Julia Programming Basics

Julia Documentation (https://docs.julialang.org/en/v1/)

Version : 1.6.1

Created By Manthan Bhikadiya

Reference Video Tutorial by Abhishek Agrawal :

- https://www.youtube.com/watch?v=lwj-1mclq0U (https://www.youtube.com/watch?v=lwj-1mclq0U)

Topics :

- Hello World
- Variable Operations
- typeof function
- Opearator Precedence
- Comments
- Variable define rules
- Arrays
- Ranges
- Tuples
- Dictionary
- Set
- Working with Date and Time
- Conditional Statements
- Loops
- Comprehensions
- Find Prime Numbers between 1 to 100
- String Manipulation

Hello World

In [1]:

```
# In Julia we have two print function

println("Hello World")
print("Hello World")
```

```
Hello World
Hello World
```

In [2]:

```
# difference : Whatever statement after println() prints in new line
# where as in print() prints in same line. we have to specify newline tag ("\n")

println("Hello ")
println("World")
print("Hello ")
print("World \n")
print("Julia")
```

```
Hello 
World
Hello World 
Julia
```

Variable Operations

In [4]:

```
loan_amount = 1000 ## int type
interest_area = 0.015 ## float type
tenture = 5
```

Out[4]:

```
5
```

In [5]:

```
loan_amount* tenture # arithmetics opeartation
```

Out[5]:

```
5000
```

In [6]:

```
str = "My First Program" ## string type variable
print(str)
```

```
My First Program
```

In [7]:

```
review = " => The best movie in the world"
movie_name = "Avengers : Endgame "

## String concatenation : 2 Ways
println(movie_name*review)  # using * operator
println(string(movie_name,review)) # string() built in function

println(movie_name^2) # print two times movie name
```

```
Avengers : Endgame  => The best movie in the world
Avengers : Endgame  => The best movie in the world
Avengers : Endgame Avengers : Endgame
```

typeof buit-in function

In [8]:

```
println(typeof(movie_name)) # type of variable
println(typeof(interest_area) ," ",typeof(loan_amount))
```

```
String
Float64 Int64
```

Opearator Precedence

In [9]:

```
2 + 5 - 6/2 * 3

# Operation follows in this order:
# 1. 6 / 2 = 3
# 2. 3 * 3 = 9
# 3. 2 + 5 = 7
# 4. 7 - 9 = -2
```

Out[9]:

```
-2.0
```

Commenting in Julia

- **Single Line Comment**
  - - '#' is used for single line comment in Julia.
- **Multiple Line Comment**
  - - '#='...'=#' is used for multiline comment in Julia.

Variable Rules

- Never start with Number or special character.
- Do not use special character except _ ('underscore') in your variable.
- You can not use buit in function/attribute name as your variable name

If you not follow above rules compiler will give you syntax error.

Arrays

- ordered collection of variables.

In [10]:

```
arr = [1,2,3,4,5] # data type = Int 64 bit
```

Out[10]:

```
5-element Vector{Int64}:
 1
 2
 3
 4
 5
```

In [11]:

```
arr = [1,2.6,3,4,3.5] # convert all int type to float64
```

Out[11]:

```
5-element Vector{Float64}:
 1.0
 2.6
 3.0
 4.0
 3.5
```

In [12]:

```
arr = ["Julia","Python","Java"] # string array
```

Out[12]:

```
3-element Vector{String}:
 "Julia"
 "Python"
 "Java"
```

In [13]:

```
func_arr = [print , println , printstyled] # function array
```

Out[13]:

```
3-element Vector{Function}:
 print (generic function with 31 methods)
 println (generic function with 3 methods)
 printstyled (generic function with 2 methods)
```

In [14]:

```
mix_array = [1, 2.5, 3 , "Julia" , "Python"] # any type array
```

Out[14]:

```
5-element Vector{Any}:
 1
 2.5
 3
  "Julia"
  "Python"
```

In [15]:

```
int64_type = Int64[1,2,3,4,5] # specific type array
```

Out[15]:

```
5-element Vector{Int64}:
 1
 2
 3
 4
 5
```

In [16]:

```
float64_type = Float64[1,2,3,9,10] # int values converted to the float64.
```

Out[16]:

```
5-element Vector{Float64}:
  1.0
  2.0
  3.0
  9.0
 10.0
```

In [17]:

```
string_type = String["This","is","Julia"] # string type array
```

Out[17]:

```
3-element Vector{String}:
 "This"
 "is"
 "Julia"
```

2D Array

In [18]:

```
array_2d = [1 2 3 4;5 6 7 8] # shape : 4 x 4
```

Out[18]:

```
2×4 Matrix{Int64}:
 1  2  3  4
 5  6  7  8
```

In [19]:

```
vector = [1 2 3 4 5] # shape : 1 x 5
```

Out[19]:

```
1×5 Matrix{Int64}:
 1  2  3  4  5
```

In [20]:

```
a1 = rand(5) # array of 5 random values
```

Out[20]:

```
5-element Vector{Float64}:
 0.004494553043690619
 0.3950438790557944
 0.6075238226157957
 0.5124315551337923
 0.16642883102200923
```

In [21]:

```
a2 = rand(3,3) # array of 3 x 3 with random values
```

Out[21]:

```
3×3 Matrix{Float64}:
 0.0496462  0.72721    0.955089
 0.201577   0.307465   0.698218
 0.923533   0.0770616  0.637665
```

getindex()

* It is used to get values at specified index * Argument : (array name , list or indexs || index || slice of index)

In [22]:

```
getindex(a2,6)

# Indexing of a2 array
# 1 4 7
# 2 5 8
# 3 6 9
```

Out[22]:

```
0.07706160620157831
```

In [23]:

```
getindex(a1 , 2) # index of a1 => [ 1 2 3 4 5 ]
```

Out[23]:

```
0.3950438790557944
```

In [24]:

```
getindex(a2 , [1,3]) # getting 1 st and 3 rd index values together..
```

Out[24]:

```
2-element Vector{Float64}:
 0.049646237718633746
 0.9235328696408989
```

In [25]:

```
getindex(a2 , 2:8) # getting values from index 2 to index 8

# Note that both the index must be inclued.( In this case 2 and 8 )
```

Out[25]:

```
7-element Vector{Float64}:
 0.20157713804054045
 0.9235328696408989
 0.727209980796329
 0.30746547339089014
 0.07706160620157831
 0.9550888867948613
 0.6982178071790035
```

In [26]:

```
a1[4] # another indexing method for 1D array or vector
```

Out[26]:

```
0.5124315551337923
```

In [27]:

```
a2[[3,4]] ## another indexing method for 2D array
```

Out[27]:

```
2-element Vector{Float64}:
 0.9235328696408989
 0.727209980796329
```

setindex!()

* It is used to set values at specified index. * Argument : (array_name , new_values , index)

In [28]:

```
setindex!(a2,[10,12],[1,3]) # using buit-in fucntion
```

Out[28]:

```
3×3 Matrix{Float64}:
 10.0        0.72721    0.955089
  0.201577  0.307465   0.698218
 12.0        0.0770616  0.637665
```

In [29]:

```
a2[[1,3]] = [0.04 , 0.05] # another method to change values at index
```

Out[29]:

```
2-element Vector{Float64}:
 0.04
 0.05
```

In [30]:

```
a2 # final array...
```

Out[30]:

```
3×3 Matrix{Float64}:
 0.04      0.72721    0.955089
 0.201577  0.307465   0.698218
 0.05      0.0770616  0.637665
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION :
Arrays (https://docs.julialang.org/en/v1/base/arrays/#Indexing-and-assignment)

Ranges

- It is used to create a array for specified ranges.

In [31]:

```
collect(1:10) # range from 1 to 10
```

Out[31]:

```
10-element Vector{Int64}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

In [32]:

```
collect(1.3:5.3) # for floting values
```

Out[32]:

```
5-element Vector{Float64}:
 1.3
 2.3
 3.3
 4.3
 5.3
```

In [33]:

```
collect(1.3:0.3:5.3) # here 2nd argument specifies hou much to addition we have to do.
```

Out[33]:

```
14-element Vector{Float64}:
 1.3
 1.6
 1.9
 2.2
 2.5
 2.8
 3.1
 3.4
 3.7
 4.0
 4.3
 4.6
 4.9
 5.2
```

In [34]:

```
collect(1:2:11) # for integer values
# make sure range values are included...
```

Out[34]:

```
6-element Vector{Int64}:
  1
  3
  5
  7
  9
 11
```

In [35]:

```
collect(100:-20:0) # reverse range
```

Out[35]:

```
6-element Vector{Int64}:
 100
  80
  60
  40
  20
   0
```

In [36]:

```
typeof(collect(1:10)) # ranges are type of array
```

Out[36]:

```
Vector{Int64} (alias for Array{Int64, 1})
```

In [37]:

```
c1 = collect(100:-20:0) # store it in variable
# now we can apply all array operation on c1.
```

Out[37]:

```
6-element Vector{Int64}:
 100
  80
  60
  40
  20
   0
```

In [38]:

```
c1[end] # end is used to give last element in the array.
```

Out[38]:

```
0
```

In [39]:

```
c1[end-2] # third last element in the array
```

Out[39]:

```
40
```

In [40]:

```
c1[1:end-1] # starting from index 1 to second last element
```

Out[40]:

```
5-element Vector{Int64}:
 100
  80
  60
  40
  20
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION :
Ranges (https://docs.julialang.org/en/v1/base/collections/#Base.collect-Tuple{Type,%20Any})

Tuples

- Tupples are immutable
- You cannot change the values of tuples once you declared.

In [41]:

```
t1 = (1,2,3,4,5)
println(typeof(t1)) # type of NTuple and elements types Int64
println(t1)
```

```
NTuple{5, Int64}
(1, 2, 3, 4, 5)
```

In [42]:

```
# accessing tuple elements same as arrays with getinde() and by using Square ([]) brackets.

println(t1[1]) # single element
println(t1[end]) # using end attribute
println(t1[[2,4]]) # accessing multiple elements
println(t1[1:4]) # accessing elements using slicing.
```

```
1
5
(2, 4)
(1, 2, 3, 4)
```

2D Tuple

In [43]:

```
t2 = ((12,21),(23,34),(55,62))
```

Out[43]:

```
((12, 21), (23, 34), (55, 62))
```

In [44]:

```
# 2D array index (1(1 , 2) ,2(1 , 2) ,3(1 , 2))
# array values : ((12, 21),(23, 34),(55,62))

println(t2[3]) # access specific tuple - 3 (55,62)
println(t2[3][1])  # access tuple - 3 1 st value (55)
println(t2[[1,3]]) # access 1 and 3 rd tuple only.
println(t2[1:3]) # access all tuples from index 1 to index 3
```

```
(55, 62)
55
((12, 21), (55, 62))
((12, 21), (23, 34), (55, 62))
```

In [45]:

```
marks = (Science = (89,100) , Maths = (97,100) , Physics = (90,100))
```

Out[45]:

```
(Science = (89, 100), Maths = (97, 100), Physics = (90, 100))
```

In [46]:

```
marks.Science # access tuples by its name
```

Out[46]:

```
(89, 100)
```

In [47]:

```
marks.Maths # access tuples by its name
```

Out[47]:

```
(97, 100)
```

In [48]:

```
marks.Science[1] # specific marks
```

Out[48]:

```
89
```

In [49]:

```
println("Total marks : ",marks.Science[1] + marks.Maths[1] +marks.Physics[1])
```

```
Total marks : 276
```

In [50]:

```
marks2 = (History = (94,100), Computer = (99,100) )
```

Out[50]:

```
(History = (94, 100), Computer = (99, 100))
```

In [51]:

```
merge(marks,marks2)
```

Out[51]:

```
(Science = (89, 100), Maths = (97, 100), Physics = (90, 100), History = (94,
100), Computer = (99, 100))
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION :
[Tuples (https://docs.julialang.org/en/v1/manual/functions/#Tuples)](https://docs.julialang.org/en/v1/manual/functions/#Tuples)

Dictionary

- Structure of Key value pair type.
- Value can be accessed by using corosponding keys.

In [52]:

```
cars = Dict("Car1"=>100000,"Car2"=>500000,"Car3"=>1000000)

# key value pair
# key type : String , Value type : Int64
```

Out[52]:

```
Dict{String, Int64} with 3 entries:
  "Car3" => 1000000
  "Car2" => 500000
  "Car1" => 100000
```

In [53]:

```
# accessing dictionary value with help of keys
println(cars["Car1"])
println(cars["Car2"])
println(cars["Car3"])
```

```
100000
500000
1000000
```

In [54]:

```
# another method to define dictonary
cars2 = Dict(:Car1=>100000 , :Car2=>500000 ,:Car3=>1000000)
```

Out[54]:

```
Dict{Symbol, Int64} with 3 entries:
  :Car3 => 1000000
  :Car1 => 100000
  :Car2 => 500000
```

In [55]:

```
# accessing this type od dictonary

println(cars2[:Car1])
println(cars2[:Car2])
println(cars2[:Car3])
```

```
100000
500000
1000000
```

haskey()

- This function checks wheather a function has key or not

In [56]:

```
haskey(cars2 , :Car2) # argument ( dictionary_name , key that we have to check present or n
```

Out[56]:

```
true
```

In [57]:

```
haskey(cars2 , :car2)
```

Out[57]:

```
false
```

delete!()

- This function is used to delete particular key from dict.

In [58]:

```
delete!(cars2 , :Car2)
```

Out[58]:

```
Dict{Symbol, Int64} with 2 entries:
  :Car3 => 1000000
  :Car1 => 100000
```

In [59]:

```
cars2
```

Out[59]:

```
Dict{Symbol, Int64} with 2 entries:
  :Car3 => 1000000
  :Car1 => 100000
```

In [60]:

```
# make sure key name is perfact
# if you enter a wrong key it will not throw any type of error

delete!(cars2 , :Cars1)
```

Out[60]:

```
Dict{Symbol, Int64} with 2 entries:
  :Car3 => 1000000
  :Car1 => 100000
```

keys()

- Return all the keys present in the dict.

In [61]:

```
keys(cars)
```

Out[61]:

```
KeySet for a Dict{String, Int64} with 3 entries. Keys:
  "Car3"
  "Car2"
  "Car1"
```

In [62]:

```
keys(cars2)
```

Out[62]:

```
KeySet for a Dict{Symbol, Int64} with 2 entries. Keys:
  :Car3
  :Car1
```

values()

- Return all the values in the dict.

In [63]:

```
values(cars)
```

Out[63]:

```
ValueIterator for a Dict{String, Int64} with 3 entries. Values:
  1000000
  500000
  100000
```

In [64]:

```
values(cars2)
```

Out[64]:

```
ValueIterator for a Dict{Symbol, Int64} with 2 entries. Values:
  1000000
  100000
```

merge()

- Used to merge two dict.

In [65]:

```
total_cars = merge(cars , cars2)
```

Out[65]:

```
Dict{Any, Int64} with 5 entries:
  :Car3  => 1000000
  "Car3" => 1000000
  "Car2" => 500000
  :Car1  => 100000
  "Car1" => 100000
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION :
Dictionary (https://docs.julialang.org/en/v1/base/collections/#Base.Dict)

Sets

- It have only unquie value in the set.

In [66]:

```
sports_brands = Set(["Adidas","Nike","Puma","Rebook"])
```

Out[66]:

```
Set{String} with 4 elements:
  "Rebook"
  "Nike"
  "Puma"
  "Adidas"
```

in()

- This function is used to check wheather a mentioned value is in the set or not.

In [67]:

```
in("Adidas",sports_brands)
```

Out[67]:

```
true
```

In [68]:

```
in("HRX",sports_brands)
```

Out[68]:

```
false
```

In [69]:

```
"Adidas" in sports_brands
```

Out[69]:

```
true
```

In [70]:

```
sports_brands_india = Set(["Adidas","Nike","HRX"])
```

Out[70]:

```
Set{String} with 3 elements:
  "Nike"
  "HRX"
  "Adidas"
```

union()

- This function is used for union of two sets.

In [71]:

```
sports = union(sports_brands , sports_brands_india)
```

Out[71]:

```
Set{String} with 5 elements:
  "Rebook"
  "Nike"
  "Puma"
  "HRX"
  "Adidas"
```

intersect()

- This function is used to find common values in both sets.

In [72]:

```
intersect(sports_brands,sports_brands_india)
```

Out[72]:

```
Set{String} with 2 elements:
  "Nike"
  "Adidas"
```

setdiff()

- It returns element of first set which are not present in second set

In [73]:

```
setdiff(sports_brands , sports_brands_india)
```

Out[73]:

```
Set{String} with 2 elements:
  "Rebook"
  "Puma"
```

In [74]:

```
setdiff(sports_brands_india,sports_brands)
```

Out[74]:

```
Set{String} with 1 element:
  "HRX"
```

push!()

- This function is used add value in set

In [75]:

```
push!(sports_brands ,"HRX","Fila")
```

Out[75]:

```
Set{String} with 6 elements:
  "Rebook"
  "Nike"
  "Puma"
  "HRX"
  "Fila"
  "Adidas"
```

pop!()

- This function is return the last value when a define the set.

In [76]:

```
println(sports_brands)
pop!(sports_brands)
```

```
Set(["Rebook", "Nike", "Puma", "HRX", "Fila", "Adidas"])
```

Out[76]:

```
"Rebook"
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION :
Sets (https://docs.julialang.org/en/v1/base/collections/#Base.Set)

Working with Date and Time

- Dates.date - working with date only
- Dates.time - working with time only
- Dates.datetime - working with bot

In [77]:

```
# import in julia

using Dates
```

In [78]:

```
now() # after importing we can use funciton like this

# represent the current time and date (Dates.datetime)
```

Out[78]:

```
2021-05-20T23:01:55.432
```

In [79]:

```
today() # represent today date only (Dates.date)
```

Out[79]:

```
2021-05-20
```

In [80]:

```
birthdate = Date(2000,5,1) # YYYY,MM,DD
```

Out[80]:

```
2000-05-01
```

In [81]:

```
typeof(birthdate) # Date type object
```

Out[81]:

```
Date
```

In [82]:

```
DateTime(2000,5,1,10,15,27) # YYYY,MM,DD,hour,minute,second
```

Out[82]:

```
2000-05-01T10:15:27
```

In [83]:

```
now(UTC) # UTC timezone
```

Out[83]:

```
2021-05-20T17:31:56.720
```

In [84]:

```
birthdate = DateTime(2000,5,1,10,15,27)
```

Out[84]:

```
2000-05-01T10:15:27
```

In [85]:

```
println(year(birthdate))
println(month(birthdate))
println(day(birthdate))
```

```
2000
5
1
```

In [86]:

```
year(now()) # nested function
```

Out[86]:

```
2021
```

In [87]:

```
hour(now()) # 2 PM => 14
```

Out[87]:

```
23
```

In [88]:

```
dayofweek(birthdate) # day of week
```

Out[88]:

```
1
```

In [89]:

```
dayname(birthdate)
```

Out[89]:

```
"Monday"
```

In [90]:

```
daysinmonth(birthdate)
```

Out[90]:

```
31
```

In [91]:

```
birthdate = Date(2000,5,1)

today()-birthdate # difference between today and birthdate in days
```

Out[91]:

```
7689 days
```

In [92]:

```
today() + Month(5) # currnet month is 5(May) add 5 months so it becomes 10 (october)
```

Out[92]:

```
2021-10-20
```

In [93]:

```
date_format = DateFormat("dd-mm-yyyy")
Dates.format(birthdate,date_format)
```

Out[93]:

```
"01-05-2000"
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION :
Date and Time (https://docs.julialang.org/en/v1/search/?q=dates)

Conditional Statements

It includes following statements :

- If - else - end Statement
- If - elseif - else - end Statement

Ternary Operator

In [94]:

```
a = 10

# use ternary operator
a > 10 ? "Yes" : "No"
```

Out[94]:

```
"No"
```

In [95]:

```
println(a < 10 ? "Yes" : "No")
println(a == 10 ? "Yes" : "No")
```

```
No
Yes
```

In [96]:

```
b = 20
```

Out[96]:

```
20
```

In [97]:

```
a >= 10 || b < 20 ## Or operator - one of them is true -> whole statement will true otherwi
```

Out[97]:

```
true
```

In [98]:

```
a > 10 || b < 20 ## both false => Final answer => False
```

Out[98]:

```
false
```

In [99]:

```
a >= 10 && b <= 20 ## Or operator - both of them is true -> whole statement will true other
```

Out[99]:

```
true
```

In [100]:

```
a >= 10 && b < 20 # one of them is false => Final Answer => False
```

Out[100]:

```
false
```

if elseif else end

- else and elseif block are not mandotory.

In [101]:

```
a = 9

if a > 10
    print("a is greater then 10")
elseif a < 10
    print("a is less then 10")
else
    print("a is equal to 10")
end
```

```
a is less then 10
```

if else end

- else block is not mandotory

In [102]:

```
country = "United States"
```

Out[102]:

```
"United States"
```

In [103]:

```
if country == "United States"
    print("Country is United States")
else
    print("You are not from USA")
end
```

```
Country is United States
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION :
Conditional Evalution (https://docs.julialang.org/en/v1/manual/control-flow/#man-conditional-evaluation)

Loops

In [104]:

```
sports
```

Out[104]:

```
Set{String} with 5 elements:
  "Rebook"
  "Nike"
  "Puma"
  "HRX"
  "Adidas"
```

For Loop

In [105]:

```
for i in sports # set traversal
    println(i)
end
```

```
Rebook
Nike
Puma
HRX
Adidas
```

In [106]:

```
for i in "Adidas" # string traversal
    print(i," ")
end
```

```
A d i d a s
```

In [107]:

```
for i in (1,2,45,6,7) # tuple traversal
    println(i)
end
```

```
1
2
45
6
7
```

In [108]:

```
cars
```

Out[108]:

```
Dict{String, Int64} with 3 entries:
  "Car3" => 1000000
  "Car2" => 500000
  "Car1" => 100000
```

In [109]:

```
for d in cars # dictionary traversal
    println(d )
end
```

```
"Car3" => 1000000
"Car2" => 500000
"Car1" => 100000
```

In [110]:

```
for s in 1:5     # using range traversal
    print(s," ")
end
```

```
1 2 3 4 5
```

In [111]:

```
for Range in 1:5
    @show Range  # inbuilt macro
end

# both Range name should be matched
```

```
Range = 1
Range = 2
Range = 3
Range = 4
Range = 5
```

In [112]:

```
for x in 1:10  # combining For loop and If else
    if x % 2 == 0
        @show x
    end
end
```

```
x = 2
x = 4
x = 6
x = 8
x = 10
```

In [113]:

```
for i in 1:10 # evalute expression in the loop
    j = i*10
    println("$(j) is multiplication of $(i) and 10")
end
```

```
10 is multiplication of 1 and 10
20 is multiplication of 2 and 10
30 is multiplication of 3 and 10
40 is multiplication of 4 and 10
50 is multiplication of 5 and 10
60 is multiplication of 6 and 10
70 is multiplication of 7 and 10
80 is multiplication of 8 and 10
90 is multiplication of 9 and 10
100 is multiplication of 10 and 10
```

In [114]:

```
for i in 1:2:10 # traverse range with step size = 2
    print(i," ")
end
```

```
1 3 5 7 9
```

While Loop

In [115]:

```
a = 1
while a < 10
    println(a)
    a = a + 1
end
```

```
1
2
3
4
5
6
7
8
9
```

In [116]:

```
# odd number between 1 to 10 in reverse order
a = 10
while a > 0
    if a%2 == 0
        println(a)
    end
    a = a-1
end
```

```
10
8
6
4
2
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION :
Loops (https://docs.julialang.org/en/v1/manual/variables-and-scoping/#Loops-and-Comprehensions)

Print All Prime Numbers Between 1 to 100

In [117]:

```
answers = [2]
is_valid = false
for number in 3:100
    is_valid = false
    for n in 2:number-1
        if number%n == 0
            is_valid = true
            break
        end
    end
    if is_valid == false
        push!(answers,number)
    end

end
```

In [118]:

```
length(answers)
```

Out[118]:

```
25
```

In [119]:

```
print("Prime Numbers :")
for i in answers
    print(i," ")
end
```

```
Prime Numbers :2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
83 89 97
```

Comprehensions

Another form of loops

In [120]:

```
x = [x for x in 1:10] # simple form of comprehensions
```

Out[120]:

```
10-element Vector{Int64}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

In [121]:

```
x = [I*2 for I in 1:6] # array comprehensions
```

Out[121]:

```
6-element Vector{Int64}:
  2
  4
  6
  8
 10
 12
```

In [122]:

```
x = Set([i for i in 1:6]) # make set
```

Out[122]:

```
Set{Int64} with 6 elements:
  5
  4
  6
  2
  3
  1
```

In [123]:

```
# generate dictionary with alphabet as key and corosponding number as value

alphabet = Dict(string(Char(x + 64)) => x for x in 1:26)
```

Out[123]:

```
Dict{String, Int64} with 26 entries:
  "Z" => 26
  "Q" => 17
  "W" => 23
  "T" => 20
  "C" => 3
  "P" => 16
  "V" => 22
  "L" => 12
  "O" => 15
  "B" => 2
  "M" => 13
  "N" => 14
  "H" => 8
  "A" => 1
  "X" => 24
  "D" => 4
  "G" => 7
  "E" => 5
  "Y" => 25
  "I" => 9
  "J" => 10
  "S" => 19
  "U" => 21
  "K" => 11
  "R" => 18
  "F" => 6
```

In [124]:

```
[(x,y) for x in 0:4,y in 0:4] # making 2d matrix of tuples
```

Out[124]:

```
5×5 Matrix{Tuple{Int64, Int64}}:
 (0, 0)  (0, 1)  (0, 2)  (0, 3)  (0, 4)
 (1, 0)  (1, 1)  (1, 2)  (1, 3)  (1, 4)
 (2, 0)  (2, 1)  (2, 2)  (2, 3)  (2, 4)
 (3, 0)  (3, 1)  (3, 2)  (3, 3)  (3, 4)
 (4, 0)  (4, 1)  (4, 2)  (4, 3)  (4, 4)
```

In [125]:

```
[x for x in 1:20 if x%2==0] # with condition
```

Out[125]:

```
10-element Vector{Int64}:
  2
  4
  6
  8
 10
 12
 14
 16
 18
 20
```

FOR MORE CHECKOUT OFFICIAL DOCUMENTATION : Comprehensions (https://docs.julialang.org/en/v1/manual/variables-and-scoping/#Loops-and-Comprehensions)

String Manipulation

In [127]:

```
string1 = "I Love Julia"
```

Out[127]:

```
"I Love Julia"
```

In [128]:

```
length(string1) # length of string

# note spaces are also calculated
```

Out[128]:

```
12
```

In [129]:

```
lastindex(string1) # last character location in string
```

Out[129]:

```
12
```

Note :

- Both are same but lastindex function require less computation.
- So as a Note Always prefer lasindex() function over length() function.

In [133]:

```
# accessing particular part of string using slice of indexing
println(string1[1])
println(string1[3:6])
println(string1[8:12])
```

```
I
Love
Julia
```

In [134]:

```
isascii(string1) # return true if stinrg have ascii character
```

Out[134]:

```
true
```

In [137]:

```
isascii("αβγ")
```

Out[137]:

```
false
```

In [138]:

```
"Love Julia"*" & Python"  # concatenation using *
```

Out[138]:

```
"Love Julia & Python"
```

In [140]:

```
"Julia "^2 # 2 times julia
```

Out[140]:

```
"Julia Julia "
```

In [141]:

```
split(string1) # split the string by default space seprated
```

Out[141]:

```
3-element Vector{SubString{String}}:
 "I"
 "Love"
 "Julia"
```

In [142]:

```
split("a,b,c,d",",") # comma seprated
```

Out[142]:

```
4-element Vector{SubString{String}}:
 "a"
 "b"
 "c"
 "d"
```

In [143]:

```
split(string1,"e") # split by some character

# make sure splitted character will not include in any splitted substring.
```

Out[143]:

```
2-element Vector{SubString{String}}:
 "I Lov"
 " Julia"
```

In [144]:

```
split(string1,"") # split by character
```

Out[144]:

```
12-element Vector{SubString{String}}:
 "I"
 " "
 "L"
 "o"
 "v"
 "e"
 " "
 "J"
 "u"
 "l"
 "i"
 "a"
```

In [145]:

```
# parse numerical or any other special character to string
# converting numerical to string type

println(typeof("100"))
println(typeof(parse(Int64,"100")))
```

```
String
Int64
```

In [146]:

```
println(typeof("100.5674"))
println(typeof(parse(Float64,"100.5674")))
```

```
String
Float64
```

In [148]:

```
in('I',string1) # "I" is present in string1 or not

# make sure use single quote here.
# alternative use occursin function
```

Out[148]:

```
true
```

In [151]:

```
println(occursin("Love",string1)) # same as in function.
println(occursin("love",string1))
```

```
true
false
```

In [152]:

```
findfirst("L",string1) # when "L" occur first in string1
```

Out[152]:

```
3:3
```

In [153]:

```
findfirst("Love",string1) # when "Love" occur first in string1
```

Out[153]:

```
3:6
```

In [154]:

```
# string replace
# replace(string , "What to replace" => "Replace with what")

replace(string1 ,"Love"=>"Adore")
```

Out[154]:

```
"I Adore Julia"
```

In [155]:

```
string1
```

Out[155]:

```
"I Love Julia"
```

THANK YOU 🎉 🎊

For More Details Do Check Out Julia Official Documentation.
https://docs.julialang.org/en/v1/ (https://docs.julialang.org/en/v1/)