

BApp Dev Lab 1

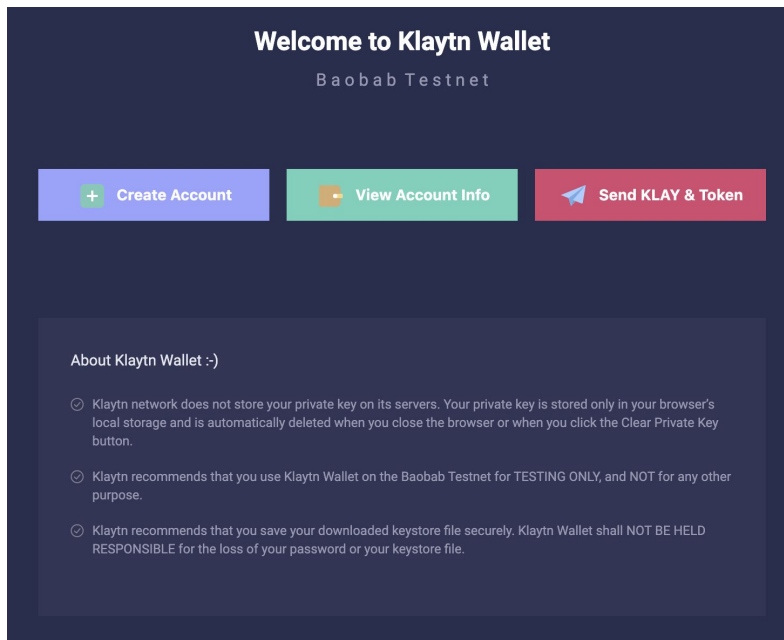
Count Smart Contract

목표

- Baobab 어카운트 생성
- KLAY Faucet 사용
- Klaytn IDE
 - Count 스마트 컨트랙트 작성
 - Count.sol 컴파일 & 배포
 - Count 컨트랙트 실행
- 로컬환경에서 컨트랙트 컴파일
- 바이트코드를 사용하여 컨트랙트 배포

Baobab 어카운트 생성 1/2

- Klaytn Wallet을 사용 - <https://baobab.wallet.klaytn.com/>



Baobab 어카운트 생성 2/2

1

2

3

STEP 1

Please Set Password for Your Keystore File

This is the first step of creating your new Klaytn account.
Please set the password for the keystore file storing your account's private key.

Password

ⓘ Klaytn Wallet is for development purpose only. KLAY on testnet has no financial value.

Next Step

✓

2

3

STEP 2

Please Download Keystore File

The password for your keystore file has been set.
Click the button below to download the file and move on to the final step.

Download & Next Step

✓

✓

3

STEP 3

Please Save Your Klaytn Wallet Key

Your new account has been created.
Please copy and securely store the Klaytn Wallet Key below.

Private Key

0xa22e8c9f87f9c2d7b61855006f85a7f2d0546c06b82f34378ff22fe54cc00fdd

COPY

Klaytn Wallet Key

0xa22e8c9f87f9c2d7b61855006f85a7f2d0546c06b82f34378ff22fe54cc00fdd0x00x49efe27ae4e95ebd86081884210905e70a4755e1

COPY

View Account Info

Send KLAY & Token

Baobab 어카운트 확인

Klaytn Wallet

+

 Create Account

+

 View Account Info

→

 Send KLAY & Token

💧

 KLAY Faucet

⋮

 More

Clear Private Key

Baobab Testnet

My Account Info

Address

Hex 0x49efe27ae4e95ebd86081884210905e70a4755e1

COPY

Private Key ⓘ

.....

COPY

Klaytn Wallet Key ⓘ

.....

COPY

Transaction List

Explore all transactions involving your account with Klaytnscope.

View Transaction List

Download Keystore File

Keystore file securely stores your private key and account address.

keystore Download

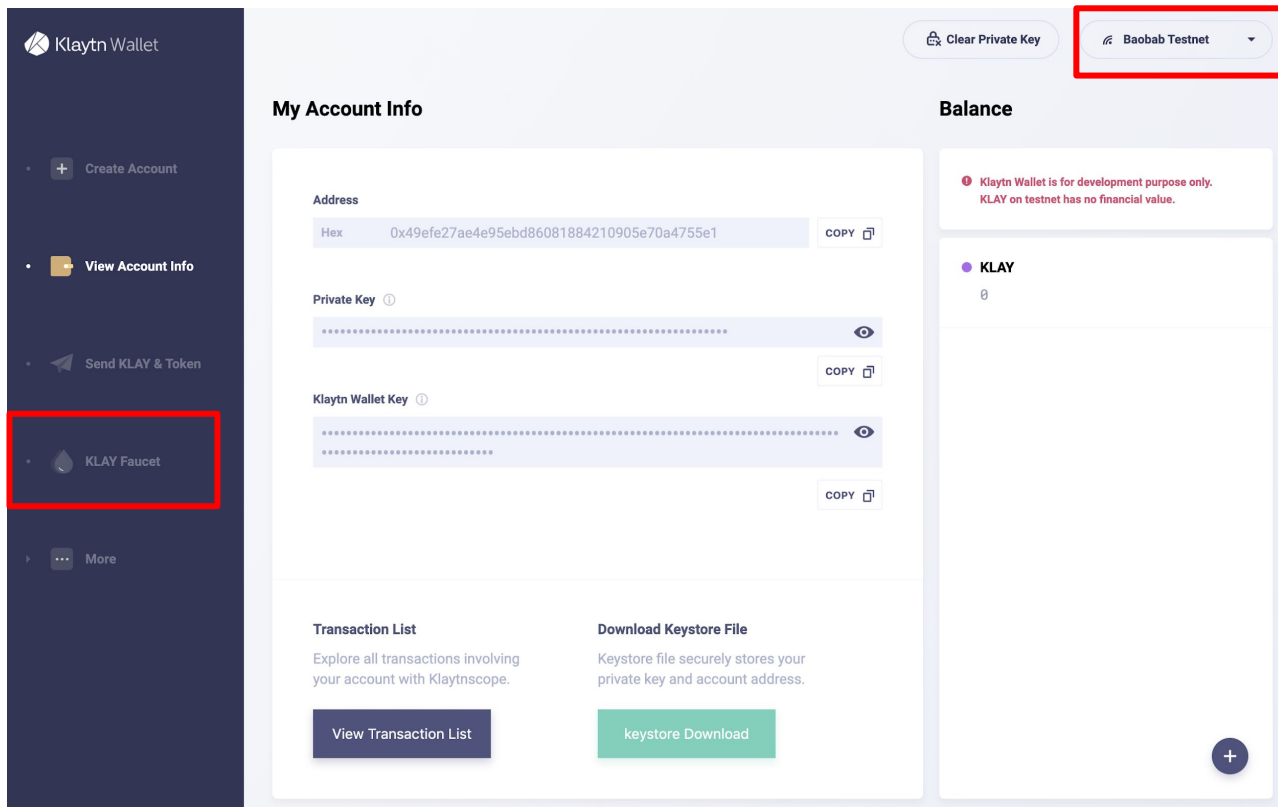
Balance

Klaytn Wallet is for development purpose only.
KLAY on testnet has no financial value.

KLAY

0

KLAY Faucet 사용



KLAY Faucet

Only for Baobab

KLAY Faucet

The KLAY Faucet runs on Baobab Testnet.

Account Address

0x49efe27ae4e95ebd86081884210905e70a4755e1

KLAY Balance

0 KLAY

✔ Faucet is ready to run.

Run Faucet

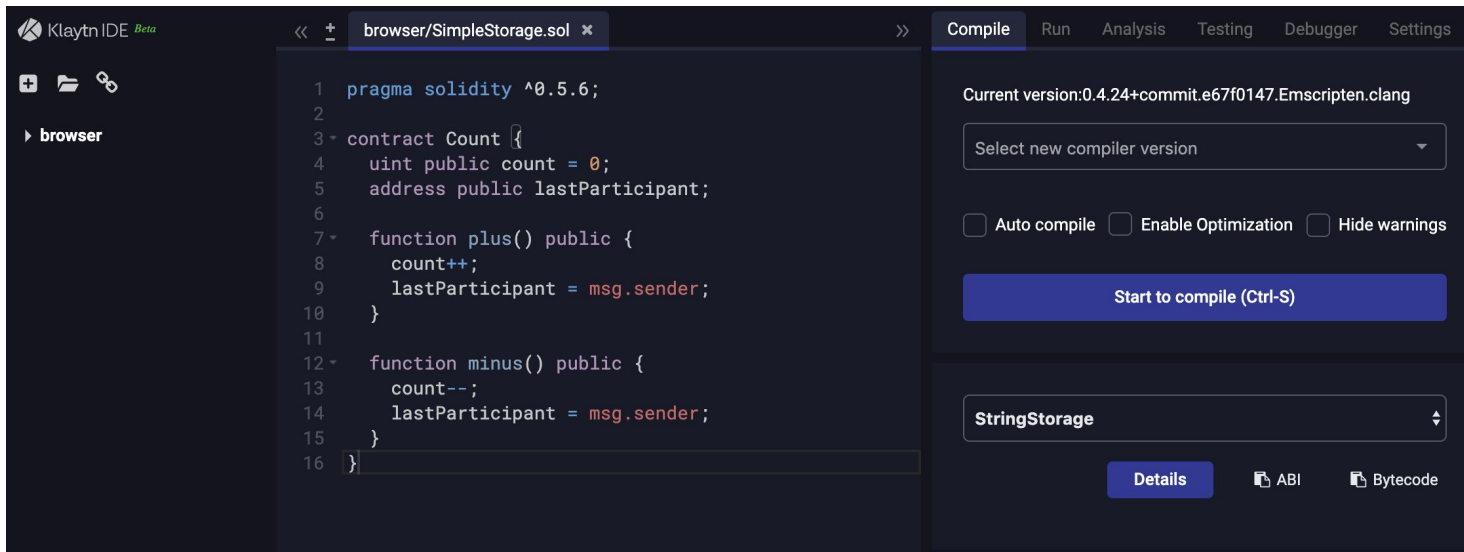
❓ How does the Klay Faucet work?

KLAY Faucet sends a small amount of KLAY to the designated address for testing. To ensure fair distribution to the community, you can run KLAY Faucet only once every 24 hours. The KLAY reserve in Faucet is replenished every 24 hours.

- 5 KLAY/day
- IP당 1,000회 리밋 설정

Klaytn IDE (Beta)

- 손쉬운 컨트랙트 작성, 컴파일, 배포, 테스트를 위한 도구
- <http://ide.klaytn.com/>



Count 스마트 컨트랙트

```
pragma solidity ^0.5.6;

contract Count {
    uint public count = 0;
    address public lastParticipant;

    function plus() public {
        count++;
        lastParticipant = msg.sender;
    }

    function minus() public {
        count--;
        lastParticipant = msg.sender;
    }
}
```

(IDE) 컨트랙트 컴파일

The screenshot shows a Solidity IDE interface. On the left, a code editor displays a Solidity contract named 'Count'. The code includes a pragma statement for Solidity version ^0.5.6, a contract definition with two public functions 'plus()' and 'minus()', and a 'lastParticipant' variable. The right panel contains a 'Compile' tab with a dropdown menu for selecting a compiler version. The current version is '0.5.6+commit.b259423e.Emscripten.clang'. A modal dialog is open, showing a list of available compiler versions: '0.5.6+commit.b259423e' and '0.4.24+commit.e67f0147'. Below the dropdown, there are checkboxes for 'Auto compile', 'Enable Optimization', and 'Hide warnings'. A large blue button labeled 'Start to compile (Ctrl-S)' is present. At the bottom of the right panel, there is a section for the compiled contract, showing the name 'Count' and buttons for 'Details', 'ABI', and 'Bytecode'. A status bar at the very bottom shows a green progress indicator and the name 'Count'.

```
1 pragma solidity ^0.5.6;
2
3 contract Count {
4     uint public count = 0;
5     address public lastParticipant;
6
7     function plus() public {
8         count++;
9         lastParticipant = msg.sender;
10    }
11
12    function minus() public {
13        count--;
14        lastParticipant = msg.sender;
15    }
16 }
```

Compile Run Analysis Testing Debugger Settings

Current version:0.5.6+commit.b259423e.Emscripten.clang

✓ Select new compiler version

- 0.5.6+commit.b259423e
- 0.4.24+commit.e67f0147

☐ Auto compile ☐ Enable Optimization ☐ Hide warnings

Start to compile (Ctrl-S)

Count

Details ABI Bytecode

Count

(IDE) 컨트랙트 배포 1/4

The screenshot displays a Solidity IDE interface with a dark theme. On the left, a code editor shows a Solidity contract named 'Count' with the following code:

```
1 pragma solidity ^0.5.6;
2
3 contract Count {
4     uint public count = 0;
5     address public lastParticipant;
6
7     function plus() public {
8         count++;
9         lastParticipant = msg.sender;
10    }
11
12    function minus() public {
13        count--;
14        lastParticipant = msg.sender;
15    }
16 }
```

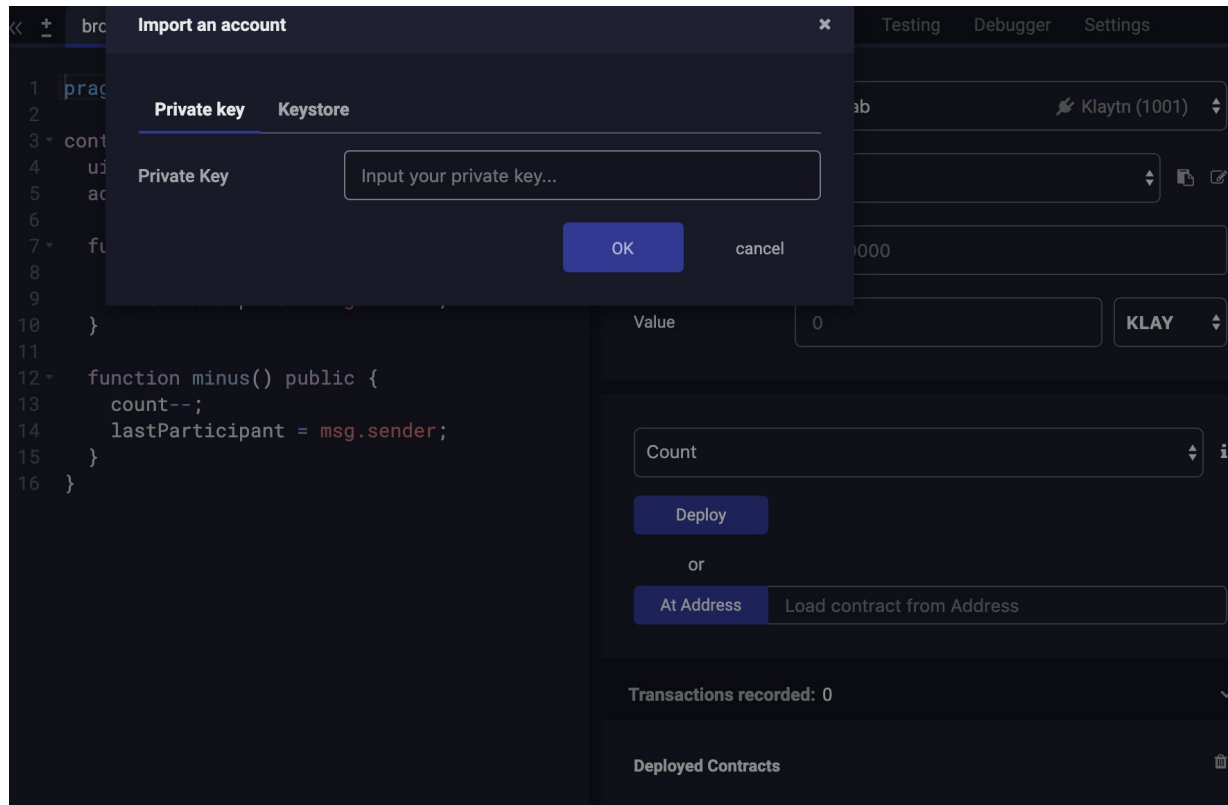
On the right, the 'Run' tab is active, showing deployment settings:

- Environment:** Baobab (Klaytn (1001))
- Account:** (empty field with a dropdown arrow and icons for file and copy)
- Gas limit:** 3000000
- Value:** 0 (with a 'KLAY' dropdown arrow)

Below these settings, there is a 'Count' dropdown menu with an information icon. Underneath is a 'Deploy' button. Below the 'Deploy' button is the text 'or' and a section with a tab labeled 'At Address' and a text input field containing 'Load contract from Address'.

At the bottom, there is a section for 'Transactions recorded: 0' with a dropdown arrow, and a section for 'Deployed Contracts' with a trash icon.

(IDE) 컨트랙트 배포 2/4



(IDE) 컨트랙트 배포 3/4



browser/SimpleStorage.sol ✕

<< >>

Compile Run Analysis Testing Debugger Settings


```
1 pragma solidity ^0.5.6;
2
3 contract Count {
4     uint public count = 0;
5     address public lastParticipant;
6
7     function plus() public {
8         count++;
9         lastParticipant = msg.sender;
10    }
11
12    function minus() public {
13        count--;
14        lastParticipant = msg.sender;
15    }
16 }
```

Environment Baobab Klaytn (1001) ▾

Account 0x49e...755e1 (5 KLAY) ▾  

Gas limit 3000000

Value 0 KLAY ▾


Count ▾ 

Deploy

or

At Address Load contract from Address

Transactions recorded: 0 ▾

Deployed Contracts 

(IDE) 컨트랙트 배포 4/4

The screenshot displays the Remix IDE interface. On the left, the Solidity code editor shows a contract named `Count` with the following code:

```
1 pragma solidity ^0.5.6;
2
3 contract Count {
4     uint public count = 0;
5     address public lastParticipant;
6
7     function plus() public {
8         count++;
9         lastParticipant = msg.sender;
10    }
11
12    function minus() public {
13        count--;
14        lastParticipant = msg.sender;
15    }
16 }
```

On the right, the **Run** tab is active, showing deployment configuration:

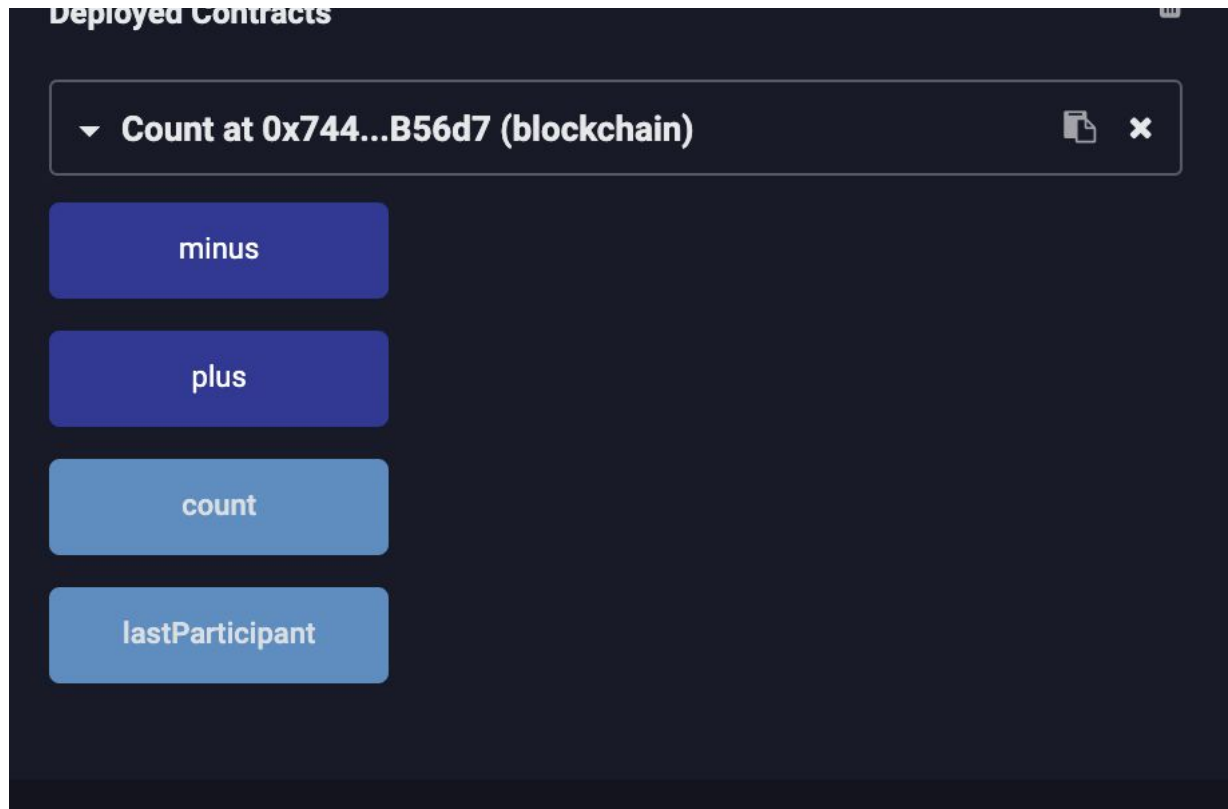
- Environment:** Baobab (Klaytn (1001))
- Account:** 0x49e...755e1 (4.995442525 KLAY)
- Gas limit:** 3000000
- Value:** 0 KLAY

Below the configuration, there are two deployment options:

- Count** (with an info icon)
- Deploy** button
- or**
- At Address** button with a text input field containing "Load contract from Address"

At the bottom, it shows **Transactions recorded: 1** and a section for **Deployed Contracts** with a trash icon. A partial entry is visible: "Count at 0x744...B56d7 (blockchain)".

(IDE) 컨트랙트 실행



로컬환경에서 컨트랙트 컴파일

- **solc** - Solidity 컴파일러
 - npm (linux, macos, Windows)으로 light version을 설치 가능 (**solcjs**, partial feature)

```
$ npm install -g solc
```

- apt (debian linux), brew (macos) 등으로 binary 설치 가능 (**solc**, full feature)

```
(debian)
$ sudo add-apt-repository ppa:ethereum/ethereum
$ sudo apt-get update
$ sudo apt-get install solc
```

```
(macos)
$ brew tap ethereum/ethereum
$ brew install solidity
```


로컬환경에서 컨트랙트 컴파일

- VS Code를 사용하여 Count.sol 파일을 생성
- 생성된 파일에는 Count 컨트랙트를 기록
- 터미널로 파일이 생성된 위치로 이동
- Count.sol이 위치한 디렉토리에서 **solc** 또는 **solcjs**를 실행

```
$ cd <path_so_Count_sol>
$ ls
Count.sol
$ solcjs --bin true --abi true -o out Count.sol
$ tree out
out
├── Count_sol_Count.abi
└── Count_sol_Count.bin

0 directories, 2 files
```

```
$ solc --bin --abi -o out Count.sol
$ tree out
out
├── Count.abi
└── Count.bin

0 directories, 2 files
```

바이트코드를 사용하여 컨트랙트 배포

- 컴파일된 컨트랙트를 caver-js를 사용하여 배포
- VS Code로 caver-js가 설치된 디렉토리에 deploy.js를 작성

caver-js 설치법

```
$ npm install caver-js
```

바이트코드를 사용하여 컨트랙트 배포

- 다음과 같이 바이트코드와 ABI를 코드에 복사

```
// BEGINNING OF deploy.js
```

```
const Caver = require('caver-js');
```

```
const caver = new Caver('https://api.baobab.klaytn.net:8651/');
```

```
const acct =
```

```
caver.klay.accounts.privateKeyToAccount('0x5fe365ada739df8ff4ece807d7565be943281f860fb814b65d2378efcf3a2f7e');
```

```
caver.klay.accounts.wallet.add(acct);
```

```
// Refer to count.sol for Solidity code
```

```
const COUNT_BYTECODE = '60806040526000805534801561001457600080fd5b5060e...';
```

```
const COUNT_ABI = [...];
```

바이트코드를 사용하여 컨트랙트 배포

- 다음과 같이 컨트랙트 객체를 생성 후 바이트코드를 Klaytn에 전송

```
new caver.klay.Contract(COUNT_ABI)
    .deploy({
        data: COUNT_BYTECODE,
        arguments: []
    })
    .send({
        from: acct.address,
        gas: 3000000,
        value: 0
    })
    .on('error', console.log)
    .on('transactionHash', function (hash) { console.log(">>> tx_hash for deployment:", hash); })
    .on('receipt', function (receipt) { console.log(">>> contract deployed at", receipt.contractAddress); });
```

바이트코드를 사용하여 컨트랙트 배포

- 터미널에서 다음과 같이 deploy.js를 실행

```
$ node deploy.js
```

```
>>> tx_hash for deployment: 0x0f0e1a3a46ab5edc595f4f8dc40c49f1b5745b9ca7c2a77cb03930930670ddf8
```

```
>>> contract deployed at 0x2425CB31E79a7076aB307db8111ca8c7da053F68
```