Setting up Julia

Download and install

- Go to http://julialang.org/downloads/ and download whatever is compatible with your machine.
- Once downloaded, run it and play around with it typing simple arithmetic equations to check that everything works.

```
julia> 1+1
2
julia> 6/3
2.0
julia> println("hello world")
hello world
julia> versioninfo()
```

Set your working directory

- You can check your default home directory by:

```
julia> homedir()
"/Users/hudanassar"
```

- You can check your current working directory by:

```
julia> pwd()
"/Users/hudanassar"
```

- You can use the command 'cd' to change the working directory:

```
julia> cd("$(homedir())/Documents/Testing_Julia")
julia> pwd()
"/Users/hudanassar/Documents/Testing_Julia"
```

Start julia from the terminal

- Open your terminal, and type:

```
julia
```

If julia was not previously installed on your machine, you will get the following error:

```
-bash: julia: command not found
```

- Now, try:

```
/Applications/Julia-0.3.7.app/Contents/Resources/julia/bin/julia
```

- But you obviously don't want to type this line every time you want to start julia, here's how you set your PATH:

```
edit ~/.bash_profile
```

- Add the following line:

```
export PATH="/Applications/Julia-0.3.7.app/Contents/Resources/
julia/bin:$PATH"
```

- Save and close. On terminal, type:

```
source $HOME/.bash_profile
```

- And to check if things changed type:

```
echo $PATH
```

- Now you should see julia in your path. Finally, type:

```
julia
```

Write a simple program

- You can write programs using the terminal, here's an example:

```
echo 'for x in ARGS; println("Hello ", x, ", how are you doing
today?"); end' > script.jl
julia script.jl Huda Julia
```

- This will output:

```
Hello Huda, how are you doing today?
```

```
Hello Julia, how are you doing today?
```

HW1a (Programming Part of CS 515 HW1a)

- link to homework: CS 515 HW1a

Watch out for:

- Packages: you need to install a package if you need it
  ```
  Pkg.add("pkg-name")
  using "pkg-name"
  ```
  More package commands:
  ```
  Pkg.status()
  Pkg.installed()
  Pkg.update()
  ```
- Preallocation of size:
  In Matlab you can do the following. In Julia, you can't
  ```
  A = rand(4,1);
  A(5) = 10;
  ```
- Location of a function matters:
  Like Matlab, you can write nested function and you can write functions in separate .jl files.
  - If writing in the same .jl file, functions come first so that you can call them
  - If writing in a separate .jl file, you have to include the function
- To refer to the i'th index in a vector v, you use square brackets:
  ```
  v[i]
  ```
- Unlike Matlab, strings come in double quote
- Types are important, here's one code example that **won't work**
  ```
  v = zeros(4,1);
  m = 6/2;
  v[m] = 2;
  ```
  That's because m is of type Float64 and indices should be of type Int64. Here's what you do:
  ```
  v = zeros(4,1);
  m = div(6,2);
  v[m] = 2;
  ```
- You can use Greek letters!

Useful links:

- Linear Algebra functions
- Functions
- Arrays
- Julia Package Ecosystem Pulse
- Juno

- Matlab to Julia

Additional Task:

Ran the ppr_push code in Matlab, Julia and C++. Here are the results:

- C++:

```
pal-nat184-143-197:julia_files hudanassar$ ./a.out
Elapsed 0.0527811 seconds.
True = 0.026124014066724
x[0] = 0.026124014066724
y[0] = 0.026124014066724
```

- Julia:

```
julia> include("pprpush\_perf4.jl")
elapsed time: 0.049447485 seconds
elapsed time: 0.014877667 seconds
elapsed time: 0.019107745 seconds (23576 bytes allocated)
True = 0.026124014066724
x\[0] = 0.026124014066724
```

- Matlab:

```
0.069144 seconds elapsed
true value x(1) = 0.026124014066724
comp value x(1) = 0.02612401406672400
```