



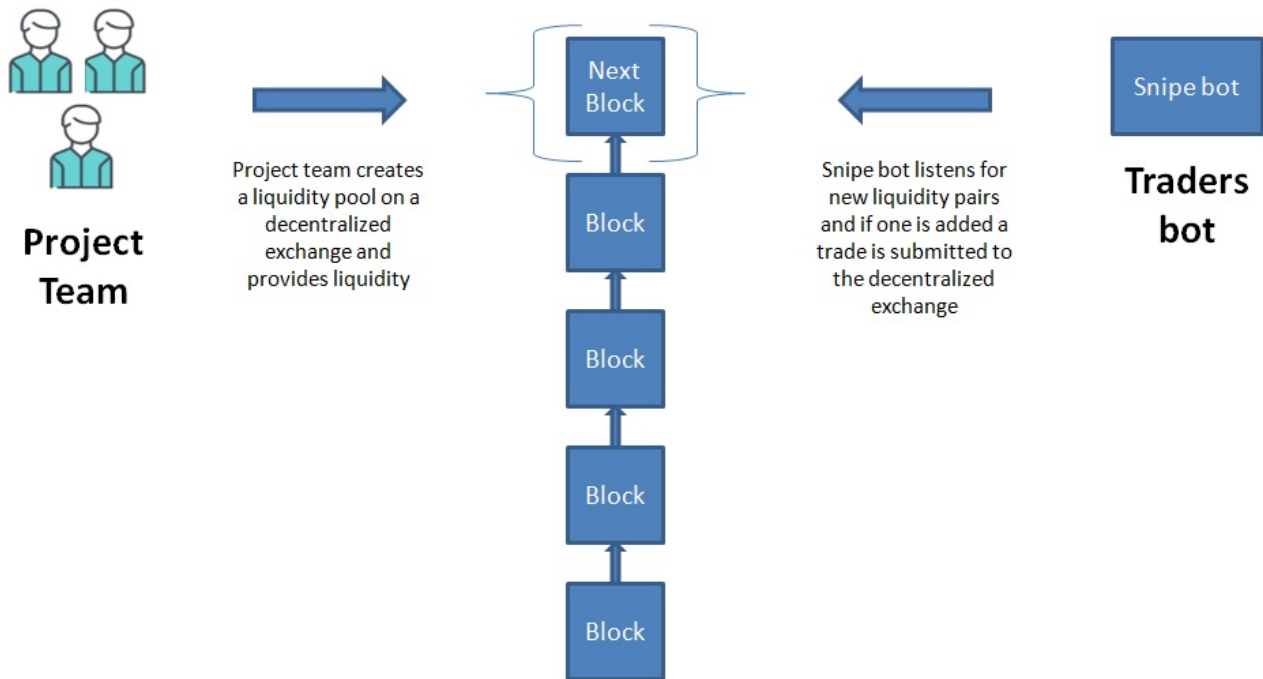
## Build a snipe bot to monitor and trade liquidity pairs

Build a snipe bot to monitor and trade liquidity pairs on decentralized exchanges. When a new liquidity pool is created on a decentralized exchange (for example [Uniswap](#)) traders can profit if they are the first to buy the new tokens and then sell them to new buyers. This strategy is called sniping and the key is to be the first to buy the tokens when the liquidity pool is added to the decentralized exchange. It is preferred to buy these tokens in the same block, or next several blocks, as the creation and funding of the liquidity pool.

Sniping is an effective strategy when you want to buy an IDO (Initial DEX “Decentralized Exchange” offering). IDO’s are a fundraising method in which token are issued on a decentralized exchange and sold to new buyers. When the liquidity pool is added to the DEX the snipe bot can place a trade to immediately buy the tokens.

In this tutorial we will build a snipe bot to monitor and trade liquidity pairs. Python and Web3.py monitor the [Ethereum](#) blockchain. When a new liquidity pool is created on Uniswap the snipe bot will place a trade.

## Decentralized exchange snipe trading bot process



The snipe trading bot will perform the following functions:

- First the Python process listens to Uniswap events for newly created token pairs
- Then the system identifies and process new token pairs
- Finally the bot submits a transaction to buy the new token using a smart contract

## Smart contract to swap tokens on a DEX

First we need to create a smart contract on the [Ethereum](#) blockchain. The snipe bot will swap in and out of tokens on Uniswap using this smart contract. Click here for detailed instructions on [how to swap tokens on Uniswap using the smart contract below](#). Read the comments in the code to better understand how the contract functions.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.0;

//import the ERC20 interface

interface IERC20 {
    function totalSupply() external view returns (uint);
```

```

function totalSupply() external view returns (uint);
function balanceOf(address account) external view returns (uint);
function transfer(address recipient, uint amount) external returns (bool);
function allowance(address owner, address spender) external view returns (uint);
function approve(address spender, uint amount) external returns (bool);
function transferFrom(
    address sender,
    address recipient,
    uint amount
) external returns (bool);
event Transfer(address indexed from, address indexed to, uint value);
event Approval(address indexed owner, address indexed spender, uint value);
}

```

```

//import the Uniswap router
//the contract needs to use swapExactTokensForTokens
//this will allow us to import swapExactTokensForTokens into our contract

```

```

interface IUniswapV2Router {
    function getAmountsOut(uint256 amountIn, address[] memory path)
        external
        view
        returns (uint256[] memory amounts);

    function swapExactTokensForTokens(
        //amount of tokens we are sending in
        uint256 amountIn,
        //the minimum amount of tokens we want out of the trade
        uint256 amountOutMin,
        //list of token addresses we are going to trade in. this is necessary to
        address[] calldata path,
        //this is the address we are going to send the output tokens to
        address to,
        //the last time that the trade is valid for
        uint256 deadline
    ) external returns (uint256[] memory amounts);
}

```

```

interface IUniswapV2Pair {
    function token0() external view returns (address);
    function token1() external view returns (address);
    function swap(
        uint256 amount0Out,
        uint256 amount1Out,
        address to,
        bytes calldata data
    ) external;
}

```

```

interface IUniswapV2Factory {
    function getPair(address token0, address token1) external returns (address);
}

```

```

contract tokenSwap {

    //address of the Uniswap v2 router
    address private constant UNISWAP_V2_ROUTER = 0x7a250d5630B4cF539739dF2C5

    //address of WETH token. This is needed because some times it is better
    //you might get a better price using WETH.
    //example trading from token A to WETH then WETH to token B might result
    address private constant WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756C

    //this swap function is used to trade from one token to another
    //the inputs are self explanatory
    //token in = the token address you want to trade out of
    //token out = the token address you want as the output of this trade
    //amount in = the amount of tokens you are sending in
    //amount out Min = the minimum amount of tokens you want out of the trad
    //to = the address you want the tokens to be sent to

    function swap(address _tokenIn, address _tokenOut, uint256 _amountIn, uin

    //first we need to transfer the amount in tokens from the msg.sender to
    //this contract will have the amount of in tokens
    IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);

    //next we need to allow the uniswapv2 router to spend the token we just
    //by calling IERC20 approve you allow the uniswap contract to spend the
    IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

    //path is an array of addresses.
    //this path array will have 3 addresses [tokenIn, WETH, tokenOut]
    //the if statement below takes into account if token in or token out is
    address[] memory path;
    if (_tokenIn == WETH || _tokenOut == WETH) {
        path = new address[](2);
        path[0] = _tokenIn;
        path[1] = _tokenOut;
    } else {
        path = new address[](3);
        path[0] = _tokenIn;
        path[1] = WETH;
        path[2] = _tokenOut;
    }

    //then we will call swapExactTokensForTokens
    //for the deadline we will pass in block.timestamp
    //the deadline is the latest time the trade is valid for
    IUniswapV2Router(UNISWAP_V2_ROUTER).swapExactTokensForTokens(_amount

}

//this function will return the minimum amount from a swap
//input the 3 parameters below and it will return the minimum amount

```

```

//this is needed for the swap function above
function getAmountOutMin(address _tokenIn, address _tokenOut, uint256 _

//path is an array of addresses.
//this path array will have 3 addresses [tokenIn, WETH, tokenOut]
//the if statement below takes into account if token in or token out
address[] memory path;
if (_tokenIn == WETH || _tokenOut == WETH) {
    path = new address[](2);
    path[0] = _tokenIn;
    path[1] = _tokenOut;
} else {
    path = new address[](3);
    path[0] = _tokenIn;
    path[1] = WETH;
    path[2] = _tokenOut;
}

uint256[] memory amountOutMins = IUniswapV2Router(UNISWAP_V2_ROUTER)
return amountOutMins[path.length -1];

}

}

```

Try it in [Remix](#)

## Build a snipe trading bot in Python to monitor liquidity pairs

Next, build a Python program that listens for new liquidity pairs created on the Uniswap decentralized exchange. This program will run in a loop and check Uniswap every 2 seconds for new liquidity pairs. The program prints new Uniswap liquidity pair information to the console.

For more information on how the code below works read the tutorial [how to listen for Ethereum events using Web3.py in Python](#). Read the comments in the Python code below to better understand how this process works.

```

# import the following dependencies
import json
from web3 import Web3
import asyncio

```

```

# add your blockchain connection information
infura_url = 'ADDYOURINFURAURL'
web3 = Web3(Web3.HTTPProvider(infura_url))

# uniswap address and abi
uniswap_router = '0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D'
uniswap_factory = '0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f'
uniswap_factory_abi = json.loads(' [{"inputs": [{"internalType": "address", "nam

contract = web3.eth.contract(address=uniswap_factory, abi=uniswap_factory_ab

# define function to handle events and print to the console
def handle_event(event):
    print(Web3.toJSON(event))
    # and whatever

# asynchronous defined function to loop

# this loop sets up an event filter and is looking for new entires for the "
# this loop runs on a poll interval
async def log_loop(event_filter, poll_interval):
    while True:
        for PairCreated in event_filter.get_new_entries():
            handle_event(PairCreated)
            await asyncio.sleep(poll_interval)

# when main is called
# create a filter for the latest block and look for the "PairCreated" event
# run an async loop
# try to run the log_loop function above every 2 seconds
def main():
    event_filter = contract.events.PairCreated.createFilter(fromBlock='lates
    #block_filter = web3.eth.filter('latest')
    # tx_filter = web3.eth.filter('pending')
    loop = asyncio.get_event_loop()
    try:
        loop.run_until_complete(
            asyncio.gather(
                log_loop(event_filter, 2)))
        # log_loop(block_filter, 2),
        # log_loop(tx_filter, 2)))
    finally:
        # close loop to free up system resources
        loop.close()

if __name__ == "__main__":
    main()

```

The code above will print new liquidity pair events to your IDE console. Below is a sample of 11 liquidity pairs that were added to Uniswap throughout the day.

```
{
  "args": {
    "token0": "0x348D86b1162e0dF83E1cf66Dd1E90489D292280b",
    "token1": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "pair": "0xb1E71a38cA5E6E3cc11293830FA7d45D287F5482"
  },
  "args": {
    "token0": "0x90b7a437ddaf105686445B928dA82D86dd447EC5",
    "token1": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "pair": "0x3Fa4eEF6Fb12f1c4b8593eca8c28ab2d0D75f763"
  },
  "args": {
    "token0": "0x1802431B72C8eccc2A8af0f2305aecA461e52b07",
    "token1": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "pair": "0x84d42E6C741298Bf03e599Fc8DA9549Aa0D7fbEe"
  },
  "args": {
    "token0": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "token1": "0xcEA5866d85A61872f5491a796E5dF355CE307b6",
    "pair": "0x3d84e5264a6a76e13409Fc200607965ed5f387Cd"
  },
  "args": {
    "token0": "0xdAC17F958D2ee523a2206206994597C13D831ec7",
    "token1": "0xF867ec2951737f0E789691b7debC374C07d4b86D",
    "pair": "0xC8185d561fB4f9F72C6Fdc7aEb0835C66eE84eF"
  },
  "args": {
    "token0": "0x7e1baaeE244f52dd860bc2686d455Ec47553D6e9",
    "token1": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "pair": "0x3e9950b0bCECEaf1A91D0eecdF0622968Ec4855CD"
  },
  "args": {
    "token0": "0xb12e4888968Fbf7A2E942F918e31EEa4f49FC7A8",
    "token1": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "pair": "0x601522F506F4708aC15fc70bcb14501F550aa118"
  },
  "args": {
    "token0": "0x11ea17e6eeA4A5fA1fE1a72e4fB9ADcDEE74d00",
    "token1": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "pair": "0x7C58a0eeF1434EF980a05a77E058dE39E71ea8A5"
  },
  "args": {
    "token0": "0x99f106c1CBE266613FbAF8bF5Ef5C07D0E6d222F",
    "token1": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "pair": "0x8340F1DC6A4E7D2546754e88029ca2b340E5938C"
  },
  "args": {
    "token0": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "token1": "0xd4f1AeE8d7F5AE037984DcCEC7C3Eaa4895c7839",
    "pair": "0x2Ce4463E81172a87883bA7Ccd42D143cAf240327"
  },
  "args": {
    "token0": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
    "token1": "0xCcCB2829e5A9019423086b07d6fd6d866A96C1118",
    "pair": "0x79A2E5e5eCe82b96E7F59d8B68871Fb7aA1472616"
  }
}
```

## Uniswap Liquidity Pair Creation Log

After the bot identifies a newly created liquidity pair send a transaction to the smart contract to execute a token swap.

# Integrate into the snipe trading bot

To complete this bot you will need to integrate the steps below into the Python code above.

1. Looking at the Uniswap Liquidity Pair Creation Log screen print above. You will need to save token0, token1 and the pair address as return values in your code. You will need these values to perform a token swap.
2. Determine which address is the new token address. Most pairs are setup with WETH vs New Token.
3. Read this tutorial on [how to send an ETH transaction to the blockchain using Web3.py in Python](#)
4. Using the new token address (token0, token1) call the function `getAmountOutMin` in the smart contract from the Python code. This will return the `amountOutMin` which is needed as an input for the swap.
5. Use the return value from `getAmountOutMin` and call the function `swap` in the smart contract from the Python code.

After finishing these steps you have a working snipe trading bot that monitors and trades liquidity pairs.

The Solidity smart contract can be modified to support sniping tokens on other Uniswap clone exchanges. As an example the Binance Smart Chain, Polygon, Ubiq, CheapEth, Fantom are [Ethereum](#) blockchain clones. Each blockchain has a copy of the Uniswap decentralized exchange. Consider adding the decentralized exchanges below to your snipe bot.

- Uniswap
- Mooniswap
- 1Inch Exchange
- Sushiswap
- Sashimiswap
- Binance Smart Chain Pancake Swap
- CheapEth CheapSwap
- Ubiq Shinobi
- Polygon SwapMatic

This code is for learning and entertainment purposes only. This code has not been audited and use at your own risk. Remember smart contracts are experimental and could contain bugs.

## Build a trading machine to run a snipe bot

Build a high performance trading machine to run an Ethereum node and your snipe trading bot. Don't use a third party service provider for your electronic trading platform. Purchase computer hardware and build your own machine. The processor, memory and M.2 SSD are extremely important.

Look into the following components:

- [AMD Ryzen 5 3600 6-Core 12-Thread cpu](#)
- [Sabrent 1TB Rocket NVMe 4.0 Gen4 PCIe M.2 Internal SSD](#)
- [ASUS Prime B550M-A motherboard](#)





For the operating system look into ubuntu Linux server. Linux is a must as there is no bloatware, it is very stable, and downtime is a minimum. From a stability standpoint you can not go wrong.

## Resources

### Blockchain Networks

Below is a list of EVM compatible Mainnet and Testnet blockchain networks. Each link contains network configuration, links to multiple faucets for test ETH and tokens, bridge details, and technical resources for each blockchain. Basically everything you need to test and deploy smart contracts or decentralized applications on each chain. For a list of popular Ethereum forums and chat applications [click here](#).



[Ethereum test network configuration and test ETH faucet information](#)



[Optimistic Ethereum Mainnet and Testnet configuration, bridge details, etc.](#)



[Polygon network Mainnet and Testnet configuration, faucets for test MATIC tokens, bridge details, etc.](#)



[Binance Smart Chain Mainnet and Testnet configuration, faucets for test BNB tokens, bridge details, etc.](#)

[Fantom network Mainnet and Testnet configuration, faucets for test FTM tokens, bridge details, etc.](#)



[Kucoin Chain Mainnet and Testnet](#) configuration, faucets for test KCS tokens, bridge details, etc.

## Web3 Software Libraries

You can use the following libraries to interact with an EVM compatible blockchain.

- [Python: Web3.py](#) Python library for interacting with Ethereum.  
[Web3.py examples](#)
- [Js: web3.js](#) Ethereum JavaScript API
- [Java: web3j](#) Web3 Java Ethereum Dapp API
- [PHP: web3.php](#) A php interface for interacting with the Ethereum blockchain and ecosystem.

## Nodes

Learn how to run a Geth node. Read [getting started with Geth to run an Ethereum node](#).

## Fix a transaction

How to [fix a pending transaction stuck](#) on Ethereum or EVM compatible chain

## Next – [Use The Graph to query Ethereum data in Python](#)

**NiceHash Miner**

Most profitable and easy to use mining software!  
Connect your GPU/CPU and start earning Bitcoins.

## PYTHON &amp; WEB3 BASICS

## How to listen for Ethereum events using Web3.py in Python

In Python Web3 application development systems use logs to capture what's going on at a specific moment in time. Applications...

## RESOURCES

## How to add an image / logo to your crypto token

If you are the owner of a token on the Ethereum network or the Binance Smart Chain (BSC) you can...

### 2 thoughts on "Build a snipe bot to monitor and trade liquidity pairs"

Pingback: [Crypto Market Pool - Use The Graph in Python to query Ethereum data](#)

Pingback: [Crypto Market Pool - Can you make money creating an arbitrage bot running on the Ethereum blockchain?](#)

### Leave a Reply

You must be [logged in](#) to post a comment.

## RECENT POSTS

---

[Vaults and the ERC-4626 token contract](#)

[What is an ERC-1404 token contract](#)

[Dynamic NFTs on Ethereum](#)

[ERC721 contract that supports sales royalties](#)

[Mint a Scary Ghost NFT on Polygon](#)

[Create a crypto faucet using a Solidity smart contract](#)

## ADVERTISEMENT

---

### CRYPTO CURRENCY REDDIT

---

[After months of research im beginning to think crypto is our only shot at surviving the upcoming financial collapse](#)

[Defi scammers be like](#)

[The Porn Star Banging Men Into Not Buying Crypto and NFTs](#)

[3 African countries Are Considering To Adopt Cryptocurrency](#)

[Kraken Rant](#)

## SUBSCRIBE

---

Get notified when new articles are added to Crypto Market Pool

**Email \***

**Submit**

test

---

Powered by the Crypto Market Pool team | All Rights Reserved | For Educational & Entertainment Purposes Only