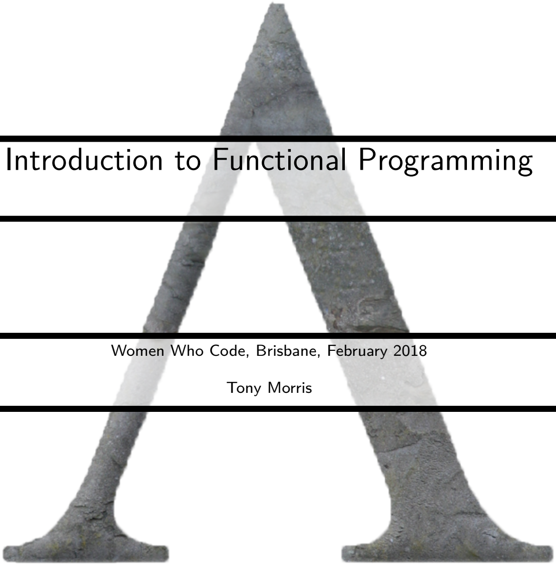


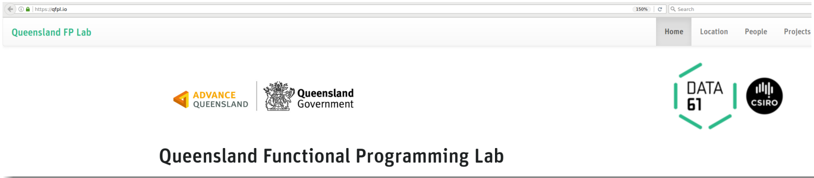
Introduction to Functional Programming

Women Who Code, Brisbane, February 2018

Tony Morris

QFPL

http://qfpl.io/



https://qfpl.io

Queensland FP Lab

Home Location People Projects

ADVANCE QUEENSLAND

Queensland Government

DATA 61 CSIRO

Queensland Functional Programming Lab

Frequently Asked Questions

FAQ

- **How can I be notified of upcoming FP courses?**
  - Subscribe to this mailing list `http://notify.qfpl.io/`
  - Sign up to YOW! conference notifications
- **Do you do non-introductory FP courses?**
  Coming in 2018. Sign up to notifications.
- **Do you *really* get paid to do whatever you want in Haskell?**
  - Yes
  - We are hiring. Wanna play?

Frequently Asked Questions

FAQ

- **How can I be notified of upcoming FP courses?**
  - Subscribe to this mailing list http://notify.qfpl.io/
  - Sign up to YOW! conference notifications
- **Do you do non-introductory FP courses?**
  Coming in 2018. Sign up to notifications.
- **Do you *really* get paid to do whatever you want in Haskell?**
  - Yes
  - We are hiring. Wanna play?

Frequently Asked Questions

FAQ

- **How can I be notified of upcoming FP courses?**
  - Subscribe to this mailing list `http://notify.qfpl.io/`
  - Sign up to YOW! conference notifications
- **Do you do non-introductory FP courses?**
  Coming in 2018. Sign up to notifications.
- **Do you *really* get paid to do whatever you want in Haskell?**
  - Yes
  - We are hiring. Wanna play?

Frequently Asked Questions

FAQ

- **How can I be notified of upcoming FP courses?**
  - Subscribe to this mailing list `http://notify.qfpl.io/`
  - Sign up to YOW! conference notifications
- **Do you do non-introductory FP courses?**
  Coming in 2018. Sign up to notifications.
- **Do you *really* get paid to do whatever you want in Haskell?**
  - Yes
  - We are hiring. Wanna play?

me

In the early 2000s, I was working for IBM, on the Java Development Kit . . .

me

navigating the principles of software engineering, I had one simple thought . . .

me

surely there is a better way and someone smarter than me has figured it out

me

I learned that yes, sound and applicable principles for software engineering have been figured out

me

It is called Functional Programming

What is Functional Programming?

What does it *mean*?

Suppose the following program . . .

```
int wibble(int a, int b) {
  counter = counter + 1;
  return (a + b) * 2;
}

/* arbitrary code */

blobble(wibble(x, y), wibble(x, y));
```

and we refactor out these common expressions . . .

```
int wibble(int a, int b) {
  counter = counter + 1;
  return (a + b) * 2;
}

/* arbitrary code */

blobble(wibble(x, y), wibble(x, y));
```

assign the expression to a value

```
int wibble(int a, int b) {
  counter = counter + 1;
  return (a + b) * 2;
}

int r = wibble(x, y);

/* arbitrary code */

blobble(r, r);
```

Did the program just change?

Yes, the program changed . . .

```
int wibble(int a, int b) {
  counter = counter + 1;
  return (a + b) * 2;
}

int r = wibble(x, y);

/* arbitrary code */

blobble(r, r);
```

Suppose this slightly different program . . .

```
int pibble(int a, int b) {
  return (a + b) * 2;
}

/* arbitrary code */

globble(pibble(x, y), pibble(x, y));
```

and we refactor out these common expressions . . .

```
int pibble(int a, int b) {
  return (a + b) * 2;
}

/* arbitrary code */

globble(pibble(x, y), pibble(x, y));
```

assign the expression to a value

```
int pibble(int a, int b) {
  return (a + b) * 2;
}

int r = pibble(x, y);

/* arbitrary code */

globble(r, r);
```

This time, did the program just change?

It's the same program

For given inputs, the same outputs are given, with no observable changes to the program

Functional Programming is the idea that

We can **always replace expressions with a value, without affecting the program behaviour**

This property of expressions is called *referential transparency*.

Consequences

A commitment to Functional Programming has many immediate consequences.

For example, no more mutable data structures

```
class Person {
    var name: String
    var address: Address
}
```

No more loops

```
for(int i = 0; i < list.length; i++)
```

No reading & writing files arbitrarily

```
contents1 = readFile("filename");
writeFile("filename", "the␣contents");
contents2 = readFile("filename");
```

So then, if all our familiar tools are taken away . . .

how do we then achieve these practical outcomes?

?

- how do we design our data structures?
- how do we write loops?
- how do we read & write files?

Let's start at a concrete example

How do I sum the integer values in a list?

Using a for loop

```
sum(list) {
  var r = 0;
  for(int i = 0; i < list.length; i++) {
    r = r + list[i];
  }
  return r;
}
```

Using a for loop

```
sum(list) {
  var r = 0;
  for(int i = 0; i < list.length; i++) {
    r = r + list[i];
  }
  return r;
}
```

Here is another way of looking at the problem

The sum of a list is . . .

- if the list is empty, return 0
- otherwise add the first element to the sum of the remainder of the list

The sum of a list is . . .

```
sum([6, 5, 9, 71, 3]) =
6 + sum (5, 9, 71, 3]) =
6 + 5 + sum([9, 71, 3]) =
6 + 5 + 9 + sum([71, 3]) =
6 + 5 + 9 + 71 + sum([3]) =
6 + 5 + 9 + 71 + 3 + sum([]) =
6 + 5 + 9 + 71 + 3 + 0 =
94
```

Here is the Haskell source code

```
sum [] = 0
sum (first:rest) = first + sum rest
```

Why?

Why would we do this? What are the practical benefits?

Why FP?

- the practical benefits are not always immediately obvious
- this is especially true when given trivial examples, such as summing a list
- but is there a point to all this?
- a benefit to throwing away familiar tools, and replacing them?

Why FP?

- the practical benefits are not always immediately obvious
- this is especially true when given trivial examples, such as summing a list
- but is there a point to all this?
- a benefit to throwing away familiar tools, and replacing them?

Why FP?

- the practical benefits are not always immediately obvious
- this is especially true when given trivial examples, such as summing a list
- but is there a point to all this?
- a benefit to throwing away familiar tools, and replacing them?

Why FP?

- the practical benefits are not always immediately obvious
- this is especially true when given trivial examples, such as summing a list
- but is there a point to all this?
- a benefit to throwing away familiar tools, and replacing them?

. . .

Some general "handwavy" benefits are

- an ability to *reason* about *discrete* programs (which may be sub-programs)
- an ability to *compose* sub-programs to make slightly less small programs, *indefinitely*

Some general “handwavy” benefits are

- an ability to *reason* about *discrete* programs (which may be sub-programs)
- an ability to *compose* sub-programs to make slightly less small programs, *indefinitely*

What are the benefits of FP?

Although this question commands a considerable amount of work, it is a seemingly endless rabbit hole, for which I have never found the bottom . . .

What are the benefits of FP?

I am committed to helping others join me in exploring this question