Working with Functions

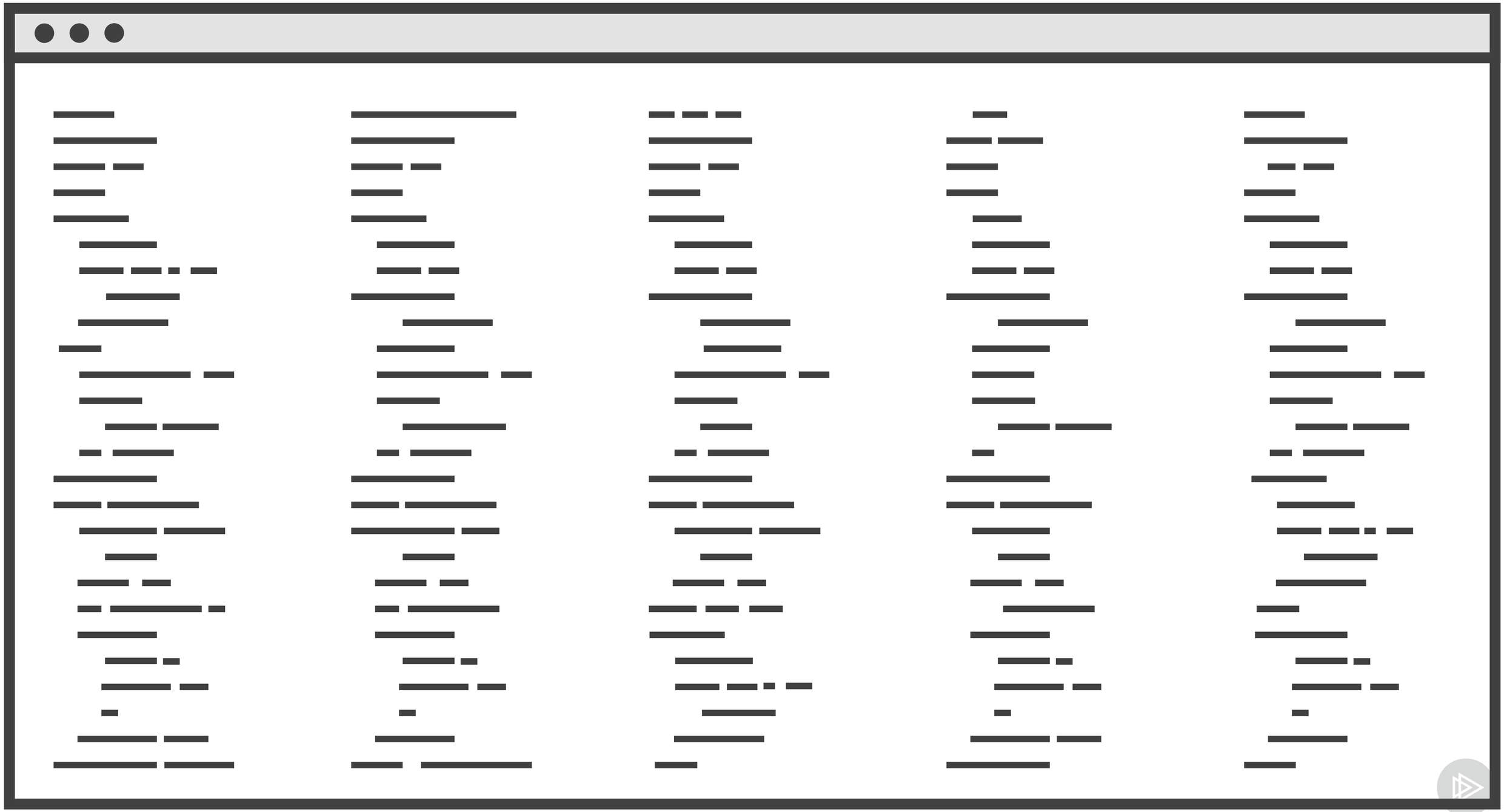
Nigel Poulton
Author & Trainer

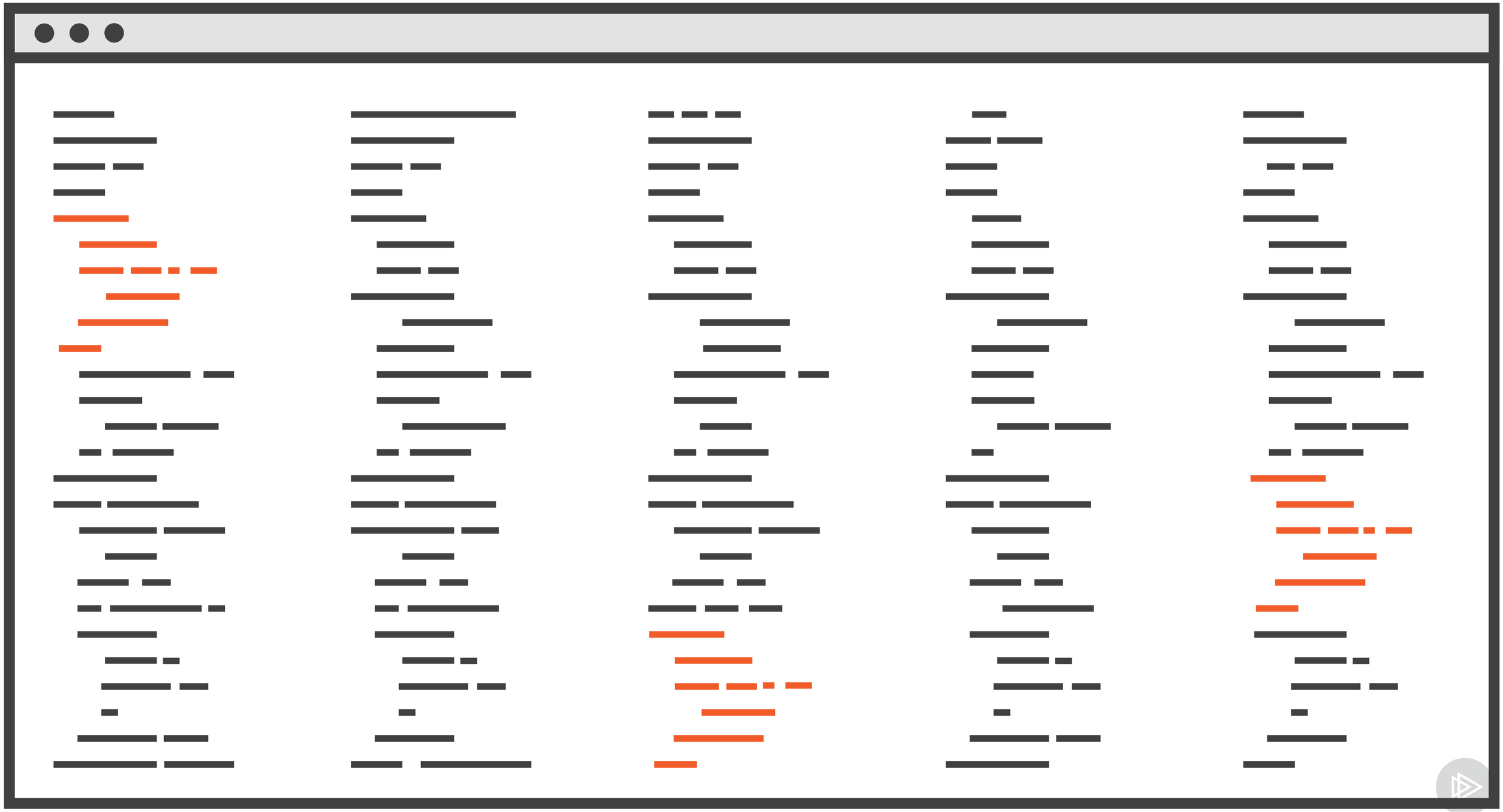
@nigelpoulton nigelpoulton.com

Agenda

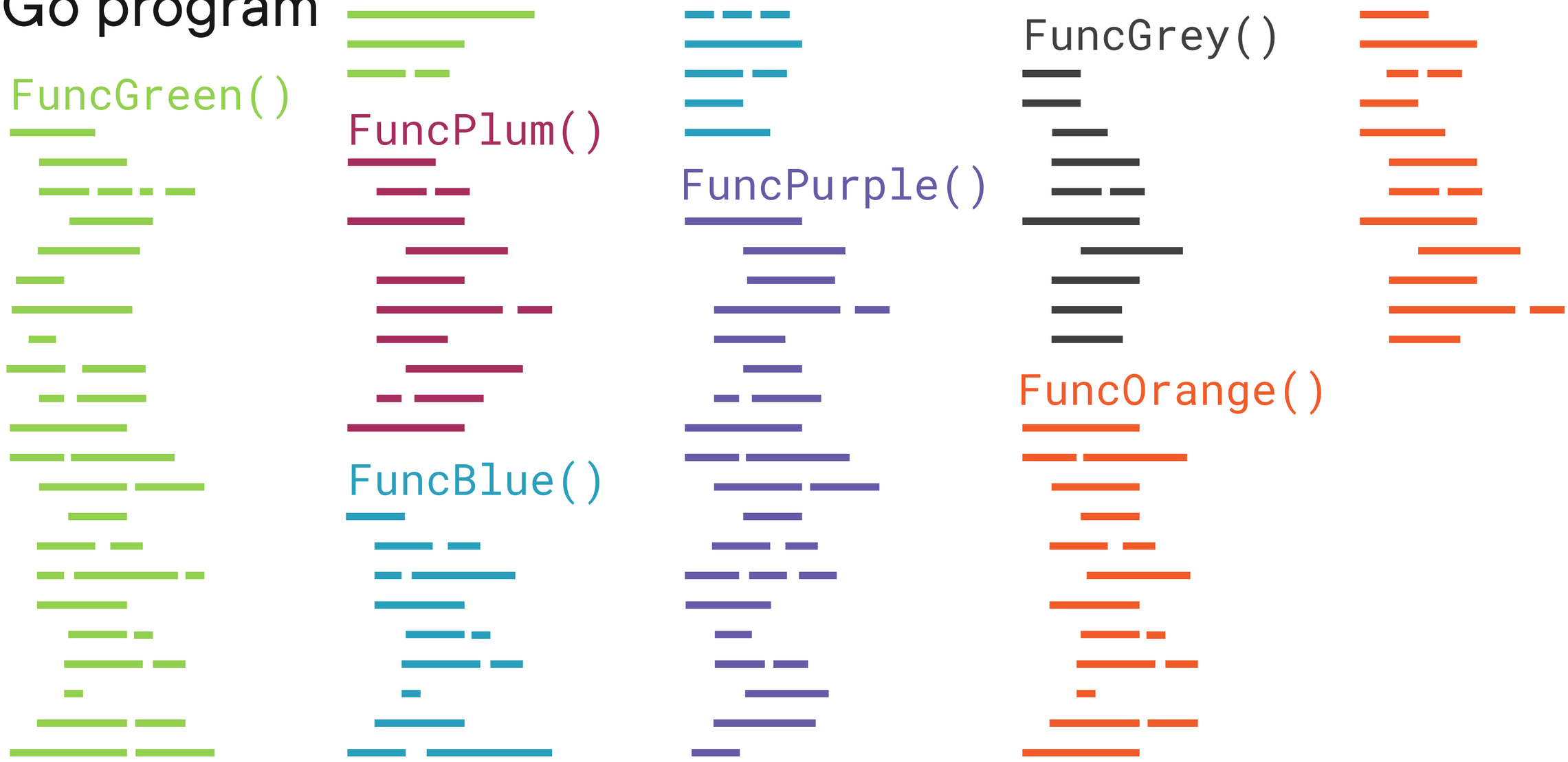
- Why functions
- Function syntax
- Writing your own functions
- Variadic functions
- Recap

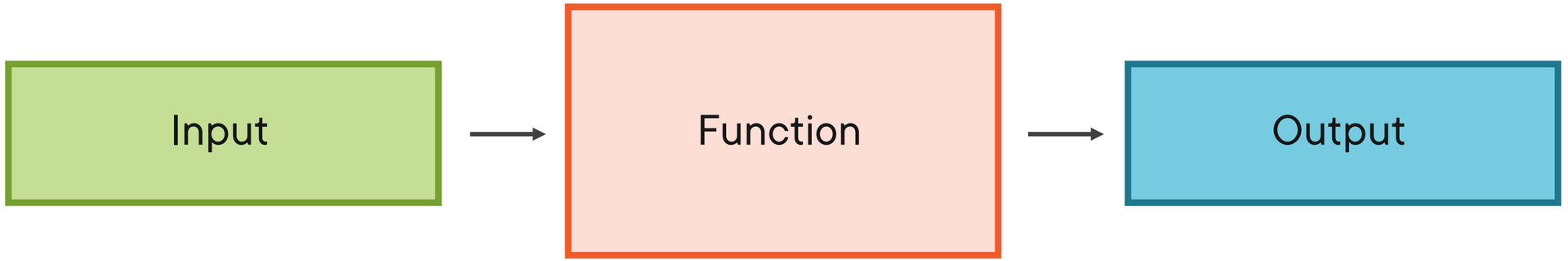
Why Functions

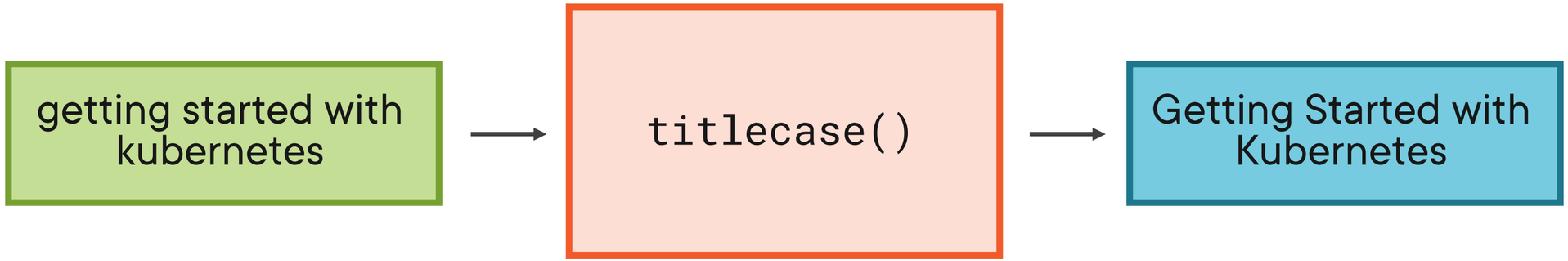




Go program


FuncGreen()
FuncPlum()
FuncPurple()
FuncGrey()
FuncBlue()
FuncOrange()


Input
Function
Output


getting started with kubernetes
titlecase()
Getting Started with Kubernetes

Function Syntax

`func`

Defined with `func` keyword

```
func titlecase(params, type) {
  <code>
}
```

Defined with `func` keyword

Function code goes inside curly braces `{}`

Parameters and returns defined in function signature

```
text := "containers on aws wavelength"
fmt.Println(titlecase(text))

func titlecase(text, string) {
  <code>
}
```

Defined with `func` keyword

Function code goes inside curly braces `{}`

Parameters and returns defined in function signature

```
text := "containers on aws wavelength"
fmt.Println(titlecase(text))

func titlecase(text, string) string {
  <code>
}
```

Defined with `func` keyword

Function code goes inside curly braces `{}`

Parameters and returns defined in function signature

```
text := "containers on aws wavelength"
fmt.Println(titlecase(text))

func titlecase(text, string) string {
  <code>
  return
}
```

Defined with `func` keyword

Function code goes inside curly braces `{}`

Parameters and returns defined in function signature

Recap

Functions

enable modular re-usable code

Go programs

are made from multiple small functions

`func`

Defined with `func` keyword

```
func name(input ...string) string {
   <code>
   return
}
```

Defined with `func` keyword

Receives inputs and returns outputs

Up Next:
Working with Conditionals