Julia

[Link to Python ACO Youtube](https://www.youtube.com/watch?v=EJKdmEbGre8)

### Variable

#### Naming

x = 1 + 2

* pi can represent π and can be assigned to other values
* name/value of the variable can be other languages
* name of the variable should be lowercase (separated by \_)
* name of types/modules: MyTypeOrModule
* name of functions/macros: myfunction

#### String

CODE

str = “Hello, World!”

str[0] ERROR

str[1] = “H”

str[end] = “!”

str[end÷2] = “,”

str[end+1] ERROR

str[4:9] = “lo, Wo”

str[6:6] = str[6] = “,”

SubString(str, 2, 3) = “el”

#Concatenation

string(str, "! ", whom, ".\n") = Hello, World!! Wow.

str \* "!" \* whom \* "!\n"

#Interpolation - evaluate the value

"$str $whom.\n" = “Hello, World! Wow.”

"1 + 2 = $(1 + 2)" = “1 + 2 = 3”

#Find first/last index

findfirst(isequal('o'), "xylophone") = 4

julia> findlast(isequal('o'), "xylophone") = 7

#Find next index

findnext(isequal('o'), "xylophone", 1) = 4

findnext(isequal('o'), "xylophone", 5) = 7 #find next index of “o” from 5

findprev(isequal('o'), "xylophone", 5) = 4

#Contains?

occursin("world", "Hello, world.") = true

* replace, nextind, match, captures, offsets,

#### Function

Initialization

f(x,y) = x + y

OR

function f(x,y)

x + y

end

* Default returned value = last expression evaluated

Tuple: (2, “how”, 0.4)

CODE

function foo(a,b)

a+b, a\*b

end

x, y = foo(2,3) = (5, 6) # x = 5, y = 6

Array: [2, 3]

### Number

#### Integer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Signed? | Number of bits | Smallest value | Largest value |
| Int8 | ✓ | 8 | -2^7 | 2^7 - 1 |
| UInt8 |  | 8 | 0 | 2^8 - 1 |
| Int16 | ✓ | 16 | -2^15 | 2^15 - 1 |
| UInt16 |  | 16 | 0 | 2^16 - 1 |
| Int32 | ✓ | 32 | -2^31 | 2^31 - 1 |
| UInt32 |  | 32 | 0 | 2^32 - 1 |
| Int64 | ✓ | 64 | -2^63 | 2^63 - 1 |
| UInt64 |  | 64 | 0 | 2^64 - 1 |
| Int128 | ✓ | 128 | -2^127 | 2^127 - 1 |
| UInt128 |  | 128 | 0 | 2^128 - 1 |
| Bool | N/A | 8 | false (0) | true (1) |

CODE

typemax(Int64) = 9223372036854775807

2.5e-4 =0.00025

2.5f-4 = 0.00025f0(float indicated)

Float32(-1.5) = -1.5f0

#### Floating number

|  |  |  |
| --- | --- | --- |
| Type | Precision | Number of bits |
| [Float16](https://docs.julialang.org/en/v1/base/numbers/#Core.Float16) | [half](https://en.wikipedia.org/wiki/Half-precision_floating-point_format) | 16 |
| [Float32](https://docs.julialang.org/en/v1/base/numbers/#Core.Float32) | [single](https://en.wikipedia.org/wiki/Single_precision_floating-point_format) | 32 |
| [Float64](https://docs.julialang.org/en/v1/base/numbers/#Core.Float64) | [double](https://en.wikipedia.org/wiki/Double_precision_floating-point_format) | 64 |

#### Complex & Rational Number

CODE

# Complex Number - functions

real(1 + 2im) = 1

imag(1 + 2im) = 2

conj(1 + 2im) = 1 - 2im

abs(1 + 2im) = 2.23606797749979

abs2(1 + 2im) = 5

angle(1 + 2im) = 1.1071487177940904

# operators

(1 + 2im)\*(2 - 3im) = 8 + 1im

2im^2 = -2 + 0im

#Rational Number

2//3 = 2/3

### Math Expression

#### Arithmetic Operators

|  |  |  |
| --- | --- | --- |
| Expression | Name | Description |
| +-x | unary plus/minus | the identity operation |
| -x | unary minus | maps values to their additive inverses |
| x +-\*/ y | binary + - \* / |  |
| x ÷ y | integer divide | x / y, truncated to an integer |
| x \ y | inverse divide | equivalent to y / x |
| x ^ y | power | raises x to the yth power |
| x % y | remainder | equivalent to rem(x,y) |
| !x | negation | true <=> false |

#### Bitwise Operators

|  |  |
| --- | --- |
| Expression | Name |
| ~x | bitwise not |
| x & y | bitwise and |
| x | y | bitwise or |
| x ⊻ y | bitwise xor (exclusive or) |
| x >>> y | [logical shift](https://en.wikipedia.org/wiki/Logical_shift) right |
| x >> y | [arithmetic shift](https://en.wikipedia.org/wiki/Arithmetic_shift) right |
| x << y | logical/arithmetic shift left |

CODE

~123 = -124

123 & 234 = 106

123 | 234 = 251

123 ⊻ 234 = 145

xor(123, 234) = 145

~UInt32(123) = 0xffffff84

~UInt8(123) = 0x84

* Math Functions: rounding(ceil, floor…), division(div, rem, mod, divrem…), sign & absolute(abs, sign, copysign…), power & root(sqrt, cbrt, exp, log…)...