	julia	python	R

System	`pwd() # print working directory`
	`cd("/Users/sswatson") # change directory`
	`readdir() # files and folders in current directory`
		`import os`
		`os.getcwd()`
		`os.chdir("/Users/sswatson")`
		`os.listdir()`
			`getwd()`
			`setwd("/Users/sswatson/")`
			`dir()`

Packages	`# press ] at a Julia prompt for package mode`
	`pkg> add Plots`
	`julia> using Plots`
		`import numpy as np`
		`import matplotlib.pyplot as plt`
		`from sympy import *`
			`install.packages('ggplot2')`
			`library(ggplot2)`

Arithmetic	`x = (1 + 2**3) % 4` `x == 1`	`x <- (1 + 2^3) %% 4` `x == 1`	`x = (1 + 2^3) % 4` `x == 1 # returns true`

Strings	`length("Hello World") # string length`
	`"Hello" * "World" # concatenation`
	`join(["Hello","World"],",") # joining`
	`split("Hello, World",",") # splitting`
	`'H' # single-quotes are for characters, not strings`
		`len('Hello world')`
		`'Hello' + 'World'`
		`','.join(['Hello','World'])`
		`'Hello, World'.split(',')`
		`"Hello, World" # alternate string syntax`
			`nchar('Hello World')`
			`paste('Hello','World')`
			`paste(c('Hello','World'),collapse='')`
			`strsplit('Hello, World',',')`
			`"Hello, World" # alternate string syntax`

Booleans	`true && false == true  # and`
	`false || true == true  # or`
	`!true == false  # not`
		`True and False == False`
		`False or True == True`
		`not True == False`
			`TRUE && FALSE == FALSE`
			`FALSE || TRUE == TRUE`
			`!TRUE == FALSE`

Loops	`for i = 1:10`
	`  print(i)`
	`end`
	
	`while x > 0`
	`  x -= 1`
	`end`
		`for i in range(10):`
		`  print(i)`
		
		`while x > 0:`
		`  x -= 1`
			`for (i in 1:10) {`
			`  print(i)`
			`}`
			
			`while (x > 0) {`
			`  x = x - 1`
			`}`

Conditionals	`if x > 0`
	`  print("x is positive")`
	`elseif x == 0`
	`  print("x is zero")`
	`else`
	`  print("x is negative")`
	`end`
	
	`# ternary conditional`
	`x > 0 ? 1 : -1`
		`if x > 0:`
		`  print('x is positive')`
		`elif x == 0:`
		`  print('x is zero')`
		`else:`
		`  print('x is negative')`
		
		`1 if x > 0 else -1`
			`if (x > 0) {`
			`  print('x is positive')`
			`}`
			`else if (x == 0) {`
			`  print('x is zero')`
			`}`
			`else {`
			`  print('x is negative')`
			`}`
			
			`ifelse(x>0,1,-1)`

Functions	`function f(x,y)`
	`  x² = x * x # \^2[tab] gives the unicode superscript`
	`  x² + sqrt(y*x²+1)`
	`end # -or-`
	`f(x) = x^2 + sqrt(y*x^2 + 1) # -or- (anonymous)`
	`x -> x^2 + sqrt(y*x^2 + 1)`
		`def f(x,y):`
		`  x2 = x * x`
		`  return x2 + (y*x2+1)**(1/2)`
		`# -or-`
		`lambda x: x**2 + (y*x**2+1)**(1/2)`
			`f <- function(x,y) {`
			`  x2 <- x * x`
			`  x2 + sqrt(y*x2+1)`
			`}`

Splatting	`args = [1,2]`
	`kwargs = (tol=0.1,maxiter=100) # a NamedTuple`
	`f(args...;kwargs...) # equiv. to f(1,2;tol=0.1,maxiter=100)`
		`args = [1,2]`
		`kwargs = {'tol':0.1,'maxiter':100} # a dictionary`
		`f(*args,**kwargs) # equiv. to f(1,2,tol=0.1)`
			`library(plyr)`
			`splat(f)(c(1,2)) # equiv. to f(1,2)`

Lists	`myArray = [1,2,"a",[10,8,9]]`
	`myArray[3] == "a"`
	`myArray[4][2] == 8`
	`myArray[end] == [10,8,9]`
	`2 in myArray`
		`myList = [1,2,"a",[10,8,9]]`
		`myList[2] == "a"`
		`myList[3][2] == 9`
		`myList[-1] == [10,8,9]`
		`2 in myList`
			`myList <- list(1,2,"a",list(10,8,9))`
			`myList[3] == "a"`
			`myList[4][2] == 8`
			`myList[length(myList)] # returns list(10,8,9)`
			`2 %in% myList`

A Julia-Python-R reference sheet — Samuel S. Watson

Mapping and filtering

julia

```
# list the subtotals for items with quantity less than 4
fruits = ["apples", "oranges", "pears"]
prices = [1.60, 1.15, 0.85]
quantities = [1, 4, 3]
[(f, p*q) for (f,p,q) in zip(fruits, prices, quantities) if q < 4]
```

python

```
# exactly the same as the Julia code
fruits = ["apples", "oranges", "pears"]
prices = [1.60, 1.15, 0.85]
quantities = [1, 4, 3]
[(f, p*q) for (f,p,q) in zip(fruits, prices, quantities) if q < 4]
```

R

```
# we use Map to replicate Julia and Python's list comprehension with zip
fruits <- c("apples", "oranges", "pears")
prices <- c(1.60, 1.15, 0.85)
quantities <- c(1, 4, 3)
Map(function(f,p,q) list(f,p*q), fruits, prices, quantities)[quantities<4]
```

Ranges

julia

```
range(0,stop=2π,step=0.1) # or 0:0.1:2π
range(0,stop=2π,length=100) # or LinRange(0,2π,100)
collect(0:5) == [0,1,2,3,4,5] # collect a range to get a vector
```

python

```
np.arange(0,stop=2*np.pi,step=0.1)
np.linspace(0,stop=2*np.pi,num=100)
list(range(5)) == [0,1,2,3,4]
```

R

```
seq(0,2*pi,by=0.1)
seq(0,2*pi,length=100)
0:5 == c(0,1,2,3,4,5)
```

Vectors and matrices

julia

```
A = [1 2; 3 4] # matrix with rows [1 2] and [3 4]
b = [1,2] # (column) vector
A' # transpose
size(A) # matrix dimension: (2,2)
A \ b # solve the equation Ax = b
b .> 0 # elementwise comparison
A.^2 # elementwise product
A * A # matrix product
findall(x -> x>0, b) # indices of positive values
fill(2,(10,10)) # 10 x 10 matrix of 2's
I # multiplicative identity
hcat(A,b') # (or [A b']) concatenate side-by-side
vcat(A,b) # (or [A;b']) concatenate vertically
```

python

```
A = np.array([[1,2],[3,4]])
b = np.array([1,2])
np.transpose(A) # or A.T
A.shape
np.linalg.solve(A,b)
b > 0 # elementwise comparison
b**2 # elementwise function application
A @ A # matrix product
np.where(b > 0)
np.full((10,10),2)
np.eye(4) # 4 x 4 identity matrix
np.hstack((A,b[:,np.newaxis]))
np.vstack((A,b))
```

R

```
A <- matrix(c(1,3,2,4),nrow=2) # column-wise!
b <- c(1,2)
t(A)
dim(A)
solve(A,b)
b > 0 # elementwise comparison
A^2 # elementwise product
A %*% A # matrix product
which(b > 0)
matrix(rep(2,100),nrow=10)
diag(4)
cbind(A,b)
rbind(A,b)
```

Slicing

julia

```
A = rand(10,10)
A[1:5,1:2:end] # first five rows, odd-indexed columns
```

python

```
A = np.random.rand(10,10)
A[:5,1::2]
```

R

```
A <- matrix(runif(100),nrow=10)
A[1:5,seq(1,10,by=2)]
```

Random numbers

julia

```
using Random; Random.seed!(1234)
rand(10,10) # matrix with Unif[0,1]'s
randn(10) # vector with N(0,1)'s
rand(10:99) # random two-digit number
```

python

```
np.random.seed(1234)
np.random.rand(10,10)
np.random.randn(10)
np.random.randint(10,100)
```

R

```
set.seed(1234)
matrix(runif(100),nrow=10)
rnorm(10)
sample(10:99,1)
```

Data frames

julia

```
using DataFrames, FileIO
myDataFrame = DataFrame(load("data.csv"))
save("mydata.csv",myDataFrame)

# Language-integrated query
using DataFramesMeta, RCall # get nycflights13 data from R
flights = rcopy(R"nycflights13::flights")
@linq flights |>
    where(:month .== 1, :day .< 5) |> # filter rows
    orderby(:day,:distance) |> # sort rows
    select(:month, :day, :distance, :air_time) |> # select columns
    transform(speed = :distance ./ :air_time * 60) |> # new columns
    by(:day, avgspeed = mean(skipmissing(:speed))) # split-apply-combine
```

python

```
import pandas as pd
myDataFrame = pd.read_csv("data.csv")
myDataFrame.to_csv("mydata.csv")

from rpy2.robjects import r, pandas2ri
from rpy2.robjects.packages import importr
importr('nycflights13'); flights = pandas2ri.ri2py(r['flights'])
flights.query('month == 1 & day < 5') \
       .sort_values(['day','distance']) \
       [['month','day','distance','air_time']] \
       .assign(speed = flights.distance/flights.air_time * 60) \
       .groupby('day') \
       .agg({'speed':'mean'})
```

R

```
myDataFrame = read.csv("data.csv")
write.csv(myDataFrame,"mydata.csv")

library(dplyr); library(nycflights13)
flights %>%
    filter(month == 1, day < 5) %>%
    arrange(day, distance) %>%
    select(month, day, distance, air_time) %>%
    mutate(speed = distance / air_time * 60) %>%
    group_by(day) %>%
    summarise(avgspeed = mean(speed,na.rm=TRUE))
```

Plotting

julia

```
using StatsPlots
# select the rows with an air_time value and plot a histogram
(@linq flights |> where((!ismissing).(:air_time)))
    |> @df histogram(:air_time)
# scatter plot (using the first 10,000 records)
flights[1:10^4,:] |> @df scatter(:air_time,:distance,group=:carrier)
```

python

```
import seaborn as sns
# histogram
sns.distplot(flights['air_time'].dropna())
# scatter plot
sns.pairplot(flights,x_vars='air_time',y_vars='distance',hue='carrier',
             plot_kws={'alpha': 0.2})
```

R

```
library(ggplot2)
# aesthetic mapping: connects data to visual elements (x, y, size, color)
# geom: geometric object used to represent data (point, line, bar)
# geom functions return layers that you add to a ggplot
ggplot(data = flights) +
    geom_point(mapping=aes(x=air_time,y=distance,color=carrier),alpha=0.2)
```

Optimization

julia

```
using Optim
rosenbrock(x) =  (1.0 - x[1])^2 + 100.0 * (x[2] - x[1]^2)^2
result = optimize(rosenbrock, zeros(2), BFGS())
```

python

```
from scipy.optimize import minimize
def rosenbrock(x):
    return (1-x[0])**2 +100*(x[1]-x[0]**2)**2
minimize(rosenbrock,[0,0],method='BFGS')
```

R

```
rosenbrock <- function(x) {
    (1-x[1])^2 +100*(x[2]-x[1]^2)^2
}
optim(c(0,0), rosenbrock, method = "BFGS")
```

Root finding

julia

```
using Roots
f(x) = exp(x) - x^4
find_zero(f,3)
```

python

```
import numpy as np
from scipy.optimize import root
def f(x):
    return np.exp(x[0]) - x[0]**4
root(f, [0])
```

R

```
f <- function(x) {
    exp(x) - x^4
}
uniroot(f,c(0,3))
```