# juliastart

Kars

June 13, 2019

## 1 Core Julia

This chapter consists of six sections:

1. Variable names;
2. Operators;
3. Types;
4. Data Structure;
5. Control flow;
6. Functions.

### 1.1 Variable names

Variable names are case sensitive and Unicode names (in UTF-8 encoding) may be used. And names must begin with a letter, no matter lowercase or uppercase, an underscore or a Unicode code point larger than 00A0, and other Unicode points, even a latex symbol. We can also redefine the constants here, such as $\pi$. But bulid-in statements are not allowed in this language. The examples are below.

```
In [1]: ## right variable names
        z = 100
        y = 1.0
        s = "my_variable"
        data_science = "true"
        datascience = true

        ## wrong variable names are here if run them, return an error
        ## if = 1.2
        ## else = true
        ##
```

```
Out[1]: true
```

### 1.2 Operators

The operators are similar to *R* and *Python*. Four main categories of operators in Julia:

1. Arithmetic;

2. Updating;
3. Numeric comparison;
4. Bitwise.

```
In [ ]: x = 2
        y = 3
        z = 4

        x + y
        x^y
        x += 2
        x
        y
```

When constructing expressing with multiple operators, the order in which these operators are applied to the expression is known as operator precedence.

With the parentheses () inclueded in the expression, we can control the order by ourselves like the following code

```
In [ ]: x*y+z^2
        x*(y+z^2)
        (x*y)+(z^2)
```

## 1.3 Types

### 1.3.1 Numeric

Julia offers full support for real and complex numbers. The internal variable `Sys.WORD_SIZE` displays the architecture type of the computer. Minimum and Maximum can be showed by `typemin()` and `typemax()`.

```
In [2]: Sys.WORD_SIZE

Out[2]: 64

In [3]: typemax(Int)

Out[3]: 9223372036854775807

In [4]: typemin(Int)

Out[4]: -9223372036854775808

In [5]: typemax(Int64)

Out[5]: 9223372036854775807

In [6]: typemax(Float32)

Out[6]: Inf32
```

Some types use leftmost bit to control the sign, such as `Int64`, but others use that bit as value and without sign like `UInt128`.

Boolean values are 8-bit integers, with false being 0 and true being 1.Overflow errors will happen if the result is larger or smaller than its allowable size.

```
In [7]: literal_int = 1
        println("typeof(literal_int):",typeof(literal_int))
```

```
typeof(literal_int):Int64
```

```
In [8]: x = typemax(Int64)
```

```
Out[8]: 9223372036854775807
```

```
In [9]: x += 1
```

```
Out[9]: -9223372036854775808
```

### 1.3.2 Floats

Floats are similar to scientific notation. They are made up of three components: asigned interger whose length determines the precision, the base used to represent te number and a signed integer htat changes the magnitude of floating point number (the exponent). `Float64` literals are distinguished by having an `e` before hte power, and can be defined in hexadecimal. `Float32` literals are distinguished by having an `f` in place of the `e`. THere are three `Float64` values that do not occur on the real line:

1. `Inf`, positive indinity: a value larger than all finite floating point numbers, equal to itself, and greater than every other floating point value but `NaN`;
2. `-Inf`, negative infinity: a value less than all finite floating point numbers, equal to itself, and less than every other floating point value but `NaN`;
3. `NaN`, not a number: a value not equal to any floating point value, and not ==,< or > than any floating point value, including itself.

Some tips:

1. digit separation using an _;

```
In [10]: x1 = 1.0
         x64 = 15e-15
         x32 = 2.5f-4
         println("x1 is ",typeof(x1))
         println("\nx64 is ", typeof(x64))
         println("\nx32 is ", typeof(x32))
```

```
x1 is Float64

x64 is Float64

x32 is Float32
```

```
In [11]: 9.2_4==9.24
         ## digit separation using an _

Out[11]: true
```

In machine, it is defined that the smallest value is $1 + z \neq 1$. In Julia, the value of epsilon for a particular machine can be found via the `eps()` function.

The spacing between floating point numbers and the value of machine epsilon is important to understand because it can help avoid certain types of errors.

There are also float underflow errors, which occur when the result of a calculation is smaller than machine epsilon or when numbers of similar precision are subtracted.

```
In [12]: eps()

Out[12]: 2.220446049250313e-16

In [13]: n1 = [1e-25,1e-5,1.,1e5,1e25]
         for i in n1
             println(*(i,eps()))
         end

2.2204460492503132e-41
2.220446049250313e-21
2.220446049250313e-16
2.220446049250313e-11
2.2204460492503133e9
```

### 1.3.3 Strings

In Julia, a string is a sequence of Unicode code points, using UTF-8 encoding. Characters in strings have an index value within the string. It is worth noting that Julia indices start at **position 1**, similar to **R** but different to **Python**.