




# mathlib: A Scala package for readable, verifiable and sustainable simulations of formal theory

Mark Blokpoel <sup>1</sup>

<sup>1</sup> Donders Institute for Brain, Cognition, and Behaviour, Radboud University, The Netherlands

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

## Software

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

Editor: [Daniel S. Katz](#) 

## Reviewers:

- [@bzz](#)
- [@larkz](#)

Submitted: 24 August 2023

Published: unpublished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](#)).

## Summary

Formal theory and computational modeling are critical in cognitive science and psychology. These methods allow scientists to ‘conceptually analyze, specify, and formalize intuitions that otherwise remain unexamined’ ([Guest & Martin, 2021](#)). They make otherwise underspecified theories precise and open for critical reflection ([van Rooij & Baggio, 2021](#)). A theory can be formally specified in a computational model using mathematical concepts such as set theory, graph theory and probability theory. The specification is often followed by analysis to understand precisely what assumptions and consequences the formal theory entails. An important method of analysis is computer simulation, which allows scientists to explore complex model behaviours and derive predictions that cannot (easily) be analytically derived.

`mathlib` is a library for Scala ([Odersky, 2008](#)) supporting functional programming that resembles mathematical expressions such as set theory, graph theory and probability theory. This library was developed to complement the theory development method outlined in the open education book *Theoretical modeling for cognitive science and psychology* by Blokpoel & van Rooij ([2021](#)).

The goal of this library is to facilitate users to implement simulations of their formal theories. Code written in Scala using `mathlib` is:

- easy to read, because `mathlib` syntax closely resembles mathematical notation
- easy to verify, by proving that the code exactly implements the theoretical model (or not)
- easy to sustain, because newer versions of Scala and `mathlib` can run old code (backwards compatibility)

## Statement of need

Writing code is not easy, writing code for which we can know that it computes what the specification (i.e., the formal theory) states is even harder. This can be facilitated by having a programming language where the syntax and semantics closely matches that of the specification. Since formal theories are specified using mathematical notation [Guest & Martin \(2021\)](#), functional programming languages bring a lot to the table in terms of syntactic and semantic resemblance to mathematical concepts and notation. `mathlib` adds mathematical concepts and notation to the functional programming language Scala ([Odersky, 2008](#)), specifically in the current version: set theory and graph theory. `mathlib` differs from other libraries in that it focuses on usability and transparency, whereas other libraries focus on computational expressiveness at the cost of accessibility and transparency.

Given the important role of theory and computer simulations, it is important that scholars can verify that the code does what the authors intent it to do. We provide an example below of a formalization of subset choice and its implementation in Scala using `mathlib` to illustrate this.

## 41 Subset choice

42 *Input:* A set of items  $I$ , a value function for single items  $v : I \rightarrow \mathbb{Z}$  and a binary value function  
43 for pairs of items  $b : I \times I \rightarrow \mathbb{Z}$ .

44 *Output:* A subset of items  $I' \subseteq I$  (or  $I' \in \mathcal{P}(I)$ ) that maximizes the combined value of the  
45 selected items according, i.e.,  $\arg \max_{I' \in \mathcal{P}(I)} \sum_{i \in I'} v(i) + \sum_{i,j \in I'} b(i,j)$ .

**type** Item = String

```
def subsetChoice(
  items: Set[Item],
  v: (Item => Double),
  b: ((Item, Item) => Double)
): Set[Set[Item]] = {

  def value(subset: Set[Item]): Double =
    sum(subset, v) + sum(subset.uniquePairs, b)

  argMax(powerset(items), value)
}
```

46 Finally, mathlib and Scala are designed to be backwards compatible, i.e., to run on future  
47 systems and future execution contexts (e.g., operating systems). Many programming con-  
48 tributions in academia are lost because of incompatibility issues between versions and newer  
49 operating systems, etc. This sometimes affects contributions within a short timeframe, and  
50 means that it is incredibly hard for anyone to run or verify the code and simulation results.

## 51 Resources

- 52 ■ Github repository: <https://github.com/markblokpoel/mathlib>
- 53 ■ Website: <https://markblokpoel.github.io/mathlib>
- 54 ■ Scaladoc: <https://markblokpoel.github.io/mathlib/scaladoc>
- 55 ■ Tutorials: <https://github.com/markblokpoel/mathlib-examples>

## 56 Acknowledgements

57 We thank the Computational Cognitive Science group at the Donders Center for Cognition  
58 (Nijmegen, The Netherlands) for useful discussions and feedback, in particular, Laura van de  
59 Braak, Olivia Guest and Iris van Rooij.

60 This project was supported by Netherlands Organization for Scientific Research Gravitation  
61 Grant of the Language in Interaction consortium 024.001.006, the Radboud School for Artificial  
62 Intelligence and the Donders Institute, Donders Center for Cognition.

## 63 References

- 64 Blokpoel, M., & van Rooij, I. (2021). *Theoretical modeling for cognitive science and psychology*.  
65 <https://computationalcognitivescience.github.io/lovelace/>
- 66 Guest, O., & Martin, A. E. (2021). How computational modeling can force theory building in  
67 psychological science. *Perspectives on Psychological Science*, 16. <https://doi.org/10.1177/1745691620970585>
- 68 Marr, D. (1982). *Vision: A computational investigation into the human representation and*  
69 *processing of visual information*. W.H. Freeman, San Francisco, CA.

- <sup>71</sup> Odersky, M. (2008). *Programming in scala*. Mountain View, California: Artima.
- <sup>72</sup> van Rooij, I., & Baggio, G. (2021). Theory before the test: How to build high-verisimilitude  
<sup>73</sup> explanatory theories in psychological science. *Perspectives on Psychological Science*, 16.  
<sup>74</sup> <https://doi.org/10.1177/1745691620970604>

DRAFT