




1 MontePy: a Python library for reading, editing, and  
2 writing MCNP input files.

3 Micah D. Gale <sup>1</sup>✉, Travis J. Labossiere-Hickman <sup>1</sup>, Brenna A. Carbno<sup>1</sup>,  
4 and Andrew J. Bascom <sup>1</sup>

5 <sup>1</sup> Idaho National Laboratory, USA ✉ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

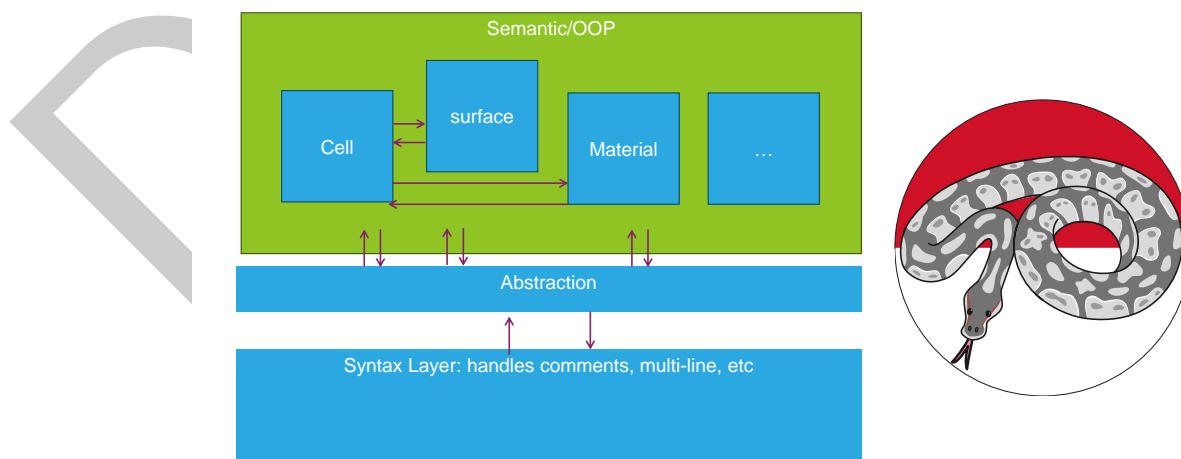
Published: unpublished

License

Authors of papers retain copyright  
and release the work under a  
Creative Commons Attribution 4.0  
International License ([CC BY 4.0](#)).

6 **Summary**

7 The Monte Carlo N-Particle (MCNP) radiation transport code is a highly capable and accurate  
8 code with a long legacy. MCNP uses the Monte Carlo simulation process to simulate the  
9 path of particles (e.g., neutrons, photons, charged particles, etc.), and their interaction with  
10 materials. It is widely used in nuclear engineering, high-energy physics, and other fields. Its  
11 origins in the mid-twentieth century predate many modern software conventions. MCNP users  
12 provide an input file to MCNP, which it then uses to create an internal representation of  
13 the simulation problem. These input files originally had to be stored as punchcard decks,  
14 and the user manual still uses the terminology of cards and decks, despite moving beyond  
15 punchcards. MCNP predates nearly all modern human readable markup or data serialization  
16 languages, such as the extensible Markup Language (XML), the Standard Generalized Markup  
17 Language (SGML), YAML (YAML Ain't Markup Language), and Javascript Object Notation  
18 (JSON). Due to this, MCNP uses an entirely custom defined syntax language for its input,  
19 making off-the-shelf libraries for XML, YAML, and JSON impossible to use for scripting various  
20 operations on MCNP input files ([Kulesza et al., 2022](#)).



**Figure 1:** Diagram of how the different models of MontePy interact to form an Object Oriented Programming (OOP) interface.

21 MCNP simulation problems use three-dimensional constructive solid geometry (CSG). The  
 22 simulation is composed of a series of cells, representing spatial regions in the modeled geometry.  
 23 These cells have assigned densities, and are linked to a material definition. These materials  
 24 define the relative amounts of different isotopes in the material, and can be shared between  
 25 cells. The cell geometry in CSG is defined by a series of Boolean set operations of geometry  
 26 primitives, which are usually quadratic surfaces. For instance, a cylindrical nuclear fuel pellet's  
 27 geometry could be defined as the inside of an axially infinite cylinder with a specified radius,  
 28 and above a bottom plane, and below a top plane.

29 MontePy is a Python library for reading, editing, and writing these MCNP input files. It provides  
 30 an object-oriented programming (OOP) interface for interacting with the simulation problems.  
 31 MontePy does not perform any of its own radiation transport, or neutronics calculations.  
 32 MontePy uses a syntax parser built on top of the Python package for parsers: SLY (Beazly &  
 33 contributors, 2016). This parser builds a concrete syntax tree of the input file. An example  
 34 syntax tree for defining a cell is given in Figure 2. This allows MontePy to parse the input file  
 35 without losing any information or formatting. The MontePy objects that represent MCNP  
 36 objects such as: cells, surfaces, materials, etc., crawl through this syntax tree and update  
 37 their internal values to reflect this tree. These attributes will then be exposed to the user as  
 38 properties. Once a file has been fully read in, all objects will be linked together. The interaction  
 39 between the components of MontePy are shown in Figure 1. For instance a cell object will  
 40 be given a “pointer” to the material object that it is defined to be filled with. When the  
 41 user wants to write their modified model to file this process will be reversed. The objects will  
 42 crawl their syntax tree as necessary to ensure it has the correct value. If a value has changed,  
 43 MontePy will try to match the numerical precision that the user used initially in the input file.

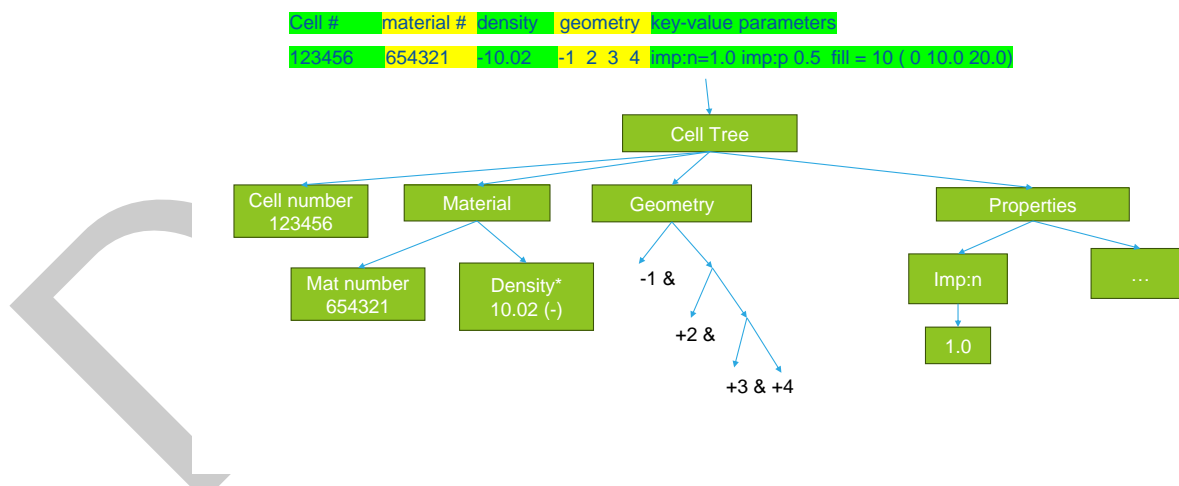


Figure 2: The syntax for defining a cell for MCNP, and the syntax tree that MontePy will extract from it.

44 MontePy is focused on using good software practices to simplify adoption as much as possible.  
 45 MontePy uses industry standard continuous integration and continuous deployment (CI/CD)  
 46 tools to test all changes, and ensure changes meet the standards of MontePy. Currently  
 47 the test suite has over 380 tests, which have over 98% code coverage of the source code.  
 48 This practice significantly reduces the risk of regressions and the introduction of new bugs.  
 49 All software changes must be reviewed by at least one person prior to being accepted for  
 50 deployment to users. MontePy is also written only in Python and only has two dependencies.  
 51 This was intentionally done to make it as easy as possible for a new user to install it, which  
 52 they can do with a single command. The decision to only use Python was made to facilitate

53 the creation of a user community. All end-users should know how to write some Python, so  
54 any user could, with some guidance, become a developer.

## 55 Statement of Need

56 MCNP is a popular Monte Carlo radiation transport code. It has nearly unmatched capabilities  
57 in its physics modeling, and its support for 37 different particle types (Kulesza et al., 2022).  
58 However, due to the fact that MCNP uses a custom syntax, it requires referencing other  
59 objects (e.g., a cell referencing a material) by their number, and tendency of radiation transport  
60 simulations to be complex, working with MCNP input files can be tedious and error-prone.  
61 This issue is well suited for automation.

62 Current packages do exist for automating the generation of new MCNP input files. One such  
63 package is the Advanced Reactor Modeling Interface (ARMI). ARMI is a modular open-source  
64 framework for coupling multiple simulation codes. A closed-source MCNP plugin for ARMI  
65 does exist. ARMI is more focused on making its internal model of a nuclear reactor fit in an  
66 MCNP input file, rather than being able to open and parse an arbitrary MCNP model (Touran  
67 et al., 2017). This is indicative of the various tools that have been created for working with  
68 MCNP in the past. They tend to be purpose-built for a specific problem, or class of problems,  
69 and are not easily generalized. MCNP input models could also be created with a templating  
70 engine, like the Workflow Template and Toolkit System (WATTS) (P. K. Romano et al., 2022).  
71 This though requires the user to create a template from the set of problems they plan to model.  
72 This would be well suited for a sensitivity study where many very similar simulations are run,  
73 but not for making large edits to an input file or making a single problem from scratch. PyNE:  
74 the Nuclear Engineering Toolkit offers some similar capabilities for input generation.

75 PyNE can create MCNP input files for specific features and extract some data from MCNP  
76 output files. However, its full capabilities extend far beyond interfacing with MCNP. PyNE can  
77 simplify material creations, analyses of cross section data, transmutations of complex systems,  
78 and interfacing with other common nuclear engineering software and data formats (?). PyNE  
79 is an excellent companion tool to MontePy.

80 All of these previous solutions were incomplete in one way or another. Neither were able to  
81 read in a previous MCNP input file and edit it in a general manner. In addition, WATTS does  
82 not have any fundamental understanding of what fields are for a user provided MCNP template.  
83 The same is true for a myriad of application-specific industry tools that are tailor-made for  
84 specific problems. There is a clear need for an object-oriented interface to these files that can  
85 both “understand” the input, and read and edit the files. This sort of model interface has  
86 been present for years in the Python API for the Monte Carlo code, OpenMC (P. Romano  
87 et al., 2015)<sup>1</sup>. Since its incorporation in the code, this interface has become by far the most  
88 dominant user interface for that code, as opposed to manual editing of the XML input files.

89 Ideally this object-oriented interface should be in Python as it is such a prolific language,  
90 especially among novice and intermediate programmers. A few such libraries do exist: MCNPpy,  
91 mckit, and others discussed later. MCNPpy is a Python wrapper for a java engine that can  
92 read, edit, and write MCNP input files. It can “understand” MCNP inputs, or as the authors  
93 put it, it has a “metamodel” for MCNP (Kowal et al., 2023). Having a library written in  
94 another language than what the user is used to, introduces another barrier to converting a  
95 user into a developer. This could present a serious barrier to developing a thriving user and  
96 developer community for this open source software. It does not appear that MCNPpy has any  
97 automated testing suite at this time, and so there is no guarantee that it will actually perform  
98 the functions it claims to. In addition it imposes additional formatting requirements on an  
99 input file that is read, beyond what MCNP requires (Kulesza et al., 2022). Mckit on the other  
100 hand is written primarily in python, and does use automated testing. Unfortunately the existing  
101 documentation is difficult to access, incomplete, and primarily in russian. It was difficult to  
102 assess the state of this project due to this. It appeared that mckit is more of a functional

103 programming style library, rather than an object-oriented programming style (Rodionov &  
104 Portnov, 2024). MontePy provides all of these listed capabilities, while also being written  
105 purely in Python, and avoiding this barrier to forming a thriving open source community.

#### 106 **TODO PyNE, mckit**

107 The authors did try to find as many Open Source Python libraries that had at least some  
108 overlap with MontePy. This was not an exhaustive search, but should cover many such libraries.  
109 Given the number of libraries found the following lists will simply be an attempt to categorize  
110 these libraries.

111 The first group of libraries are those which attempt to have a read, edit, write capability for  
112 MCNP input files. These all do not fully parse the inputs as they do not use context-free  
113 parsers, and are generally feature limited, and may lack sufficient documentation. These libraries  
114 are:

- 115     ▪ numjuggler (Travleev et al., 2022)
- 116     ▪ MCNP Input Reader (Mariano, 2022)
- 117     ▪ mctools (Laghi, 2023)
- 118     ▪ mc-tools (Batkov et al., 2024)
- 119     ▪ PyMCNP (Persaud et al., 2024)

120 There are even more tools that specialize in input templating and generation. These are clearly  
121 not complete alternatives as they lack the ability to read MCNP input files. These libraries are:

- 122     ▪ CardSharpForMCNP (Pacific Northwest National Laboratory, 2025)
- 123     ▪ wig (Hagen, 2021)
- 124     ▪ Plugin-MCNP [for Funz] (Richet, 2023)
- 125     ▪ GDNF (niess, 2018)
- 126     ▪ map-stp (Portnov, 2024)
- 127     ▪ MCNP Input Generator (ikarino, 2021)
- 128     ▪ Neutronics Material Maker (Shimwell et al., 2024)

129 There are also libraries that specialize in parsing an MCNP input file in order to convert the  
130 model to be an input for another program:

- 131     ▪ MCNP Conversion tools for OpenMC (Paul Romano et al., 2024)
- 132     ▪ t4\_geom\_convert (Mancusi, 2024)

133 There are also libraries that have to parse MCNP inputs to some extent as they provide MCNP  
134 syntax highlighting support for various text editors:

- 135     ▪ MCNP-syntax-highlighting (Turkoglu, 2018)
- 136     ▪ NPP\_MCNP\_Plugin (Marcinkevicius, 2025)
- 137     ▪ vscode\_mcnf (Repositony, 2024)

138 Finally there are the libraries that have been purpose built for working with and automating a  
139 specific type of MCNP models:

- 140     ▪ BEMP\_Thesis (Galdon, 2024)
- 141     ▪ MCNP6-HPGe\_Detector\_simulation (Hung, 2023)
- 142     ▪ rodcal-mcnf (Park, 2021)

143 MontePy is currently targeting two primary communities. First, Nuclear Engineers with  
144 moderate Python experience as a user base. The goal is to get these users to use the interface  
145 to remove the tedium from their work when they need to make some modification to their  
146 model. In addition, these users can use MontePy to quickly interrogate, and retrieve information  
147 from their models in order to validate them, or to just answer some questions they had about  
148 them. The other target user is the Nuclear Engineer developer, making automation tools.  
149 Many nuclear engineering departments have a large MCNP model that they need to frequently  
150 update. For instance, the authors of MontePy use a model of the Advanced Test Reactor

151 (Campbell et al., 2021) for their work on a daily basis. This large and complex model needs to  
152 be updated every reactor cycle with the new fuel compositions, the specific control element  
153 configurations, etc. Their department does have an automation tool that relies heavily on  
154 template-like use of regular expressions. This tool will fail to run if the model is modified in a  
155 way that is allowed by MCNP, but which the tool cannot handle. This tool is a prime example  
156 of a real-life case where MontePy could be applied to improve the workflow and increase  
157 robustness.

## 158 Status of MontePy

159 As of MontePy 0.5.4 many of the most commonly used MCNP inputs (cards) are supported.  
160 These include:

- 161 ■ Cells, which are the base of an MCNP geometry and contain a material and a CSG  
162 geometry definition.
  - 163 – Cell modifier inputs:
    - 164 \* IMP inputs which specify a cell's importance for variance reduction, and other  
165 uses.
    - 166 \* FILL, LAT, and U inputs which are used for defining universes, and filling cells  
167 with those universes.
    - 168 \* VOL input which specifies the volume for a cell
- 169 ■ Surface inputs, which are used to define the primitive surfaces used. All surfaces are  
170 supported at a basic level. The following surface types are supported in a semantic way  
171 where the constants are tied to their geometric meaning:
  - 172 – PX, PY, and PZ surfaces, which are planes perpendicular to a specific axis.
  - 173 – CX, CY, and CZ surfaces, which are cylinders parallel to a specific axis and centered  
174 at the origin.
  - 175 – C\X, C\Y, and C\Z surfaces, which are cylinders parallel to a specific axis, and not  
176 centered at the origin.
- 177 ■ M inputs, which define the composition of a specific material.
- 178 ■ MT inputs, which define a thermal scattering law to use for a specific material.
- 179 ■ mode inputs, which define which particle types to run in the simulation.
- 180 ■ TR inputs, which define a geometry transformation.

181 MontePy does not support reading output files, and there are no current plans to add such  
182 support. First, MCNP is export controlled software, with a publicly released manual. MontePy  
183 was based solely on this manual. It does not document the formatting of the MCNP output  
184 files, so this feature is not included. Secondly, there is already an Open-Source tool available  
185 to read some MCNP output files, MCNPtools. This is a Python wrapper for a C++ tool to  
186 read meshtal, and mctal files output by MCNP (Bates et al., 2022). So for the time being, to  
187 avoid scope creep, the core MontePy developers will not be adding support for output files to  
188 allow development to focus on supporting more input features.

## 189 Future Work

190 MCNP supports over 140 different inputs (cards). For almost all of the remaining input types  
191 that MontePy doesn't support the information from the input is still available to the user.  
192 The next planned release at the time of publication is version 1.0.0. This new release is  
193 significant redesign of the material definition interface, making the material interface much  
194 more user-friendly. The exceptions are those inputs with syntax that conflicts with the rest  
195 of MCNP, which need to be handled specifically on their own. Adding more object-oriented  
196 support for all of these inputs is an ongoing project. Development is primarily prioritized by  
197 most commonly used inputs. Finally, MontePy and OpenMC's Python interface have many  
198 similar features. Harmonizing MontePy and OpenMC to be intercompatible would unlock

199 a whole new set of possibilities. It would then be possible to translate OpenMC models to  
200 MCNP, and vice versa, which would be ideal for code-to-code comparisons.

## 201 Acknowledgments

202 Work supported through the Advanced Fuels Campaign (AFC) under DOE Idaho Operations  
203 Office Contract DE-AC07-05ID14517. The authors wish to thank the U.S. Department of  
204 Energy Office of Isotope R&D and Production for their vital and continued support and funding  
205 of the Co-60 program at INL under Contract No. DE-AC07-05ID14517. Co-60 is sold by the  
206 National Isotope Development Center (NIDC). Quotes on Co-60 can be obtained from NIDC  
207 at [www.isotopes.gov/products/cobalt](http://www.isotopes.gov/products/cobalt). This research made use of Idaho National Laboratory's  
208 High Performance Computing systems located at the Collaborative Computing Center and  
209 supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear  
210 Science User Facilities under Contract No. DE-AC07-05ID14517.

## 211 References

- 212 Bates, C. R., Bolding, S. R., Josey, C. J., Kulesza, J. A., Solomon, C. J., Jr., & Zukaitis, A. J.  
213 (2022). *The MCNPTools package: Installation and use* (Report LA-UR-22-28935). Los  
214 Alamos National Laboratory. <https://doi.org/10.2172/1884737>
- 215 Batkov, K., Borghi, N., Furutaka, K., Ansell, S., & Vezhlev, E. (2024). *Mc-tools* (Version  
216 1.0.post1). <https://github.com/kbat/mc-tools>
- 217 Beazly, D., & contributors, S. (2016). *SLY (sly lex yacc)* [Online Multimedia]. <https://sly.readthedocs.io/en/latest/>
- 219 Campbell, J., Marshall, F., & Longhurst, G. (2021). *Advanced test reactor user guide* (Report  
220 INL/EXT-21-64328). Idaho National Laboratory. <https://doi.org/10.2172/1826354>
- 221 Galdon, M. (2024). *BEMP\_thesis* (Version 1.0.0). [https://github.com/mgaldon17/BEMP\\_Thesis](https://github.com/mgaldon17/BEMP_Thesis)
- 223 Hagen, A. (2021). *Wig*. <https://github.com/alexhagen/wig>
- 224 Hung, B. T. (2023). *MCNP6-HPGE\_detector\_simulation*. [https://github.com/hungbt1908/MCNP6-HPGE\\_Detector\\_Simulation](https://github.com/hungbt1908/MCNP6-HPGE_Detector_Simulation)
- 226 ikarino. (2021). *MCNP input generator*. [https://github.com/ikarino/mcnp\\_input\\_generator](https://github.com/ikarino/mcnp_input_generator)
- 227 Kowal, P. J., Blake, C. E., Dominesey, K. A., Lefebvre, R. A., Brown, F. B., & Ji, W.  
228 (2023). Enhancing monte carlo workflows for nuclear reactor analysis with metamodel-  
229 driven modeling [Journal Article]. *Nuclear Science and Engineering*, 197(8). <https://doi.org/10.1080/00295639.2022.2153617>
- 231 Kulesza, J., Adams, T., Armstrong, J. C., Bolding, S. R., Brown, F. B., Bull, J. S., Burke, T. P.,  
232 Clark, A. R., Forster, R. A. I., Giron, J. F., Grieve, T. S., Josey, C. J., Martz, R. L., McKinney,  
233 G. W., Pearson, E. J., Rising, M. E., Solomon, C. J. Jr., Swaminarayan, S., Trahan, T. J.,  
234 ... Zukaitis, A. J. (2022). *MCNP code version 6.3.0 theory & user manual* (Report LA-UR-  
235 22-30006, Rev. 1). Los Alamos National Laboratory. <https://doi.org/10.2172/1889957>
- 236 Laghi, D. (2023). *Mctools*. <https://github.com/dodu94/mctools>
- 237 Mancusi, D. (2024). *t4\_geom\_convert* (Version 1.1.2). [https://github.com/arekfu/t4\\_geom\\_convert](https://github.com/arekfu/t4_geom_convert)
- 239 Marcinkevicius, B. (2025). *NPP\_MCNP\_plugin*. [https://github.com/kordusas/npp\\_mcnp\\_plugin](https://github.com/kordusas/npp_mcnp_plugin)
- 241 Mariano, G. (2022). *MCNP input reader* (Version 0.2.1). <https://github.com/>

- 242 [ENEА-Fusion-Neutronics/MCNP-Input-Reader](#)
- 243 niess. (2018). *GДNP*. <https://github.com/niess/gdnp>
- 244 Pacific Northwest National Laboratory. (2025). *CardSharpForMCNP* (Version 1.4.2). <https://github.com/pnnl/CardSharpForMCNP>
- 245
- 246 Park, P. (2021). *Rodcal-mcnp*. <https://github.com/patrickpark910/rodcal-mcnp>
- 247 Persaud, A., Unzueta, M. A., Surry, E. K., & Parsons, A. M. (2024). Python-based software  
248 tools for MCNP. *2024 IEEE Nuclear Science Symposium (NSS), Medical Imaging Conference*  
249 *(MIC) and Room Temperature Semiconductor Detector Conference (RTSD)*, 1–1. <https://doi.org/10.1109/NSS/MIC/RTSD57108.2024.10655039>
- 250
- 251 Portnov, D. (2024). *Map-stp*. <https://github.com/MC-kit/map-stp>
- 252 Repository. (2024). *Vscode\_mcnp*. [https://github.com/repository/vscode\\_mcnp](https://github.com/repository/vscode_mcnp)
- 253 Richet, Y. (2023). *Funz plugin-MCNP* (Version 1.16-0). <https://github.com/Funz/plugin-MCNP>
- 254
- 255 Rodionov, R., & Portnov, D. (2024). *Mckit* (Version 0.8.3). <https://github.com/MC-kit/mckit>
- 256 Romano, P. K., Stauff, N. E., Ooi, Z. J., Miao, Y., Lund, A., & Zou, L. (2022). WATTS:  
257 Workflow and template toolkit for simulation [Journal Article]. *Journal of Open Source*  
258 *Software*, 7(79). <https://doi.org/10.21105/joss.04735>
- 259 Romano, P., Horelik, N. E., Herman, B. R., Nelson, A. G., Forget, B., & Smith, K. (2015).  
260 OpenMC: A state-of-the-art monte carlo code for research and development [Journal  
261 Article]. *Annals of Nuclear Energy*, 82. <https://doi.org/10.1016/j.anucene.2014.07.048>
- 262 Romano, Paul, Li, K., Shriwise, P., & Valderrama, J. (2024). *MCNP conversion tools for*  
263 *OpenMC*. [https://github.com/openmc-dev/openmc\\_mcnp\\_adapter](https://github.com/openmc-dev/openmc_mcnp_adapter)
- 264 Shimwell, J., Billingsley, J., Buendia, C., & Neutronics Material Material Contributors. (2024).  
265 *Neutronics material maker* (Version 1.2.1). [https://github.com/fusion-energy/neutronics\\_](https://github.com/fusion-energy/neutronics_material_maker)  
266 [material\\_maker](https://github.com/fusion-energy/neutronics_material_maker)
- 267 Touran, N., Gilleland, J., Malmgren, G., Whitmer, C., & Gates, W. H. (2017). Computational  
268 tools for the integrated design of advanced nuclear reactors [Journal Article]. *Engineering*,  
269 3(4). <https://doi.org/10.1016/J.ENG.2017.04.016>
- 270 Travleev, A., Previti, A., & Portnov, D. (2022). *Numjuggler* (Version 2.42.36). <https://github.com/travleev/numjuggler>
- 271
- 272 Turkoglu, D. (2018). *MCNP-syntax-highlighting*. [https://github.com/danyalturkoglu/](https://github.com/danyalturkoglu/MCNP-syntax-highlighting)  
273 [MCNP-syntax-highlighting](https://github.com/danyalturkoglu/MCNP-syntax-highlighting)