Abstract Factory

```
function ClassFactory() {
      thi s.type = 'class'
      thi s.l ogType = function () {
            con sol e.l og( 'Type', this.type)
     }
}
function Warrio rCl ass() {
      Cla ssF act ory.ca ll( this)
      thi s.s ubtype = 'warrior'
      thi s.l ogS ubtype = function () {
            con sol e.l og( 'Su btype', this.s -
ubtype)
     }
}
function Wizard Class() {
      Cla ssF act ory.ca ll( this)
      thi s.s ubtype = 'wizard'
      thi s.l ogS ubtype = function () {
            con sol e.l og( 'Su btype', this.s -
ubtype)
     }
}
const $warrior = new Warrio rCl ass()
$warri or.l og Type()
$warri or.l og Sub type()
```

Type class
Subtype warrior

Dynamic Function From String

```
global.runFoo = function runFoo() {
     if (argum ent s.l ength) {
           let args = []
           for (let i = 0; i < argume nts.le ngth;
i++) {
                 arg s.p ush (ar gum ent s[i])
           }
           con sole.log(`executed runFoo (${ arg -
s.j oin()})`)
     } else {
           con sol e.l og( 'ex ecuted runFoo()')
     }
}
const fn = 'runFoo'
const fnParams = [1,2,3]
global [fn]()
global[fn].apply(null, fnParams)
```

executed runFoo()
executed runFoo(1,2,3)

Functional Interface (Interface Segregation)

```
global.oven = function Oven() {
      this.on = false
      thi s.t urnOn = function () {
            this.on = true
     }
      thi s.t urnOff = function () {
            this.on = false
     }
      thi s.cook = function (item) {
            con sol e.log( `Cooking ${item} in the o
ven)
```

Functional Interface (Interface Segregation) (cont)

```
>  }
}
// function Stove() { // ES6
global.stove = function Stove() {
  this.on = false
  this.turnOn = function () {
    this.on = true
  }
  this.turnOff = function () {
    this.on = false
  }
  this.cook = function (item) {
    console.log(Cooking ${item} in the stove)
  }
}
function ICooker(cooker) {
  // let fn = window[cooker]; // ES6
  let fn = global[cooker] // Node.js
  return new fn()
}
function Restaurant(name, cooker) {
  this.name = name
  this.cooker = new ICooker(cooker)
  this.cook = function (item) {
    this.cooker.turnOn()
    this.cooker.cook(item)
    this.cooker.turnOff()
```

Functional Interface (Interface Segregation) (cont)

```
>  }
}
const cookerTypes = {
  OVEN: 'oven',
  STOVE: 'stove',
}
const bakery = new Restaurant('Bakery', cookerTypes.OVEN)
bakery.cook('cookies')
const crepery = new Restaurant('Crepery', cookerTypes.STOVE)
crepery.cook('crepes')
```

Cooking cookies in the oven
Cooking crepes in the stove

Functional Constructor

```
function Robot(name, job) {
      thi s.name = name
      thi s.job = job
      thi s.i ntr oduce = function () {
            con sol.log(Name ${this.name})
            con sol.log(Job ${this.job})
     }
}
const walle = new Robot( 'Wa ll-E', 'clean')
consol e.l og( walle instanceof Robot)
walle.i nt rod uce()
```

true
Name Wall-E
Job clean

Functional Inheritance (Liskov Substitution)

```
const Scientist = {
      init: function (name) {
            thi s.name = name
     },
      run _ex per iment: function () {
            con sol e.l og( `${this.name} is running e
xperiment)
     },
}
const MadSci entist = {
     ...S ci entist,
      run _ex per iment: function () {
            const sabotage = !!Math.fl oor (Ma -
th.r an dom() * 2)
           if (sabotage) {
                 con sol e.l og( `${this.name} is sab
otaging experi ment!)
           } else {
                 con sol e.l og( `${this.name} is run
ning experiment)
           }
     },
}
const neil = Object.cr eat e(S cie ntist)
neil.i nit ('N eil')
const hubert = Object.cr eat e(M adS cie ntist)
hubert.in it( 'Hu bert')
neil.r un_ exp eri ment()
hubert.ru n_e xpe rim ent()
```

Neil will always return "Neil is running experiment"
Hubert will possibly return "Hubert is running experiment" or "Hubert is sabatoging experiment!"

Open/Close Principle

```
function announce(collection) {
      con sol e.l og( col lec tio n.d esc rip tion)
      col lec tio n.l ogI tems()
}
var favori teC ities = {
      items: {
            Den mark: 'Copen hagen',
            Uganda: 'Kampala',
            Ura guay: 'Monte video',
     },
      des cri ption: 'My favorite cities around the
world:',
      log Items: function () {
            Obj ect.ke ys( thi s.i tem s).f or -
Eac h(f unction (key) {
                  con sol e.l og( thi s.i tem -
s[key])
           }, this)
     },
}
announ ce( fav ori teC ities)
```

My favorite cities around the world:
Copenhagen
Kampala
Montevideo