System

```julia
pwd() # print working directory
cd("/Users/sswatson") # change directory
readdir() # files and folders in current directory
```

```python
import os
os.getcwd()
os.chdir("/Users/sswatson")
os.listdir()
```

```r
getwd()
setwd("/Users/sswatson/")
dir()
```

Packages

```julia
# press ] at a Julia prompt for package mode
pkg> add Plots
julia> using Plots
```

```python
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
```

```r
install.packages('ggplot2')
library(ggplot2)
```

Arithmetic

```julia
x = (1 + 2^3) % 4
x == 1 # returns true
```

```python
x = (1 + 2**3) % 4
x == 1
```

```r
x <- (1 + 2^3) %% 4
x == 1
```

Strings

```julia
length("Hello World") # string length
"Hello" * "World" # concatenation
join(["Hello","World"],",") # joining
split("Hello, World",",") # splitting
'H' # single-quotes are for characters, not strings
```

```python
len('Hello world')
'Hello' + 'World'
','.join(['Hello','World'])
'Hello, World'.split(',')
"Hello, World" # alternate string syntax
```

```r
nchar('Hello World')
paste('Hello','World')
paste(c('Hello','World'),collapse='')
strsplit('Hello, World',',')
"Hello, World" # alternate string syntax
```

Booleans

```julia
true && false == true  # and
false || true == true  # or
!true == false  # not
```

```python
True and False == False
False or True == True
not True == False
```

```r
TRUE && FALSE == FALSE
FALSE || TRUE == TRUE
!TRUE == FALSE
```

Loops

```julia
for i = 1:10
    print(i)
end

while x > 0
    x -= 1
end
```

```python
for i in range(10):
    print(i)

while x > 0:
    x -= 1
```

```r
for (i in 1:10) {
    print(i)
}

while (x > 0) {
    x = x - 1
}
```

Conditionals

```julia
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
```

```python
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
```

```r
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

```julia
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)
```

```python
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)
```

```r
f <- function(x,y) {
  x2 <- x * x
  x2 + sqrt(y*x2+1)
}
```

Splatting

```julia
args = [1,2]
kwargs = (tol=0.1,maxiter=100) # a NamedTuple
f(args...;kwargs...) # equiv. to f(1,2;tol=0.1,maxiter=100)
```

```python
args = [1,2]
kwargs = {'tol':0.1,'maxiter':100} # a dictionary
f(*args,**kwargs) # equiv. to f(1,2,tol=0.1)
```

```r
library(plyr)
splat(f)(c(1,2)) # equiv. to f(1,2)
```

Lists

```julia
myArray = [1,2,"a",[10,8,9]]
myArray[3] == "a"
myArray[4][2] == 8
myArray[end] == [10,8,9]
2 in myArray
```

```python
myList = [1,2,"a",[10,8,9]]
myList[2] == "a"
myList[3][2] == 9
myList[-1] == [10,8,9]
2 in myList
```

```r
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
```

Mapping and filtering

julia

```
# Even perfect squares up to 10^2:
[x^2 for x=1:10 if x % 2 == 0]
# -or-
square(x) = x^2
square.(filter(iseven,1:10))
```

python

```
[x**2 for x in range(1,11) if x % 2 == 0] # -or-
map(lambda x: x**2,filter(lambda x: x % 2 == 0,range(1,11)))
```

R

```
A <- sapply(1:10,function(x) {x^2})
A[A %% 2 == 0]
```

Ranges

julia

```
range(0,stop=2π,step=0.1)
range(0,stop=2π,length=100)
0:5:20 == [0,5,10,15,20]
```

python

```
np.arange(0,stop=2*np.pi,step=0.1)
np.linspace(0,stop=2*np.pi,num=100)
```

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]
b = [1,2]
A'
size(A)
A \ b
b .> 0 # elementwise comparison
A.^2 # elementwise product
A * A # matrix product
findall(x -> x>0, b) # indices of true values
fill(2,(10,10)) # 10 x 10 matrix of 2's
I # multiplicative identity
hcat(A,b') # stack side by side
vcat(A,b) # stack 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)

using DataFramesMeta, Feather
Feather.read("flights.feather") # see R column to write this file
@linq flights |>
    where(:month .== 1, :day .< 5) |>
    orderby(:day,:distance) |>
    select(:month, :day, :distance, :air_time) |>
    transform(speed = :distance ./ :air_time * 60) |>
    by(:day, avgspeed = mean(skipmissing(:speed)))
```

python

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

import feather
flights = feather.read_dataframe("flights.feather")
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({'day':'mean'})
```

R

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

library(dplyr); library(nycflights13)
flights %>%
    filter(month == 1, day < 5) %>% # filter rows
    arrange(day, distance) %>% # sort by day and distance
    select(month, day, distance, air_time) %>% # select columns
    mutate(speed = distance / air_time * 60) %>% # add a column
    group_by(day) %>% # group by day
    summarise(avgspeed = mean(speed,na.rm=TRUE)) # collapse columns
library(feather) # write flight data to disk for Python & Julia
write_feather(flights,"flights.feather")
```

Plotting

julia

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

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))
```