## Parens are Weird

While LillyPy representation may look a lot like text, it’s really hierarchical, like the underlying language itself. The place that will look the weirdest at first is when you’re typing parentheses, brackets and braces. You won’t see much difference when you’re typing a new expression, but when you start editing existing text, the differences will start to appear.

### Different Kinds of Parens

One difference that you’ll see immediately when you start using LillyPy; is that parentheses, brackets, and braces, do not have the same uniform appearance that they do in text. There are multiple kinds of each, and while you type them with the same keys, once typed, they each have different meanings. Unlike syntax highlighting in an IDE, they indicate an identity that generally persists across edits.

Parentheses:

1. Arithmetic auto-parens – Expressions in LillyPy, while they may look like plain text, are actually a hierarchy of operations. In a text editor, if you type “3 \* ” and then paste “2 + 2” after the “\*”, you’ll get “3 \* 2 + 2”. In LillyPy, you’ll get “3 \* (2 + 2)”. The parentheses are actually part of the “+” operation. If you then grabbed the “(2 + 2)” back out of that expression and pasted it after “3 + “, the parens would disappear and you would get “3 + 2 + 2”.
2. Tuple parens – If you type a comma inside of a paren, you will get a tuple.
3. Call parens – If you type a paren after an identifier, LillyPy will produce an italic-looking paren referred to as a call paren.
4. Grouping parens – Used for emphasis but not syntactically necessary, around arithmetic grouping that already conforms to expression rules for precedence and associativity.

Brackets:

1. Lists brackets – Surround a list or list comprehension
2. Subscript brackets – Surround an array reference or slice

Braces:

1. Dict/Set – Surround a dictionary or set constant

Constructive Parens/Brackets/Braces

1. Temporary open paren/bracket/brace to help type one of the above types.

LillyPy inserts a “constructive” paren, bracket or brace when it is not yet sure where the matching end will go, or in some cases, what kind of paren you will be typing. Until these are closed, it’s best to think of them as placeholders, as opposed to “real” parens, because they don’t balance like normal parens. For example, if you typed fn(1\*2 +3), and then inserted an open paren before the 3, you would see:

fn(1\*(2+3)

In a text editor, you could type an end paren at the end of the expression to balance it. In LillyPy, you could do that if all the parens were of the normal arithmetic/grouping type. However, here they are part of a function call, which are not interchangeable.

Another aspect of parenthesis to you will need to become accustomed, is building them left-to-right. None of these restrictions should lead to additional keystrokes or mouse movementsThe LillyPy representation is hierarchical to is need to type it after the 3, because. In LillyPy, cursor parens don’t count until they are closed, and you can’t close it by typing an end paren at the end of the expression

(It would be possible to allow this, and probably worth doing. That is, make the rule that the cursor paren can cause rematching of all of the parens in the expression between it and the matching paren. This would only hold for arithmetic parens, not braces, brackets, calls, or tuples.)

The rule of thumb with parens is to type them left to right. If you want to enclose something, type the left one first, then the right. To save effort, you don’t actually need to move the cursor. You can also use the ^) key to toggle between all of the possible matching end positions). Likewise, deleting the right paren re-opens the paren, whereas deleting the right paren removes both.

Backspacing has a different effect on an end paren/bracket/brace than it does on a start-paren.