Bitwise boolean Operations

from julialang.org

Performs boolean operations on
- individual bits of one argument
- same-index bits of two arguments

Examples of these ops
in program 'bitwise.jl' on the web site

Expression	Name	
`~x`	bitwise not	
`x & y`	bitwise and	
`x | y`	bitwise or	
`x ⊻ y`	bitwise xor (exclusive or)	- same as xor(x,y)
`x >>> y`	logical shift right	- shifts all bits
`x >> y`	arithmetic shift right	- leaves sign bit (1s are shifted in if negative)
`x << y`	logical/arithmetic shift left	- does not preserve sign (0s shifted in on right)

Vectorized operators

All operators acting on single variables have vectorized “dot” versions

For an array x (any number of dimensions):

`.op x` performs “op” on each element

Example, for a vector x of lengt n

```
for i=1:n
   x[i] = x[i]^2
end
```

does the same as

```
x .= x.^2
```

x = x.^2 also works, but allocates
a new x if x already exists (slower)

can also be expressed with the @. macro

```
@. x = x^2
```

Examples in program timing.jl online
- this program also introduces functionality for timing code for performance

Complex numbers

These complex types are available:

```
ComplexF16 – same as Complex{Float16}
ComplexF32 – same as Complex{Float32}
ComplexF64 – same as Complex{Float64}
```

The numbers refer to the number of bits in both real and imag part

The imaginary constant i is denoted `im`

A complex number can be assigned by adding real and imag parts:

```
c = 1.7 + 4.0im
```

Note a literal constant multiplying a named variable or constant does not need * in Julia

or with the complex function

```
c = complex(1.7,4.0)
```

This is the recommended way

Many functions for complex operations are available

Some examples in complex.jl online

Rational numbers

There is a type for rational numbers, notation a//b

- check the Julia documentation if you need to use

Characters

A single character is of the type Char; using 4 bytes (32 bits)

The Unicode system is used

- Char(c) is the Unicode character corresponding to integer c
- A character is entered within ‘ ’

```
a = ‘A’
```
assigns the value A to the variable a

- A character can be converted to its number by Int()

```
println(Int(‘A’),”  “,Int(’大’))
```
gives the output:
```
65   22823
```

A character can be referred to using \u or \U

- followed by the number of a character in hexadecimal format
- characters are in windows 0-D7FF and E000 - 10FFFF (not all assigned)

```
c=‘\U5927’
println(c)
```
5927 is hexadecimal for 22823

produces 大

Unocodes 0-127 are the conventional ASCII characters

Examples in prgram unicode.jl online

Strings (character strings) - text

An object of type `String` consists of one or more characters

```
a = “Hello”
```

assigns the word Hello to the variable a; using “ ” (not ‘ ’)

A string of length 1 is not the same as a Char

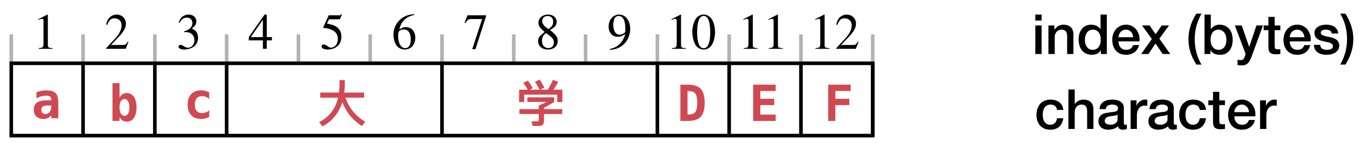
```
a = “H”
b = ‘H’
```

length-1 string (type is String)
character (type is Char)

`a == b` **`false`**

- a Char always uses 4 bytes
- a character stored in a string uses 1-4 bytes

Example: `a = “abc大学DEF”`



- The size of the string in bytes (number of indices, here 12): `lastindex(a)`
- The length of the string, `length(a)`, is the number of characters (8)

a[i] is the character starting at index i; error if no start at i

- cumbersome feature, avoided if only ASCII characters (1 byte each)

Further illustrations in online program string.jl