CM20214/221

Now for something completely different

So now you have seen

CM20214/221

Now for something completely different

So now you have seen

- Procedural style programming: C and others

CM20214/221

Now for something completely different

So now you have seen

- Procedural style programming: C and others
- Object Oriented style programming: Java and others

CM20214/221

Now for something completely different

So now you have seen

- Procedural style programming: C and others
- Object Oriented style programming: Java and others
- No style: unstructured things like Basic and assembler

Now for something completely different

So now you have seen

- Procedural style programming: C and others
- Object Oriented style programming: Java and others
- No style: unstructured things like Basic and assembler

We now turn to the *Functional* style

Books

For Lisp and functional programming I like

- “The Little Lisper” Friedman and Felleisen
- “The Little Schemer” Friedman and Felleisen
- “Structure and Interpretation of Computer Programs” Abelson and Sussman. Probably ought to be read by all Computer Scientists whether they are interested in Lisp or not.
- “Object-Oriented Programming: The CLOS Approach” Paepcke
- “The Art of the Metaobject Protocol” Kiczales et al

Functional Style

The functional style is quite different from OO and procedural

Functional Style

The functional style is quite different from OO and procedural

Some people have problems with the functional style

Functional Style

The functional style is quite different from OO and procedural

Some people have problems with the functional style

But quite often, the functional style is *closer* to the way we think than other styles

Functional Style

The functional style is quite different from OO and procedural

Some people have problems with the functional style

But quite often, the functional style is *closer* to the way we think than other styles

We've been corrupted by the other styles so much that we can find it harder to get a natural solution to a problem

Functional Style

The functional style is quite different from OO and procedural

Some people have problems with the functional style

But quite often, the functional style is *closer* to the way we think than other styles

We've been corrupted by the other styles so much that we can find it harder to get a natural solution to a problem

Learning the Functional style means warping your brain

Functional Style

The functional style is quite different from OO and procedural

Some people have problems with the functional style

But quite often, the functional style is *closer* to the way we think than other styles

We've been corrupted by the other styles so much that we can find it harder to get a natural solution to a problem

Learning the Functional style means warping your brain

Actually, *unwarping* your brain…

Functional Style

A problem: add together a list of numbers

Functional Style

A problem: add together a list of numbers

We think: “take the numbers and add them”

Functional Style

A problem: add together a list of numbers

We think: “take the numbers and add them”

We *don’t* think: “Well, I need a loop variable to iterate over this list and a variable to accumulate the sum and . . . ”

Functional Style

A problem: add together a list of numbers

We think: “take the numbers and add them”

We *don’t* think: “Well, I need a loop variable to iterate over this list and a variable to accumulate the sum and . . . ”

We *don’t* think: “We need to get the add method for integers and . . . ”

Functional Style

A problem: add together a list of numbers

We think: “take the numbers and add them”

We *don’t* think: “Well, I need a loop variable to iterate over this list and a variable to accumulate the sum and . . . ”

We *don’t* think: “We need to get the add method for integers and . . . ”

The functional solution is “take the numbers and add them”

Functional Style

A problem: add together a list of numbers

We think: “take the numbers and add them”

We *don’t* think: “Well, I need a loop variable to iterate over this list and a variable to accumulate the sum and . . . ”

We *don’t* think: “We need to get the add method for integers and . . . ”

The functional solution is “take the numbers and add them”

“Apply ‘plus’ down this list of numbers”

Functional Style

A problem: add together a list of numbers

We think: “take the numbers and add them”

We *don’t* think: “Well, I need a loop variable to iterate over this list and a variable to accumulate the sum and . . . ”

We *don’t* think: “We need to get the add method for integers and . . . ”

The functional solution is “take the numbers and add them”

“Apply ‘plus’ down this list of numbers”

“`apply(+, [1,2,3])`”

Functional Style

Then multiplying a list of numbers is the same

Functional Style

Then multiplying a list of numbers is the same

“`apply(*, [1,2,3])`”

Functional Style

Then multiplying a list of numbers is the same

“`apply(*, [1,2,3])`”

Not a new separate `for` loop with multiplies instead of adds

Functional Style

Then multiplying a list of numbers is the same

“`apply(*, [1,2,3])`”

Not a new separate `for` loop with multiplies instead of adds

You have a much higher view of what you are trying to achieve rather than language-level details of how to implement it

Functional Style

To some extent, you need to ~~forget~~ put aside the stuff you learned in C and Java and start again

Functional Style

To some extent, you need to ~~forget~~ put aside the stuff you learned in C and Java and start again

One big error for someone new to the functional style is to try to program a functional language in a procedural style

Functional Style

To some extent, you need to ~~forget~~ put aside the stuff you learned in C and Java and start again

One big error for someone new to the functional style is to try to program a functional language in a procedural style

It doesn't work

Functional Style

To some extent, you need to ~~forget~~ put aside the stuff you learned in C and Java and start again

One big error for someone new to the functional style is to try to program a functional language in a procedural style

It doesn't work

For example, some versions of Lisp (a functional language) don't have iterative ('for') loops in the way C and Java do

Functional Style

To some extent, you need to ~~forget~~ put aside the stuff you learned in C and Java and start again

One big error for someone new to the functional style is to try to program a functional language in a procedural style

It doesn’t work

For example, some versions of Lisp (a functional language) don’t have iterative (‘for’) loops in the way C and Java do

If you find yourself having problems programming in a functional style: stop, step back and reappraise the situation

Functional Style

To some extent, you need to ~~forget~~ put aside the stuff you learned in C and Java and start again

One big error for someone new to the functional style is to try to program a functional language in a procedural style

It doesn't work

For example, some versions of Lisp (a functional language) don't have iterative ('for') loops in the way C and Java do

If you find yourself having problems programming in a functional style: stop, step back and reappraise the situation

You are most likely trying to force a procedural (or other) style

Functional Style

But, in the other direction, the things we learn from the functional style are often applicable to other languages

Functional Style

But, in the other direction, the things we learn from the functional style are often applicable to other languages

The “map-reduce” operation that Google relies on for its massive scalability is a functional idea re-implemented in C

Functional Style

But, in the other direction, the things we learn from the functional style are often applicable to other languages

The “map-reduce” operation that Google relies on for its massive scalability is a functional idea re-implemented in C

Exercise. Look up Hadoop

Functional Style

The functional style is based on

Functional Style

The functional style is based on

- the evaluation of functions.

Functional Style

The functional style is based on

- the evaluation of functions. So far no difference

Functional Style

The functional style is based on

- the evaluation of functions. So far no difference
- avoiding global state.

Functional Style

The functional style is based on

- the evaluation of functions. So far no difference
- avoiding global state. Just like OO

Functional Style

The functional style is based on

- the evaluation of functions. So far no difference
- avoiding global state. Just like OO
- avoid changes of state.

Functional Style

The functional style is based on

- the evaluation of functions. So far no difference
- avoiding global state. Just like OO
- avoid changes of state. Eh?

Functional Style

State

State is essentially the value of the variables

Functional Style

State

State is essentially the value of the variables

Global state is bad as different parts of the program can interfere with the state causing unexpected results

Functional Style
State

State is essentially the value of the variables

Global state is bad as different parts of the program can interfere with the state causing unexpected results

Particularly when we come to large systems with many programmers; and parallel programming

Functional Style

State

State is essentially the value of the variables

Global state is bad as different parts of the program can interfere with the state causing unexpected results

Particularly when we come to large systems with many programmers; and parallel programming

So OO captures state within objects and only allows controlled access via methods

Functional Style

State

State is essentially the value of the variables

Global state is bad as different parts of the program can interfere with the state causing unexpected results

Particularly when we come to large systems with many programmers; and parallel programming

So OO captures state within objects and only allows controlled access via methods

Functional programs capture state within functions and only allow access via function evaluation

Functional Style

State

A big problem is when we try to analyse a program for correctness

Functional Style

State

A big problem is when we try to analyse a program for correctness

The issue is that **variables vary**

Functional Style

State

A big problem is when we try to analyse a program for correctness

The issue is that **variables vary**

In mathematics an *x* here is the same as the *x* there

Functional Style

State

A big problem is when we try to analyse a program for correctness

The issue is that **variables vary**

In mathematics an x here is the same as the x there

We can make deductions and proofs and so on

Functional Style
State

A big problem is when we try to analyse a program for correctness

The issue is that **variables vary**

In mathematics an x here is the same as the x there

We can make deductions and proofs and so on

In a program `x` might have changed while we were not looking

Functional Style

State

A big problem is when we try to analyse a program for correctness

The issue is that **variables vary**

In mathematics an *x* here is the same as the *x* there

We can make deductions and proofs and so on

In a program `x` might have changed while we were not looking

```
x = 7;
wibble(y);
// what is the value of x here?
```

Functional Style

State

In the functional style we mimic the mathematical idea by *never updating a variable*

Functional Style

State

In the functional style we mimic the mathematical idea by *never updating a variable*

So the `x` here has the same value as the `x` there (within a block)

Functional Style

State

In the functional style we mimic the mathematical idea by *never updating a variable*

So the `x` here has the same value as the `x` there (within a block)

We can then do a mathematical style analysis to prove things, e.g., correctness, about our program

Functional Style

State

In the functional style we mimic the mathematical idea by *never updating a variable*

So the `x` here has the same value as the `x` there (within a block)

We can then do a mathematical style analysis to prove things, e.g., correctness, about our program

It also gives us *referential transparency*

Functional Style

Referential Transparency

A chunk of code is *referentially transparent* if it is not dependent on its *environment*: this is the values of the variables outside of the chunk

Functional Style

Referential Transparency

A chunk of code is *referentially transparent* if it is not dependent on its *environment*: this is the values of the variables outside of the chunk

So it can’t read any variable from its environment

Functional Style

Referential Transparency

A chunk of code is *referentially transparent* if it is not dependent on its *environment*: this is the values of the variables outside of the chunk

So it can’t read any variable from its environment

And it can’t update any variable from its environment

Functional Style

Referential Transparency

As a referentially transparent chunk of code does not depend on its environment we can pick it up and use it somewhere else

Functional Style

Referential Transparency

As a referentially transparent chunk of code does not depend on its environment we can pick it up and use it somewhere else

And it will work correctly!

Functional Style

Referential Transparency

As a referentially transparent chunk of code does not depend on its environment we can pick it up and use it somewhere else

And it will work correctly!

This is software reuse

Functional Style

Referential Transparency

A function is called *purely functional* if it has no *side effects*

Functional Style

Referential Transparency

A function is called *purely functional* if it has no *side effects*

That is it doesn’t interfere with the environment/global state

Functional Style

Referential Transparency

A function is called *purely functional* if it has no *side effects*

That is it doesn’t interfere with the environment/global state

So a purely functional function is referentially transparent

Functional Style

Referential Transparency

A function is called *purely functional* if it has no *side effects*

That is it doesn’t interfere with the environment/global state

So a purely functional function is referentially transparent

Many functions, e.g., `sin`, `sqrt`, are naturally purely functional

Functional Style

Referential Transparency

A function is called *purely functional* if it has no *side effects*

That is it doesn't interfere with the environment/global state

So a purely functional function is referentially transparent

Many functions, e.g., `sin`, `sqrt`, are naturally purely functional

Which is why we can use them anywhere

Functional Style

Referential Transparency

A function is called *purely functional* if it has no *side effects*

That is it doesn't interfere with the environment/global state

So a purely functional function is referentially transparent

Many functions, e.g., `sin`, `sqrt`, are naturally purely functional

Which is why we can use them anywhere

And get the same result for the same argument every time

Functional Style

Referential Transparency

A function is called *purely functional* if it has no *side effects*

That is it doesn't interfere with the environment/global state

So a purely functional function is referentially transparent

Many functions, e.g., `sin`, `sqrt`, are naturally purely functional

Which is why we can use them anywhere

And get the same result for the same argument every time

And using them does not affect any other part of the system

Functional Style

Referential Transparency

3354
HERE'S THE PILL THE VET TOLD ME TO GIVE YOU...
I'LL BE INTERESTED TO SEE IF IT HAS ANY SIDE-EFFECTS...
ME, TOO
ONE OF THE SECRETS OF LIFE IS TO HAVE GOOD SIDE-EFFECTS
SCHULZ

Schulz

Functional Style

Referential Transparency

Code like

```
int f(int x) {
  count++;
  return x+1;
}
```

which counts the number of times `f` has been called is not purely functional

Functional Style

Referential Transparency

Code like

```
int f(int x) {
  count++;
  return x+1;
}
```

which counts the number of times `f` has been called is not purely functional

It modifies a non-local variable `count` from its environment, but *which* variable named `count` it modifies depends on where the function definition happens to be placed

Functional Style

one.c:
```
#include <stdio.h>

static int count = 1;
extern void otherfoo(void);

static void foo(void)
{
  printf("foo %d\n", count);
}

int main(void)
{
  foo();
  otherfoo();
  return 0;
}
```

two.c:
```
#include <stdio.h>

static int count = 2;


static void foo(void)
{
  printf("foo %d\n", count);
}

void otherfoo(void)
{
  foo();
}
```

Functional Style

Referential Transparency

So the behaviour of this code depends on where it is: it is not referentially transparent

Functional Style

Referential Transparency

So the behaviour of this code depends on where it is: it is not referentially transparent

These are trivial examples but the idea expands to all code

Functional Style

Referential Transparency

OO tries to be referentially transparent by hiding state within an object to prevent unexpected interactions between parts of state; but then has global objects, so we’ve just pushed the problem up a bit

Functional Style

Referential Transparency

OO tries to be referentially transparent by hiding state within an object to prevent unexpected interactions between parts of state; but then has global objects, so we've just pushed the problem up a bit

It happens to be straightforward to implement OO inside a functional language

Functional Style

Referential Transparency

OO tries to be referentially transparent by hiding state within an object to prevent unexpected interactions between parts of state; but then has global objects, so we've just pushed the problem up a bit

It happens to be straightforward to implement OO inside a functional language

As it originally was. Functional is much older than OO

Functional Style

Referential Transparency

Question to think on

Functional Style

Referential Transparency

Question to think on

If we can’t interact with the global environment, how can we do input and output?