




Lambdas

Lambdas

```
button.addActionListener { println("Hi") }
```

Lambdas vs anonymous classes

```
button.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Hi");  
    }  
});
```



```
button.addActionListener { println("Hi") }
```



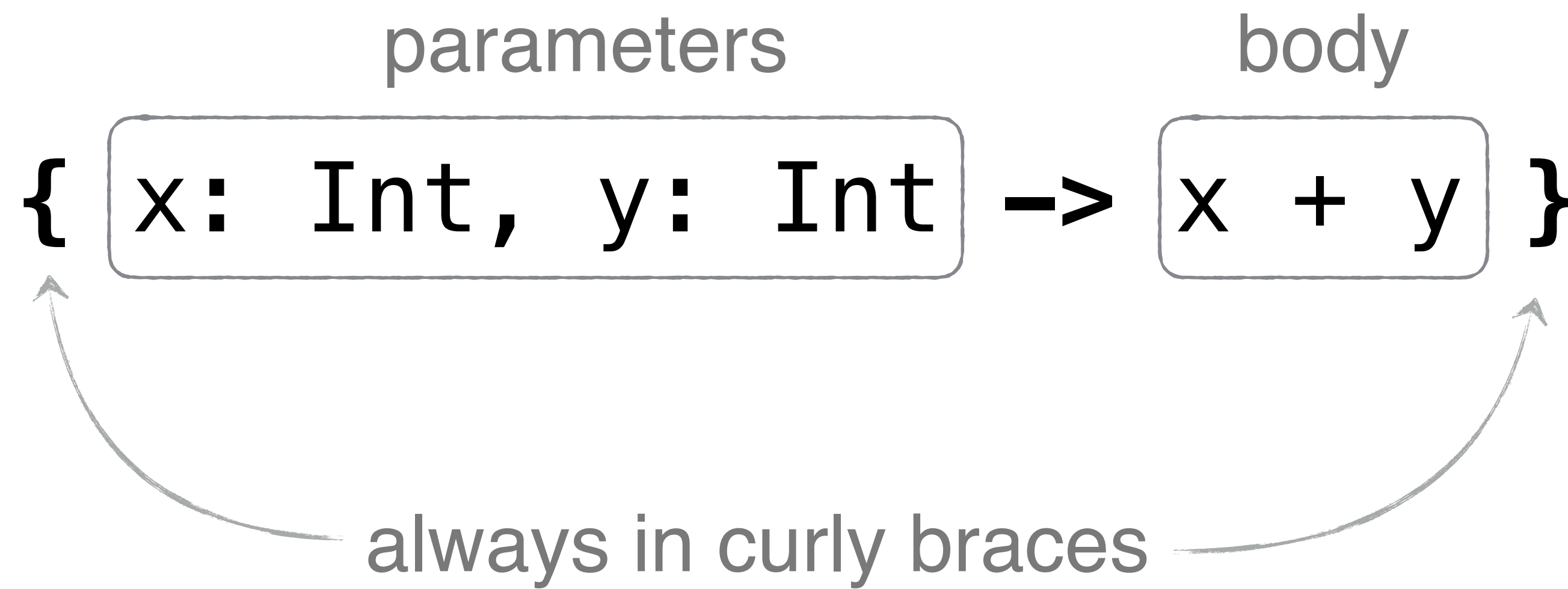
Working with collections in a functional style

```
val employees: List<Employee>  
data class Employee(  
    val city: City, val age: Int  
)
```

What's an average
age of employees
working in Prague?

```
employees.filter { it.city == City.PRAGUE }  
    .map { it.age }  
    .average()
```

Lambda syntax



Lambda syntax

```
list.any({ i: Int -> i > 0 })
```



full syntax

Lambda syntax

```
list.any() { i: Int -> i > 0 }
```



when lambda is the last argument,
it can be moved out of parentheses

Lambda syntax

```
list.any { i: Int -> i > 0 }
```

empty parentheses can be omitted



Lambda syntax

```
list.any { i -> i > 0 }
```

type can be omitted if it's clear from the context



Lambda syntax

`list.any { it > 0 }`



it denotes the argument if it's only one

Multi-line lambda

```
list.any {  
  println("processing $it")  
  it > 0  
}
```

the last expression is the result



Lambda syntax

```
map.mapValues { entry -> "${entry.key} -> ${entry.value!}" }
```

use destructuring declarations syntax instead



Lambda syntax

```
map.mapValues { (key, value) -> "$key -> $value!" }
```



destructuring declarations syntax

Lambda syntax

```
map.mapValues { (_, value) -> "$value!" }
```

omit the parameter name if the parameter is unused

