are NOT transparent

$x = x + 1$

Functional Programming Concepts

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

If x and y are the same, then
`addOne(x)` and `addOne(y)`
give same result

Functional Programming Concepts

More languages are starting to have immutable variables by default

Functional Programming Concepts

Lambda function (or anonymous function):

A function without a name

Functional Programming Concepts

Why lambdas?

For higher level functions or to pass arguments to a function

Usually used once to few times

Functional Programming Concepts

Lambdas can't be recursive*

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

** which is possible but outside of this scope

@geekygirlsarah #MusicCityCode

geekygirlsarah.com/primer-fp

https://stackoverflow.com/questions/2067988/recursive-lambda-functions-in-c11

C++11:

```
std::function<int(int,int)> sum;
sum = [term,next,&sum](int a, int b)->int {
    if(a>b)
        return 0;
    else
        return term(a) + sum(next(a),b);
};
```

C++14:

```
void f() {
    static int (*self)(int) = [](int i)->int { return i>0 ? self(i-1)*i : 1; };
    std::cout<<self(10);
}
```

Functional Programming Concepts

variable

lambda argument

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

Function definition

x

Functional Programming Concepts

Functions ARE values

Can be passed as values into other functions

@geekygirlsarah #MusicCityCode

geekygirlsarah.com/primer-fp

- A lot of the core of functional programming
- Because of referential transparency and pure functions, this can be like passing in a regular value

```
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)

Functional Programming Concepts

Who has heard of map/filter/reduce?

Functional Programming Concepts

Who has *used* map/filter/reduce?

Functional Programming Concepts

Map – apply a function to all terms in a list

map(function, list)

- Depending on language, the order of arguments may swap

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

squared = map(lambda x: x**2, items)
                # same as
squared = []
for i in items:
    squared.append(i**2)
```

Functional Programming Concepts

Filter – creates a list for all items that match a filter (function returns true)

filter(function, list)

```
items = [1, 2, 3, 4, 5, 6, 7, 8]

under_5 = filter(lambda x: x<5, items)
                    # same as
under_5 = []
for i in items:
    if (i < 5):
        under_5.append(i)
```

Functional Programming Concepts

Reduce – returns result of some computation on a list

reduce(function, list)

```
items = [1, 2, 3, 4, 5, 6, 7, 8]

product = reduce(lambda x, y: x * y, items)
                # same as
product = 1
for i in items:
    product = product * num
```

Functional Programming Concepts

Note: Reduce works differently in different languages

reduce_left()
reduce_right()

https://twitter.com/steveluscher/status/741089564329054208

- A lot of things thrown at you
- The beginning might seem promising
- The later stuff might just be weird
- So…why?

Why Use Functional Programming?

Pure functions are simpler and faster to write

Why Use Functional Programming?

Pure functions that work correctly will *always* work correctly

Why Use Functional Programming?

Stack traces are a pain in OOP

Stack traces in FP simplify things

Why Use Functional Programming?

Unit testing IS* functional programming. No side effects make unit tests pass reliably.

* Well, should be anyway

@geekygirlsarah #MusicCityCode

geekygirlsarah.com/primer-fp

- Set up environment
- Run unit test
- Assert the result
- Clear out what you set up
- Should affect nothing else

Why Use Functional Programming?

Global state of program isn't affected by pure functions

Why Use Functional Programming?

Concurrency is WAY easier

Why Use Functional Programming?

As code grows larger, it's all more reliable

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. Each person will take the previous number, add 1 to it
3. Say the number out loud
4. Sit down
5. Last person reports the total

Activity 2

In a moment, everyone will stand up.

1. You are 1 person, so number is 1
2. Find a neighbor
3. Total your two numbers together
4. One of you sits down
5. Repeat steps 2-4 for each person in the row
6. Extra volunteer will count the array of results (end of each row), add them up
7. Volunteer will return the final result

Why Use Functional Programming?

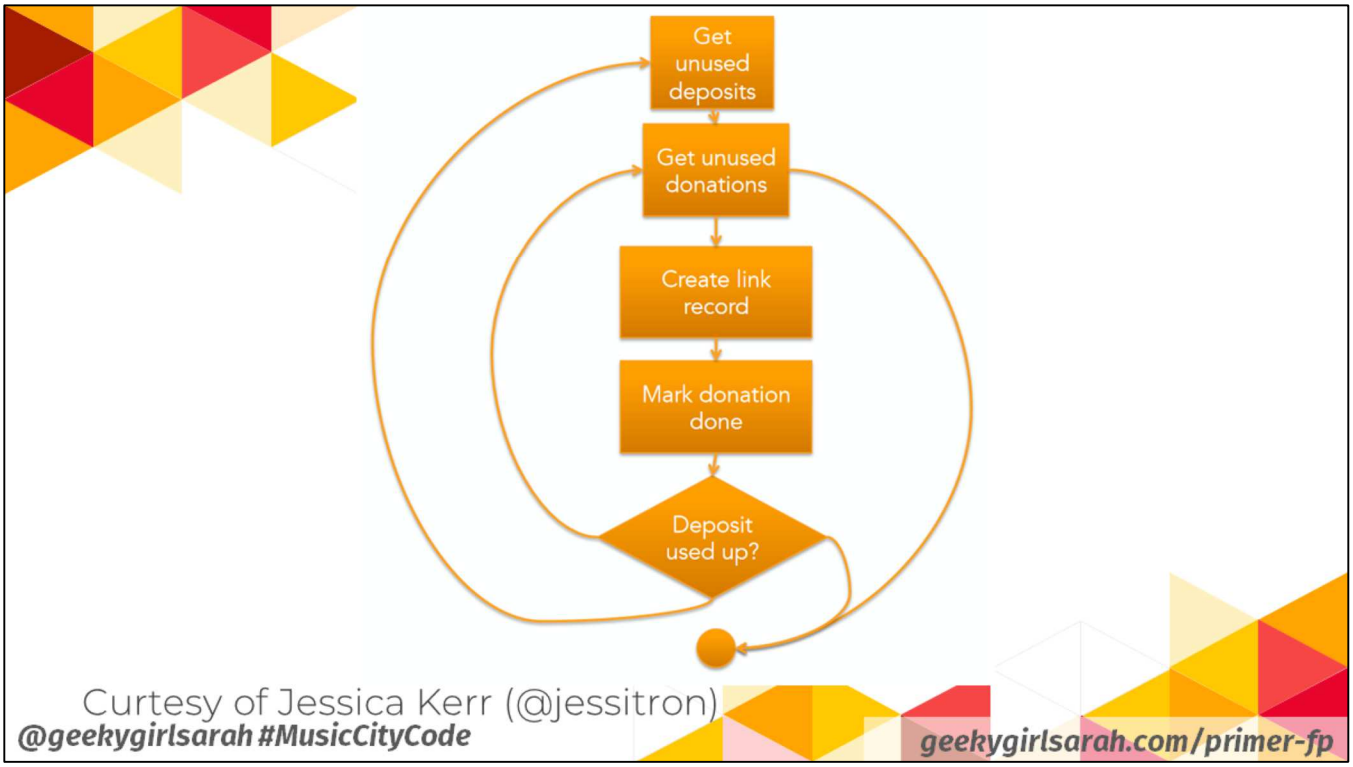
Activity 1 was like a for/while loop

- x = x + 1
- Took a long time

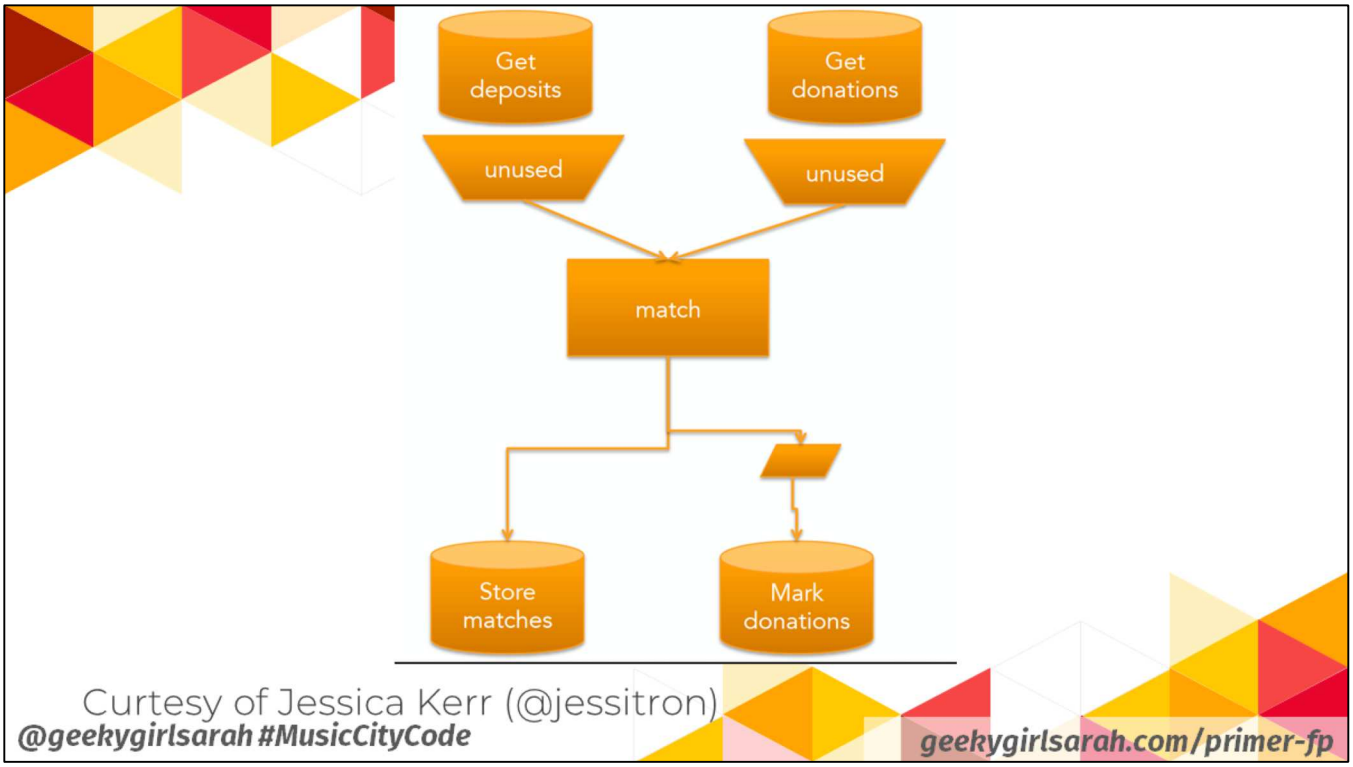
Why Use Functional Programming?

Activity 2 was recursive and concurrent

- Rows counted independently (no side effects)
- Reduce all the totals together



- A more imperitive/OOP way of doing non-profit donations



- A more imperitive/OOP way of doing non-profit donations

3.
Brief Glance at Functional Programming Languages

Functional Languages (Pure)

- Agda
- Charity
- Clean
- Coq
- Cuneiform
- Curry
- Disciple
- Elm
- Frege
- Futhark
- Haskell
- Hope
- Idris
- Joy
- KRC
- Mars
- Mercury
- Miranda
- Purescript
- SAC
- SASL
- SequenceL

@geekygirlsarah #MusicCityCode

geekygirlsarah.com/primer-fp

- 22 languages (at least)
- They don't allow side effects and guarantee referential transparency
- KRC = Kent Recursive Calculator
- SAC = Single Assignment C
- SASL = St. Andrews Static Language

Functional Languages (Impure)

- ActionScript
- Alice
- APL
- ATS
- CAL
- C++ (since C++11)
- C#
- Ceylon
- Clojure
- Common Lisp
- Curl
- D
- Dart
- Dylan
- ECMAScript
- Emacs Lisp
- Erlang
- Elixir
- F#
- Groovy
- Hop
- J
- Java (since JDK8)
- JavaScript
- JMP Scripting Language (JSL)
- JScript
- Julia
- Kotlin
- LFE
- Little b
- Logo
- Mathematica
- Nemerle
- Nim
- OCaml
- Opal
- OPS5
- Python
- Q (both of them)
- R
- Red
- Ruby
- REFAL
- Rust
- Scala
- Scheme/Racket
- Spreadsheets
- Standard ML (SML)
- Tea
- Wolfram Language

@geekygirlsarah #MusicCityCode geekygirlsarah.com/primer-fp

- 50 languages (at least)
- For the most part, they implement lambdas and passing functions around (map/filter/reduce)
- Any surprises in here?

4.
Conclusion

Conclusion

- Functional programming is getting more popular, but been around for decades
- Functional principles make your code simpler, smaller, more reliable
- Functional concepts can work in nearly any language

Thanks!

Any questions?

You can find me at

- @geekygirlsarah
- hello@sarahwithee.com

Slides:
geekygirlsarah.com/primer-fp

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Example images by Jessica Kerr (@jessitron)

Presentation design

This presentation uses the following typographies and colors:

- Titles: Montserrat bold
- Body copy: Montserrat light

You can download the fonts on this page:

https://www.fontsquirrel.com/fonts/montserrat

Yellow #ffc800 · Orange #ffa400 · Raspberry #f64646 · Crimson #e8062f