**Building a Crypto Trading Bot in Python for Beginners (Part 1: stop-loss and take-profit with Uniswap)**

[[crjameson](https://medium.com/@crjameson?source=post_page-----67cc2c7b6344--------------------------------)](https://medium.com/@crjameson?source=post_page-----67cc2c7b6344--------------------------------)

[crjameson](https://medium.com/@crjameson?source=post_page-----67cc2c7b6344--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fc2815d464763&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40crjameson%2Fbuilding-a-crypto-trading-bot-in-python-for-beginners-part-1-stop-loss-and-take-profit-with-67cc2c7b6344&user=crjameson&userId=c2815d464763&source=post_page-c2815d464763----67cc2c7b6344---------------------post_header-----------)

14 min read

·

Oct 26, 2023

9

1



Welcome back. In this tutorial we will combine our knowledge from the previous articles and build something really useful:

**A trading bot which can automatically sell our position when certain thresholds are met to automatically limit our losses or take profits.**

If you havent read the previous articles, make sure to briefly review them to make sure you have everything setup.

If you want to make money with crypto trading, one of the most important things is to manage your risk. And a simple way to do so is the risk reward ratio.

***Hint****: In the second part of this tutorial I will write some automated unit tests which explain a few more things and show how to use this bot. So in case there are any open questions, make sure to check that out.*

**A simple risk management strategy for spot trading based on the risk/reward ratio**

Lets say you buy a token X for 1 ETH (I will write about different entry / sniper strategies later on my [Substack](https://substack.com/@crjameson)). Now two things can happen:

1. The price rises — well done
2. The price declines — shit happens

In the first case you have to realize your gains somehow and in the second you have to limit your losses.

A very simple strategy to do so is the so called risk/reward ratio.

**The risk/reward ratio measures the prospective reward a trader can make on every dollar he risks on a trade.**

So for example you buy any amount of token X for 1 ETH.

You sell the X token when the value of the position falls below 0.9 ETH with a small loss of 0.1 ETH. Your risk is 0.1 ETH:

And you sell the X tokens when their value raises above 1.5 ETH with a gain of 0.5 ETH. Your reward is 0.5ETH.

So your risk/reward ratio would be 1 to 5. For every 1 ETH you risk on that trade you can earn 5 ETH.

Which means, you don’t have to be right every trade to be profitable. This also means, that you never can lose all your money on a single trade, because you automatically sell the token, when you lost a certain amount. This is called a stop-loss order in trading and it is one of the most important building bricks of a profitable trading strategy.

In traditional finance it is recommended that you set your stop-loss to 0.5% — 1% of your trading balance. So lets say you have 10 ETH in your account, a good stop-loss could be 0.1 ETH. In this tutorial we are going to do spot trading on Uniswap without leverage, so you can increase the percentage until you still feel comfortable. But don’t gamble and go above 10%. A good night sleep is more important than money …

Our goal now is to automate this trading strategy in Python on Uniswap:

We want to automatically sell a token, when it’s position value raised above our take-profit limit and we want to sell it, when the value declined below our stop-loss limit.

**There is no fixed price on a decentralized Exchange like Uniswap**

Before we start with the code i just want to highlight a very important concept for DEX trading:

**We can not use the current price of a token for out stop-loss or take-profit orders. Actually a “price” like on a centralized exchange or in traditional finance doesn’t exist.**

This is valid for any automated market maker (AMM) following the y\*x=k formula. In this formula k is the constant price and y and x are the reserve amounts of the tokens.

When we alter the amounts of the reserves by swapping tokens, we would get a different price for each token we sell. If you’re trading bigger amounts you might see this in the Uniswap front end as “price impact”.

So if we want to sell like 1 Million token X, the price for each token we sell would slightly decrease. So we cant just take the current price and use that for our stop and take profit but instead we either have to either reverse the AMM formula to calculate the real price or we just call the get\_amounts\_out function we have already used in the previous tutorial to let Uniswap calculate how much token we would get out of a swap.

Calling get\_get\_amounts out is the easier approach, so we will do that.

**The basic bot structure**

Most simple trading bots follow the same structure. You have some kind of endless loop running which executes the strategy and sleeps.

So our bot stores an initial state like the current value of our tokens during startup and then starts the bot loop.

while True:  
 position\_value = get\_position\_value()  
 if position\_value > take\_profit:  
 sell\_token()  
 if position\_value < stop\_loss:  
 sell\_token()  
 time.sleep(60)

Every 60 seconds we execute our strategy and check the value of our position. If the predefined thresholds are met, we sell the token.

This approach, where you check for something in a regular interval is also called “polling” and is a popular and very simple solution which we will use in this example bot as well.

***Hint****: A more advanced approach I use for my bots handling thousands of requests per second is event based. I know that the value of my position only changes, when a swap event happened. So i don’t use polling here but the so called publisher/subscriber model. This means i create a web socket connection to a blockchain node (*[*you can do this with Quicknode for example*](https://www.quicknode.com/?tap_a=67226-09396e&tap_s=4216385-4b6531&utm_source=affiliate&utm_campaign=generic&utm_content=affiliate_landing_page&utm_medium=generic)*) and scan every incoming new block for all relevant swap events and only check the position value after a new swap event happened. Even more advanced would be to frontrun certain transactions and calculating the reserves locally by not only watching confirmed blocks but yea … not today. This would involve to many new concepts and for most spot trading strategies the simple polling approach is just fine.*

**The code for the Python trading bot**

I will walk you through the code now, and you can find the complete code in the [Python and DeFi repository on Github](https://github.com/crjameson/python-defi-tutorials/tree/main/bots/autosellbot)as well.

A few notes before we start:

* The code is working and can be used in production with a bit tweaking but it is still not perfect — so double check everything
* I will make this example as simple as possible (less than 200 lines of code) so yea, there is room for improvement
* I will limit the bot examples to a single working Python file— for production code I would usually split things up and write more reusable code components

**The imports:**

import os  
from web3 import Account, Web3  
import json  
import dotenv  
import time  
import argparse  
  
dotenv.load\_dotenv()

The only new things here are argparse, which Iwill explain later and is part of the Python standard lib, and dotenv ( [https://pypi.org/project/python-dotenv )/](https://pypi.org/project/python-dotenv/). Python dotenv allows us to work with environment variables. This makes handling certain things like private keys or API keys a bit more secure.

**The constants:**

MIN\_ERC20\_ABI = json.loads(  
 """  
 [{"constant": true, "inputs": [], "name": "name", "outputs": [ { "name": "", "type": "string" } ], "payable": false, "stateMutability": "view", "type": "function" }, { "constant": false, "inputs": [ { "name": "\_spender", "type": "address" }, { "name": "\_value", "type": "uint256" } ], "name": "approve", "outputs": [ { "name": "", "type": "bool" } ], "payable": false, "stateMutability": "nonpayable", "type": "function" }, { "constant": true, "inputs": [], "name": "totalSupply", "outputs": [ { "name": "", "type": "uint256" } ], "payable": false, "stateMutability": "view", "type": "function" }, { "constant": false, "inputs": [ { "name": "\_from", "type": "address" }, { "name": "\_to", "type": "address" }, { "name": "\_value", "type": "uint256" } ], "name": "transferFrom", "outputs": [ { "name": "", "type": "bool" } ], "payable": false, "stateMutability": "nonpayable", "type": "function" }, { "constant": true, "inputs": [], "name": "decimals", "outputs": [ { "name": "", "type": "uint8" } ], "payable": false, "stateMutability": "view", "type": "function" }, { "constant": true, "inputs": [ { "name": "\_owner", "type": "address" } ], "name": "balanceOf", "outputs": [ { "name": "balance", "type": "uint256" } ], "payable": false, "stateMutability": "view", "type": "function" }, { "constant": true, "inputs": [], "name": "symbol", "outputs": [ { "name": "", "type": "string" } ], "payable": false, "stateMutability": "view", "type": "function" }, { "constant": false, "inputs": [ { "name": "\_to", "type": "address" }, { "name": "\_value", "type": "uint256" } ], "name": "transfer", "outputs": [ { "name": "", "type": "bool" } ], "payable": false, "stateMutability": "nonpayable", "type": "function" }, { "constant": true, "inputs": [ { "name": "\_owner", "type": "address" }, { "name": "\_spender", "type": "address" } ], "name": "allowance", "outputs": [ { "name": "", "type": "uint256" } ], "payable": false, "stateMutability": "view", "type": "function" }, { "payable": true, "stateMutability": "payable", "type": "fallback" }, { "anonymous": false, "inputs": [ { "indexed": true, "name": "owner", "type": "address" }, { "indexed": true, "name": "spender", "type": "address" }, { "indexed": false, "name": "value", "type": "uint256" } ], "name": "Approval", "type": "event" }, { "anonymous": false, "inputs": [ { "indexed": true, "name": "from", "type": "address" }, { "indexed": true, "name": "to", "type": "address" }, { "indexed": false, "name": "value", "type": "uint256" } ], "name": "Transfer", "type": "event"}]  
 """  
)  
  
UNISWAPV2\_ROUTER\_ABI = json.loads(  
 """  
 [{"inputs":[{"internalType":"address","name":"\_factory","type":"address"},{"internalType":"address","name":"\_WETH","type":"address"}],"stateMutability":"nonpayable","type":"constructor"},{"inputs":[],"name":"WETH","outputs":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"tokenA","type":"address"},{"internalType":"address","name":"tokenB","type":"address"},{"internalType":"uint256","name":"amountADesired","type":"uint256"},{"internalType":"uint256","name":"amountBDesired","type":"uint256"},{"internalType":"uint256","name":"amountAMin","type":"uint256"},{"internalType":"uint256","name":"amountBMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"addLiquidity","outputs":[{"internalType":"uint256","name":"amountA","type":"uint256"},{"internalType":"uint256","name":"amountB","type":"uint256"},{"internalType":"uint256","name":"liquidity","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"token","type":"address"},{"internalType":"uint256","name":"amountTokenDesired","type":"uint256"},{"internalType":"uint256","name":"amountTokenMin","type":"uint256"},{"internalType":"uint256","name":"amountETHMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"addLiquidityETH","outputs":[{"internalType":"uint256","name":"amountToken","type":"uint256"},{"internalType":"uint256","name":"amountETH","type":"uint256"},{"internalType":"uint256","name":"liquidity","type":"uint256"}],"stateMutability":"payable","type":"function"},{"inputs":[],"name":"factory","outputs":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"internalType":"uint256","name":"reserveIn","type":"uint256"},{"internalType":"uint256","name":"reserveOut","type":"uint256"}],"name":"getAmountIn","outputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"}],"stateMutability":"pure","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"},{"internalType":"uint256","name":"reserveIn","type":"uint256"},{"internalType":"uint256","name":"reserveOut","type":"uint256"}],"name":"getAmountOut","outputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"}],"stateMutability":"pure","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"}],"name":"getAmountsIn","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"}],"name":"getAmountsOut","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountA","type":"uint256"},{"internalType":"uint256","name":"reserveA","type":"uint256"},{"internalType":"uint256","name":"reserveB","type":"uint256"}],"name":"quote","outputs":[{"internalType":"uint256","name":"amountB","type":"uint256"}],"stateMutability":"pure","type":"function"},{"inputs":[{"internalType":"address","name":"tokenA","type":"address"},{"internalType":"address","name":"tokenB","type":"address"},{"internalType":"uint256","name":"liquidity","type":"uint256"},{"internalType":"uint256","name":"amountAMin","type":"uint256"},{"internalType":"uint256","name":"amountBMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"removeLiquidity","outputs":[{"internalType":"uint256","name":"amountA","type":"uint256"},{"internalType":"uint256","name":"amountB","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"token","type":"address"},{"internalType":"uint256","name":"liquidity","type":"uint256"},{"internalType":"uint256","name":"amountTokenMin","type":"uint256"},{"internalType":"uint256","name":"amountETHMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"removeLiquidityETH","outputs":[{"internalType":"uint256","name":"amountToken","type":"uint256"},{"internalType":"uint256","name":"amountETH","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"token","type":"address"},{"internalType":"uint256","name":"liquidity","type":"uint256"},{"internalType":"uint256","name":"amountTokenMin","type":"uint256"},{"internalType":"uint256","name":"amountETHMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"removeLiquidityETHSupportingFeeOnTransferTokens","outputs":[{"internalType":"uint256","name":"amountETH","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"token","type":"address"},{"internalType":"uint256","name":"liquidity","type":"uint256"},{"internalType":"uint256","name":"amountTokenMin","type":"uint256"},{"internalType":"uint256","name":"amountETHMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"},{"internalType":"bool","name":"approveMax","type":"bool"},{"internalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"removeLiquidityETHWithPermit","outputs":[{"internalType":"uint256","name":"amountToken","type":"uint256"},{"internalType":"uint256","name":"amountETH","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"token","type":"address"},{"internalType":"uint256","name":"liquidity","type":"uint256"},{"internalType":"uint256","name":"amountTokenMin","type":"uint256"},{"internalType":"uint256","name":"amountETHMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"},{"internalType":"bool","name":"approveMax","type":"bool"},{"internalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"removeLiquidityETHWithPermitSupportingFeeOnTransferTokens","outputs":[{"internalType":"uint256","name":"amountETH","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"tokenA","type":"address"},{"internalType":"address","name":"tokenB","type":"address"},{"internalType":"uint256","name":"liquidity","type":"uint256"},{"internalType":"uint256","name":"amountAMin","type":"uint256"},{"internalType":"uint256","name":"amountBMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"},{"internalType":"bool","name":"approveMax","type":"bool"},{"internalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"removeLiquidityWithPermit","outputs":[{"internalType":"uint256","name":"amountA","type":"uint256"},{"internalType":"uint256","name":"amountB","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapETHForExactTokens","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"payable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOutMin","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapExactETHForTokens","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"payable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOutMin","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapExactETHForTokensSupportingFeeOnTransferTokens","outputs":[],"stateMutability":"payable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"},{"internalType":"uint256","name":"amountOutMin","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapExactTokensForETH","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"},{"internalType":"uint256","name":"amountOutMin","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapExactTokensForETHSupportingFeeOnTransferTokens","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"},{"internalType":"uint256","name":"amountOutMin","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapExactTokensForTokens","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"},{"internalType":"uint256","name":"amountOutMin","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapExactTokensForTokensSupportingFeeOnTransferTokens","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"internalType":"uint256","name":"amountInMax","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapTokensForExactETH","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"internalType":"uint256","name":"amountInMax","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapTokensForExactTokens","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"nonpayable","type":"function"},{"stateMutability":"payable","type":"receive"}]  
 """  
)  
  
UNISWAP\_V2\_SWAP\_ROUTER\_ADDRESS = "0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D"  
WETH\_TOKEN\_ADDRESS = "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2"

The same as in the last tutorials, just this time, we define the ABIs in the same file. You can get the JSON ABI in Etherscan (or any other blockexplorer), by going to a contract and check the code section there ( <https://arbiscan.io/address/0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D#code> ).

The ABI tells our Python script how to talk to a certain smart contract by defining which functions are available, which parameters they expect and what values they return.

**The bot constructor**

The bot needs a state, to store some variables like the initial position value. Whenever we have functions combined with a state, we use a Python class.

So all of our bot logic will be in the **AutoSellBot** class.

class AutoSellBot:  
 def \_\_init\_\_(  
 self,  
 name: str,  
 token\_address: str,  
 stop\_loss\_percent: int = 10,  
 take\_profit\_percent: int = 50,  
 check\_interval: int = 60,  
 slippage\_percent: int = 3,  
 private\_key: str = "",  
 rpc: str = "",  
 ) -> None:  
 self.name = name  
 self.token\_address = token\_address  
 self.stop\_loss\_percent = stop\_loss\_percent  
 self.take\_profit\_percent = take\_profit\_percent  
 self.check\_interval = check\_interval  
 self.slippage\_percent = slippage\_percent  
 self.private\_key = private\_key if private\_key else os.getenv("PRIVATE\_KEY")  
 if not self.private\_key:  
 raise ValueError("error: no private key provided")  
  
 self.sell\_path = [self.token\_address, WETH\_TOKEN\_ADDRESS]  
  
 self.rpc = rpc if rpc else os.getenv("RPC\_ENPOINT")  
 if not self.rpc:  
 raise ValueError("error: no rpc endpoint configured")  
  
 self.account = Account.from\_key(self.private\_key)  
  
 self.web3 = Web3(Web3.HTTPProvider(os.getenv("RPC\_ENPOINT")))  
 self.router\_contract = self.web3.eth.contract(  
 address=UNISWAP\_V2\_SWAP\_ROUTER\_ADDRESS, abi=UNISWAPV2\_ROUTER\_ABI  
 )  
 self.token\_contract = self.web3.eth.contract(  
 address=self.token\_address, abi=MIN\_ERC20\_ABI  
 )  
  
 self.token\_balance = self.get\_balance()  
 if self.token\_balance == 0:  
 raise ValueError("error: token\_balance is 0")  
  
 self.initial\_value = self.get\_position\_value()  
 self.stop\_loss\_value = self.initial\_value \* (1 - self.stop\_loss\_percent / 100)  
 self.take\_profit\_value = self.initial\_value \* (  
 1 + self.take\_profit\_percent / 100  
 )  
  
 approved = self.approve\_token()  
 assert approved, f"{self.name}: error: could not aprove token"  
 print(f"{self.name}: bot started")

The constructor initializes a few variables like the stop\_loss\_percentage or the take\_profit\_percentage for our risk/reward ratio. To do so we calculate the initial value of our position in Ether\* and add/subtract the percentage value of our take-profit/stop-loss.

Later in our bot loop we just compare the current value of the position to this initial values and react.

*\*****Hint****: I use Ether/WETH in this example with 18 decimals but you could use any token pair. You just have to switch the addresses and the decimals.*

We also load the bots account to trade with. Here you see how i use os.getenv(“PRIVATE\_KEY”). This is possible because i used the dotenv package.

To make this work, you need to create a .env file in the same directory as the bot script, with the following content:

PRIVATE\_KEY = "0x5d9d3c897ad4f2b8b51906185607f79672d7fec086a6fb6afc2de423c017330c"  
RPC\_ENPOINT = "http://127.0.0.1:8545"

***Hint****: If you are using Windows, try the WSL (*[*Windows Subsystem for Linux*](https://learn.microsoft.com/en-us/windows/wsl/install)*, that works quite well with all of this)*

The advantage of this approach is, that if you’re working with Github like me, you can commit your code without risking leaking sensitive contents like private keys or API keys stored in your .env file. And yea … i know i just did that … but it’s a local Ganache account as shown in the first tutorial ;)

The rest should be easy to understand. I am checking that the private key and the RPC endpoint is provided and that the bot account has a balance of the token we want to monitor. For the sake of simplicity, we don’t buy with our bot and don’t allow manual trading.

**So our bot just handles the case where we have already bought a token and want to manage this token position by automatically selling when our thresholds are met.**

I also don’t catch all possible exceptions here, but that’s OK, as we are monitoring the bots startup process and its always good to fail fast, if something is off. As soon as we start the bots endless loop later, this is different and needs to be much more reliable.

One more thing to note here, we are already approving the Uniswap router to sell our token in the constructor. We are using an unlimited amount here, but might use the balance as well. This is again, to fail as fast as possible in case there is something wrong with the key or the RPC provided but also, to don’t have to do this, when the market is crashing or the fees might be much higher. So typically the situations where our bot needs to act.

***Hint****: If you need a reliable RPC Provider suitable for crypto trading, check out*[*Quicknode*](https://www.quicknode.com/?tap_a=67226-09396e&tap_s=4216385-4b6531&utm_source=affiliate&utm_campaign=generic&utm_content=affiliate_landing_page&utm_medium=generic)*.*

**Some helper functions for our trading bot**

def get\_balance(self):  
 return self.token\_contract.functions.balanceOf(self.account.address).call()  
  
 def get\_position\_value(self):  
 amounts\_out = self.router\_contract.functions.getAmountsOut(  
 self.token\_balance, self.sell\_path  
 ).call()  
 # amounts\_out[0] is the token amount in - amounts out [1] is the ETH amount out  
 return amounts\_out[1]  
  
 def approve\_token(self):  
 approve\_tx = self.token\_contract.functions.approve(  
 UNISWAP\_V2\_SWAP\_ROUTER\_ADDRESS, 2\*\*256 - 1  
 ).build\_transaction(  
 {  
 "gas": 500\_000,  
 "maxPriorityFeePerGas": self.web3.eth.max\_priority\_fee,  
 "maxFeePerGas": 100 \* 10\*\*10,  
 "nonce": self.web3.eth.get\_transaction\_count(self.account.address),  
 }  
 )  
  
 signed\_approve\_tx = self.web3.eth.account.sign\_transaction(  
 approve\_tx, self.account.key  
 )  
  
 tx\_hash = self.web3.eth.send\_raw\_transaction(signed\_approve\_tx.rawTransaction)  
 tx\_receipt = self.web3.eth.wait\_for\_transaction\_receipt(tx\_hash)  
  
 if tx\_receipt and tx\_receipt["status"] == 1:  
 print(  
 f"{self.name}: approve successful: approved {UNISWAP\_V2\_SWAP\_ROUTER\_ADDRESS} to spend unlimited token"  
 )  
 return True  
 else:  
 raise Exception(f"{self.name} error: could not approve token")  
  
 def sell\_token(self, min\_amount\_out=0):  
 sell\_tx\_params = {  
 "nonce": self.web3.eth.get\_transaction\_count(self.account.address),  
 "from": self.account.address,  
 "gas": 500\_000,  
 "maxPriorityFeePerGas": self.web3.eth.max\_priority\_fee,  
 "maxFeePerGas": 100 \* 10\*\*10,  
 }  
 sell\_tx = self.router\_contract.functions.swapExactTokensForETH(  
 self.token\_balance, # amount to sell  
 min\_amount\_out, # min amount out  
 self.sell\_path,  
 self.account.address,  
 int(time.time()) + 180, # deadline now + 180 sec  
 ).build\_transaction(sell\_tx\_params)  
  
 signed\_sell\_tx = self.web3.eth.account.sign\_transaction(  
 sell\_tx, self.account.key  
 )  
  
 tx\_hash = self.web3.eth.send\_raw\_transaction(signed\_sell\_tx.rawTransaction)  
 tx\_receipt = self.web3.eth.wait\_for\_transaction\_receipt(tx\_hash)  
 if tx\_receipt and tx\_receipt["status"] == 1:  
 # now make sure we sold them  
 self.token\_balance = self.get\_balance()  
 if self.token\_balance == 0:  
 print(f"{self.name}: all token sold: tx hash: {Web3.to\_hex(tx\_hash)}")  
 return True  
 else:  
 print(f"{self.name}: error selling token")  
 return False

This functions help us to do the basics like checking the balance, get the value of our position, approving a token or sell it. They are all explained in more detail in the previous two tutorials.

**The bot strategy**

def execute(self):  
 position\_value = self.get\_position\_value()  
 print(f"{self.name}: position value {position\_value/10\*\*18:.6f}")  
 # we check if we hit our stop loss or take profit level  
 if (position\_value <= self.stop\_loss\_value) or (  
 position\_value >= self.take\_profit\_value  
 ):  
 # we calculate the min amount out here based on the current value and the configured slippage  
 min\_amount\_out = int(position\_value \* (1 - self.slippage\_percent / 100))  
 print(f"{self.name}: position value hit limit - selling all token")  
 self.sell\_token(min\_amount\_out)

Our strategy is quite simple. To use the bot you just buy any token and provide the bot with a token address together with the stop-loss and take-profit limits.

Whenever the value of our token position in Ether is less than our stop-loss value or more than our take-profit value we calculated in the constructor, we sell all our tokens.

You could improve various things here like having multiple take-profits / stop-losses and sell only a fraction of our tokens.

If one of our thresholds is met, we calculate the min\_amount\_out we expect to get for our tokens using the slippage parameter from the bot and sell the token.

**Important**:

**Having a stop-loss of 10% doesn’t mean, we can not lose more**. It means that the stop-loss will be triggered when the position value during the check hit our targets and our bot tries to sell the token.

Let’s say we have a check interval of every 60 seconds, an initial position value of 1 ETH and a stop-loss at 10%, so 0.9 ETH.

Now the first check returns 1 ETH and the second returns 0.8 ETH position value, because there where some bad news during our 60 seconds check interval. Our stop-loss would be triggered only now during the second check and we lost 20%!

To make up for this, you should set a short check interval for your polling depending on the available liquidity. Token with very little liquidity (<100k) should be checked much more often than a token with a deep liquidity pool (500k+). This is technically no problem and even most free [RPC API provider](https://www.quicknode.com/?tap_a=67226-09396e&tap_s=4216385-4b6531&utm_source=affiliate&utm_campaign=generic&utm_content=affiliate_landing_page&utm_medium=generic) allow you to check positions as often as every second.

***Hint****: I would recommend a paid plan with one of the fastest RPC-Providers available if you are serious about crypto trading —*[*i am using Quicknode for all of this*](https://www.quicknode.com/?tap_a=67226-09396e&tap_s=4216385-4b6531&utm_source=affiliate&utm_campaign=generic&utm_content=affiliate_landing_page&utm_medium=generic)*.*

Another thing to consider, but usually not a big thing with automated bots, is the so called slippage which i explained earlier in more detail. That’s why we allow to set that value in the bot parameters.

The min\_amount\_out value is the real value we get. If the value of the token position falls very drastically we might get a slightly different amount.

How to set this parameters is up to you. I prefer to have my trades executed when the limits are hit.

The alternative would be to set the min\_amount\_out to the exact value of your stop-loss, but this might lead to a situation where your tokens are not sold at all.

**The bot loop**

And finally the last piece of our bot: the endless loop i already mentioned in the bot structure section:

def bot\_loop(self):  
 # the bot is supposed to run until all tokens are sold  
 while self.token\_balance:  
 # we catch any runtime / network errors here - just to make sure it runs  
 try:  
 self.execute()  
 time.sleep(self.check\_interval)  
 except Exception as e:  
 print(f"exception: {e}")  
 print(f"{self.name}: stopping bot - all token sold")

This is quite simple. As long as we have tokens in our bot account (while self.token\_balance) execute our bot strategy function and sleep for the configured interval.

We separate the strategy execution from the bot loop, to make the strategy easier to test. I will go into more detail about automated testing in the next tutorial.

We also surround the bot strategy with try/except to catch any issues which might arise during bot execution. Most likely this could be network errors like connection timeout at this point. The proper way would be to do this more detailed in the execution function, but for this tutorial it is just important, that we keep the bot running in case of any exceptions.

And that’s it. Now we just need to instantiate the bot in our main function and run it.

**The main function**

if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser(  
 description="Simple stop loss and take profit trading bot (crjameson.xyz)."  
 )  
  
 # mandatory arguments  
 # example token on mainnet uniswap v2: ELON\_TOKEN\_ADDRESS = "0x761D38e5ddf6ccf6Cf7c55759d5210750B5D60F3"  
  
 parser.add\_argument(  
 "token\_address", type=str, default="", help="token address you want to monitor"  
 )  
 # optional arguments - we set the default values here  
 parser.add\_argument("--name", type=str, default="simple bot", help="bot name")  
 parser.add\_argument("--sl", type=int, default=5, help="stop-loss percentage")  
 parser.add\_argument("--tp", type=int, default=20, help="take profit percentage")  
 parser.add\_argument(  
 "--interval", type=int, default=60, help="check interval in seconds"  
 )  
 parser.add\_argument("--slippage", type=int, default=3, help="slippage percentage")  
 parser.add\_argument("--key", type=str, default="", help="your private key")  
 parser.add\_argument(  
 "--rpc", type=str, default="http://127.0.0.1:8545", help="your rpc endpoint"  
 )  
  
 args = parser.parse\_args()  
  
 bot = AutoSellBot(  
 args.name,  
 args.token\_address,  
 stop\_loss\_percent=args.sl,  
 take\_profit\_percent=args.tp,  
 check\_interval=args.interval,  
 slippage\_percent=args.slippage,  
 private\_key=args.key,  
 )  
 bot.bot\_loop()

This should be easy to understand. We are parsing the different command line arguments like the mandatory token address and the optional bot configuration parameters.

We are using the argparse module here from the standard lib.

Saving all of this into a file named autosell\_bot.py we could now run our bot like this:

python autosell\_bot.py 0x<TOKEN\_ADDRESS> --name "autobot" --sl 10 --tp 50

**The End**

And that was it. If you followed along, you have your own useful little trading bot in python running capable of managing your position on Uniswap v2.

A homework might be to add more stop-loss/take-profit levels, allow different trading pairs than WETH, or Uniswap v3. There are plenty of ways to optimize it but in the next tutorial in this series we will do something much more important:

**We will look into how to write automated tests for this bot and really make sure it works, before we put any real money into it.**

The code for the bot is as always available in my GitHub here:

<https://github.com/crjameson/python-defi-tutorials/tree/main/bots/autosellbot>

And it would be nice if you follow me here on [SubStack](https://substack.com/@crjameson) where i write about more advanced and profitable stuff around Python and DeFi.

[Cryptotradingbot](https://medium.com/tag/cryptotradingbot?source=post_page-----67cc2c7b6344---------------cryptotradingbot-----------------)

[Crypto Trading](https://medium.com/tag/crypto-trading?source=post_page-----67cc2c7b6344---------------crypto_trading-----------------)

[Uniswap](https://medium.com/tag/uniswap?source=post_page-----67cc2c7b6344---------------uniswap-----------------)

[Trading Strategy](https://medium.com/tag/trading-strategy?source=post_page-----67cc2c7b6344---------------trading_strategy-----------------)

[Trading Bot](https://medium.com/tag/trading-bot?source=post_page-----67cc2c7b6344---------------trading_bot-----------------)