BAPL3 Design Choice Doc

**Table of Contents**

[References 3](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.og7ujgo9hggn)

[What design choices have you made for your language or plan on using? 4](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.2vwebi4u9zqp)

[Language Design Choices by: [your name] 4](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.74peeg317cav)

[Learning 4](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.wd2lk4eh31zp)

[Elegance 4](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.phyokv915xne)

[Experiments 4](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.xtytrujlf4q3)

[Aiming for the Sky Goals 4](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.o4zwtoyfjq6k)

[Safety 4](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.afv9cx69fj0q)

[EXAMPLE OF HOW YOU MIGHT ENTER YOUR INFORMATION ABOUT YOUR DESIGN CHOICES FOR YOUR LANGUAGE 5](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.hn3yyznuko5)

[Language design choices by: John Doe 5](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.i2dcosogltu2)

[Learning 5](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.1pecfzdsmwp0)

[Avoid Rabbit Holes, Focus on Class Goals First 5](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.j8l19logejz2)

[Not Aiming for Performance 5](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.5ryc679anbug)

[Elegance 5](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.357nrzk8dbxa)

[Experiments 5](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.kaz7ey37rdli)

[Aiming for the Sky Goals 5](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.bq5glvkcmppx)

[Safety 6](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.tygkxc2ehs0w)

[Type System 6](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.r1rdqj8m8n2q)

[Borrow Checking/Lifetime System 6](https://docs.google.com/document/d/1Snx1DdanRnBXlFnAYLtJvA9UbC1b6sJM3pWhWIlNFY0/edit#heading=h.3froucax72ix)

**References**

* Fun inspirations here <https://esolangs.org/wiki/Language_list>

* Full source code for the language at the end of each lesson is available on GitHub for reference, if you have issues with things not working it's helpful to compare:  <https://github.com/classpert/bapl2-class-language>

**What design choices have you made for your language or plan on using?**

Copy the following section and fill it with your information:

|  |
| --- |
| **Language Design Choices by: [Martin Friesen]**  Learning   * PEGs, LPeg interacting with Lua in particular, to produce ASTs * Compiling ASTs to bytecode (jumps and lexical scoping especially) * Executing bytecode in a VM (stack, memory, functionality) * Design choices (namespaces, function/operator overloading, recursion, closures) * Recursion, recursion, recursion   Elegance   * Lua feels like a VisiCalc of languages * LPeg feels like a VisiCalc of PEGs * Could there be a VisiCalc of DSLs?   Experiments   * implemented 8 Queens in my language (felt like BAPL came full circle)   + many Queens died and some of my juicy bugs suffered horrible deaths (RIP)   + used “unless” and string interpolation - take that Lua 😊 * deviated from class implementation somewhat to strive for error message precision   + feel it is important for DSLs to be accepted by users   + tested my understanding (or not) of the principles taught * thanks to Matt for inspiring me to implement relational operators mathematically   + a < b < c   =   a < b  and  b < c * making “and” higher priority than “or” reduced need for parentheses in some cases * added a zone type e.g. 1:10 early on (tested understanding and boosted enjoyment)   + pleasant surprises     - served for multiple return values (ok, maybe just amusement)     - just by adding size attribute, zone inherited all array features!       * hmmm… inheritance by attributes?   + late realization: may serve for substr, e.g.     - “string”[3 : -1] → “ring”     - or, “string” ~ 3:-1 → “ring” with ~ operator     - unary ~ to give string size     - given 1 based arrays - use a[0] to give array size? * added a string type and concatenation early on   + a little went a long way to help debugging and it boosted enjoyment   + adding interpolation felt like a step toward multiple return values * added destructuring of arrays and associations (felt like another step toward MRVs) * added lowering of for loops to while loops in the parser (no compiler/vm changes needed)   + reusing proven stuff seemed like cheating (but ftw) * learning a lot from peer’s superior approaches in many areas   Aiming for the Sky Goals   * redo project ground-up targeting a DSL vs general level   + if a Lua/LPeg alternative course offered that, I would have picked it * wrap in and around Lua/LPeg/C++ and apply to my professional domain * correctly answer a programmer joke/riddle for once   Safety   * my for loops do nothing given a 0 step size; while 1 { } seems clearer for infinite loops |

|  |
| --- |
| EXAMPLE OF HOW YOU MIGHT ENTER YOUR INFORMATION ABOUT YOUR DESIGN CHOICES FOR YOUR LANGUAGE  Language design choices by: John Doe  Learning  My main goal is to learn how to make a language.  Avoid Rabbit Holes, Focus on Class Goals First  Make sure to submit a language that passes the class requirements and has all the basic functions desired. An overly complicated implementation will result in frustration and failure to learn important concepts.  Not Aiming for Performance  A focus on performance would be a distraction from the other things I need to learn right now.  Elegance   * Simplicity. * Consistency. * Try to avoid 'more than one way to do something.' * Aim for Lua-like ease of reading, which is made possible by its keywords for logic conditionals and blocks. * Attempt Lua style of no separators between statements.   Experiments  Try to see if it's possible to do some things that I've wanted in other languages.     * Allow single spaces in identifiers. * Digit grouping through commas: this conflicts with parameter lists, is there a way around it? * Generic base numbers with "<number> b<base of number>" format.   Aiming for the Sky Goals   * Mixing dynamic and static typing. * Multiple return values. * First-class functions. * Rust-style lifetime-guided memory management.   Safety  Type System  It's been a lot of fun to implement this, and I enjoy having the additional checks on errors.  It's surprising to me how natural implementing it has been.  Borrow Checking/Lifetime System  Curious if this can be done in a less confusing way that Rust does it. |