https://julialang.org/

https://julialang.org/downloads/

http://junolab.org/

https://juliacomputing.com/

https://www.youtube.com/user/JuliaLanguage/videos

REPL/Shell

juno

juliapro

jupyter https://jupyter.org/, see IJulia below

julia mycode.jl

include("mycode.jl")

versioninfo()

arrow up and down

# This is a comment

#= This is another comment =#

? # switches to help mode

cos # help on cosene

autocompletion

; shell commands

ctrl+L # cleans consol, also clearconsole()

exit()

\alpha (+ press Tab)

\int (+ press Tab)

\:whale: (+ press Tab)

\:pizza: (+ press Tab)

\:hamburger: (+ press Tab)

🍕>🍔

# Here is where unicode is supercool

∑(x,y) = x + y

∑(1,2)

ans

ans;

ans+1

pi (+ press Tab) # returns 3.14...

ℯ

Base.MathConstants.golden

println(ans)

println("I like economics")

println("""I like economics "with" quotes""")

] # for package manager

ctrl+C # to exit

st # status of packages

add PyPlot

up PyPlot # update PyPlot

rm PyPlot # remove package

update # updates all packages

using Pyplot

List of packages

https://pkg.julialang.org/

Printf # Why in this way? Two reasons

Gadfly # ggplot2-like plotting

Pandas

PyCall

TensorFlow

DifferentialEquations

JuMP

StatsBase

ForwardDiff

DataFrames # linear/logistic regression

Distributions # Statistical distributions

Flux # Machine learning

LightGraphs # Network analysis

TextAnalysis # NLP

ODBC

using IJulia

notebook() # Jupyter

using PyPlot

x = range(0,stop=5,length=101)

y = cos.(2x .+ 5)

plot(x, y, linewidth=2.0, linestyle="--")

title("a nice cosinus")

xlabel("x axis")

ylabel("y axis")

ctrl+C # to exit

a = time()

b = time()-a

The LLVM

code\_llvm(sqrt, (Int64,))

#############################

Variables

#############################

a = 1

typeof(a)

bitstring(a)

a = 1.0

typeof(a)

bitstring(a)

isa(a,Float64)

iseven(2)

isodd(2)

ispow2(4)

isfinite(a)

isinf(a)

isnan(a)

eltype(a) # types of an interated list

typemax(Int64)

typemin(Int64)

typemin(Float64) # returns -Inf (just a convention)

typemin(Float64) # returns Inf (just a convention)

eps(Float64) # returns 2.22e-16

1.0 + eps(Float64)

precision(Float64) # returns 53, effective number of bits in the mantissa

typeof(pi (+ press Tab))

a::Float64 # fixes type of a to generate type-stable code

a = "Hello"

a::Float64 # It also asserts type

a = 0x3 # unsigned integer, hexadecimal base

a = 0b11 # unsigned integer, binary base

a = 3.0 # Float64

a = 4 + 3im # imaginary

a = complex(4,3) # same as above

a = true # boolean

a = "String" # string

const aa = 1 # constant

# type promotion system

a = Any[1 2 3; 4 5 6]

convert(Array{Float64}, a)

Array{Float64}(a)

promote(1, 1.0) # promotes both variables to 1.0, 1.0

# Union types

Union{Int, String}

# arbitrary precision arithmetic with GNU Multiple Precision Arithmetic Library (GMP) and the GNU MPFR Library

BigFloat(2.0^66) / 3

supertype(Float64) # supertype of Float64

subtypes(Integer) # subtypes of Integer

a = 1 // 2 # note // operator instead of /

b = 3//7

c = a+b

numerator(c) # finds numerator of c

denominator(c) # finds denominator of c

a = 1 // 0

a = 0 // 0

a = [1, 2, 3] # vector

a = [1; 2; 3] # same vector

first(a) # returns 1

last(a) # returns 3

a = 1:0.5:4

typeof(a)

a[2]

a = collect(1.0:0.5:4) # vector from 1.0 to 4.0 with step 0.5

a[2]

b = [1 2 3] # 1x3 matrix (i.e., row vector)

b = [1 2 3]' # 3x1 matrix (i.e., column vector)

a = [1 2; 3 4] # create a 2x2 matrix

a[2,2] # access element 2,2

a[1,:] # access first row

a[:,1] # access first column

a = zeros(2,2) # zero matrix

a = ones(2,2) # unitary matrix

using LinearAlgebra

a = 1.0\*Matrix(I,2,2) # identity matrix

a = diagm(0 => [2,2,3]) # diagonal matrix, identity matrix

a = diagm(1 => [1,2,3]) # diagonal matrix, identity matrix

a = fill(2,3,4) # fill a 3x4 matrix with 2's

a = trues(2,2) # 2x2 matrix of trues

a = falses(2,2) # 2x2 matrix of falses

a = rand(2,2) # random matrix (uniform)

a = randn(2,2) # random matrix (gaussian)

a = Array{Float64,2}

a = ["Economics" 2;

3.1 true]

ndims(a) # number of dimensions of a

size(a) # size of each dimension of a

length(a) # length (factor of the sizes) of a

a = [1 2; 3 4] # create a 2x2 matrix

a' # complex conjugate transpose of a

a[:] # convert matrix a to vector

vec(a) # vectorization of a

b = [1 2]'

a\*b # multiplication of two matrices

a\b # solution of linear system ax = b

inv(a) # inverse of a

pinv(a) # pseudo-inverse of a

rank(a) # rank of a

norm(a) # Euclidean norm of a

det(a) # determinant of a

diag(a) # diagonal of a

eigvals(a) # eigenvalues

eigvecs(a) # eigenvectors

tril(a) # lower triangular matrix of a

triu(a) # upper triangular matrix of a

show(a) # shows a

sum(a) # sum of a

maximum(a) # max of a

minimum(a) # min of a

b = [1 2;3 4]

dot(a, b) # inner product of two vectors

a[end] # gets last element of a

a[end-1] # gets element of a -1

-------------------------------------------------------------

# Sparse Matrices

using SparseArrays

a = sparse([1, 2, 3], [1, 2, 3], [0, 2, 0])

a = spzeros(3)

# Passing by sharing (not by value)!!!!!!!

# Somewhat imprecissely, passing by reference

a = ["My string" 2; 3.1 true]

b = a

b[1,1]

a[1,1] = "Example of passing by sharing"

b[1,1]

pointer\_from\_objref(a)

pointer\_from\_objref(b)

# If you want passing by value

a = ["My string" 2; 3.1 true]

b = copy(a) # shallow copy

a[1,1] = "Example of passing by reference"

b[1,1]

# also, a deep copy

b = deepcopy(a)

# Julia deals very well with sets

a = [1,2,3]

2 in a # returns true

in(2,a) # same as above

4 in a # returns false

a = [2,1,3]

b = [2,4,5]

union(a,b) # returns 2,1,3,4,5

intersect(a,b) # returns 2

setdiff(a,b) # returns 1,3

setdiff(b,a) # returns 4,5

# Also, tuples are important

a = ("This is a tuple", 2018) # definition of a tuple

a[2] # accessing element 2 of tuple a

a = [1 2]

b = [3 4]

c = zip(a,b)

first(c)

#############################

Basic functions

#############################

polymorphic multiple dispatch

methods(+)

# Lazy evaluation

2 > 3 && println("I am lazy")

2 > 1 && println("I am lazy")

a = 1.2

abs(a) # absolute value of a

abs2(a) # square of a

sqrt(a) # square root of a

isqrt(a) # integer square root of a

cbrt(a) # cube root of a

exp(a) # exponent of a

exp2(a) # power a of 2

exp10(a) # power a of 10

expm1(a) # exponent e^a-1 (accurate)

ldexp(a,n) # a\*(2^n)

log(a) # log of a

log2(a) # log 2 of a

log10(a) # decimal log of a

log(n,a) # log base n of a

log1p(a) # log of 1+a (accurate)

# Some syntaxic sugar

isapprox(1.0, 1.1; atol = 0.1)

+ - \* / ^ # arithmetic operations

+. -. \*. /. ^. # element-by-element operations (for vectors and matrices)

// # division for rationals that produces another rational

+a # identity operator

-a # negative of a

a+=1 # a = a+1, can be applied to any operator

a\b # back division

x = 3

7\*x # this delivers 21

7x # this also delivers 21

x7 # this delivers an error message (Julia searches for variable "x7")

eval(a) # evaluates expression a in a global scope

real(a) # real part of a

imag(a) # imaginary part of a

reim(a) # real and imaginary part of a (a tuple)

conj(a) # complex conjugate of a

angle(a) # phase angle of a in radians

cis(a) # exp(i\*a)

sign(a) # sign of a

round(a) # rounding a to closest floating point natural

ceil(a) # round up

floor(a) # round down

trunc(a) # truncate toward zero

clamp(a,low,high) # returns a clamped to [a,b]

mod2pi(a) # module after division by 2\pi

modf(a) # tuple with the fractional and integral part of a

div(a,b) # same as above

cld(a,b) # ceiling division

fld(a,b) # flooring division

rem(a,b) # remainder of a/b

gcd(a,b) # greatest positive common denominator of a,b

gcdx(a,b) # gcd of a and and and their minimal Bezout coefficients

mod(a,b) # module a,b

mod1(a,b) # module a,b after flooring division

lcm(a,b) # least common multiple of a,b

min(a,b) # min of a and (can take as many arguments as desired)

max(a,b) # max of a and (can take as many arguments as desired)

minmax(a,b) # min and max of a and b (a tuple return)

muladd(a,b,c) # a\*b+c

+(a,b)

a = true

b = false

c = 1.0

a+c # this delivers 2.0

b+c # this delivers 1.0

a\*c # this delivers 1.0

b\*c # this delivers 0.0

! # not

&& # and

|| # or

== # is equal?

!== # is not equal?

=== # is equal? (enforcing type 2===2.0 is false)

!=== # is not equal? (enforcing type)

> # bigger than

>= # bigger or equal than

< # less than

<= # less or equal than

3 > 2 && 4<=8 || 7 < 7.1

~ # bitwise not

& # bitwise and

| # bitwise or

xor # bitwise xor (also typed by \xor or \veebar + tab)

>> # right bit shift operator

<< # left bit shift operator

>>> # unsigned right bit shift operator

#############################

Control structures

#############################

# Conditionals

x = 1

y = 1

if x < y

println("x is less than y")

elseif x > y

println("x is greater than y")

else

println("x is equal to y")

end

# Loops

for i in 1:5

println(i)

end

for i in 1:0.1:5

println(i)

end

a = [1, 2, 3]

for i in a

println(i)

end

for i ∈ 1:5

println(i)

end

for i = 1:5

println(i)

end

for i = 1:2, j = 3:4

println((i, j))

end

for i = 1:2, j = 3:4

println((i, j))

if condition break

end

for i = 1:2, j = 3:4

if condition continue

println((i, j))

end

# Comprenhensions

[n^2 for n in 1:5] # basic comprehensions

Float64[n^2 for n in 1:5] # comprehension fixing type

[x+y for x in 1:3, y = 1:4]

# Generators

sum(1/n^2 for n=1:1000)

i = 0

while i <= 5

println(i)

i += 1

end

#############################

Functions

#############################

# functions are first-class citizens

a = [exp, abs]

a[1](3)

# operators are functions

1+2

+(1,2)

# all arguments to functions are passed by sharing

sort vs. sort!

a = [2, 1, 3];

sort(a)

sort!(a)

# One-line

foo(var) = var+1

fooalt = function (var)

var+1

end

# passing functions (also by sharing!!!!!!!!!)

foo1 = foo

multiplicacion = \*

# multiple dispatch

methods(foo)

foo(var1,var2) = var1+var2+1

methods(foo)

# Broadcasting

a = [1, 2, 3]

foo.(a)

# Several lines, also show multiple dispatch

function foo15(var1, var2::Float64, var3=1)

output1 = var1+2

output2 = var2+4

output3 = var3+3 # var3 is optional, by default var3=1

return output1, output2, output3

end

# empty argument

function foo()

output1 = 1

end

# keywords

function foo(var1, var2; keyword=2)

output1 = var1+var2+keyword

end

# fixing types

function foo3(var1::Int64, var2; keyword=2)

output1 = var1+var2+keyword

end

foo3(2.0,2)

foo3(2,2)

function foo4(x,y)::Int8

return x\*y

end

foo4(1.2,1.3)

foo4(1,1)

# Higher-order

function foo(var1)

function foo1(var2)

answer = var1+var2

return answer

end

return foo1

end

foo2 = foo(1) # creates a function foo2 that produces 1+var2

foo5 = foo(2) # creates a function foo3 that produces 2+var2

x -> x^2 # anonymous function

a = x -> x^2 # named anonymous function

a(3)

a = x -> x.^2 # named anonymous function

a([3.0,2.0])

code\_llvm(x ->x^2, (Float64,))

code\_native(x ->x^2, (Float64,))

# recursion

function outer(a)

b = a+2

function inner(b)

b = b+3

end

inner(b)

end

fib(n) = n < 2 ? n : fib(n-1) + fib(n-2)

# Closure

function counter()

n = 0

() -> n += 1

end

# we name it

addOne = counter()

addOne() # Produces 1

addOne() # Produces 2

# Currying: transforms the evaluation of a function with multiple arguments into the evaluation of a sequence of functions, each with a single argument

Haskell Curry

function mult(a)

return function f(b)

return a\*b

end

end

foo5 = mult(3)

foo5(9)

map(floor,[1.2, 5.6, 2.3]) # applies floor to vector [1.2, 5.6, 2.3]

map(x ->x^2,[1.2, 5.6, 2.3]) # applies abstract to vector [1.2, 5.6, 2.3]

reduce(+,[1,2,3]) # generic reduce

foldl(-,[1,2,3]) # folding (reduce) from the left

foldr(-,[1,2,3]) # folding (reduce) from the right

mapreduce(x->x^2, +, [1,3])

a = [1,5,8,10,12]

filter(isodd,a) # select elements of a bigger than 5

#############################

Macros

#############################

macro welcome(name)

return :(println("Hello, ", $name, " likes economics"))

end

@welcome("Jesus")

#############################

Types (constructors and methods) and Named Tuples

#############################

struct MicroSurveyObservation

id::Int64

year::Int64

quarter::Int64

region::String

ageHouseholdHead::Int64

familySize::Int64

numberChildrenunder18::Int64

consumption::Float64

end

household1 = MicroSurveyObservation(12,2017,3,"angushire",23,2,0,345.34)

fieldnames(MicroSurveyObservation)

household1.familySize

totalPopulation = household1.familySize

household1.id = 31 # it will give you an error

# Mutable

mutable struct MutableMicroSurveyObservation

id::Int64

year::Int64

quarter::Int64

region::String

ageHouseholdHead::Int64

familySize::Int64

numberChildrenunder18::Int64

consumption::Float64

end

household1 = MutableMicroSurveyObservation(12,2017,3,"angushire",23,2,0,345.34)

household1.id = 31

function EquivalenceScale(x::MicroSurveyObservation)

if x.familySize == 1

return x.consumption

else

return x.consumption/(1+0.5\*(x.familySize-1))

end

end

household1 = MicroSurveyObservation(12,2017,3,"angushire",23,2,0,345.34)

EquivalenceScale(household1)

function AverageConsumption(x::MicroSurveyObservation,y::MicroSurveyObservation)

return 0.5\*(x.consumption+y.consumption)

end

import Base: +

+(x::MicroSurveyObservation,y::MicroSurveyObservation) = x.consumption + y.consumption

household1 = MicroSurveyObservation(12,2017,3,"angushire",23,2,0,345.34)

household2 = MicroSurveyObservation(13,2015,2,"Wolpex",35,5,2,645.34)

household = Vector{MicroSurveyObservation}(undef, 10)

household[1] = MicroSurveyObservation(12,2017,3,"angushire", 23, 2,0,345.34)

household[2] = MicroSurveyObservation(13,2015,2,"Wolpex", 35, 5,2,645.34)

for i in 1:10

# read file with observation

household[i] = MicroSurveyObservation(#data from previous step)

end

household1 = (id = 12,

year = 2017,

quarter = 3,

region = "angushire",

ageHouseholdHead = 23,

familySize = 2,

numberChildrenunder18 = 0,

consumption = 345.34)

using DataFrames, Statistics

microSurveyObservations = DataFrame(;household1...) #Creating with named tuple

household2 = ( id = 15,

year = 2017,

quarter = 3,

region = "angushire",

ageHouseholdHead = 26,

familySize = 2,

numberChildrenunder18 = 0,

consumption = 1345.34)

push!(microSurveyObservations, household2) #Push named tuples onto the dataframe

mean(microSurveyObservations[:consumption]) #Statistics.

#############################

Dictionaries

#############################

# Creating a dictionary

a = Dict("University of Pennsylvania" => "Philadelphia", "Boston College" => "Boston")

a["University of Pennsylvania"] # access one key

a["Harvard"] = "Cambridge" # adds an additional key

delete!(a,"Harvard") # deletes a key

keys(a)

values(a)

haskey(a,"University of Pennsylvania") # returns true

haskey(a,"MIT") # returns false

#############################

Metaprogramming

#############################

a = quote

"I like economics"

end

typeof(a) # returns Expr

eval(a)

name = "Jesus"

a = :(name\*" likes economics")

eval(a) # returns "Jesus likes economics"

name = "Pablo"

eval(a) # returns "Pablo likes economics"

a = :($name\*" likes economics")

function math\_expr(op, op1, op2)

expr = Expr(:call, op, op1, op2)

return expr

end

ex = math\_expr(:+, 1, Expr(:call, :\*, 4, 5))

eval(ex)

ex = math\_expr(:\*, 1, Expr(:call, :\*, 4, 5))

eval(ex)

#############################

Strings

#############################

string('a','b') # returns ab

string("a","b") # returns ab

"a"\*"b" # returns ab

" " # white space

"a"\*" "\*"b" # returns a b

\*("a","b") # returns ab

repeat("a",2) # returns aa

"a"^2 # returns aa also

join(["a","b"]," and ") # returns "a and b"

a = 3

string("a=$a") # returns a=3

b = true

string(b) # returns "true"

a = 1

print(a) # basic printing functionality, no formatting

println(a) # as before, plus a newline

using Printf

# first an integer, second a float with two decimals, third a character

@printf("%d %.2f % c\n", 32, 34.51, 'a')

# Now a composed string

name = "Jesus"

@printf("%s likes economics \n", name)

# It will print with color

printstyled(a;color=:red)

printstyled(a;color=:magenta)

printstyled(a;color=:blue)

a = readline()

f = open("results.txt", "w") # open file "results.txt"

using CSV

CSV.read(filename)

open("results.txt", "w") do f

write(f, "I like economics")

close(f)

end

open("results.txt", "r") do f

mystring = readline(f)

close(f)

end

##############################################################

Plots

##############################################################

using Plots

pyplot()

x = 1:10

y = x.^2

plot(x,y)

plot(x,y,title="A nicer plot", label = "Square function", xlabel = "x-axis", ylabel ="y-axis")

plot!(x,y.+1,title="A second plot", label = "Square function", xlabel = "x-axis", ylabel ="y-axis")

savefig("figure1.pdf")

using Distributed

using LinearAlgebra

M = Matrix{Float64}[rand(1000,1000) for i = 1:10];

addprocs(4)

@time pmap(svdvals, M);

using Profile

@profile main()

Profile.print(format=:flat)