ROBCO INDUSTRIES UNIFIED OPERATING SYSTEM
COPYRIGHT 2075-2077 ROBCO INDUSTRIES
-Server 1-

Ulabox IT department

Rub&eaccute;n Sospedra

Ulalearners edition

> Loading functional programming . . .
> Loaded
> Press any key to continue

```
> man functional-programming

ALIAS fp

DESCRIPTION

    --deterministic
    --declarative
```

DETERMINISTC

```
const x = 1337
const plus = (x) => ++x
```

```
plus(x) // 1338
x // 1337
```

DECLARATIVE

```
const n = [4, 8, 15, 16]
const pairs = n.filter((x) => x % 2)

pairs // [15]
```

```
> unlocked purity
> unlocked no-side-effects
```

Error: File “Detonation.paf” has crashed.

F4: Reboot F5: Restart Operation
F6: Rebuild File F7: Exit

> Side effects are coming

```
let x = 1337
const plus = () => {
  x = x + 1
  return x
}

plus() // 1338
x // 1338
```

> Imperator imperative

```
const n = [4, 6, 8, 15]
const pairs = []

n.forEach((x) => {
  if (x % 2) pairs.push(x)
})
pairs // [15]
```

```
> man fp-principles

DESCRIPTION
```

```
--first-class-citizens
--high-order
--immutability
```

FIST CLASS CITIZENS

```
const clbk = (x) => x * x
const n = [1, 1, 2, 3, 5]

n.map(clbk)
// [1, 1, 4, 9, 25]
```

HIGH ORDER FUNCTIONS

```
const pow = (exp) => {
  return (base) => base ** exp
}

pow(2)(5)
// 25
```

IMMUTABILITY

```
const n = [4, 9, 2, 3]
const m = n.map((x) => {
  return (x !== 3) ? x : 11
})

n // [4, 9, 2, 3]
m // [4, 9, 2, 11]
```

```
> man fp-category-theory

DESCRIPTION
```

```
--map/reduce
--functor
--monad
```

MAP/REDUCE

```
[1, 'zero', 1]
  .map((x) => !isNaN(x))
  .reduce((memo, x) => memo + x)

// 2
```

FUNCTOR

```
const f = [0, 1, 3, 2, 6]
const g = f.map((x) => x)

f === g // false
f.map // [Function: map]
g.map // [Function: map]
```

MONAD

```
const fmap = (l, f) => {
 return [].concat.apply([], l.map(f))
}
const seventh = (x) => [x, x / 7]

fmap([7, 49, 91], seventh)
// [7, 1, 49, 7, 91, 13]
```

```
> man fp-benefits

DESCRIPTION
```

```
--stateless
--referencial-transparency
--composition
--testability
```

STATELESS

```
const cart = { k: 9 }
const max = (x, c) => Object.assign({},
  c,
  c[x] < 10 && { [x]: ++c[x] }
)

const nc = max('k', cart) // { kiwi: 10 }
max('k', nc) // { kiwi: 10 }
```

REFERENCIAL TRANSPARENCY

```
const sqrt = (x) => x * x
const mult2 = (x) => x * 2
const sum3 = (x) => x + 3

sqrt(mult2(sum3(10)))
// 676
```

COMPOSITION

```
const fourLegs = (x) => ({
  ...x, legs: 4
})
const hasLasers = (x) => ({
  ...x, lasers: true
})

const soundsLikeACat = { meow: true }
const cat = fourLegs(soundsLikeACat)
const catinator4000 = hasLasers(cat)
```

TESTABILITY

```
const p = (p1, p2) => p1 + p2
```

```
p(mock1, mock2)
// mock1 - mock2

p(type1 - type2)
// (yN) throwing errors
```

FUNCTIONAL

is now