Photo by Michael Niessl (https://unsplash.com/@antipodos) on Unsplash (https://unsplash.com)

Variables and Types

3 minute read

In this lesson you will learn what are variables and how to perform simple mathematical operations. Furthermore we will deal with the concept of “types” and their role in Julia.

Variables

If we want to really simplify what a program is and what it does, we may say that a program is a series of instructions to be performed on various kind of data. Such data may be the name of a place, the height of a person or a series of measurements made by a scientist. In programming, when we want to “label” a piece of information we give it a name and call it a **variable**, so we may say that a variable is a box which contains some kind of data.

In Julia, when we want to assign some data to a variable, we do it by typing:

```
my_name = "Aurelio"
my_favourite_number = 42
my_favourite_pie = 3.1415
```

Here `my_name` contains a *string*, which is a piece of text, while `my_favourite_number` contains an *integer* and `my_favourite_pie` a *floating point* number. Contrary to other programming languages like C++ and similarly to Python, Julia can infer the type of the object on the right side of the equal sign, so there is no need to specify the type of the variable as you would do in C++.

If we want to print the value of a variable, we use `print` :

```
>>>print(my_name)
Aurelio
```

It is possible to perform mathematical operations on numbers and variables and assign the results to new variables:

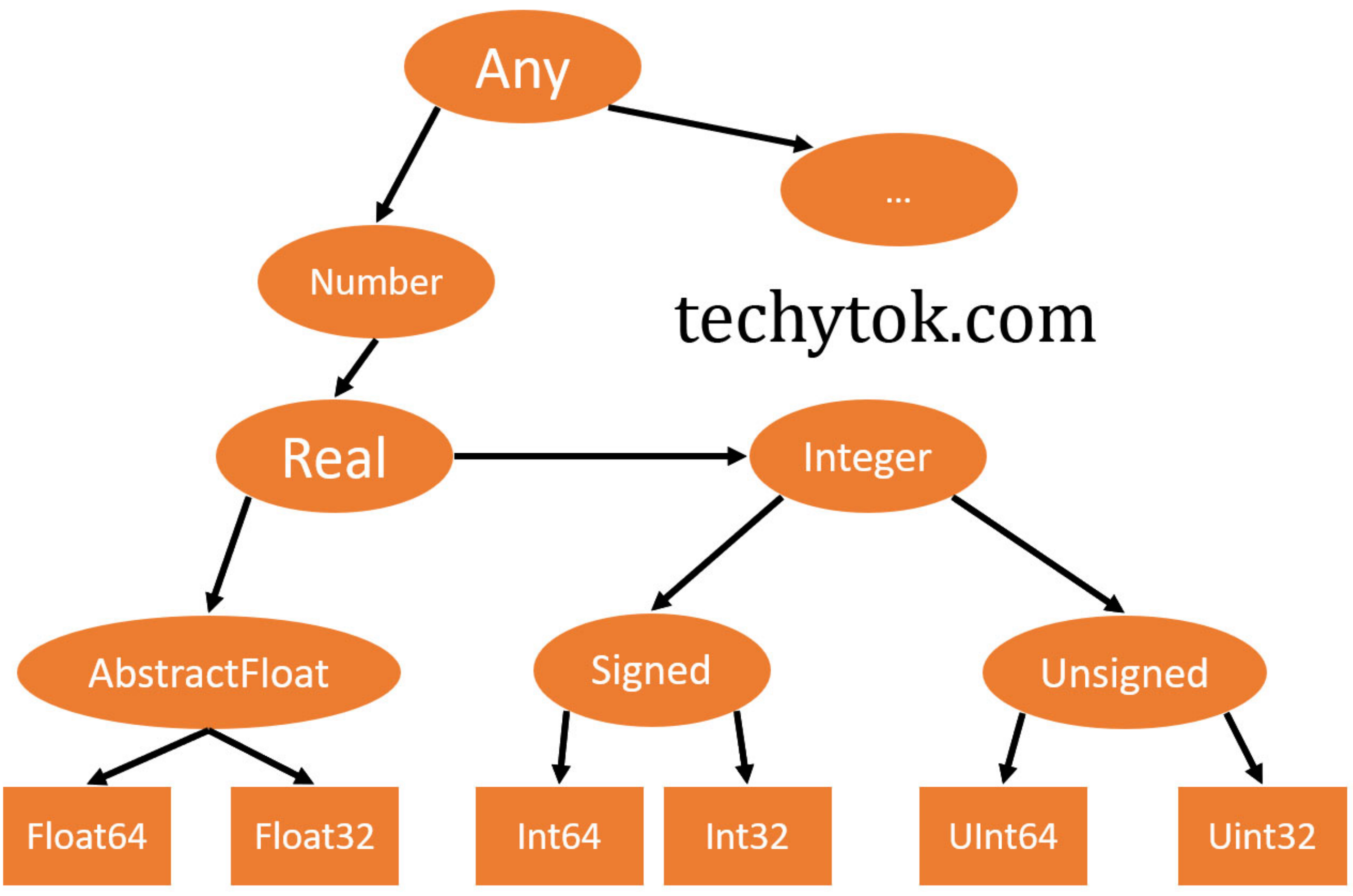
```
a = 2
b = 3
sum = a + b
difference = a - b
product = a * b
quotient = b / a
power = a^3
modulus = b % a
```

It is also possible to use rational numbers, if it is needed for exact numerical computations, and they take the form of `numerator//denominator` , e.g. `1//3` .

For a complete list of the mathematical operations see the official guide (https://docs.julialang.org/en/v1/manual/mathematical-operations/index.html).

Types

In Julia every element has a type. The type system is a hierarchical structure: at the top of the tree there is the type `Any` , which means that every element belongs to it, then there are many other sub-types, for example `Number` which includes `Real` and `Complex` , and `Real` contains for example `Int` (integer) numbers and `Float64` numbers.



We will deal with types again later, for now what we need to remember is that since every variable has a type (which can be inferred by the compiler or specified by the programmer), Julia can build machine code optimised for the specific types which are being used. This is one of the main reasons why Julia is a really fast language! Furthermore, we can expect that if a function can work on a `Type`, for example a `Number`, it will also work on all of its `Subtypes`, like `Float64` or `Int64`.

To determine the type of a variable, we can use the `typeof` function:

```
>>>typeof(0.1)
Float64

>>>typeof(42)
Int64

>>>typeof("TechyTok")
String
```

Although it is possible to change the value of a variable inside a program (it is a *variable*, after all) it is good programming practice and is also critical for performance that inside a program a variable is “type stable”. This means that if we have assigned `a = 42` it is better **not** to assign a new value to `a` which cannot be converted into an `Int` without losing information, like a `Float64` `a = 0.42` (if we convert a `Float64` to an `Int`, the decimal part gets truncated).

If we know that a variable (such as `a`) will have to contain values of type `Float64` it is better to initialise it with a value that is already of that type.

```
a = 2 # if we need to operate with ints
b = 2.0 # if we need to operate with floats
```

It is possible, if needed, to convert a value from a type to another using the function `convert`, for example:

```
>>>convert(Float64, 2)
2.00

a = 2
b = convert(Float64, a)

>>>a
2

>>>b
2.00
```

Conclusion

We have learned what variables are, how to perform basic operations and we have dealt about types.

In the next lesson we will try to understand what **functions** are and how they can be used to write versatile and reusable code!

If you liked this lesson and you would like to receive further updates on what is being published on this website, I encourage you to subscribe to the **newsletter** (https://techytok.com/newsletter/)! If you have any **question** or **suggestion**, please post them in the **discussion below**!

Thank you for reading this lesson and see you soon on TechyTok!

Tags: Julia

Categories: Lessons Tutorial

Updated: November 2, 2019