Functions

John Cinnamond

@jcinnamond | panagile.com

We’re going to need…

We’re going to need…

Some mathematics

We’re going to need…

Some mathematics

Some fancy academic terms

What is functional programming?

What is functional programming?

Using mathematical functions…

What is functional programming?

Using mathematical functions…

…to perform calculations

A function…

A function…

Is a relation

A function…

Is a relation

between a set of input values

A function…

Is a relation
between a set of input values
and a set of output values

$$\begin{tikzcd}
\text{Input} & & \text{Output} \\
2 \arrow[dr] & & 4 \\
3 \arrow[r] & \text{double} \arrow[ur] \arrow[r] \arrow[dr] & 6 \\
4 \arrow[ur] & & 8
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Input} & & \text{Output} \\
x \arrow[r] & \text{double} \arrow[r] & x * 2
\end{tikzcd}$$

x ( ) = double x * 2

$$\max(x,y) = \begin{cases} x & \text{if } x > y \\ y & \text{otherwise} \end{cases}$$

```
def max(x, y)
  if x > y
    x
  else
    y
  end
end
```

$$\text{first\_even(list)} = \left\{ \right.$$

```
def first_even(list)
  for i in list
    if i.even?
      break i
    end
  end
end
```

$$first\_even(\text{list}) = \begin{cases} \text{nil} & if \quad \text{empty?}(\text{list}) \\ \text{head}(\text{list}) & if \quad \text{even?}(\text{head}(\text{list})) \\ first\_even(\text{tail}(\text{list})) & \text{otherwise} \end{cases}$$

$$first\_even(\text{list}) = \begin{cases} \text{nil} & if \quad \text{empty?(list)} \\ \text{head(list)} & if \quad \text{even?(head(list))} \\ first\_even(\text{tail(list)}) & \text{otherwise} \end{cases}$$

```ruby
def first_even(list)
  i = list.shift
  if i.even?
    i
  else
    first_even(list)
  end
end
```

```ruby
def first_even(list)
  return nil if list.empty?
  i = list.shift
  if i.even?
    i
  else
    first_even(list)
  end
end
```

```ruby
def first_even(list)
  return nil if list.empty?
  i = list.shift
  if i.even?
    i
  else
    first_even(list)
  end
end
```

```
def first_even(list)
  for i in list
    if i.even?
      break i
    end
  end
end
```

```
def first_even(list)
  for i in list
    if i.even?
      break i
    end
  end
end
```

$$a = 10$$

$$a = 10$$

$$\xcancel{a = 11}$$

Not OK

$$a = 10$$
$$double\_a = a \times 2$$

$$a = 10$$
$$double\_a = a \times 2$$
$$also\_double\_a = 10 \times 2$$

```ruby
a = 10
def double_a
  a * 2
end

double_a # => 20
```

```ruby
a = 10
def double_a
  a * 2
end

double_a # => 20

a = 11

double_a # => 22
```

$$even(x) = x \ \% \ 2 \equiv 0$$

Functional Programming Concepts

Functions are definitions, not lists of instructions

Immutable definitions, not variables

Functions are first class citizens

Currying

Haskell Curry

Functional Thinking

Advantages of the Functional Approach

Add features without changing existing code

Create each step of the report with very little code

No need to support unrelated code from other steps

If you like functional programming…

…maybe try a proper functional programming language

(e.g., Clojure)

Recommended Functional Techniques

map, select, inject, …

Blocks for lists of data

Lambdas for running code later

Not so much

~~Currying and Composing Lambdas?~~

$$\begin{tikzcd}\text{Still awesome} \arrow[d] \\ \text{Functional Composition}\end{tikzcd}$$

Immutability
and
Referential Transparency

make it easier to reason about code

Composition
Referential Transparency
Small, specific solutions

Write Better Object Oriented Code