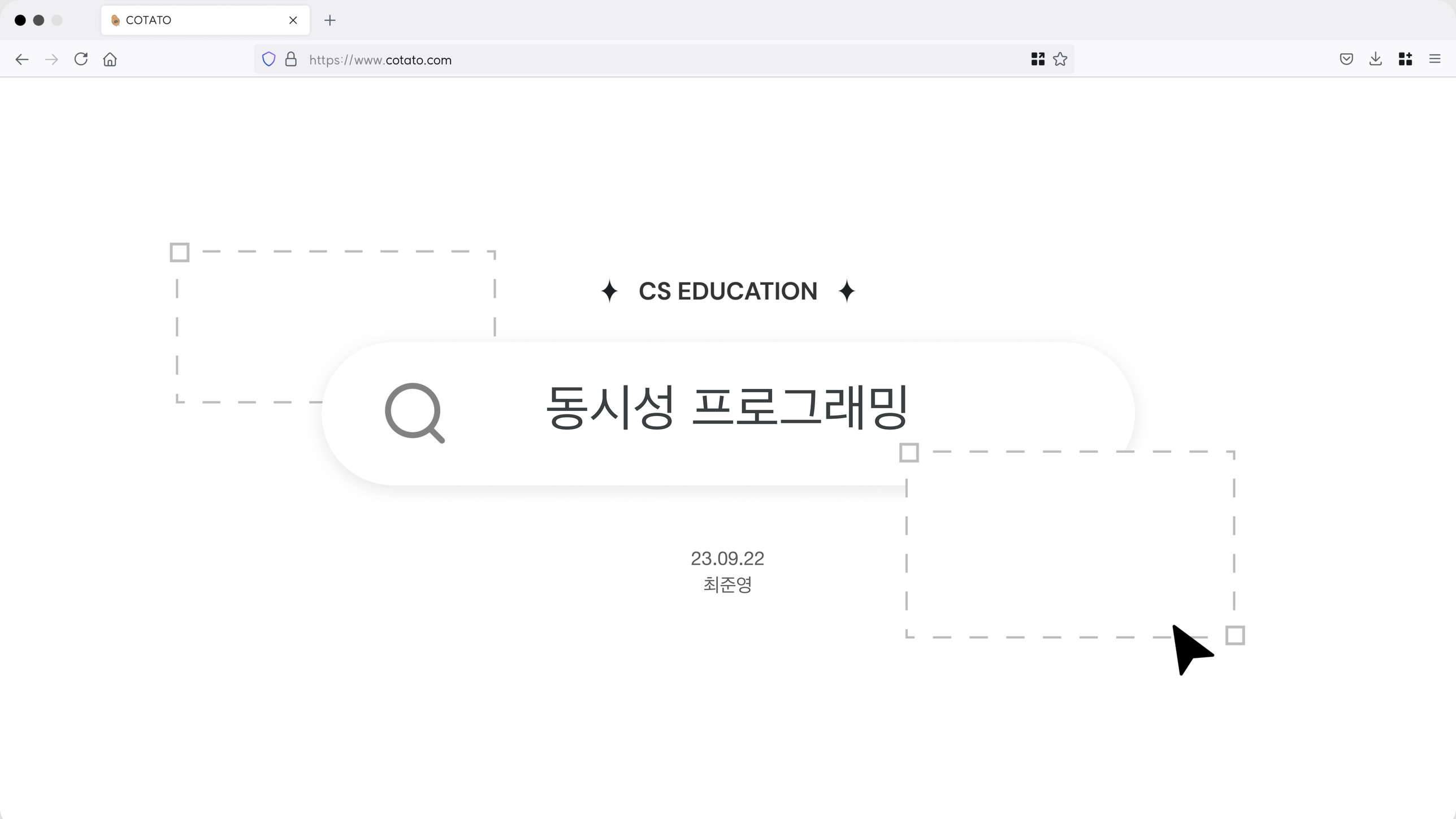
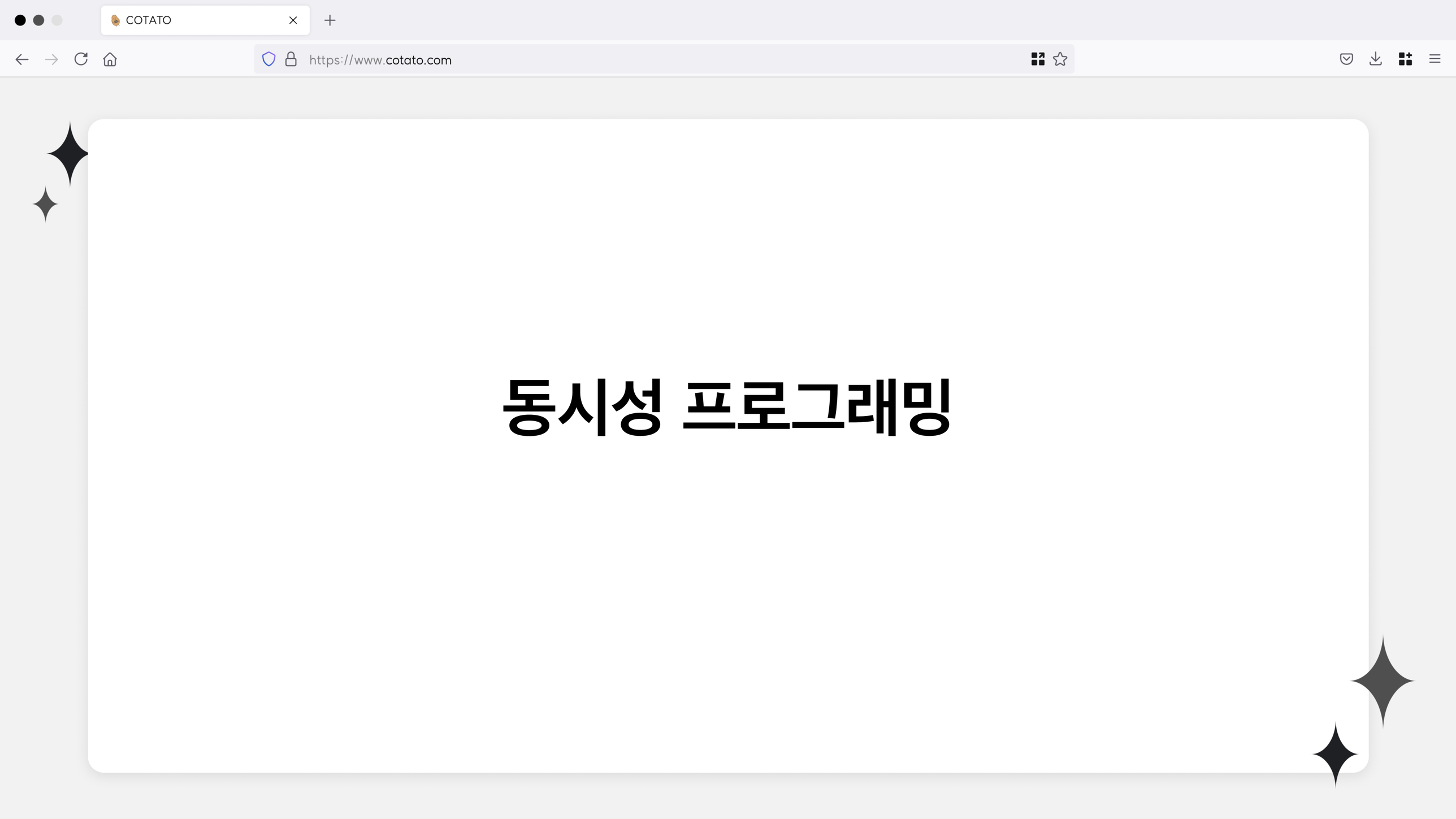
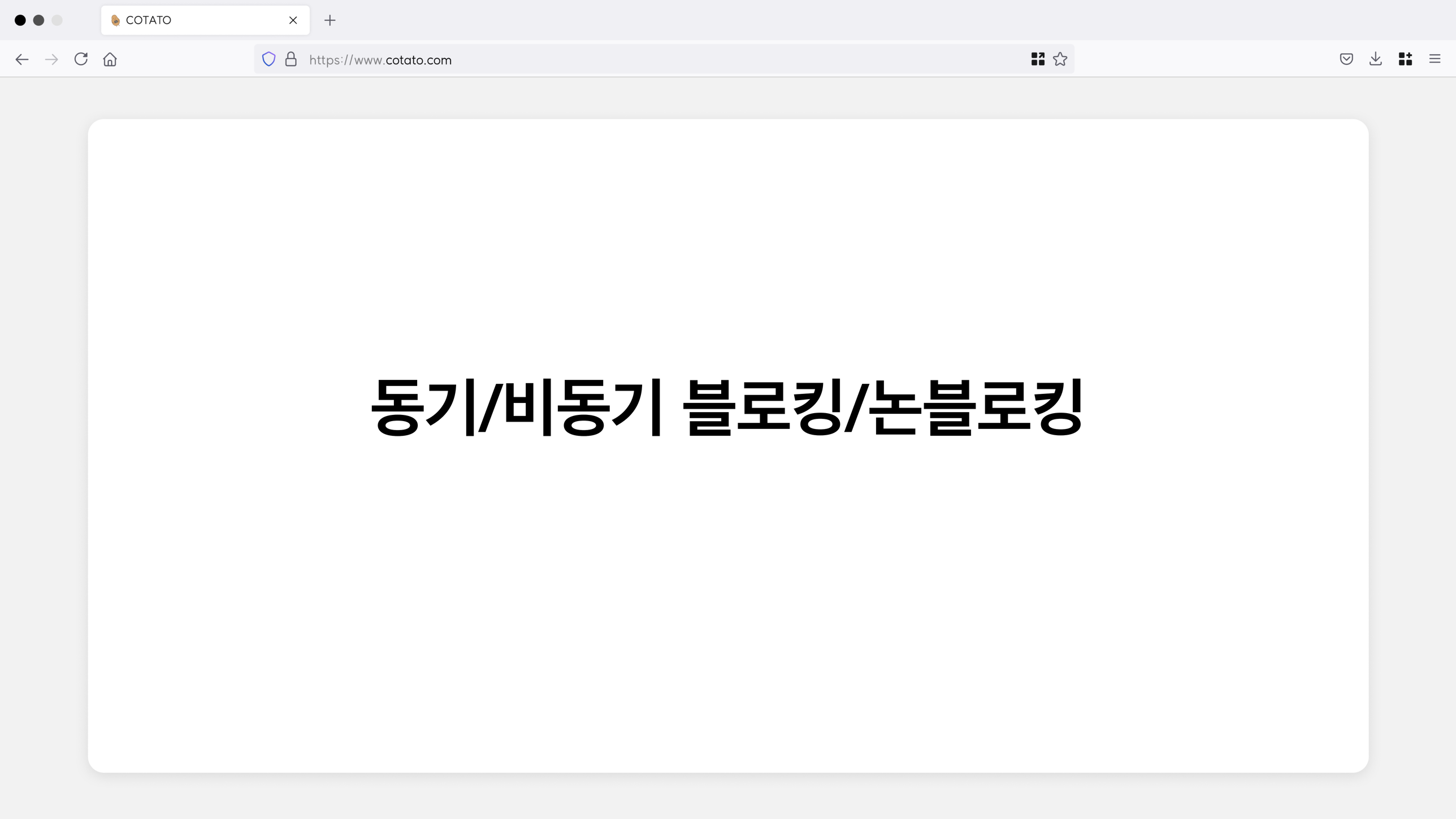

COTATO
https://www.cotato.com
CS EDUCATION
동시성 프로그래밍
23.09.22
최준영

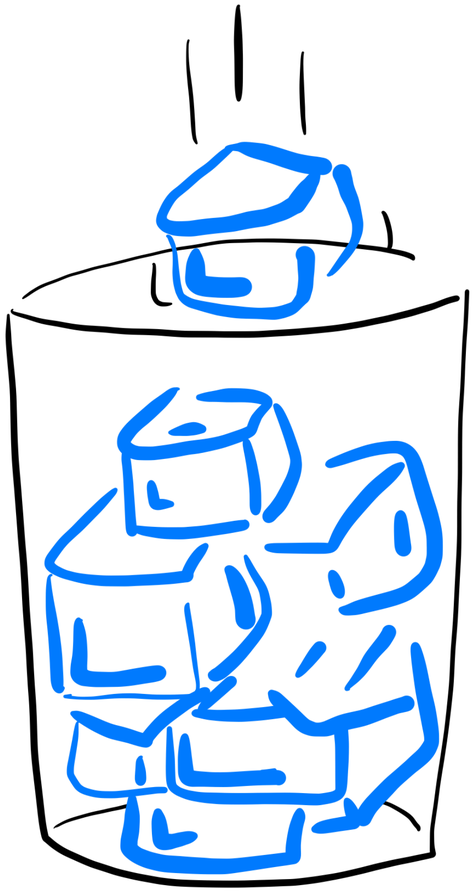


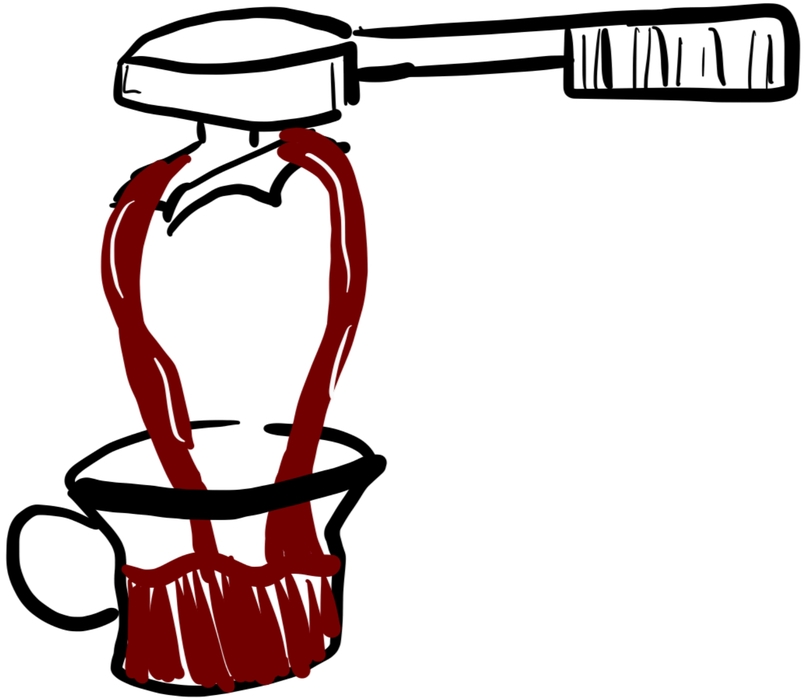
동시성 프로그래밍

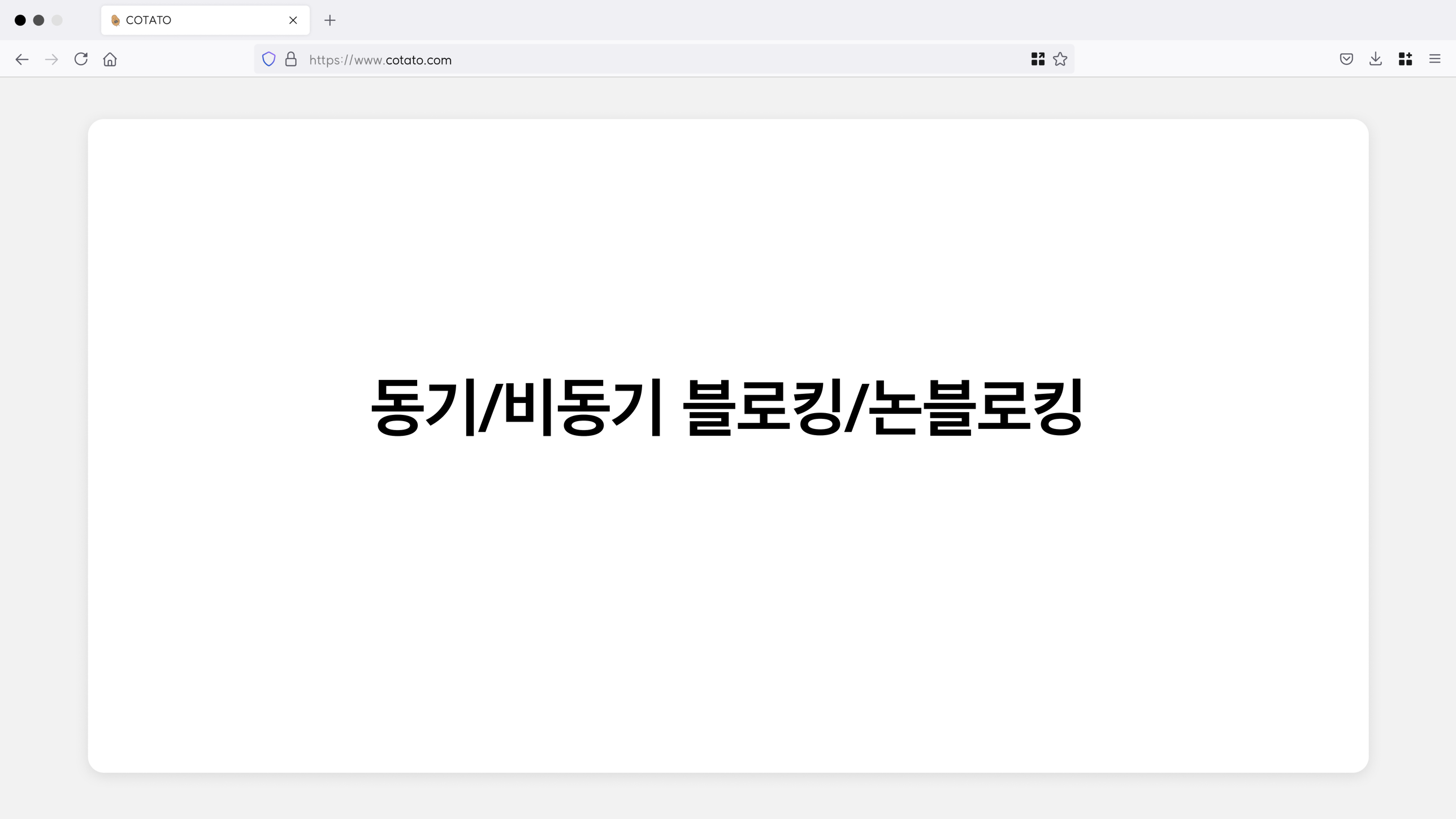

COTATO
https://www.cotato.com
동기/비동기 블로킹/논블로킹

COTATO

https://www.cotato.com

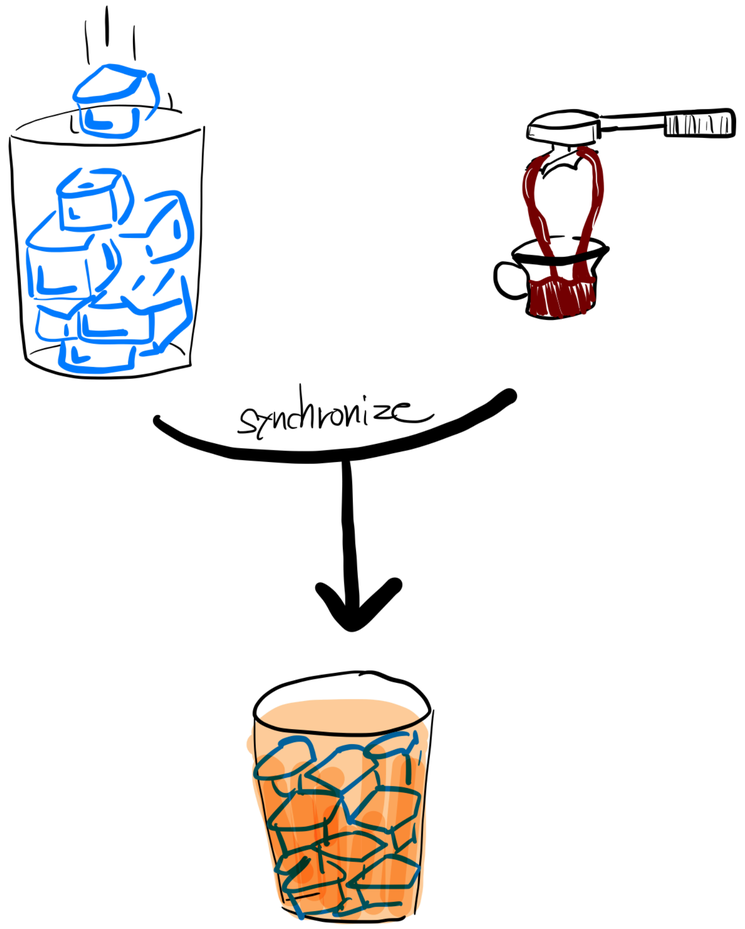


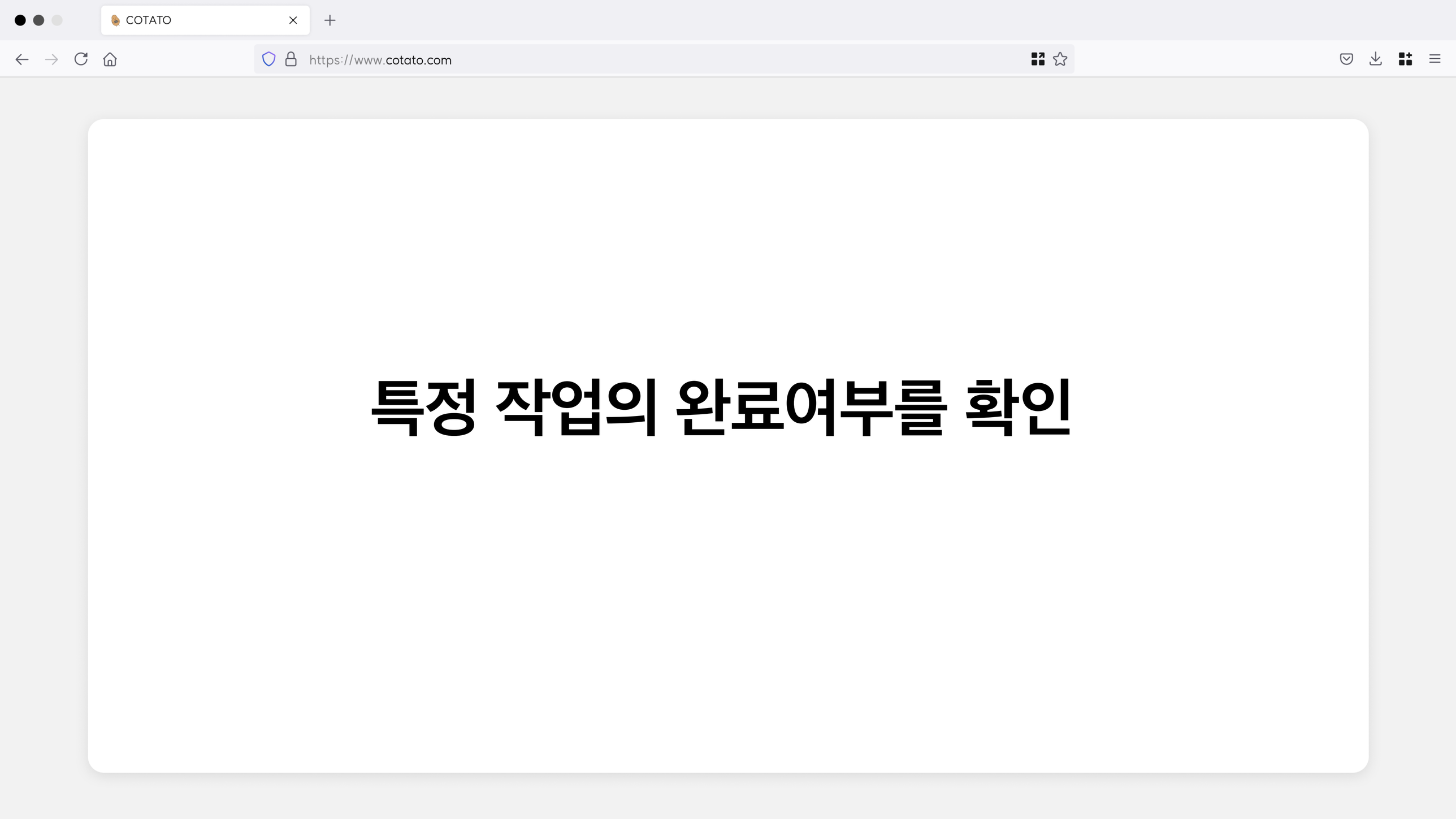



COTATO
https://www.cotato.com
동기/비동기 블로킹/논블로킹

동기(Synchronous)

비동기(Asynchronous)

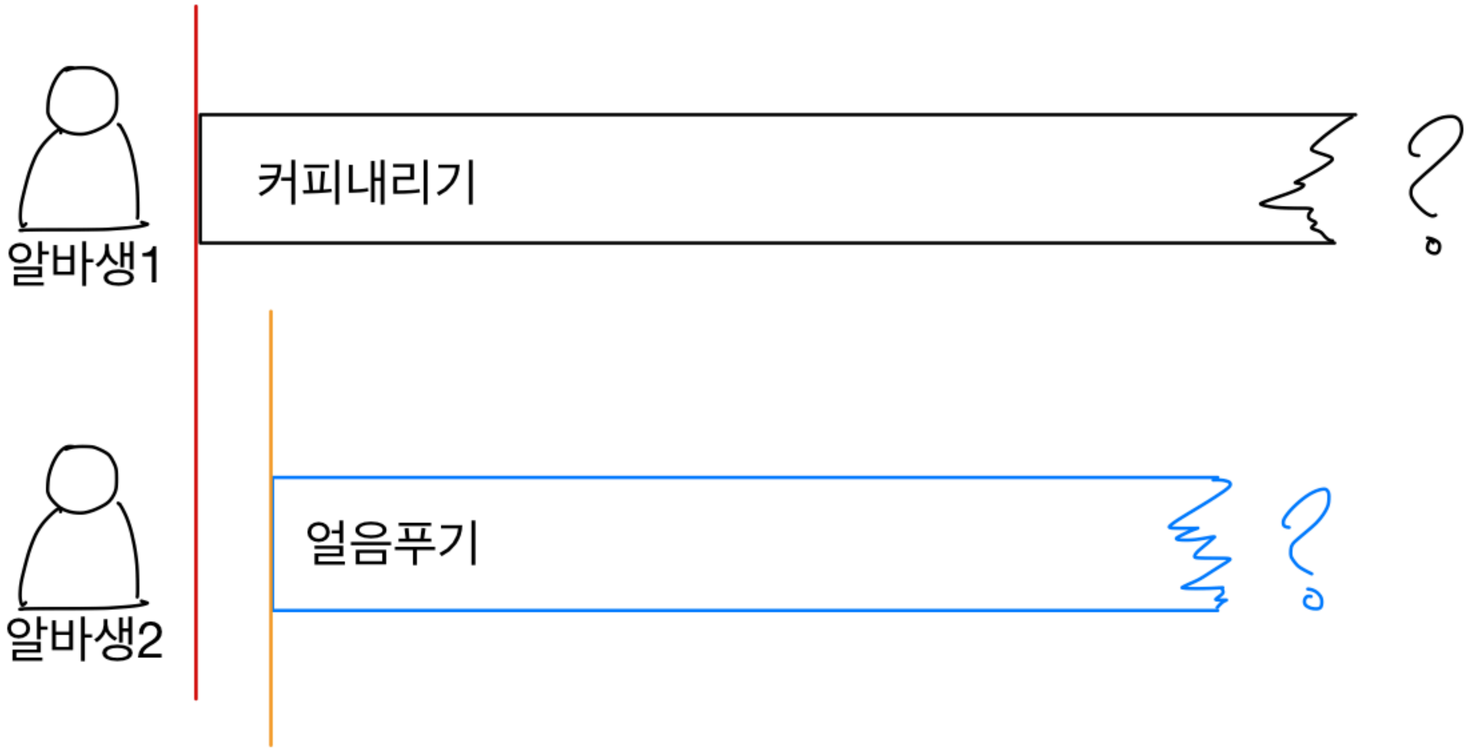

synchronize


COTATO
https://www.cotato.com
특정 작업의 완료여부를 확인

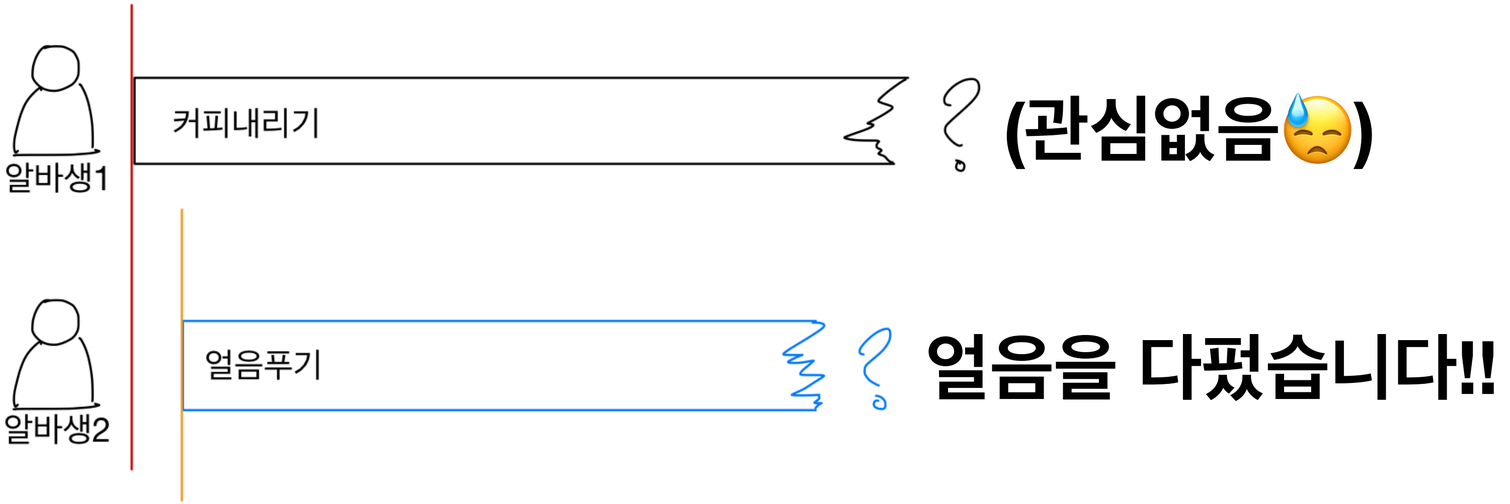
```swift
import Foundation

func makeCoffee() {
    for index in 0..<6 {
        print("커피 내림☕: \(20 * index)%")
        sleep(1)
    }
    print("알바생1: 얼음을 다폈습니다!!")
    isIceFinished = true
}


func scoopIce() {
    for index in 0..<5 {
        print("얼음 푸기❄: \(25 * index)%")
        sleep(1)
    }
    print("알바생1: 커피다내렸어요!!")
    isCoffeeFinished = true
}

var isCoffeeFinished = false
var isIceFinished = false

var iceOperation = BlockOperation(block: scoopIce)

OperationQueue().addOperations([iceOperation], waitUntilFinished: false)

makeCoffee()

while(!isCoffeeFinished && !isIceFinished) {
    sleep(1)
}
print("커피가 완성됐습니다.")
```

```swift
import Foundation

func makeCoffee() {
    for index in 0..<6 {
        print("커피 내림☺: \(20 * index)%")
        sleep(1)
    }
    print("알바생1: 얼음을 다펐습니다!!")
    isIceFinished = true
}


func scoopIce() {
    for index in 0..<5 {
        print("얼음 푸기❄: \(25 * index)%")
        sleep(1)
    }
    print("알바생1: 커피다내렸어요!!")
    isCoffeeFinished = true
}

var isCoffeeFinished = false
var isIceFinished = false

var iceOperation = BlockOperation(block: scoopIce)

OperationQueue().addOperations([iceOperation], waitUntilFinished: false)

makeCoffee()

while(!isCoffeeFinished && !isIceFinished) {
    sleep(1)
}
print("커피가 완성됐습니다.")
```

얼음 푸기❄: 0%
커피 내림☺: 0%
커피 내림☺: 20%
얼음 푸기❄: 25%
커피 내림☺: 40%
얼음 푸기❄: 50%
커피 내림☺: 60%
얼음 푸기❄: 75%
커피 내림☺: 80%
얼음 푸기❄: 100%
커피 내림☺: 100%
알바생1: 커피다내렸어요!!
알바생1: 얼음을 다펐습니다!!
커피가 완성됐습니다.

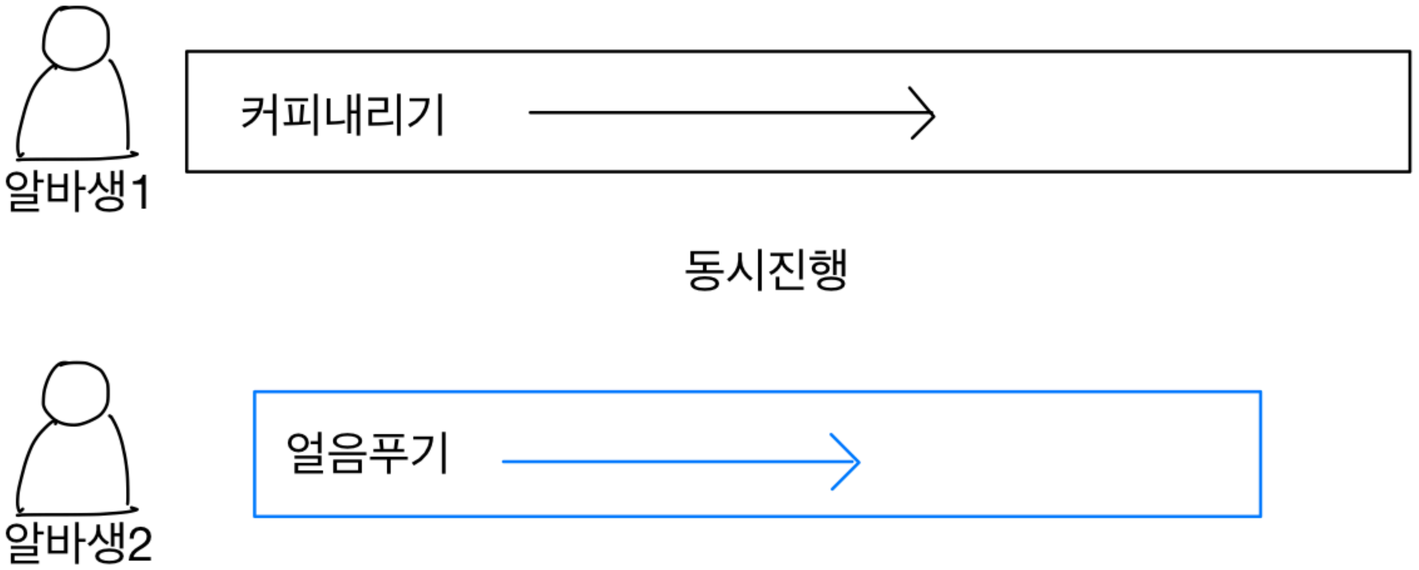


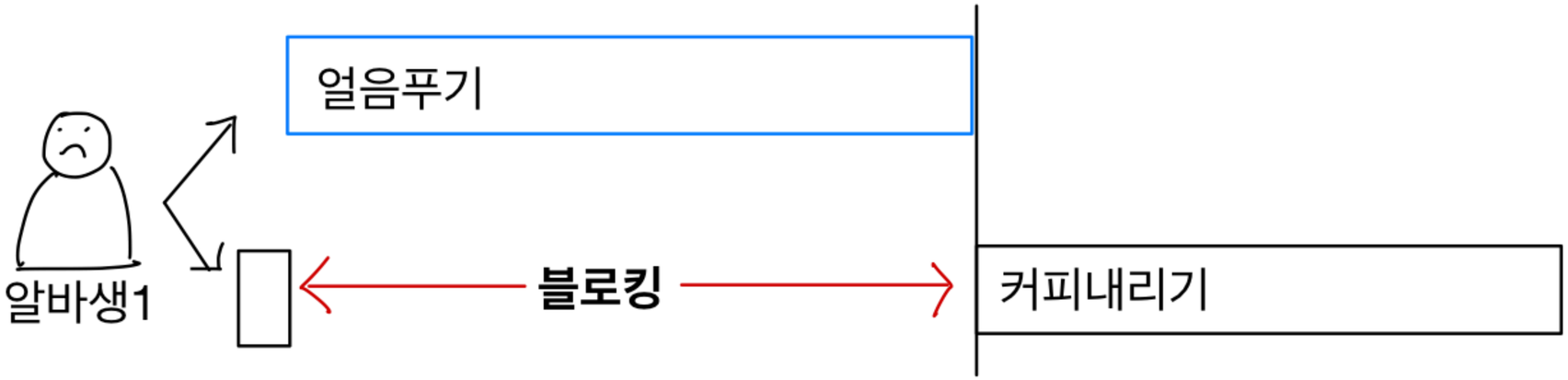
```
iceOperation.completionBlock = {
    print("알배생1: 얼음을 다펐습니다!!")
}
```



블로킹(Blocking)

논블로킹(Non-blocking)

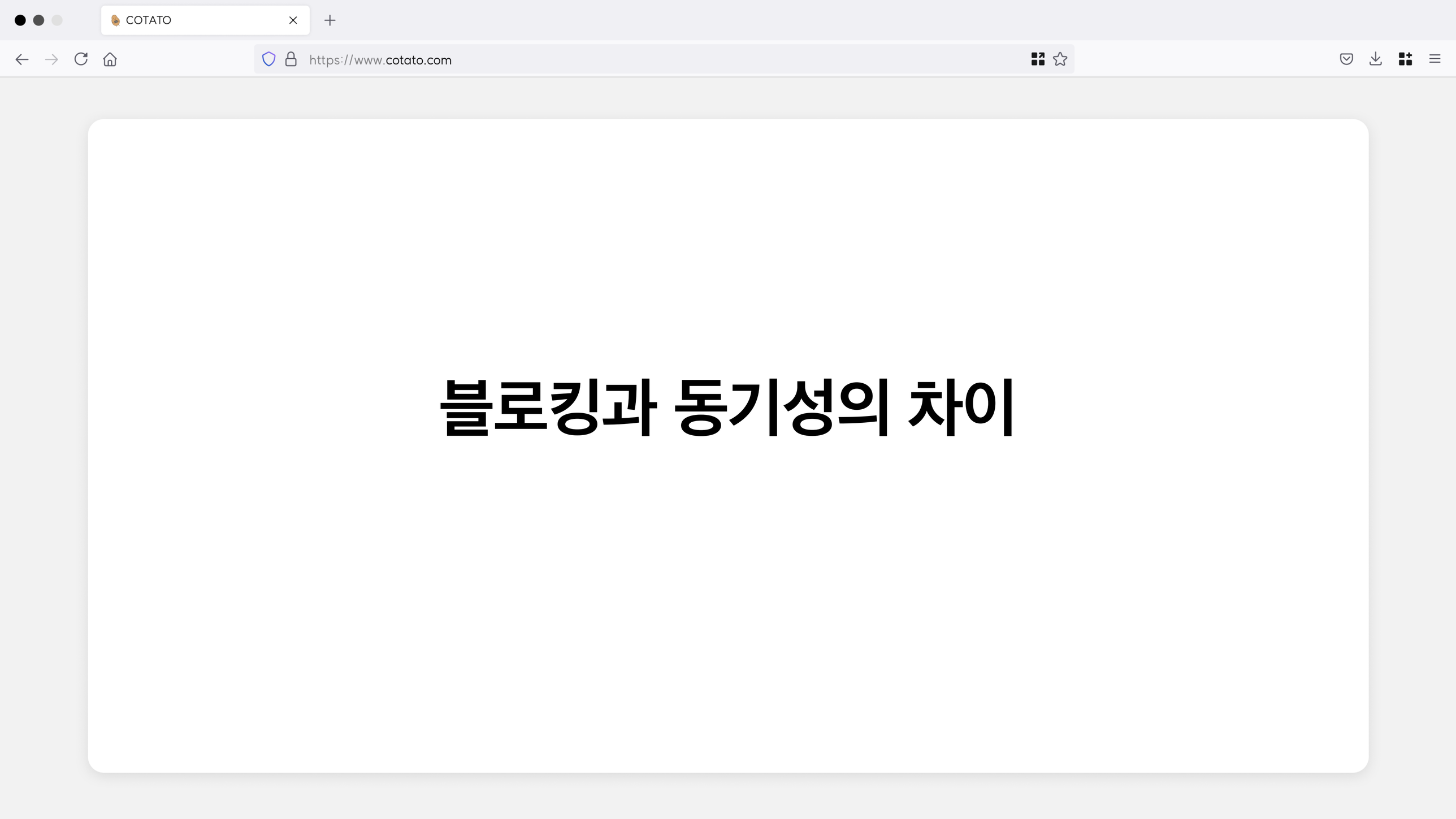


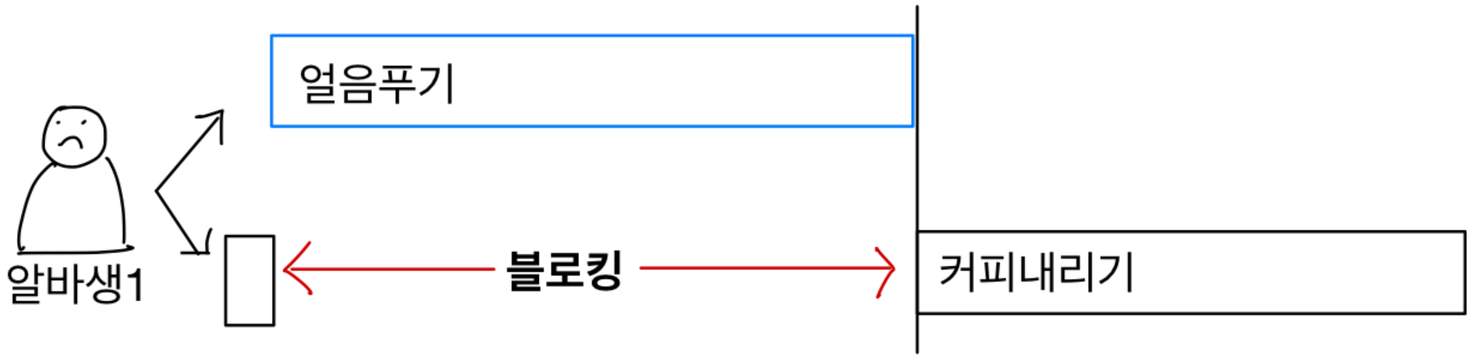


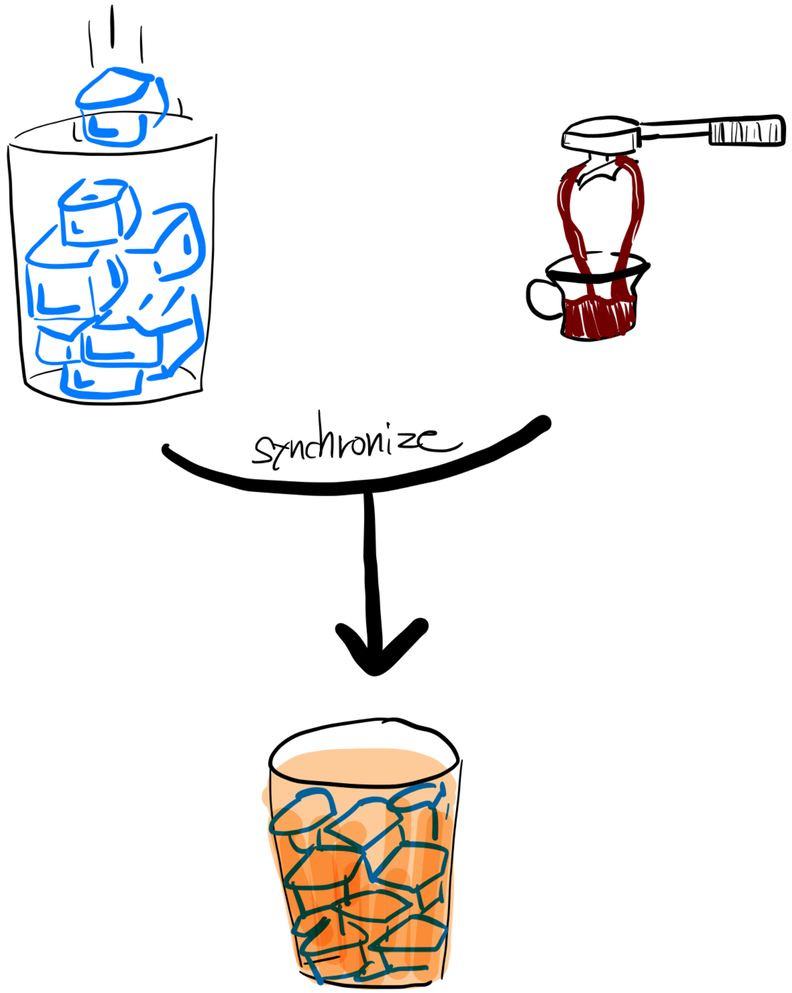
```
var iceOperation = BlockOperation(block: scoopIce)

OperationQueue().addOperations([iceOperation], waitUntilFinished: true)
//블로킹 시작지점

makeCoffee()
```

```swift
var iceOperation = BlockOperation(block: scoopIce)

OperationQueue().addOperations([iceOperation], waitUntilFinished: true)
//블로킹 시작지점

makeCoffee()
```

얼음 푸기❄️: 0%
얼음 푸기❄️: 25%
얼음 푸기❄️: 50%
얼음 푸기❄️: 75%
얼음 푸기❄️: 100%
알바생1: 얼음을 다펐습니다!!
커피 내림☺️: 0%
커피 내림☺️: 20%
커피 내림☺️: 40%
커피 내림☺️: 60%
커피 내림☺️: 80%
커피 내림☺️: 100%
알바생1: 커피다내렸어요!!


COTATO
https://www.cotato.com
블로킹과 동기성의 차이

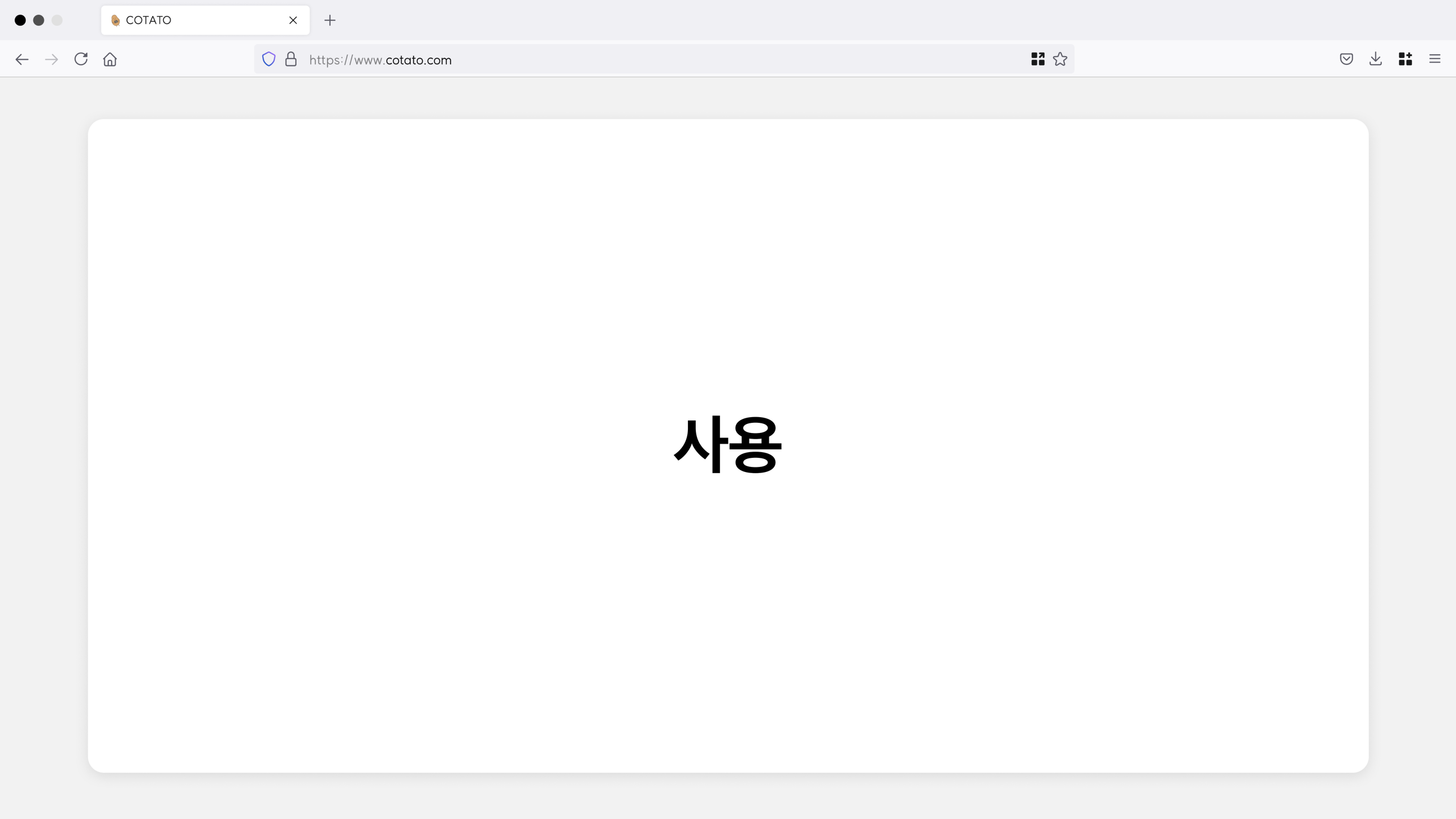


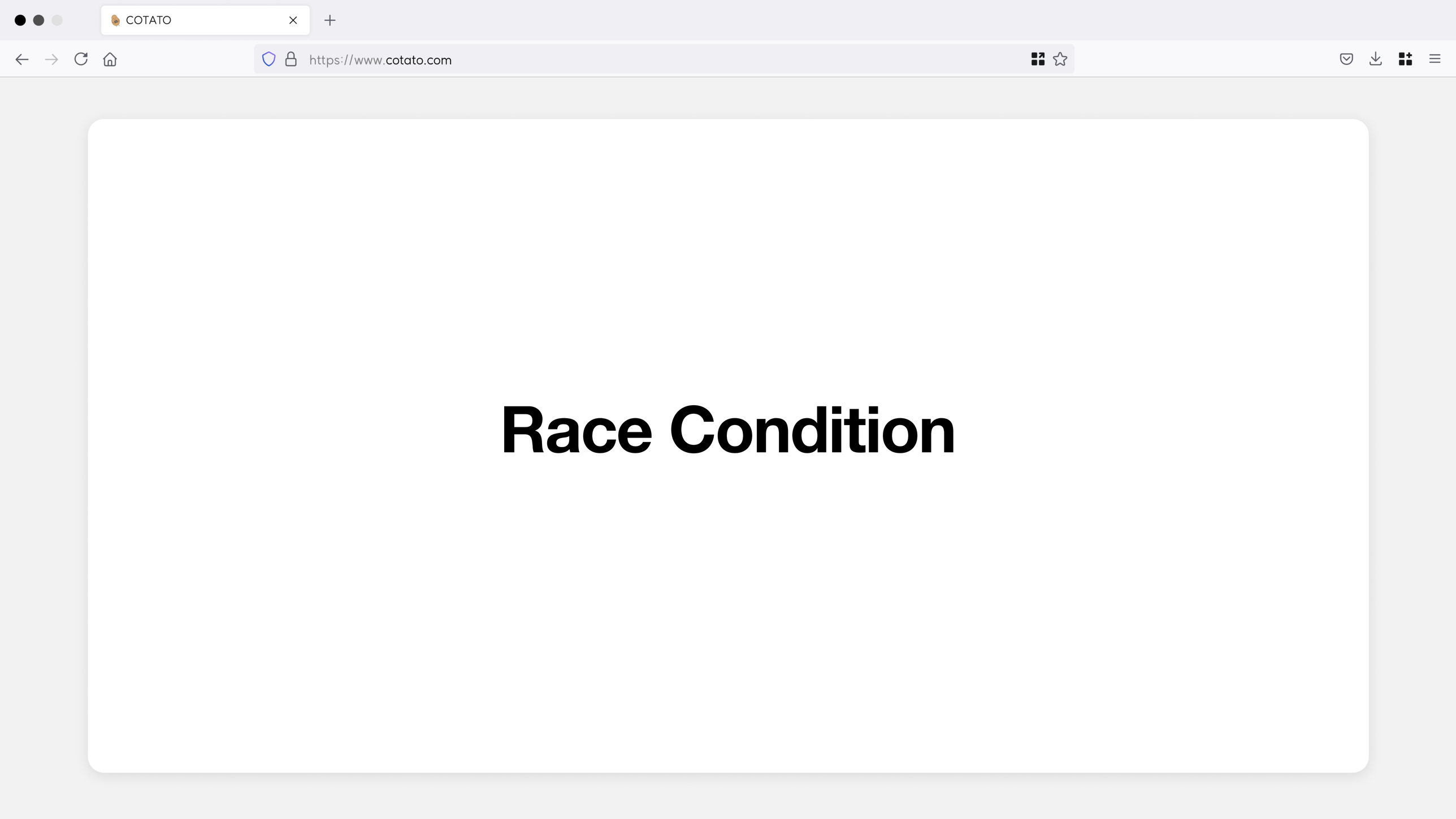

synchronize

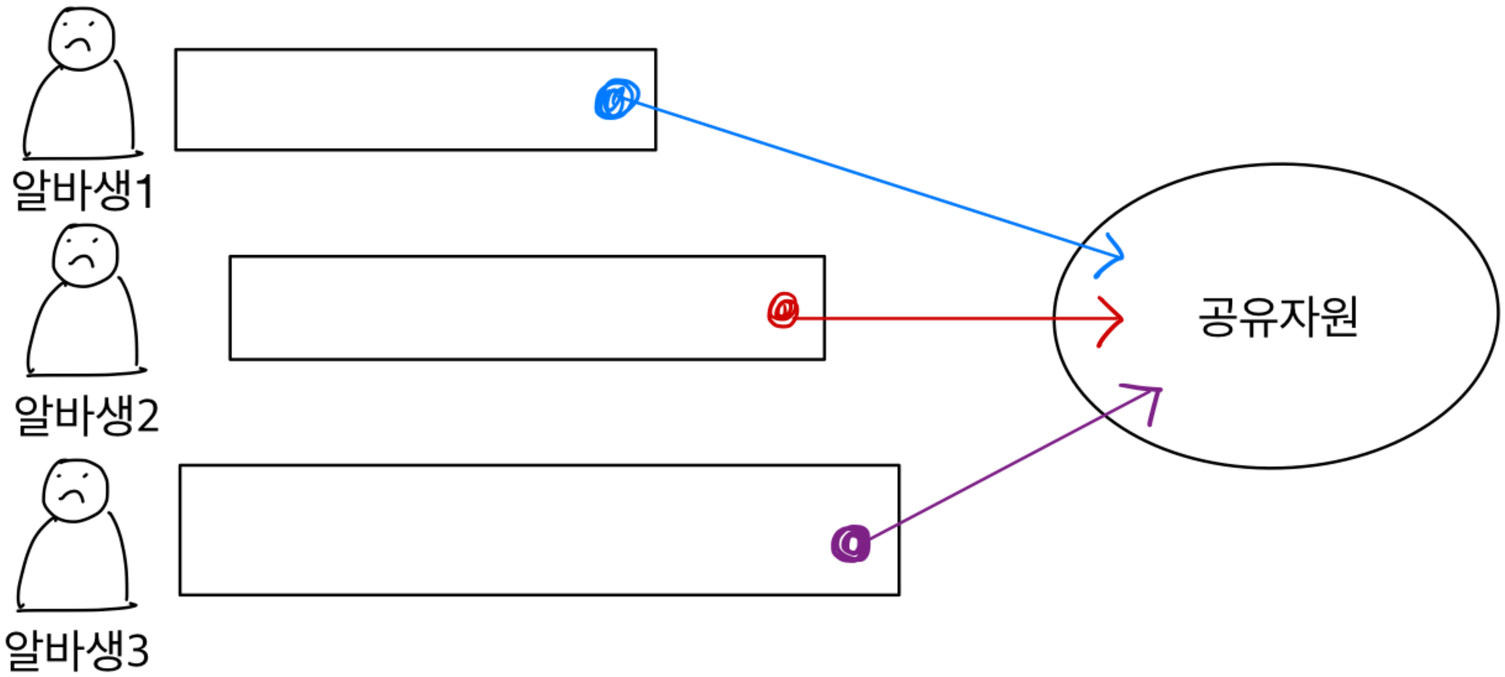
```swift
import Foundation

func makeCoffee() {
    for index in 0..<6 {
        print("커피 내림☕: \(20 * index)%")
        sleep(1)
    }
    print("알바생1: 얼음을 다펐습니다!!")
    isIceFinished = true
}


func scoopIce() {
    for index in 0..<5 {
        print("얼음 푸기❄: \(25 * index)%")
        sleep(1)
    }
    print("알바생1: 커피다내렸어요!!")
    isCoffeeFinished = true
}

var isCoffeeFinished = false
var isIceFinished = false

var iceOperation = BlockOperation(block: scoopIce)

OperationQueue().addOperations([iceOperation], waitUntilFinished: false)

makeCoffee()

while(!isCoffeeFinished && !isIceFinished) {
    sleep(1)
}
print("커피가 완성됐습니다.")
```

```swift
import Foundation

func makeCoffee() {
    for index in 0..<6 {
        print("커피 내림☺: \(20 * index)%")
        sleep(1)
    }
    print("알바생1: 얼음을 다펐습니다!!")
    isIceFinished = true
}


func scoopIce() {
    for index in 0..<5 {
        print("얼음 푸기❄: \(25 * index)%")
        sleep(1)
    }
    print("알바생1: 커피다내렸어요!!")
    isCoffeeFinished = true
}

var isCoffeeFinished = false
var isIceFinished = false

var iceOperation = BlockOperation(block: scoopIce)

OperationQueue().addOperations([iceOperation], waitUntilFinished: false)

makeCoffee()

while(!isCoffeeFinished && !isIceFinished) {
    sleep(1)
}
print("커피가 완성됐습니다.")
```

얼음 푸기❄: 0%
커피 내림☺: 0%
커피 내림☺: 20%
얼음 푸기❄: 25%
커피 내림☺: 40%
얼음 푸기❄: 50%
커피 내림☺: 60%
얼음 푸기❄: 75%
커피 내림☺: 80%
얼음 푸기❄: 100%
커피 내림☺: 100%
알바생1: 커피다내렸어요!!
알바생1: 얼음을 다펐습니다!!
커피가 완성됐습니다.


COTATO
https://www.cotato.com
사용

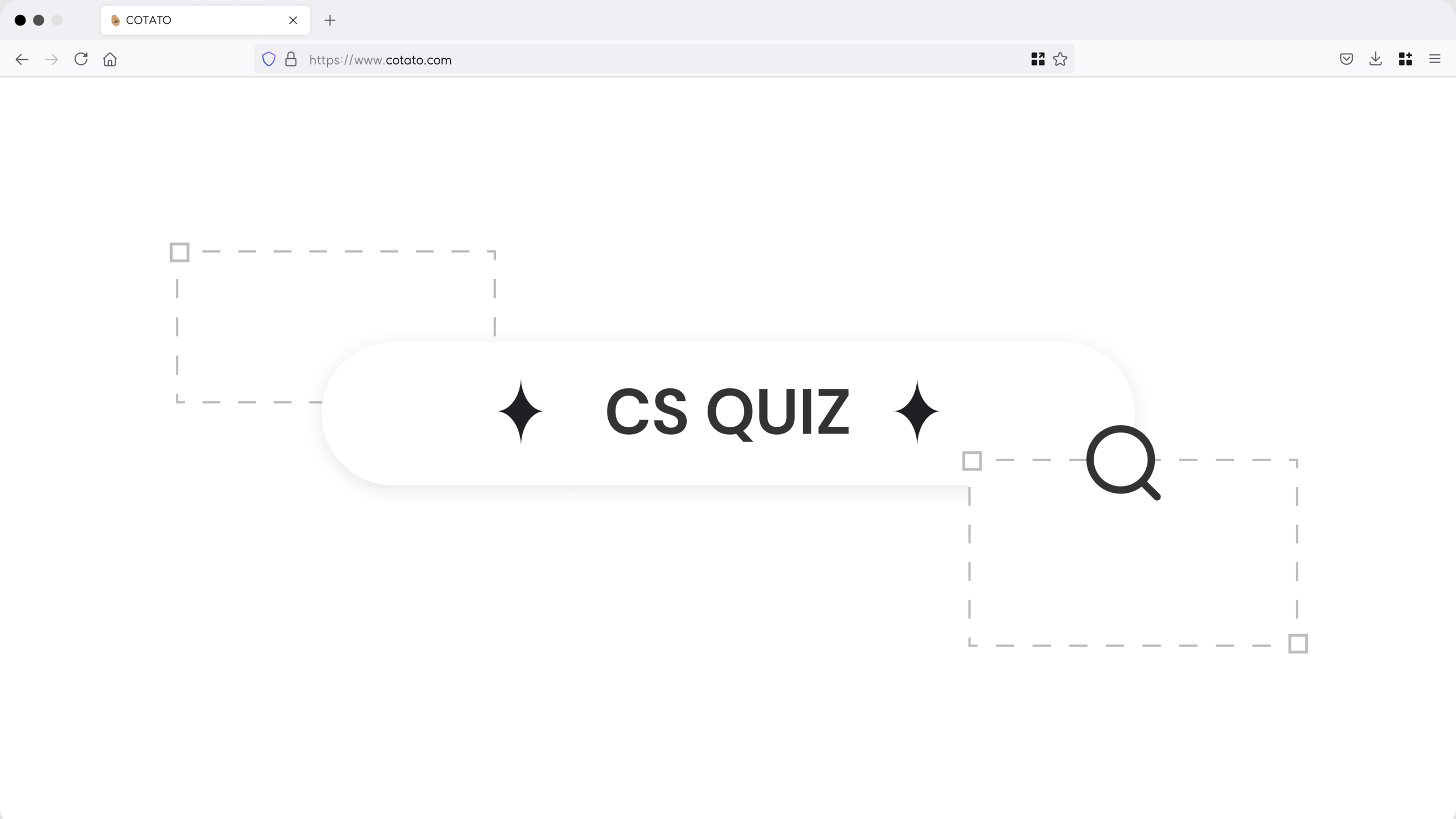

COTATO
https://www.cotato.com
Race Condition



```swift
var array = Array(1...10)

DispatchQueue.global().async {
    for _ in 1...3 {
        print("준영", array.removeFirst())
    }
}

DispatchQueue.global().async {
    for _ in 1...3 {
        print("창훈", array.removeFirst())
    }
}

DispatchQueue.global().async {
    for _ in 1...3 {
        print("동수", array.removeFirst())
    }
}
```

```swift
var array = Array(1...10)

DispatchQueue.global().async {
    for _ in 1...3 {
        print("준영", array.removeFirst())
    }
}

DispatchQueue.global().async {
    for _ in 1...3 {
        print("창훈", array.removeFirst())
    }
}

DispatchQueue.global().async {
    for _ in 1...3 {
        print("동수", array.removeFirst())
    }
}
```

```
창훈 1
준영 1
동수 2
창훈 3
창훈 5
동수 6
준영 6
준영 8
동수 8
```


COTATO
https://www.cotato.com
CS QUIZ

Q.1

멀티스레드 환경에서는 각 스레드에서 공유된 자원에 접근하는 경우 서로 같은 자원에 접근해도 다른 값을 읽는 문제가 발생할 수 있다.
이러한 문제를 [ ]이라고 한다.

A.1

멀티스레드 환경에서는 각 스레드에서 공유된 자원에 접근하는 경우 서로 같은 자원에 접근해도 다른 값을 읽는 문제가 발생할 수 있다.
이러한 문제를 Race Condition 이라고 한다.

Q.2

다음 설명은 어떠한 프로그래밍과 관련된 개념인지 고르시오.

코테이토 운영진은 IT동아리인만큼 세션 준비를 할때 특정 프로그래밍 방법을 사용해 일을 나눠서 진행한다. 회장 박감자와 부회장 김감자는 세션 PPT를, 교육팀은 CS 교육 자료 준비를, 운영지원팀 박감자는 대면, 비대면 참가자를 구분하고 이를 세션전에 통합한다. 이와 같이 서로 다른일을 동시에 진행하고 마지막에 결과물로 동기화하는 방식으로 진

1. 페어 프로그래밍
2. 동시성 프로그래밍
3. 객체지향 프로그래밍
4. 선언형 프로그래밍

A.2

다음 설명은 어떠한 프로그래밍과 관련된 개념인지 고르시오.

코테이토 운영진은 IT동아리인만큼 세션 준비를 할때 특정 프로그래밍 방법을 사용해 일을 나눠서 진행한다. 회장 박감자와 부회장 김감자는 세션 PPT를, 교육팀은 CS 교육 자료 준비를, 운영지원팀 박감자는 대면, 비대면 참가자를 구분하고 이를 세션전에 통합한다. 이와 같이 서로 다른일을 동시에 진행하고 마지막에 결과물로 동기화하는 방식으로 진

1. 페어 프로그래밍
2. 동시성 프로그래밍
3. 객체지향 프로그래밍
4. 선언형 프로그래밍

Q.3 다음 보기를 읽고 알맞은 것을 모두 고르시오

카페에서 일하는 영준이는 같이 일하는 승우를 도저히 믿을 수 없다. 그래서 영준이는 승우가 하는 작업이 끝날때까지 도저히 자기 일을 진행할 수 없다.

1. 승우의 작업이 영준의 작업을 블로킹했군

2. 영준은 승우의 작업 완료를 확인한다는 점에서 두 작업은 비동기성을 띄어

3. 불안하지만 영준이 승우의 작업이 끝나기 전에 작업을 진행한다면 두 작업은 동시성을 띄어

A.3 다음 보기를 읽고 알맞은 것을 모두 고르시오

카페에서 일하는 영준이는 같이 일하는 승우를 도저히 믿을 수 없다. 그래서 영준이는 승우가 하는 작업이 끝날때까지 도저히 자기 일을 진행할 수 없다.

1. 승우의 작업이 영준의 작업을 블로킹했군

2. 영준은 승우의 작업 완료를 확인한다는 점에서 두 작업은 비동기성을 띄어

3. 불안하지만 영준이 승우의 작업이 끝나기 전에 작업을 진행한다면 두 작업은 동시성을 띄어

Q.4

다음 밑줄 친 부분에 해당하는 기술 용어를 입력하시오.

식당 홀에서 일하는 두식은 주문이 들어오면 주문 내역을 주방에 전달한다. 주문만 받는 두식은 음식의 완료여부를 신경쓰지 않는다. 주방장은 주문내역의 요리가 완성되면 음식의 완료(종료)를 확인하기 위해 주방 한쪽에 부착된 벨을 울려 서빙을 지시한다.

A.4

다음 밑줄 친 부분에 해당하는 기술 용어를 입력하시오.

식당 홀에서 일하는 두식은 주문이 들어오면 주문 내역을 주방에 전달한다. 주문만 받는 두식은 음식의 완료여부를 신경쓰지 않는다. 주방장은 주문내역의 요리가 완성되면 음식의 완료(종료)를 확인하기 위해 주방 한쪽에 부착된 벨을 울려 서빙을 지시한다.

callback 함수/블록

Q.5

빈칸에 들어갈 알맞은 말을 순서대로 쓰세요.

학생이 시험지를 선생에게 건넨 직후 가만히 앉아 채점이 끝나서 시험지를 돌려받기만을 기다린다면 학생은 현재 [ ]상태입니다. 하지만 학생이 시험지를 건넨 후 선생에게 채점이 완료되었다는 확인을 받기 전까지 다른 과목을 공부한다거나 게임을 하거나 다른 일을 하게 되면 현재 학생 상태는 [ ]상태가 됩니다.

A.5

빈칸에 들어갈 알맞은 말을 순서대로 쓰세요.

학생이 시험지를 선생에게 건넨 직후 가만히 앉아 채점이 끝나서 시험지를 돌려받기만을 기다린다면 학생은 현재 [블록] 상태입니다. 하지만 학생이 시험지를 건넨 후 선생에게 채점이 완료되었다는 확인을 받기 전까지 다른 과목을 공부한다거나 게임을 하거나 다른 일을 하게 되면 현재 학생 상태는 [논블록] 상태가 됩니다.

Q.6

동기적으로 작동되는 프로그램은 멀티쓰레드 환경에서 사용될 수 없다. (o/x)

A.6

동기적으로 작동되는 프로그램은 멀티쓰레드 환경에서 사용될 수 없다. (o/x)

Q.7 올바른 설명을 모두 고르시오

1. 동기성 프로그램은 작업 완료 여부를 논 블로킹 방식으로 확인할 수 있다.
2. 멀티쓰레드 프로그램은 최대한 많은 수의 쓰레드를 생성하는게 효율적이다.
3. 콜백 함수는 동기 프로그래밍 환경에서 작업의 완료 시 실행되는 함수를 의미한다.
4. 비동기성은 특정 작업의 완료시점을 신경쓰지 않는 개념이다.

A.7 올바른 설명을 모두 고르시오

1. 동기성 프로그램은 작업 완료 여부를 논 블로킹 방식으로 확인할 수 있다.
2. 멀티쓰레드 프로그램은 최대한 많은 수의 쓰레드를 생성하는게 효율적이다.
3. 콜백 함수는 동기 프로그래밍 환경에서 작업의 완료 시 실행되는 함수를 의미한다.
4. 비동기성은 특정 작업의 완료시점을 신경쓰지 않는 개념이다.

Q.8

동기성과 비동기성은 작업의 완료시점을 신경쓰느냐의 여부에 따른 차이가 있다. [ ]은 다른 작업의 완료시점을 신경쓰는 방식이고[ ]은 다른 작업의 완료 시점을 신경쓰지 않는 방식이다.

A.8

동기성과 비동기성은 작업의 완료시점을 신경쓰느냐의 여부에 따른 차이가 있다. 비동기성은 다른 작업의 완료시점을 신경쓰는 방식이고 동기성은 다른 작업의 완료 시점을 신경쓰지 않는 방식이다.

Q.9 다음 중 틀린 내용을 고르시오.

1. 블로킹은 특정 작업이 끝날때까지 기다려 CPU자원을 불필요하게 낭비할 수 있다.

2. 동기성방식을 사용하기 위해서는 블로킹 방식을 통해 모든 작업들의 종료 여부를 확인해야 한다.

3. 논블로킹에서 함수가 종료되면 곧바로 다음 함수를 실행시킨다.

4. 해당작업이 끝나면 호출되는 코드블록이 전달되는데, 이때 전달되는 코드블록을 콜백함수라 한다.

A.9 다음 중 틀린 내용을 고르시오.

1. 블로킹은 특정 작업이 끝날때까지 기다려 CPU자원을 불필요하게 낭비할 수 있다.

2. 동기성방식을 사용하기 위해서는 블로킹 방식을 통해 모든 작업들의 종료 여부를 확인해야 한다.

3. 논블로킹에서 함수가 종료되면 곧바로 다음 함수를 실행시킨다.

4. 해당작업이 끝나면 호출되는 코드블록이 전달되는데, 이때 전달되는 코드블록을 콜백함수라 한다.

다음 중 동기와 비동기에 대해 옳지 않은 이야기를 하는 친구를 고르세요.

1. 제니: 동기는 설계가 매우 간단하고 직관적이지만, 비동기는 동기보다 복잡해.
2. 민지: 동기는 작업의 완료 여부를 확인하고 기다리는 반면, 비동기는 완료 여부를 신경 쓰지 않고 다른 작업을 수행해.
3. 주혁: 비동기방식은 실제 프로그래밍 환경에서 작업의 종료여부를 콜백함수를 통해 알 수 있어.
4. 은우: 동기는 추구하는 행위(목적)가 다를 수도 있고, 동시에 이루어지지도 않아. 하지만 비동기는 추구하는 행위(목적)가 동시에 이루어져.

다음 중 동기와 비동기에 대해 옳지 않은 이야기를 하는 친구를 고르세요.

1. 제니: 동기는 설계가 매우 간단하고 직관적이지만, 비동기는 동기보다 복잡해.
2. 민지: 동기는 작업의 완료 여부를 확인하고 기다리는 반면, 비동기는 완료 여부를 신경 쓰지 않고 다른 작업을 수행해.
3. 주혁: 비동기방식은 실제 프로그래밍 환경에서 작업의 종료여부를 콜백함수를 통해 알 수 있어.
4. 은우: 동기는 추구하는 행위(목적)가 다를 수도 있고, 동시에 이루어지지도 않아. 하지만 비동기는 추구하는 행위(목적)가 동시에 이루어져.