Installation https://narimiran.github.io/nim-basics/

Installation mit Scoop (empfohlen):

[Beschreibung Scoop] Scoop ist ein Kommandozeilen-Installationsprogramm für Windows.
Scoop installiert standardmäßig Programme im Home-Verzeichnis. Es werden also keine Administratorrechte benötigt, um Programme zu installieren, und es werden nicht jedes Mal UAC-Popups angezeigt, wenn ein Programm hinzugefügt oder entfernt wird.
Nicht sicher, ob 32-Bit oder 64-Bit benötigt wird, oder welche Befehle notwendig sind um nach der Installation ergänzende Pakete zu erhalten? Scoop liest die README und ermittelt somit alle Abhängigkeiten.
Einfach `scoop install <Anwendung>` in der PowerShell Konsole eingeben, alles andere erledigt Scoop.

Installation von Scoop
Scoop wird über die PowerShell Kommandozeile installiert. Das erfordert PS-Version 5 oder höher. Ich empfehle `PowerShell v7.0`, braucht nicht installiert werden. Download entpacken und Konsole mit Aufruf `pwsh.exe` starten. Download hier: https://github.com/PowerShell/PowerShell/releases/tag/v7.0.0
Windows 10 hat bereits mindestens Version 5 installiert.
Eingabe in die PS-Kommandozeile: `iwr -useb get.scoop.sh | iex`
Falls eine Fehlermeldung erscheint, muss möglicherweise die Ausführungsrichtlinie geändert werden mit `Set-ExecutionPolicy RemoteSigned -scope CurrentUser`

Nim installieren in PS-Kommandozeile: `scoop install nim`
Mitinstalliert wird MingW.
Die Umgebungsvariable "`PATH`" muss um den Eintrag `C:\Users\USER\scoop\apps\nim\1.2.0\bin` ergänzt werden. (Versionsnummer kann variieren)
ACHTUNG! Bei Updates (`scoop update nim`) wird `PATH` nicht automatisch geändert und muss per Hand angepasst werden.

Installation manuell:

Download der letzten Windows Version (32/64 bit Version) von der Installations-Seite:
https://nim-lang.org/install_windows.html
Entpacken in das gewünschte Installationsverzeichnis.
Nach Installation, das Installationsverzeichnis öffnen und die "`finish.exe`" ausführen. Dieses Tool prüft auch, ob ein C-Compiler vorhanden ist, und kann MingW, die GNU C-Compilersammlung für Windows, installieren.
Die Umgebungsvariable "`PATH`" muss um 2 Einträge ergänzt werden
- für das Verzeichnis "`\bin`" des Installationsverzeichnisses
- für "`%USERPROFILE%\.nimble\bin`"
Das sollte bereits von der finish.exe erledigt werden, trotzdem überprüfen.

Es gibt eine Reihe anderer Abhängigkeiten, die eventuell installiert werden müssen, um Nim verwenden zu können. Sie müssen im selben Ordner gespeichert werden, wie die `nim.exe`. Im Ordner überprüfen, falls nicht vorhanden, von hier: https://nim-lang.org/download/dlls.zip herunterladen.
Sie beinhalten u.a. PCRE und OpenSSL, es handelt sich um folgende Dateien:
`libcurl.dll` `libeay32.dll` `libeay64.dll` `pcre.dll` `pcre32.dll` `pcre64.dll` `pdcurses.dll` `pdcurses32.dll` `pdcurses64.dll` `sqlite3_32.dll` `sqlite3_64.dll` `ssleay32.dll` `ssleay64.dll`

Zusammen mit Nim wird auch der zugehörige Paketmanager Nimble installiert. Darüber lassen sich Module über die Konsole installieren. Der Paketmanager verwendet Git. Falls nicht installiert, Download von hier: https://git-scm.com/downloads.

IDE

Grundsätzlich ist jeder Texteditor möglich. Um jedoch Syntaxhervorhebung etc. nutzen zu können, ist ein Editor zu wählen, der PlugIns für Nim anbietet. Visual Studio Code ist dafür geeignet zusammen mit Nim extension und Code Runner (für schnelles Kompilieren und Ausführen). Für Liebhaber von Vim ist Neovim zu empfehlen.

Hinweis zu Visual Studio Code:

Microsoft's vscode source code is open source (MIT-licensed), but the product available for download (Visual Studio Code) is licensed under this not-FLOSS license and contains telemetry/tracking.

Aus diesem Grund gibt es das Projekt VSCodium.
The VSCodium project exists so that you don't have to download+build from source. This project includes special build scripts that clone Microsoft's vscode repo, run the build commands, and upload the resulting binaries for you to GitHub releases. These binaries are licensed under the MIT license. Telemetry is disabled.

VSCodium ist somit identisch zu VSCode, nur ohne Telemetrie.
Installation mit scoop:
"extras" zu scoop hinzufügen - `scoop bucket add extras`
VSCodium installieren - `scoop install vscodium`

VSCode/*VSCodium* sollte noch etwas konfiguriert werden, um optimal mit Nim arbeiten zu können. Das beinhaltet auch Einstellungen für *Code Runner*.
Hier mal meine Einstellungen, mit eigenem *Font* (`'Source Code Pro'`). Die anderen Einstellungen sind ohne Zusätze möglich. Hier sind auch die Werte zum Deaktivieren der Telemetrie von VSCode gesetzt.

settings.json:

```
{
    //==> Zoomfaktor des Fensters, je Schritt +/- 20 %
    "window.zoomLevel": 0,
    //==> Steuert die Sichtbarkeit der Aktivitätsleiste
    "workbench.activityBar.visible": true,
    //==> Gibt das im Arbeitsbereich verwendete Farbdesign an
    "workbench.colorTheme": "Tomorrow Night Blue",
    //==> Aktiviert Absturzberichte, die an Microsoft gesendet werden.
    // Diese Option erfordert einen Neustart, damit sie wirksam wird.
    "telemetry.enableCrashReporter": false,
    //==> Aktivieren Sie das Senden von Nutzungsdaten und Fehlern an Microsoft.
    "telemetry.enableTelemetry": false,
    //==> Steuert die automatische Speicherung ungespeicherter Editoren.
    "files.autoSave": "afterDelay",
    //==> Steuert, ob die Minikarte angezeigt wird.
    "editor.minimap.enabled": false,
    //==> Steuert die Schriftfamilie. *** KEIN STANDARD-FONT, MUSS ZUSÄTZLICH INSTALLIERT WERDEN ***
    "editor.fontFamily": "SourceCodePro, 'Source Code Pro', monospace",
    //==> Steuert den Schriftgrad in Pixeln.
    "editor.fontSize": 14,
    //==> Schriftgröße des Editors mit STRG+Mausrad verändern
    "editor.mouseWheelZoom": true,
    //==> Steuert, wie der Editor die aktuelle Zeilenhervorhebung rendern soll.
    // - none
    // - gutter
    // - line
    // - all: Hebt den Bundsteg und die aktuelle Zeile hervor.
    "editor.renderLineHighlight": "gutter",
    //==> Legt fest, ob der Editor die aktuelle Zeichenhervorhebung nur dann rendern soll, wenn der
    Editor fokussiert ist.
    "editor.renderLineHighlightOnlyWhenFocus": true,
    //==> Tabulator & Ruler
    //==> Tabulatorweite
    "editor.tabSize": 2,
    //==> Tab-Vervollständigungen aktivieren.
    "editor.tabCompletion": "off",
    //==> Das Einfügen und Löschen von Leerzeichen erfolgt nach Tabstopps
```

```
    "editor.useTabStops": true,
    //==> Vertikale Linien nach einer bestimmten Anzahl von Monospacezeichen rendern. Verwenden Sie
    mehrere Werte für mehrere Linien. Wenn das Array leer ist, werden keine Linien gerendert.
    "editor.rulers": [80],
    //==> Steuert, wie der Zeilenumbruch durchgeführt werden soll.
    "editor.wordWrap": "wordWrapColumn",
    //==> Code Runner
    // vor Run letzte Ausgabe löschen
    "code-runner.clearPreviousOutput": true,
    //==> die Standardsprache
    "code-runner.defaultLanguage": "nim",
    //==> aktuelle Datei vor Ausführen speichern
    "code-runner.saveFileBeforeRun": true,
    //==> Bestimmt die Ausführungsoption für bestimmte Dateierweiterungen
    "code-runner.executorMap": {
        "html": "\"C:\\Program Files (x86)\\Google\\Chrome\\Application\\chrome.exe\"",
        "nim": "nim compile --verbosity:0 --hints:off --run $fullFileName"
        // "c": "cd $dir && gcc $fileName -o $fileNameWithoutExt && $dir$fileNameWithoutExt"
    },
    //==> Wenn aktiviert, werden Updates für Erweiterungen automatisch installiert
    "extensions.autoUpdate": false,
    //==> Der Standardsprachmodus, der neuen Dateien zugewiesen ist. Wenn "${activeEditorLanguage}"
    dafür konfiguriert ist, wird, falls möglich, der Sprachmodus des aktuell aktiven Text-Editors
    verwendet.
    "files.defaultLanguage": "nim",
    //==> Erlauben von Automatischen Updates für VSCode
    "update.mode": "none",
    //==> Gibt an, ob Tastaturzuordnungen im Terminal zugelassen werden sollen
    "terminal.integrated.allowChords": false,
}
```

In den Tastatureinstellungen habe ich noch angepasst
`Blockkommentar umschalten STRG+UMSCHALTTASTE+#`
Dann ist es in logischem Einklang zu
`Zeilenkommentar umschalten STRG+#`

Start

Beginnen wir mit dem üblichen "Hello World!".
Im Editor schreiben wir die Zeile `echo "Hello World!"` und speichern die Datei als "`helloworld.nim`".
Wichtig! Strings (`Typ: string`) müssen immer in *doppelten* Anführungszeichen eingefasst werden. *Einfache* Anführungszeichen werden für einzelne Zeichen (`Typ: char`) verwendet.

Kompiliert wird mit `nim c helloworld.nim`. Nach erfolgreicher Ausführung wird im selben Ordner die Datei `helloworld.exe` erstellt. Diese lässt sich nun in der Konsole mit `helloworld.exe` starten.
Bei Verwendung von VSCode mit der Code Runner Erweiterung wird mit `Strg+Alt+N` kompiliert und sofort die exe gestartet. Das entspricht dem Aufruf `nim c -r "pfad\zu\meineDatei.nim"`. Alle Kommandozeilenoptionen sind mit `nim --help` abfragbar.

Grundlagen

Anweisungen eines Blocks werden durch Einzüge markiert. Ein Einzuglevel ist fix 2 Leerzeichen. Tab darf nicht verwendet werden (also Tabweite auf 2 stellen führt zu Fehler).
Für ein Programm können mehrere `*.nim` verwendet werden. Um darauf zuzugreifen wird `import meineDatei.nim` am Dateianfang eingefügt. Da Funktionen, Variablen etc. nur im Scope ihrer Datei gültig sind, müssen diejenigen Funktion, Variablen, Konstanten, die allgemeingültig sein sollen, dafür kenntlich gemacht werden. Das geht wie folgt:
`var <name>*: <type>`
Also ein `*` an den jeweiligen Namen anhängen (die spitzen Klammern dienen nur der Hervorhebung).
Kommentare werden durch eine `#` gekennzeichnet, am Zeilenanfang oder inline.

```
var <name>: <type>  # comment inline
# comment line
```

Blockkommentare sind wie folgt möglich:

```
#[
block
...
comment
]#
```

Die Darstellung von Strings wurde schon beim Start kurz angesprochen. Weitere Möglichkeiten sind Multiline Strings:

```
"""
this is a
multiline
string
"""
```

oder Raw String
`r"c:\muster\pfad"` im Vergleich zur Normalstringvariante, in der spezielle Zeichen maskiert werden müssen: `"c:\\muster\\pfad"`.

Variablen (veränderlich)

Nim ist eine statisch typisierte Programmiersprache, d.h. der Typ einer Zuweisung muss deklariert werden, bevor der Wert verwendet werden kann.
Wichtig! Nim unterscheidet nicht zwischen Groß- und Kleinschreibung und Unterstrich, was bedeutet, dass `helloWorld` und `hello_world` identisch sind. Die Ausnahme ist das erste Zeichen, bei dem zwischen Groß- und Kleinschreibung unterschieden wird. Namen können auch sowohl Zahlen als auch andere UTF-8-Zeichen enthalten, (sogar Emojis, falls gewünscht).
Üblicherweise werden Variablen im CamelCase Stil geschrieben, erster Buchstabe klein.
Die Deklaration erfolgt über das Schlüsselwort `var`.
Einzelne Variable:
`var <name>: <type>`
Einzelne Variable mit Wertzuweisung, Zuweisungsoperator ist `=`:
`var <name>: <type> = <value>`
Nim erkennt automatisch den Typ eines zugewiesenen Wertes, sodass die Deklaration auch so möglich ist:
`var <name> = <value>`

Mehrere Variablen werden im Block deklariert:

```
var
  <name>: <type>
  <name>: <type>
```

Variablen (unveränderlich)

Nim erlaubt das Deklarieren von Variablen, denen einmalig ein Wert zugewiesen werden kann. Nach erfolgter Zuweisung ist die Variable nicht mehr veränderlich.
Diese Deklaration erfolgt mit dem Schlüsselwort `let`.

```
let a = 5
a = 7 # error
```

Konstanten

Konstanten sind unveränderlich und müssen bei ihrer Deklaration mit einem Wert belegt werden. Eine Typisierung ist nicht erforderlich, da durch die Wertzuweisung der Typ automatisch erkannt wird. Zur Deklaration wird das Schlüsselwort `const` verwendet.

```
const <name> = <value>
```

In vielen Sprachen ist es üblich, Konstantennamen kpl. in Großbuchstaben zu schreiben. In Nim werden sie wie jede andere Variable geschrieben.