PHEP: 3
Title: PyHC Python & Upstream Package Support Policy
Author: Shawn Polson <pyhc-confidential@lasp.colorado.edu> <https://orcid.org/0000-0003-0619-5745>
Discussions-To: https://github.com/heliophysicsPy/standards/pull/29
Revision: 1
Status: Draft
Type: Standards Track
Content-Type: text/markdown; charset=UTF-8; variant=CommonMark
Created: 06-Jun-2024
Post-History: 06-Jun-2024, 11-Jun-2024, 02-Jul-2024, 17-Jul-2024, 23-Jul-2024, 05-Sep-2024,
09-Sep-2024, 18-Jun-2025
Resolution: https://doi.org/10.5281/zenodo.15080483

Abstract

This PHEP recommends that all projects across the PyHC ecosystem adopt a common time-based policy for support of dependencies, inspired by SPEC 0. Specifically, for Python versions and the upstream Scientific Python packages covered by SPEC 0, it recommends that projects:

1. Support Python versions for at least **36 months** (3 years) after their initial release.
2. Support upstream core Scientific Python packages for at least **24 months** (2 years) after their initial release.
3. Adopt support for new versions of these dependencies within **6 months** of their release.

At the time of writing, the upstream core Scientific Python packages are: `numpy`, `scipy`, `matplotlib`, `pandas`, `scikit-image`, `networkx`, `scikit-learn`, `xarray`, `ipython`, `zarr`.

This policy replaces the current standard #11 which simply mandates Python 3 support.

Motivation

The current PyHC standard #11, which mandates compatibility with Python 3, is outdated. Python 3 support is virtually universal now, so it would be more beneficial to replace this standard with a policy for how to support new minor Python versions and key upstream dependencies. SPEC 0 provides a structured support timeline that balances stability and progress, essential for software in the heliophysics community. Adopting a similar policy ensures consistency and predictability in support timelines. Additionally, limiting the scope of supported versions is an effective way for packages to limit maintenance burden while promoting interoperability.

Rationale

Following SPEC 0's 24/36-month support timeline keeps PyHC in better sync with the broader Scientific Python community, maintaining compatibility with newer Python features and key upstream dependencies, while providing adequate time for package maintainers to adapt. Allowing 6 months to adopt new versions ensures packages stay current with development cycles while providing a reasonable timeframe for testing and integration.

Specification

This PHEP refers to feature releases of dependencies (e.g., Python 3.12.0, NumPy 2.0.0; not Python 3.12.1, NumPy 2.0.1).

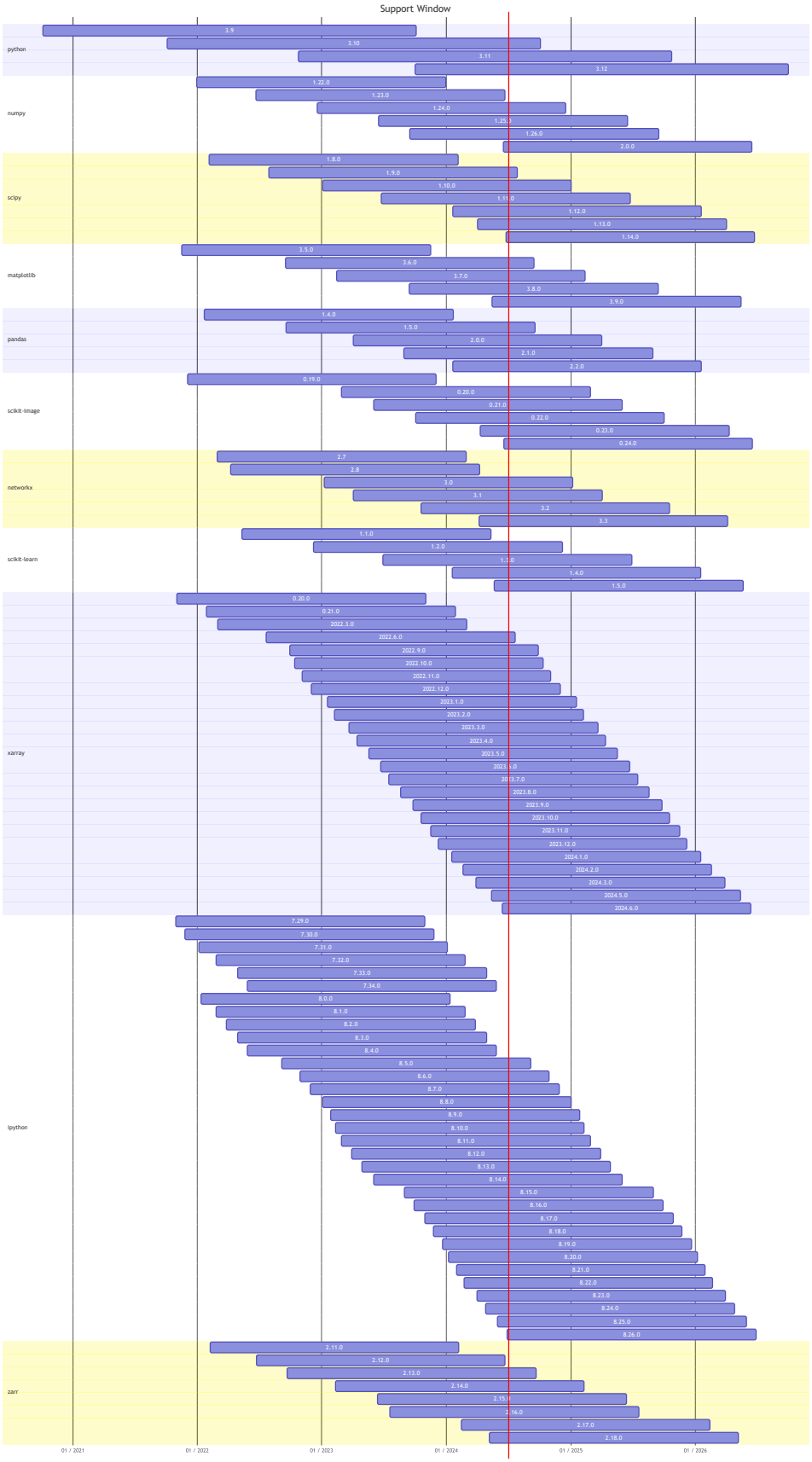
This PHEP adopts Scientific Python's SPEC 0 and specifies that all PyHC packages should:

1. Support Python versions for at least **36 months** (3 years) after their initial release.
2. Support upstream core Scientific Python packages for at least **24 months** (2 years) after their initial release.
3. Adopt support for new versions of these dependencies within **6 months** of their release.

At the time of writing, the upstream core Scientific Python packages are: `numpy`, `scipy`, `matplotlib`, `pandas`, `scikit-image`, `networkx`, `scikit-learn`, `xarray`, `ipython`, `zarr`. If their core packages are updated, this policy applies to the updated list instead.

Since new minor Python versions are released annually every October (PEP 602), this effectively means that PyHC packages should be supporting about three minor Python versions at any given time. Upstream packages have more varied release schedules, but several recent versions should typically be supported concurrently. Providing ongoing support for older versions beyond the specified support periods is optional.

The following shows the dependency support window as of this PHEP's adoption:


Support Window
python
3.9
3.10
3.11
3.12
numpy
1.22.0
1.23.0
1.24.0
1.25.0
1.26.0
2.0.0
scipy
1.8.0
1.9.0
1.10.0
1.11.0
1.12.0
1.13.0
1.14.0
matplotlib
3.5.0
3.6.0
3.7.0
3.8.0
3.9.0
pandas
1.4.0
1.5.0
2.0.0
2.1.0
2.2.0
scikit-image
0.19.0
0.20.0
0.21.0
0.22.0
0.23.0
0.24.0
networkx
2.7
2.8
3.0
3.1
3.2
3.3
scikit-learn
1.1.0
1.2.0
1.3.0
1.4.0
1.5.0
xarray
0.20.0
0.21.0
2022.3.0
2022.6.0
2022.9.0
2022.10.0
2022.11.0
2022.12.0
2023.1.0
2023.2.0
2023.3.0
2023.4.0
2023.5.0
2023.6.0
2023.7.0
2023.8.0
2023.9.0
2023.10.0
2023.11.0
2023.12.0
2024.1.0
2024.2.0
2024.3.0
2024.5.0
2024.6.0
ipython
7.29.0
7.30.0
7.31.0
7.32.0
7.33.0
7.34.0
8.0.0
8.1.0
8.2.0
8.3.0
8.4.0
8.5.0
8.6.0
8.7.0
8.8.0
8.9.0
8.10.0
8.11.0
8.12.0
8.13.0
8.14.0
8.15.0
8.16.0
8.17.0
8.18.0
8.19.0
8.20.0
8.21.0
8.22.0
8.23.0
8.24.0
8.25.0
8.26.0
zarr
2.11.0
2.12.0
2.13.0
2.14.0
2.15.0
2.16.0
2.17.0
2.18.0
01 / 2021
01 / 2022
01 / 2023
01 / 2024
01 / 2025
01 / 2026

PyHC packages should clearly document their dependency version policy (e.g., like PlasmaPy and SpacePy) and be tested against the minimum and maximum supported versions. Testing with CI against release candidates is encouraged, too, as a way to stay ahead of future releases. Packages that use semantic versioning should consider using their version number to indicate versions that drop support for older dependencies. There is no expectation that a package "deprecate" an older dependency before dropping support for it. However, there is an expectation that maximum or exact requirements (e.g., `numpy<2` or `matplotlib==3.5.3`) be set only when absolutely necessary (and that issues be immediately created to remove such requirements), and packages must not require versions of any dependency older than 24 months. Additionally, if a package has been supporting specific OS versions and CPU architectures (e.g., releasing binary wheels), this support should continue for new OS versions and architectures to maintain the same level of support as before.

This new policy replaces the current standard #11 in the PyHC standards document with the following new text:

> **11. Python and Upstream Package Support:** All packages should support minor Python versions released within the last 36 months (3 years) and upstream core Scientific Python packages released within the last 24 months (2 years). Additionally, packages should support new versions within 6 months of their release (see PHEP 3).

Lastly, if there is a Python 4 or other significant changes in dependencies, this policy will have to be reviewed in light of the community's and projects' best interests.

Backwards Compatibility

This policy potentially introduces backwards incompatibilities by enforcing a new support timeline, which may encourage some packages to drop support for older dependency versions sooner than planned.

Security Implications

There are no direct security implications of this policy. However, ensuring packages are updated to newer dependency versions may improve security by incorporating fixes and improvements from newer releases.

How to Teach This

- The PyHC Tech Lead will maintain a new web page on the PyHC website detailing the support policy and include a graphical timeline of the schedule (similar to the Gantt chart above).
- Automated email reminders will be sent via the PyHC mailing list quarterly and near important drop/support dates to remind package maintainers of the schedule.

Reference Implementation

Multiple PyHC packages already follow this version support policy. One notable example is PlasmaPy which currently documents their SPEC 0-based policy and even mentions it in comments inside their pyproject.toml file.

Code to generate support and drop schedules:

The following code can be used to generate support and drop schedules, including the Gantt chart above.

```
import requests
import collections
from datetime import datetime, timedelta

import pandas as pd
```

```
from packaging.version import Version


py_releases = {
    "3.9": "Oct 5, 2020",
    "3.10": "Oct 4, 2021",
    "3.11": "Oct 24, 2022",
    "3.12": "Oct 2, 2023",
}
core_packages = [
    "numpy",
    "scipy",
    "matplotlib",
    "pandas",
    "scikit-image",
    "networkx",
    "scikit-learn",
    "xarray",
    "ipython",
    "zarr",
]
plus36 = timedelta(days=int(365 * 3))
plus24 = timedelta(days=int(365 * 2))
plus6 = timedelta(days=int(365 * 0.5))

# Release data

# put cutoff 3 quarters ago - we do not use "just" -9 month,
# to avoid the content of the quarter to change depending on when we generate this
# file during the current quarter.

current_date = pd.Timestamp.now()
current_quarter_start = pd.Timestamp(
    current_date.year, (current_date.quarter - 1) * 3 + 1, 1
)
cutoff = current_quarter_start - pd.DateOffset(months=9)


def get_release_dates(package, support_time=plus24):
    releases = {}

    print(f"Querying pypi.org for {package} versions...", end="", flush=True)
    response = requests.get(
        f"https://pypi.org/simple/{package}",
        headers={"Accept": "application/vnd.pypi.simple.v1+json"},
    ).json()
    print("OK")

    file_date = collections.defaultdict(list)
    for f in response["files"]:
        ver = f["filename"].split("-")[1]
        try:
            version = Version(ver)
        except:
```

```
            continue

        if version.is_prerelease or version.micro != 0:
            continue

        release_date = None
        for format in ["%Y-%m-%dT%H:%M:%S.%fZ", "%Y-%m-%dT%H:%M:%SZ"]:
            try:
                release_date = datetime.strptime(f["upload-time"], format)
            except:
                pass

        if not release_date:
            continue

        file_date[version].append(release_date)

    release_date = {v: min(file_date[v]) for v in file_date}

    for ver, release_date in sorted(release_date.items()):
        drop_date = release_date + support_time
        if drop_date >= cutoff:
            releases[ver] = {
                "release_date": release_date,
                "drop_date": drop_date,
                "support_by_date": release_date + plus6
            }

    return releases


package_releases = {
    "python": {
        version: {
            "release_date": datetime.strptime(release_date, "%b %d, %Y"),
            "drop_date": datetime.strptime(release_date, "%b %d, %Y") + plus36,
            "support_by_date": datetime.strptime(release_date, "%b %d, %Y") + plus6
        }
        for version, release_date in py_releases.items()
    }
}

package_releases |= {package: get_release_dates(package) for package in core_packages}

# filter all items whose drop_date are in the past
package_releases = {
    package: {
        version: dates
        for version, dates in releases.items()
        if dates["drop_date"] > cutoff
    }
    for package, releases in package_releases.items()
}
```

```
# Save Gantt chart
# You can paste the contents into https://mermaid.live/ to generate the chart image.

print("Saving Mermaid chart to chart.md (render at https://mermaid.live/)")
with open("chart.md", "w") as fh:
    fh.write(
        """gantt
dateFormat YYYY-MM-DD
axisFormat %m / %Y
title Support Window"""
    )

    for name, releases in package_releases.items():
        fh.write(f"\n\nsection {name}")
        for version, dates in releases.items():
            fh.write(
                f"\n{version} : {dates['release_date'].strftime('%Y-%m-%d')},{dates['drop_date'].strftim
            )
    fh.write("\n")

# Print drop schedule

data = []
for k, versions in package_releases.items():
    for v, dates in versions.items():
        data.append(
            (
                k,
                v,
                pd.to_datetime(dates["release_date"]),
                pd.to_datetime(dates["drop_date"]),
                pd.to_datetime(dates["support_by_date"]),
            )
        )

df = pd.DataFrame(data, columns=["package", "version", "release", "drop", "support_by"])

df["quarter_drop"] = df["drop"].dt.to_period("Q")
df["quarter_support_by"] = df["support_by"].dt.to_period("Q")

dq_drop = df.set_index(["quarter_drop", "package"]).sort_index()
dq_support_by = df.set_index(["quarter_support_by", "package"]).sort_index()


print("Saving support schedule to schedule.md")


def pad_table(table):
    rows = [[el.strip() for el in row.split("|")] for row in table]
    col_widths = [max(map(len, column)) for column in zip(*rows)]
    rows[1] = [
        el if el != "----" else "-" * col_widths[i] for i, el in enumerate(rows[1])
    ]
```

```
    padded_table = []
    for row in rows:
        line = ""
        for entry, width in zip(row, col_widths):
            if not width:
                continue
            line += f"| {str.ljust(entry, width)} "
        line += f"|"
        padded_table.append(line)

    return padded_table


def make_table(sub):
    table = []
    table.append("|    |    |    |")
    table.append("|----|----|----|")
    for package in sorted(set(sub.index.get_level_values(0))):
        vers = sub.loc[[package]]["version"]
        minv, maxv = min(vers), max(vers)
        rels = sub.loc[[package]]["release"]
        rel_min, rel_max = min(rels), max(rels)
        version_range = str(minv) if minv == maxv else f"{minv} to {maxv}"
        rel_range = (
            str(rel_min.strftime("%b %Y"))
            if rel_min == rel_max
            else f"{rel_min.strftime('%b %Y')} and {rel_max.strftime('%b %Y')}"
        )
        table.append(f"|{package:<15}|{version_range:<19}|released {rel_range}|")

    return pad_table(table)


def make_adopt_table(sub):
    table = []
    table.append("|    |    |    |")
    table.append("|----|----|----|")
    for package in sorted(set(sub.index.get_level_values(0))):
        vers = sub.loc[[package]]["version"]
        minv, maxv = min(vers), max(vers)
        support_bys = sub.loc[[package]]["support_by"]
        support_by_min, support_by_max = min(support_bys), max(support_bys)
        version_range = str(minv) if minv == maxv else f"{minv} to {maxv}"
        support_by_range = (
            str(support_by_min.strftime("%b %Y"))
            if support_by_min == support_by_max
            else f"{support_by_min.strftime('%b %Y')} and {support_by_max.strftime('%b %Y')}"
        )
        table.append(f"|{package:<15}|{version_range:<19}|support by {support_by_range}|")

    return pad_table(table)


def make_quarter(quarter, dq_drop, dq_support_by):
```

```
    table = ["#### " + str(quarter).replace("Q", " - Quarter ") + ":\n"]

    # Add new versions adoption schedule if not empty
    if quarter in dq_support_by.index.get_level_values(0):
        table.append("###### Adopt support for:\n")
        adopt_sub = dq_support_by.loc[quarter]
        adopt_table = make_adopt_table(adopt_sub)
        table.extend(adopt_table)

    table.append("\n###### Can drop support for:\n")
    sub = dq_drop.loc[quarter]
    table.extend(make_table(sub))

    return "\n".join(table)


with open("schedule.md", "w") as fh:
    # we collect package 6 month in the past, and drop the first quarter
    # as we might have filtered some of the packages out depending on
    # when we ran the script.
    tb = []
    for quarter in list(sorted(set(dq_drop.index.get_level_values(0))))[1:]:
        tb.append(make_quarter(quarter, dq_drop, dq_support_by))

    fh.write("\n\n".join(tb))
    fh.write("\n")
```

Rejected Ideas

- NEP 29's more lenient 42-month support timeline was originally considered instead of SPEC 0's 36 months, but it was ultimately decided to follow SPEC 0 because it supersedes NEP 29.
- The scope of this PHEP was originally limited to Python version support. However, it was decided that including the upstream package support policy from SPEC 0 would better promote PyHC package interoperability and avoid the need for a future separate PHEP.
- It was considered making this a requirement rather than a recommendation (i.e., using "must" instead of "should" language) but it was decided that this policy is better suited as a recommendation.

Open Issues

There are no remaining open issues.

Footnotes

1. SPEC 0: https://scientific-python.org/specs/spec-0000/
2. NEP 29: https://numpy.org/neps/nep-0029-deprecation_policy.html

Revisions

Revision 1 (pending): Initial draft.

Copyright


```
@techreport(phep3,
  author = {Shawn Polson},
  title  = {PyHC Python Support Policy},
  year = {2024--2025},
  type = {PHEP},
  number = {3},
  doi = {10.5281/zenodo.15692276}
)
```