What are we going to be talking about today?

- SAT Solvers
  - Enumerating Versions
- Declarative Parser
  - Feature Blocks
- Some small VSCode improvements:
  - VSCode highlights propagated exceptions
  - Better nimble integration with vscode
- Setup Nimble Action

WHY SAT SOLVERS?

Traditional greedy solvers make locally optimal choices, often leading to dependency conflicts or suboptimal selections.

EXAMPLE ISSUE:

- Project A requires B >= 0.1.4 and C <= 0.1.0
- Project B (version 0.1.4) depends on C (any version)
- Project C has versions 0.1.0 and 0.2.1

A greedy solver might pick C 0.2.1, breaking Project A's constraints

SAT SOLVER ADVANTAGE:

- ✔ Global optimization: Ensures all dependencies are satisfied simultaneously
- ✔ Backtracking & constraint solving: Finds a valid solution where C 0.1.0 is chosen
- ✔ Correct & complete solutions: Avoids resolution failures that greedy solvers cannot handle

ENUMERATING VERSIONS:

Smarter Dependency Resolution:

Traditional solvers pick a single version per package upfront, often leading to failures when dependencies are not satisfiable.

NEW APPROACH:

🔄 Iterative Version Selection: Instead of failing early, the solver downloads multiple versions and retries when conflicts arise.

HOW IT WORKS:

- If a dependency constraint allows any version, we fetch multiple versions
- If the first attempted solution fails, we backtrack and try another version
- This increases the chance of finding a valid solution automatically

EXAMPLE:

- A package requires B 0.1.4, which depends on C (any version)
- If picking C 0.2.1 fails, the solver automatically tries C 0.1.0
- The solver adapts dynamically to available versions, reducing resolution failures

ADVANTAGES:

- ✔ More robust package resolution
- ✔ Fewer manual fixes needed
- ✔ Improved user experience

DECLARATIVE PARSER

- 🚀 New in Nimble 0.18.0
- Enable with `--parser:declarative`
- ➡️ SOON Will become the default parser in future versions

ADVANTAGES:

- ✔ Deterministic dependencies → Reliable caching
- ✔ Improved speed → No need to spin up the Nim VM for every dependency

CONDITIONAL DEPENDENCIES WITH FEATURE BLOCKS

- Supports optional dependencies
- Avoids unnecessary packages when not needed

EXAMPLE:

A library supporting both asyncdispatch and chronos:

```
#nimble file
feature "chronos":
  require "chronos"
```

ACTIVATING FEATURES

Command line

```
`nimble --feature:"chronos" install`
```

Dependency level activation

```
`require "awesomeAsyncPackage[chronos]"`
```

CHECKING ACTIVE FEATURES IN CODE

```
1 when defined(feature.awesomeAsyncPackage.chronos):
2   import chronos
3 else:
4   import std/asyncdispatch
```

THE DEV FEATURE

Useful for development-specific dependencies

```
1 feature "dev":
2   require "unittest2"
```

```
1 #alias
2 dev:
3   require "unittest2"
```

VSCode highlights propagated exceptions

```
## Initialises a new   Submodule   object.
syncio.writeFile: proc (filename: string, content: string){.gcsafe, raises: <inferred> [ref IOError].}
Opens a file named `filename` for writing. Then writes the
`content` completely to the file and closes the file afterwards.
Raises an IO exception in case of an error.
propagated exceptions: @[ref IOError]
writeFile "whatever.txt", "wrote"
"ok"
```

Better nimble integration with vscode

```
code-nim > nimvscode.nimble > ...
    cmd = "powershell.exe " & cmd
  exec cmd

Run task
task installVsix, "Install the VSIX package":
  initialNpmInstall()
  exec "code --install-extension out/nimvscode-" & version & ".vsix"

# Tasks for maintenance
Run task
task auditNodeDeps, "Audit Node.js dependencies":
  initialNpmInstall()
  exec "npm audit"
  echo "NOTE: 'engines' versions in 'package.json' need manually audited"

Run task
task upgradeNodeDeps, "Upgrade Node.js dependencies":
  initialNpmInstall()
  exec "npm exec -c 'ncu -ui'"
  exec "npm install"
  echo "NOTE: 'engines' versions in 'package.json' need manually upgraded"

Run task
```

Editing instructions... (↑↓ for history, @ for code / documentation)
Esc to close · claude-3.5-sonnet · ⇧⌘K to toggle

```
NIM
> LSP Status
v Nimble Tasks
  Refresh Nimble Tasks
  main  This compiles the vscode Nim extension
  release  This compiles a release version
  vsix  Build VSIX package
  installVsix  Install the VSIX package
  auditNodeDeps  Audit Node.js dependencies
  upgradeNodeDeps  Upgrade Node.js dependencies
  anotherTask  This is another task
> LSP Notifications
```

Setup Nimble Action

```yaml
- name: Setup Nimble
        uses: nim-lang/setup-nimble-action@v1
        with:
          nimble-version: "latest"
          repo-token: ${{ secrets.GITHUB_TOKEN }}
```