

function oldFn (arg1, arg2, arg3) {

}

function newFn (arg) {

return oldFn (1, 2, arg);

}

`new Intro = intro.bind(obj, ①);`

`new Intro(, 2);`

`Intro (①, ②)`

The diagram illustrates how arguments are passed to the `Intro` function. A curved arrow originates from the `①` argument in the first call (`new Intro = intro.bind(obj, ①);`) and points to the `①` parameter in the function definition (`Intro (①, ②)`). Another curved arrow originates from the `2` argument in the second call (`new Intro(, 2);`) and points to the `②` parameter in the function definition. A third curved arrow originates from the empty space before the `2` in the second call and points to the `①` parameter, indicating that the first parameter is not explicitly provided in that call.

★ newIntro = intro.bind(obj, ... fixedargs)

★ newIntro (... nonfixedargs);

↓
intro.call(obj, ,);

function intro (arg1, arg2, arg3){

}

newIntro = intro.bind(obj, 99);

* newIntro (3, 100);

Current
Customer = { } + { --proto-- }

Object.prototype.
withdraw

Saving
Customer

```
objectCreationUsingFuncs.js:13
▼ {nam: 'jerico', branch: 'sbi xyz', accountBalance: 10000, withdraw: f} ⓘ
  accountBalance: 10000
  branch: "sbi xyz"
  nam: "jerico"
  ▶ withdraw: f (amount)
  ▶ [[Prototype]]: Object

objectCreationUsingFuncs.js:16
▼ {nam: 'Tez', branch: 'icic abc', accountBalance: 100000, withdraw: f} ⓘ
  accountBalance: 100000
  branch: "icic abc"
  nam: "Tez"
  ▶ withdraw: f (amount)
  ▶ [[Prototype]]: Object
```

object.prototype

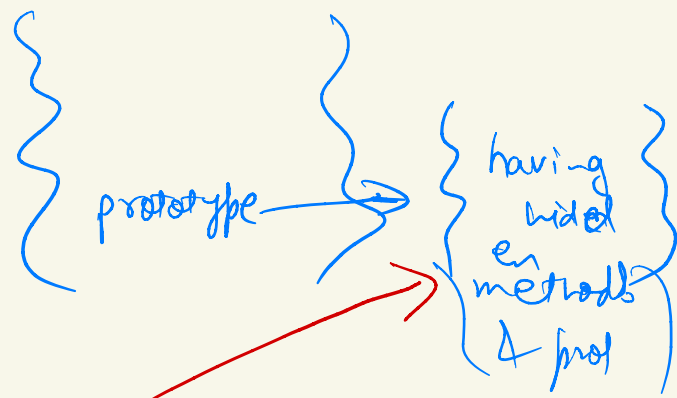
Create Customer \Rightarrow fn +

It creates instance (it's constr fn)

prototype: { Stores hidden methods and props }

Create Customer \Rightarrow fn + { prototype \rightarrow { constr fn : Create Customer } }

Create Customer \Rightarrow fn +



(instance of Create Customer)

customer1 \rightarrow

customer1. -- proto --

★ customer. -- proto -- === CreateCustomer.prototype

↪ true

CreateCustomer.prototype.withdraw = function() {

}