**Функции обратного вызова**

**Callback-функции**

Как мы знаем, функция является частным случаем объекта, а коли так, функция может быть передана в аргумент другой функции.

Функция переданная в аргумент другой функции называется **Функцией обратного вызова** или **Callback-функцией**.

Разберем на абстрактном примере.

const foo1 = (obj) => obj.id

const foo2 = (f) => f({id: 1}) + 1

console.log(foo2(foo1)) //получим 2

Упростим данную запись, для большей наглядности

const foo1 = (obj) => obj.id

const foo2 = (f) => {

const funcResult = f({id: 1})

return funcResult + 1

}

console.log(foo2(foo1)) //получим 2

Мы рассмотрим только одну причну применения **callback**-функций - модификацию поведения.

**Модификация поведения**

Рассмотрим модификацию поведения главной функции на примере

const process = (a, b, func) => {

return { operationResult: func(a, b) }

}

//выводы

console.log(process(1, 2, (a, b) => a + b))

console.log(process(1, 2, (a, b) => a - b))

console.log(process(1, 2, (a, b) => (a + b) \* 2))

//усложним - выведем всё в однои объекте

console.log({

add: process(1, 2, (a, b) => a + b),

sub: process(1, 2, (a, b) => a - b),

dblAdd: process(1, 2, (a, b) => (a + b) \* 2),

})

По сути, мы создём "скелет", от которого нельзя отходить, но какую-то его часть наш коллега-программист может модифицировать под свои нужды. Тем самым мы уменьшаем количество кода и делаем его более гибким.