// MODULE · CODING FUNDAMENTALS

Coding, terminal, Git.

The floor every trading automator stands on.

04	50	10	08
LANGUAGES	COMMANDS	GIT STEPS	FAILURES

NEXURAL · SAGE IDEAS LLC
31 pages · Educational only · v1.0
nexural.com/learn/automation

// contents

What's inside

Six sections. Thirty pages. Production-grade fundamentals.

// section 1 · languages

Four languages, one playbook

Each of these languages dominates a specific runtime. Learn one well; the others rhyme.

Python

Universal glue. Backtests, ML, API clients, ETL. Runs anywhere — Alpaca, IBKR, QuantConnect, custom desks.

Pine Script v5

TradingView's strategy/indicator DSL. Smallest learning curve. Lives inside the chart.

C# / NinjaScript

NinjaTrader 8 strategies & indicators. Compiles to DLLs. Tight integration with order management.

MQL5

MetaTrader 5 expert advisors. Forex/CFDs. C-like syntax, broker-native execution.

How to pick

If you trade on TradingView → Pine. If you trade on NinjaTrader → C#. If you trade forex on MT5 → MQL5. For everything else (research, ML, multi-broker, custom infra) → Python. Most automators end up writing Python and one of the three platform languages.

// section 1 · syntax map

Same logic, four languages

Buy when RSI < 30, exit when RSI > 70. Spot the patterns.

Concept	Python	Pine v5	C# / NS	MQL5
Variable	`x = 14`	`x = 14`	`int x = 14;`	`int x = 14;`
If	`if rsi < 30:`	`if rsi < 30`	`if (rsi < 30)`	`if(rsi < 30)`
Function	`def buy(): ...`	`buy() => strategy.entry("L", strategy.long)`	`void Buy() { EnterLong(); }`	`void Buy() { trade.Buy(0.1); }`
Loop	`for i in range(10):`	`for i = 0 to 9`	`for (int i=0;i<10;i++)`	`for(int i=0;i<10;i++)`
Type	`dynamic`	`implicit`	`static / strict`	`static / strict`
Entry call	`broker.buy('AAPL', 100)`	`strategy.entry("L", strategy.long)`	`EnterLong(100, "L");`	`trade.Buy(0.1, _Symbol);`
Indicator	`ta.rsi(close, 14)`	`ta.rsi(close, 14)`	`RSI(Close, 14)[0]`	`iRSI(_Symbol,_Period,14,PRICE_CLOSE,0)`
Plot / print	`print(rsi)`	`plot(rsi)`	`Print(rsi);`	`Print(rsi);`

// section 1 · language 1/4

Python — the universal glue

Used by every modern broker SDK. Run a backtest, fit a model, ping an API.

Strategy: Buy when RSI < 30, exit when RSI > 70

```
import pandas as pd
import talib
from broker import client  # alpaca / ibkr / ccxt

df = pd.read_csv('aapl.csv')
df['rsi'] = talib.RSI(df['close'], timeperiod=14)

position = 0
for i, row in df.iterrows():
    if row['rsi'] < 30 and position == 0:
        client.buy('AAPL', qty=100)
        position = 100
    elif row['rsi'] > 70 and position > 0:
        client.sell('AAPL', qty=position)
        position = 0
```

What to learn first

- pandas for time-series. Indexing, resampling, rolling windows.
- numpy for math. Vectorize loops — 100×+ faster.
- requests / httpx for REST APIs. Add retries with `tenacity`.
- asyncio for streaming. Websockets, parallel order fills.
- pytest for tests. Run on every commit (you'll thank yourself).

// section 1 · language 2/4

Pine Script v5 — chart-native

Runs inside TradingView. No infra, no servers. Compiled to bytecode by their engine.

Same strategy, in Pine v5

```
//@version=5
strategy("RSI Reversion", overlay=false)

length = input.int(14, "RSI length")
oversold = input.int(30)
overbought = input.int(70)

rsi = ta.rsi(close, length)

if rsi < oversold and strategy.position_size == 0
    strategy.entry("Long", strategy.long)

if rsi > overbought and strategy.position_size > 0
    strategy.close("Long")

plot(rsi, "RSI", color=color.aqua)
hline(oversold, "OS", color=color.lime)
hline(overbought, "OB", color=color.fuchsia)
```

Gotchas

- Indentation matters — Pine is whitespace-sensitive like Python.
- `strategy.entry` is idempotent per ID — won't double-fire.
- Repaints: if you reference `close` on the current bar, results change as the bar forms. Use `close[1]` for confirmed values.
- Alerts ≠ live trading. To execute, route through a webhook bridge.

// section 1 · language 3/4

C# / NinjaScript — NinjaTrader 8

Strongly typed. Compiles to a DLL. Runs in the NT8 process with broker-level access.

Strategy skeleton

```
public class RsiReversion : Strategy {
  private RSI rsi;

  protected override void OnStateChange() {
    if (State == State.Configure) {
      rsi = RSI(14, 3);
    }
  }

  protected override void OnBarUpdate() {
    if (CurrentBar < 14) return;

    if (rsi[0] < 30 && Position.MarketPosition == MarketPosition.Flat)
      EnterLong(100, "L");
    else if (rsi[0] > 70 && Position.MarketPosition == MarketPosition.Long)
      ExitLong("L");
  }
}
```

Lifecycle

State.SetDefaults → name and properties. State.Configure → wire up indicators and data series. State.DataLoaded → safe to read historical bars. OnBarUpdate → fires per bar (or tick if `Calculate = OnEachTick`).

Recompile via F5 in the NinjaScript Editor. Errors block all strategies until fixed.

// section 1 · language 4/4

MQL5 — MetaTrader 5 expert advisors

C-like syntax. Strict types. Broker-native execution, mostly FX and CFDs.

Expert advisor skeleton

```
#include <Trade\Trade.mqh>
CTrade trade;

input int RsiPeriod = 14;
input double Lots = 0.10;

int rsi_handle;

int OnInit() {
  rsi_handle = iRSI(_Symbol, _Period, RsiPeriod, PRICE_CLOSE);
  return INIT_SUCCEEDED;
}

void OnTick() {
  double rsi[];
  CopyBuffer(rsi_handle, 0, 0, 1, rsi);

  bool flat = PositionsTotal() == 0;
  if (rsi[0] < 30 && flat)        trade.Buy(Lots);
  else if (rsi[0] > 70 && !flat)   trade.PositionClose(_Symbol);
}
```

Build & deploy

Compile inside MetaEditor (F7). Output is a `.ex5` file in `MQL5/Experts`. Drag onto a chart, enable AutoTrading, confirm the smiley face icon.

Test in Strategy Tester first. Always with realistic spreads and slippage.

// section 2 · terminal

Navigate & files

If you can't move around your filesystem fluently, you can't ship.

`pwd`	Print working directory.
`ls -la`	List all files (incl. hidden), long format.
`cd ~/strategies`	Change directory; ~ is your home.
`cd -`	Jump back to previous directory.
`mkdir -p a/b/c`	Create nested directories without error if they exist.
`cp -r src/ dst/`	Recursive copy.
`mv old.py new.py`	Rename or move.
`rm -i file`	Interactive delete (prompts before each).
`rm -rf node_modules`	Recursive force delete. Triple-check the path.
`touch run.py`	Create empty file (or update mtime).
`cat config.json`	Print whole file to stdout.
`less log.txt`	Pageable viewer; q to quit, / to search.

// section 2 · terminal

Process & search

Find what's running, find what's not, kill what shouldn't be.

`ps aux | grep python`	Show all Python processes.
`top / htop`	Live process viewer (htop nicer, install separately).
`kill -9 12345`	Force-kill PID 12345.
`pkill -f 'strategy_runner'`	Kill by command-line pattern.
`lsof -i :8080`	What's holding port 8080?
`nohup python run.py &`	Background process that survives logout.
`grep -r 'API_KEY' .`	Recursive search for string in current dir.
`grep -rn 'TODO' src/`	Add line numbers.
`find . -name '*.csv'`	Find files by glob.
`find . -mtime -1`	Files modified in last 24h.

// section 2 · terminal

Network & download

Fetch market data, hit broker APIs, prove the network works.

`curl https://api.io`	GET request. Default verb.
`curl -X POST -d @body.json url`	POST a JSON file.
`curl -H 'Authorization: Bearer X' url`	Bearer-auth a request.
`wget url`	Download to current dir. Resumes with `-c`.
`ping api.broker.com`	Round-trip latency check.
`dig api.broker.com`	Resolve DNS — name → IPs.
`ssh user@host`	Open remote shell.
`scp file user@host:~/`	Copy file to remote host.

Tip. When a broker API fails, run `curl -v` first — the verbose output tells you whether the issue is DNS, TCP, TLS, auth, or response body.

// section 2 · terminal

Git on the command line

The minimum git you need before you're effective.

`git status`	Always start here. What's changed?
`git add -p`	Stage hunks interactively. Forces you to read the diff.
`git commit -m 'msg'`	Commit with inline message.
`git push`	Push current branch to its tracked remote.
`git pull --rebase`	Pull, replay your commits on top.
`git checkout -b feat/x`	Create + switch to new branch.
`git switch main`	Switch (modern alternative to checkout).
`git log --oneline -20`	Last 20 commits, one line each.
`git diff --staged`	Show what's about to be committed.
`git stash`	Park dirty changes; restore with `stash pop`.

// section 2 · terminal

Python toolchain

Once your Python project has more than one file, this is your tax.

`python -V`	Print Python version. Confirm before installing anything.
`python -m venv .venv`	Create isolated environment.
`source .venv/bin/activate`	Activate env. Prompt should change.
`deactivate`	Leave the env.
`pip install pandas`	Install a package into the active env.
`pip install -r requirements.txt`	Install everything pinned in the file.
`pip freeze > requirements.txt`	Snapshot exact versions.
`python -m pytest -q`	Run tests.
`python -m black .`	Auto-format.
`python -m mypy src/`	Static type check.

// section 3 · git

The 10-step daily workflow

Burn this into muscle memory. Every shipped feature follows the same dance.

01	`git pull --rebase`	Bring main up to date before you touch anything.
02	`git checkout -b feat/x`	Branch off main. Name it after the change, not the date.
03	`Edit files`	Small, focused changes. One concern per branch.
04	`git status / git diff`	Read what you actually did, not what you remember doing.
05	`git add -p`	Stage hunk-by-hunk. Catch debug prints here.
06	`git commit -m 'feat: ...'`	Conventional Commits keep changelogs readable.
07	`git push -u origin feat/x`	Push and track. First push only needs `-u`.
08	`Open PR on GitHub`	Description = what + why + how to test.
09	`Address review`	Push more commits — don't rewrite history mid-review unless asked.
10	`Squash & merge`	One feature → one commit on main. Delete the branch.

// section 3 · git

Branching & merging

Two trees. Replay one onto the other.

Merge vs rebase

Merge creates a merge commit. History shows a literal branch in the graph. Safer; easier to undo. Rebase rewrites your commits on top of the target. History is linear; reads cleanly. Pick one per project and stick with it. We use rebase for feature branches, merge for main.

```
# Replay feature commits on latest main:
git checkout feat/x
git fetch origin
git rebase origin/main
# resolve conflicts → git add . → git rebase --continue
git push --force-with-lease     # safer than --force
```

When conflicts hit

Open conflicted files. Look for <<<<<<< / ======= / >>>>>>> markers. Keep the version you want, delete the rest, delete the markers. `git add`, then `git rebase --continue`.

// section 3 · git

Recovery playbook

Almost nothing is unrecoverable. Calm down. Run these.

I committed to the wrong branch

```
git log -1   # note the SHA
git reset --hard HEAD~1
git checkout right-branch
git cherry-pick <SHA>
```

I want to undo my last commit but keep changes

```
git reset --soft HEAD~1
```

I want to discard local changes entirely

```
git restore .          # tracked files
git clean -fd          # untracked + dirs
```

I rebased and lost commits

```
git reflog             # find lost SHA
git reset --hard <SHA>
```

I want a one-line patch from another branch

```
git checkout other-branch -- path/to/file
```

// section 3 · git

Cheat sheet — the 20

Print this. Tape it next to your monitor.

`git init`	Create a new repo here.
`git clone url`	Copy a remote repo locally.
`git status`	Working tree state.
`git add -p`	Stage hunks interactively.
`git commit -m 'msg'`	Commit staged changes.
`git commit --amend`	Edit the previous commit.
`git log --oneline -20`	Recent commits, terse.
`git diff`	Unstaged changes.
`git diff --staged`	Staged changes.
`git branch`	List branches.
`git checkout -b name`	New branch + switch.
`git switch name`	Switch (modern).
`git fetch origin`	Pull metadata only.
`git pull --rebase`	Fetch + replay locals.
`git push -u origin name`	Push + set upstream.
`git stash`	Park dirty changes.
`git stash pop`	Restore stashed changes.
`git rebase main`	Replay onto main.
`git reflog`	All HEAD movements — your safety net.
`git tag v1.0.0`	Mark a release.

// section 4 · env

Python — venv + pip

Never `pip install` system-wide. Always a venv.

```
# 1. Create
python -m venv .venv

# 2. Activate (macOS / Linux)
source .venv/bin/activate

# 2. Activate (Windows PowerShell)
.venv\Scripts\Activate.ps1

# 3. Install
pip install pandas numpy talib-binary requests

# 4. Snapshot
pip freeze > requirements.txt

# 5. Reproduce on another machine
python -m venv .venv && source .venv/bin/activate
pip install -r requirements.txt
```

Verify. Run `which python` — it should point inside `.venv/`. If it points to `/usr/bin/python`, the venv isn't active and your installs are leaking into the system.

// section 4 · env

Node — nvm + pnpm

Pin the Node version per project. Don't fight global Node.

```
# Install nvm (once)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

# Per project
echo 20.11.1 > .nvmrc
nvm install
nvm use

# Faster, deterministic installs
corepack enable
corepack prepare pnpm@9 --activate

pnpm install
pnpm dev
```

Why pnpm

Hard-linked dependencies (one copy on disk per version) → faster installs, smaller `node_modules`. Strict by default — your code can't accidentally import undeclared packages.

// section 4 · env

Docker — Compose quickstart

Reproducible, isolated, identical on every machine.

```
# docker-compose.yml
services:
  redis:
    image: redis:7-alpine
    ports: ['6379:6379']
  strategy:
    build: .
    env_file: .env
    depends_on: [redis]
    restart: unless-stopped

# Daily commands
docker compose up -d           # start in background
docker compose logs -f         # tail logs
docker compose ps              # what's running
docker compose down            # stop
docker compose build --no-cache strategy   # force rebuild
```

// section 4 · env

Troubleshooting — the 8 common errors

If you've installed Python more than twice, you've hit at least three of these.

`ModuleNotFoundError`

venv not activated, or you installed into the wrong one. Run `which python`.

`SSL: CERTIFICATE_VERIFY_FAILED`

Corporate proxy or stale certifi. `pip install --upgrade certifi`.

`Permission denied: '/usr/local/lib/...'`

You're trying to install system-wide. Use a venv.

`port 8080 already in use`

Another process holds it. `lsof -i :8080` → kill PID.

`docker: Cannot connect to daemon`

Docker Desktop isn't running. Start it; wait for the whale icon.

`npm ERR! ERESOLVE`

Peer dependency conflict. Try `npm install --legacy-peer-deps` or fix the offender.

`zsh: command not found: nvm`

Add `nvm.sh` source line to `~/.zshrc`, restart shell.

`Killed (exit 137)`

Out of memory. Increase Docker memory or reduce concurrency.

// scenario · 22

Sev 1

Double-fill

Your strategy submits a buy order. Network blips. Strategy resubmits. Broker accepts both. You're now long 200 instead of 100.

Symptom

Position size = 2× expected. Two fills with timestamps within 200ms.

Root cause

No client order ID. Retry logic doesn't dedupe.

Fix path

Send a UUID as `client_order_id`. Broker rejects duplicates server-side.

Pin test

Mock the broker. Submit twice with same ID. Assert one fill.

// scenario · 23

Sev 1

Backtest ≠ live

Backtest shows 18% CAGR. Live runs at 2%. You haven't changed code.

Symptom

Live trades on data the backtest never saw. Slippage 5× higher than modeled.

Root cause

Backtest used adjusted close. Live trades on bid/ask. Look-ahead bias on indicators.

Fix path

Use raw OHLC in backtest. Add realistic slippage (1× spread). Confirm indicators only reference past bars.

Pin test

Bar-by-bar replay through the live execution path. Diff should be zero.

// scenario · 24

Sev 1

Position drift

Strategy thinks it's flat. Broker says you're long 50.

Symptom

`strategy.position_size == 0`, but broker dashboard shows 50 long.

Root cause

Restart lost in-memory state. Manual close at broker not reflected back.

Fix path

On startup, reconcile against broker positions API. Trust the broker, not the cache.

Pin test

Kill the process mid-position. Restart. Assert reconciled state matches broker.

// scenario · 25

Sev 2

Silent stop

Strategy hasn't traded in 6 hours. You didn't notice until you checked.

Symptom

Process is alive (or dead). No errors. No trades. No heartbeat.

Root cause

Exception caught silently. Or thread deadlocked. Or websocket disconnected without retry.

Fix path

Emit a heartbeat every 30s to a watchdog. Alert when missed for 2 cycles. Re-raise unexpected exceptions.

Pin test

Suspend the process for 60s. Watchdog alert must fire.

// scenario · 26

Sev 2

Latency spike

Median fill latency: 80ms. Today: 800ms. Slippage doubled.

Symptom

Latency histogram has a fat right tail. Worst case used to be 200ms; now 4s.

Root cause

Garbage collection pause, or feed lag, or new code on the hot path doing I/O.

Fix path

Profile. Move I/O off the order path. Pre-allocate. Add p95/p99 latency to dashboards.

Pin test

Synthetic load test asserting p99 < 200ms. Run on every PR.

// scenario · 27

Sev 2

Edge crash

Strategy crashes once a week on what looks like a random tick.

Symptom

Stack trace points to `NoneType` or `KeyError`. Always after a wide bar or thin volume.

Root cause

Edge case — gap, zero volume, missing field. Code assumes presence.

Fix path

Validate inputs at the boundary. Default values for missing fields. Log the bar that triggered it.

Pin test

Replay the offending bar. Assert no crash.

// scenario · 28

Sev 3

Rate limit

Broker returns 429 Too Many Requests. Orders pile up.

Symptom

Cluster of 429s followed by a few seconds of dropped orders.

Root cause

Burst of orders exceeded broker's per-second cap.

Fix path

Token bucket rate limiter. Back off on 429 with exponential delay. Cap concurrent in-flight.

Pin test

Mock broker that 429s every 5th request. Assert no orders lost; retries succeed.

// scenario · 29

Sev 3

Auth token expiry

Strategy ran fine for 23 hours. Hour 24: 401 on every request.

Symptom

Every API call returns `401 Unauthorized` simultaneously.

Root cause

Access token expired. No refresh logic; or refresh token also expired.

Fix path

Refresh proactively at 80% of TTL. On 401, refresh and retry once. Alert if refresh fails.

Pin test

Mock broker that 401s after N calls. Assert refresh + retry succeeds without manual intervention.

// section 6 · next

What's next

Six deep dives, fifteen PDFs, one Nexural Academy.

Per-platform deep dives

TradingView, NinjaTrader, cTrader, Alpaca, QuantConnect, IBKR. Step-by-step walkthroughs of the same RSI strategy on each.

Backtesting

How to backtest without lying to yourself. Walk-forward, Monte Carlo, slippage, costs.

Data engineering

Tick → bar → feature. Storage, schemas, time zones, corporate actions.

Machine learning

Feature engineering for finance. Why most ML in trading fails. When it doesn't.

Operations

Monitoring, on-call, runbooks. Recovery from the worst day.

Glossary & legal

Every term defined. Every disclaimer in plain English.

Continue at

nexural.com/learn/automation

nexural.com/learn/automation/coding-fundamentals

Continue reading → Next module: Platforms