pyglenn — Cheat Sheet

> **v0.2.0** · 2,035 species · 3,779 intervals · zero runtime dependencies
> **Author:** Dr. Reginaldo G. Leão Jr. · ProfLeao/pyglenn

🚀 30-second start

```
pip install pyglenn                  # PyPI
conda install conda-forge::pyglenn   # conda-forge
```

```
from pyglenn import ThermochemicalCalculator

with ThermochemicalCalculator() as calc:
    sid  = calc.get_available_species('CO2', exact_match=True)[0].id
    prop = calc.calculate_properties(sid, 1000.0)

    print(f"Cp = {prop.cp:.2f} J/(mol·K)")
    print(f"H° = {prop.h_relative:.1f} J/mol")
    print(f"S° = {prop.s:.3f} J/(mol·K)")
```

📖 Recipes

🔍 Find a species

```
# Exact match (recommended) — 'N2' returns N₂, not Be₃N₂
calc.get_available_species('N2', exact_match=True)[0].id

# Substring search
calc.get_available_species('CH4')

# Full alphabetical listing (single query)
calc.list_all_species()
```

🔥 Single-point properties

```
p = calc.calculate_properties(species_id, 1000.0)
# p.cp            → J/(mol·K)
# p.h_relative    → J/mol (absolute H°(T), NASA-7 convention)
# p.s             → J/(mol·K)
# p.species_name  → 'CO2'
# p.phase         → 'gas' | 'condensed'
# p.temp_interval → (T_min, T_max)   # tuple
```

📈 Temperature sweep

```
sid = calc.get_available_species('CO2', exact_match=True)[0].id

# Typed, ordered list — loads intervals once (fast for long sweeps)
props = calc.calculate_properties_range(sid, [300, 500, 1000, 1500])

# Temperature-keyed dict (legacy)
by_t = calc.get_properties_range(sid, [300, 500, 1000, 1500])
# → {300.0: ThermoProperties, 500.0: ThermoProperties, ...}
```

💉 Enthalpy

```python
# Formation at 298.15 K
hf = calc.calculate_formation_enthalpy(species_id)  # J/mol

# Sensible ΔH between two temperatures
dh = calc.calculate_enthalpy_change(species_id, 298.15, 1000.0)  # J/mol
```

🛡️ Error handling

```python
from pyglenn import ThermoCalcError

try:
    p = calc.calculate_properties(sid, T)
except ThermoCalcError as e:
    print(f"Failed: {e}")
```

Exception	Meaning	Context carried
`ThermoCalcError`	Base — catches all pyglenn errors	—
`DatabaseNotConnectedError`	Forgot `.connect()` or context manager	—
`SpeciesNotFoundError`	Invalid species ID	`.species_id`
`TemperatureOutOfRangeError`	T outside valid intervals	`.temperature`, `.species_name`, `.temp_bounds`

🧮 NASA-7 Polynomials

pyglenn evaluates the standard NASA-7 (Gordon–McBride) form:

$$\frac{C_p^\circ}{R} = \frac{a_1}{T^2} + \frac{a_2}{T} + a_3 + a_4 T + a_5 T^2 + a_6 T^3 + a_7 T^4$$

$$\frac{H^\circ}{RT} = -\frac{a_1}{T^2} + a_2 \frac{\ln T}{T} + a_3 + a_4 \frac{T}{2} + a_5 \frac{T^2}{3} + a_6 \frac{T^3}{4} + a_7 \frac{T^4}{5} + \frac{b_1}{T}$$

$$\frac{S^\circ}{R} = -\frac{a_1}{2T^2} - \frac{a_2}{T} + a_3 \ln T + a_4 T + a_5 \frac{T^2}{2} + a_6 \frac{T^3}{3} + a_7 \frac{T^4}{4} + b_2$$

Dimensional values are obtained by denormalizing with the **dataset reference gas constant** `R_GLENN` (read automatically from the database `metadata` table) — not the modern universal value:

Constant	Value (J/(mol·K))	Meaning
`R_GLENN`	8.314510	NASA Glenn/CEA `thermo.inp` reference (CODATA 1986) — used for denormalization
`R_UNIVERSAL`	8.31446261815324	CODATA 2018/2022 universal value — legacy fallback

Using the universal value instead of `R_GLENN` introduces a systematic **~5.7 ppm** bias in every dimensional result.

🔧 CLI

```
pyglenn query -s O2                    # quick lookup
pyglenn build -i thermo.inp -o my.db -v    # rebuild database
pyglenn migrate-metadata -d my.db          # add dataset metadata to legacy DB
pyglenn validate -d my.db                  # read-only integrity checks
```

⚡ Tips

✅ Do	❌ Don't
Use context manager (`with`)	Call `.connect()` / `.close()` manually
`exact_match=True`	Rely on substring search for exact species
Catch `ThermoCalcError`	Assume a missing species returns `None`
`calculate_properties_range()` for sweeps	Loop `calculate_properties()` for many points
Use attribute access (`props.cp`)	Index results like dicts (`props['cp']`)
Bundled `thermo.db` (no path needed)	Ship your own DB unless customised

🔄 Migrating to v0.2.0

```
# BEFORE (v0.1.x — dicts)                    # AFTER (v0.2.0 — dataclasses)
props = calc.calculate_properties(sid, T)    props = calc.calculate_properties(sid, T)
cp    = props['cp']                          cp    = props.cp
name = props['species_name']                 name = props.species_name
```

- Public methods (`calculate_properties` , `get_available_species` , `find_species` , `get_species_data` , `get_statistics` , …) now return frozen dataclasses instead of `dict` .
- To recover the old `dict` shape: `from dataclasses import asdict; asdict(props)` .
- Legacy databases are migrated automatically on `connect()` (metadata only — thermochemical rows are never touched).
- `R` remains available as an alias of `R_UNIVERSAL` ; denormalization now uses the dataset's `R_GLENN` reference constant.

📦 API at a glance

Symbol	Purpose
`ThermochemicalCalculator`	High-level — properties, enthalpy, species lookup
`ThermoDBQuery`	Low-level — SQL access, coefficient evaluation, validation
`ThermoDBBuilder`	Build DB from FORTRAN `thermo.inp`
`ThermoProperties` , `SpeciesInfo` , `NASACoefficients` , `IntervalData` , `SpeciesData` , `DatabaseStats`	Typed result dataclasses
`R_GLENN` , `R_UNIVERSAL` , `R`	Dataset reference constant, universal constant, alias
`ThermoCalcError` (+ 3 subclasses)	Exception hierarchy