Variables

A variable, in Julia, is a name associated (or bound) to a value. It's useful when you want to store a value (that you obtained after some math, for example) for later use. For example:

```
# Assign the value 10 to the variable x
julia> x = 10
10

# Doing math with x's value
julia> x + 1
11

# Reassign x's value
julia> x = 1 + 1
2

# You can assign values of other types, like strings of text
julia> x = "Hello World!"
"Hello World!"
```

Julia provides an extremely flexible system for naming variables. Variable names are case-sensitive, and have no semantic meaning (that is, the language will not treat variables differently based on their names).

```
julia> x = 1.0
1.0

julia> y = -3
-3

julia> Z = "My string"
"My string"

julia> customary_phrase = "Hello world!"
"Hello world!"

julia> UniversalDeclarationOfHumanRightsStart = "人人生而自由，在尊严和权利上一律平等。"
"人人生而自由，在尊严和权利上一律平等。"
```

Unicode names (in UTF-8 encoding) are allowed:

```
julia> δ = 0.00001
1.0e-5

julia> 안녕하세요 = "Hello"
"Hello"
```

In the Julia REPL and several other Julia editing environments, you can type many Unicode math symbols by typing the backslashed LaTeX symbol name followed by tab. For example, the variable name `δ` can be entered by typing `\delta`-*tab*, or even `α̂₂` by `\alpha`-*tab*-`\hat`- *tab*-`\_2`-*tab*. (If you find a symbol somewhere, e.g. in someone else's code, that you don't know how to type, the REPL help will tell you: just type `?` and then paste the symbol.)

Julia will even let you redefine built-in constants and functions if needed (although this is not recommended to avoid potential confusions):

```
julia> pi = 3
3

julia> pi
3

julia> sqrt = 4
4
```

However, if you try to redefine a built-in constant or function already in use, Julia will give you an error:

```
julia> pi
π = 3.1415926535897...

julia> pi = 3
ERROR: cannot assign a value to variable MathConstants.pi from module Main

julia> sqrt(100)
10.0

julia> sqrt = 4
ERROR: cannot assign a value to variable Base.sqrt from module Main
```

Allowed Variable Names

Variable names must begin with a letter (A-Z or a-z), underscore, or a subset of Unicode code points greater than 00A0; in particular, Unicode character categories Lu/Ll/Lt/Lm/Lo/Nl (letters), Sc/So (currency and other symbols), and a few other letter-like characters (e.g. a subset of the Sm math symbols) are allowed. Subsequent characters may also include ! and digits (0-9 and other characters in categories Nd/No), as well as other Unicode code points: diacritics and other modifying marks (categories Mn/Mc/Me/Sk), some punctuation connectors (category Pc), primes, and a few other characters.

Operators like `+` are also valid identifiers, but are parsed specially. In some contexts, operators can be used just like variables; for example `(+)` refers to the addition function, and `(+) = f` will reassign it. Most of the Unicode infix operators (in category Sm), such as `⊕`, are parsed as infix operators and are available for user-defined methods (e.g. you can use `const ⊗ = kron` to define `⊗` as an infix Kronecker product). Operators can also be suffixed with modifying marks, primes, and sub/superscripts, e.g. `+̂ₐ″` is parsed as an infix operator with the same precedence as `+`.

The only explicitly disallowed names for variables are the names of the built-in Keywords:

```
julia> else = false
ERROR: syntax: unexpected "else"

julia> try = "No"
ERROR: syntax: unexpected "="
```

Some Unicode characters are considered to be equivalent in identifiers. Different ways of entering Unicode combining characters (e.g., accents) are treated as equivalent (specifically, Julia identifiers are NFC-normalized). The Unicode characters `ɛ` (U+025B: Latin small letter open e) and `µ` (U+00B5: micro sign) are treated as equivalent to the corresponding Greek letters, because the former are easily accessible via some input methods.

Stylistic Conventions

While Julia imposes few restrictions on valid names, it has become useful to adopt the following conventions:

- Names of variables are in lower case.
- Word separation can be indicated by underscores (`'_'`), but use of underscores is discouraged unless the name would be hard to read otherwise.
- Names of `Type`s and `Module`s begin with a capital letter and word separation is shown with upper camel case instead of underscores.
- Names of `function`s and `macro`s are in lower case, without underscores.

- Functions that write to their arguments have names that end in `!`. These are sometimes called "mutating" or "in-place" functions because they are intended to produce changes in their arguments after the function is called, not just return a value.

For more information about stylistic conventions, see the Style Guide.

« Getting Started Integers and Floating-Point Numbers »