Northwestern
INFORMATION TECHNOLOGY

Julia – Week 1 Day 1

Sophie van Genderen, Assoc Computational Specialist
Julia Giannini, Computational Specialist

Northwestern IT Research Computing and Data Services

November 18th 2025

Covered in today's class:

- Why Julia?
- Comparing Python to Julia
- Demo
- Intro to Quest
- Starting up Julia on Quest

Julia

- **Why Julia?**
- Comparing Python to Julia
- Demo
- Intro to Quest
- Starting up Julia on Quest

Why Julia?

- Superior performance for numerical analysis and scientific computing because of “Just-In-Time” (JIT) compilation vs. Interpreted
  - Compiled vs. Interpreted languages
- Built-in parallelism, great for heavy computations
- No external libraries needed for Mathematics
- Data Models

Julia

- Why Julia?
- **Comparing Python to Julia**
- Demo
- Intro to Quest
- Starting up Julia on Quest

Comparing Python to Julia

Key Indicator	Julia	Python
Maturity	Created in 2012	Created in 1991
Scope	General-purpose, but data-oriented	General-purpose and used for almost everything
Language Type	High Level, (Just in Time) Compiled	High Level, Interpreted
Typing	Dynamically-typed language, but also offers the ability to specify types (Static)	Dynamic, the type for a variable is decided at runtime
Open-source	Yes	Yes
Usage	Data Science and Machine Learning – especially work with data models	Mobile/web Dev, AI, Data Science, web scripting, game development, security ops.
Data Science	Math functions are easy to write and understand – no external libraries are needed for math functions	Requires NumPy or other libraries for advanced math
Performance	Fast development and production, high speed runtime, can handle millions of data threads	Fast for development, slow for production

Comparing Python to Julia

- Indexing
- Loops
- Functions / Methods
- Installing / Loading / Using external packages
- Dictionaries
- Global
- Modules
- Classes

Comparing Python to Julia

- **Indexing**
- Loops
- Functions / Methods
- Installing / Loading / Using external packages
- Dictionaries
- Global
- Modules
- Classes

Comparing Python to Julia - **Indexing**

- Julia: 1, 2, 3 (indexing is 1-based)
- Python: 0, 1, 2 (indexing is 0-based)

Python 

```
list_numbers = [1,
2, 5, 10, 4, 9, 7, 12, 9]
element = 10
list_numbers.index(element)

> 3
```

Julia 

```
str = "Hello, world!"
println(str[1])
println(str[4:9])

> H
> lo, wo
```

Comparing Python to Julia

- Indexing
- **Loops**
- Functions / Methods
- Installing / Loading / Using external packages
- Dictionaries
- Global
- Modules
- Classes

Comparing Python to Julia - **Loops**

Python

Loops

```
while
for

(if/else)
```

Control Statements

```
continue
break

(pass)
```

Julia

Loops

```
while
for

(foreach)
```

Control Statements

```
continue
break
```

Comparing Python to Julia - **Loops**

Python 

```
for i in range(10):
        i += 5
        print(i)
```

```
for i in range(5):
    if i % 2 == 0:
        print(f"{i} is even")
    else:
        pass  # Do nothing for
odd numbers
```

Julia 

```
for i in 1:10
        i += 5
        print(i)
end
```

```
foreach(i -> begin
    if i % 2 == 0
        println("$i is
even")
    end
end, 0:4)
```

https://docs.julialang.org/en/v1/manual/control-flow/#man-loops

Comparing Python to Julia

- Indexing
- Loops
- **Functions / Methods**
- Installing / Loading / Using external packages
- Dictionaries
- Global
- Modules
- Classes

Comparing Python to Julia - **Functions**

Python

```
def new_function():
    print("Hello!")

new_function()

def name_function(name):
    print("Hello, " + name +
"!")

name_function("James")
```

Julia

```
new_function(x,y)
    x + y
end


new_function(2,3)
```

==

```
f(x,y) = x + y


f(2,3)
```

https://docs.julialang.org/en/v1/manual/functions/

Comparing Python to Julia - **Methods**

Python

A method in Python is a function that belongs to an object/class.

```
class C:
    def my_method(self):
        print("I am a C")

c = C()
c.my_method()   # Prints("I am a C")
```

Julia

A method in Julia is the implementation of **multiple dispatch**, which allows the call/execution of different definitions of a function based on the data types of the arguments.

(example in Jupyter Notebook later)

https://docs.julialang.org/en/v1/manual/methods/

Comparing Python to Julia

- Indexing
- Loops
- Functions / Methods
- **Installing / Loading / Using external packages**
- Dictionaries
- Global
- Modules
- Classes

Comparing Python to Julia

- Indexing
- Loops
- Functions / Methods
- Installing / Loading / Using external packages
- **Dictionaries**
- Global
- Modules
- Classes

Comparing Python to Julia - **Dictionaries**

Python 

Creating an empty dictionary:

```
dict_ex = {}
```

Creating a filled dictionary with different types:

```
dict_ex = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
print(type(dict_ex)) ⟶ class 'dict'
```

`dict()` method to make a dictionary:

```
dict_ex = dict(brand= "Ford",
electric= False, year= 1964, colors
= ["red", "white", "blue"])
```

Referring to values using keys:

```
print(dict_ex["brand"])
> Ford
```

`.keys()` returns the keys through a dict_keys object.
`.values()` returns the values through a dict_values object.
`.items()` returns both the keys and values through a dict_items object.

Comparing Python to Julia - **Dictionaries**

Julia 

Creating an Empty dictionary:

```
Dict1 = Dict()
println("Empty Dictionary = ", Dict1)
```

Creating an Untyped Dictionary:

```
Dict2 = Dict("a" => 101, "b" => 102,
"c" => "Hello")
println("\nUntyped Dictionary = ",
Dict2)
```

Creating a Typed Dictionary:

```
Dict3 = Dict{String, Integer}("a" =>
101, "c" => 102)
println("\nTyped Dictionary = ",
Dict3)
```

Accessing dictionary values using keys:

```
println(Dict2["b"])
println(Dict2["c"])
```

Creating a Dictionary with Integer keys:

```
Dict2 = Dict(1 => 10, 2 => 20, 3
=> "Geeks")
println(Dict2[1])
println(Dict2[3])
```

Creating a Dictionary with Symbols (immutable):

```
Dict3 = Dict(:a => 1, :b => "one")
println(Dict3[:b])
```

`Keys = keys(Dictionary_name)` Returns all the keys of the dictionary

`Values = values(Dictionary_name)` Returns all the values of the dictionary

https://docs.julialang.org/en/v1/base/collections/

Comparing Python to Julia

- Indexing
- Loops
- Functions / Methods
- Installing / Loading / Using external packages
- Dictionaries
- **Global**
- Modules
- Classes

Comparing Python to Julia – **Global variables**

Python

Any variable defined outside the scope of a function is a global variable.

To create or modify the value of a global variable inside the scope of a function, use the keyword `global`

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

The original value of the global variable `x` is `awesome`. This returns:

```
Python is fantastic
```

Comparing Python to Julia – **Global variables**

Julia 

It's complicated, and has to do with the fact that Julia is JIT compiled.

A `global` variable is *reasonable* if it's a constant (`const`). It is *borderline reasonable* if it is a constant type.

https://gist.github.com/flcong/2eba0189d7d3686ea9633a6d14398931
https://docs.julialang.org/en/v1/manual/performance-tips/#Avoid-untyped-global-variables
https://docs.julialang.org/en/v1/manual/variables-and-scoping/#man-typed-globals

```
global x::Int = 1000
global const z::Float64 = rand(1)[1]
## This is nice for the compiler 🙂

function my_func()
    ## stuff
end
```

Will lead to (much) better performance than

```
x = 1000
y = rand(1)[1]
## If one of these changes type
throughout the program, it will be
difficult for the compiler 😫

function my_func()
    ## stuff
end
```

Comparing Python to Julia

- Indexing
- Loops
- Functions / Methods
- Installing / Loading / Using external packages
- Dictionaries
- Global
- **Modules**
- Classes

Comparing Python to Julia – **Modules** (Python)

In the command line:

```
[user@computer ~/py_dir]$ python
./load_fibo.py
0 1 1 2 3 5 8
Done
```

Python caches the compiled version of each module in the `__pycache__` directory

```
[user@computer ~/py_dir]$ ls
fibo.py  load_fibo.py  __pycache__
```

https://docs.julialang.org/en/v1/manual/modules/
https://docs.python.org/3/tutorial/modules.html

Definition of module - **`fibo.py`**

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series
up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

Use the module - **`load_fibo.py`**

```
# load my module
import fibo

# call a function defined in that
module
fibo.fib(10)
print('done')
```

Comparing Python to Julia – **Modules** (Julia)

In the command line:

```
[user@computer ~/julia_dir]$ julia
./load_Fibo.jl
0 1 1 2 3 5 8
done
```

Julia does not cache compiled versions of modules

```
[user@computer ~/julia_dir]$ ls
Fibo.jl  load_Fibo.jl
```

https://docs.julialang.org/en/v1/manual/modules/
https://docs.python.org/3/tutorial/modules.html

Definition of module - **`Fibo.jl`**

```
module Fibo

function fib(n::Int) # write Fibonacci series
    a, b = 0, 1
    while a < n
        print(a, " ")
        a, b = b, a + b
    end
    println()
end

# End of module
end
```

Use the module - **`load_Fibo.jl`**

```
# load my module
include("Fibo.jl")
using .Fibo

# call a function defined in that module
Fibo.fib(10)
println("done")
```

Comparing Python to Julia

- Indexing
- Loops
- Functions / Methods
- Installing / Loading / Using external packages
- Dictionaries
- Global
- Modules
- **Classes**

Comparing Python to Julia - **Classes**

Python

```
class Person:
  def __init__(pyobject, name, age):
    pyobject.name = name
    pyobject.age = age

  def myfunction(x):
    print("Hello my name is " + x.name)

p = Person("Matthew", 46)
p.myfunction()

>> Hello my name is Matthew
```

Julia

Julia has no class!

Seriously though, Julia isn't an object-oriented language. Using structs, generic functions, and constructors you can create something class and object-like but it's a pain.

If an OOP paradigm is what you seek, look elsewhere.

Julia

- Why Julia?
- Comparing Python to Julia
- **Demo**
- Intro to Quest
- Starting up Julia on Quest

Live Demo!

Live Demo will Cover…

- Python vs. Julia vs. Python w/ Numba
- Python vs. Julia Linear Regression
- Python vs. Julia DataFrames

Note: we will use the **Python runtime notebook** to compare the performance of Julia and Python – follow along if you like

Julia

- Why Julia?
- Comparing Python to Julia
- Demo
- **Intro to Quest**
- Starting up Julia on Quest

Quest

What is Quest

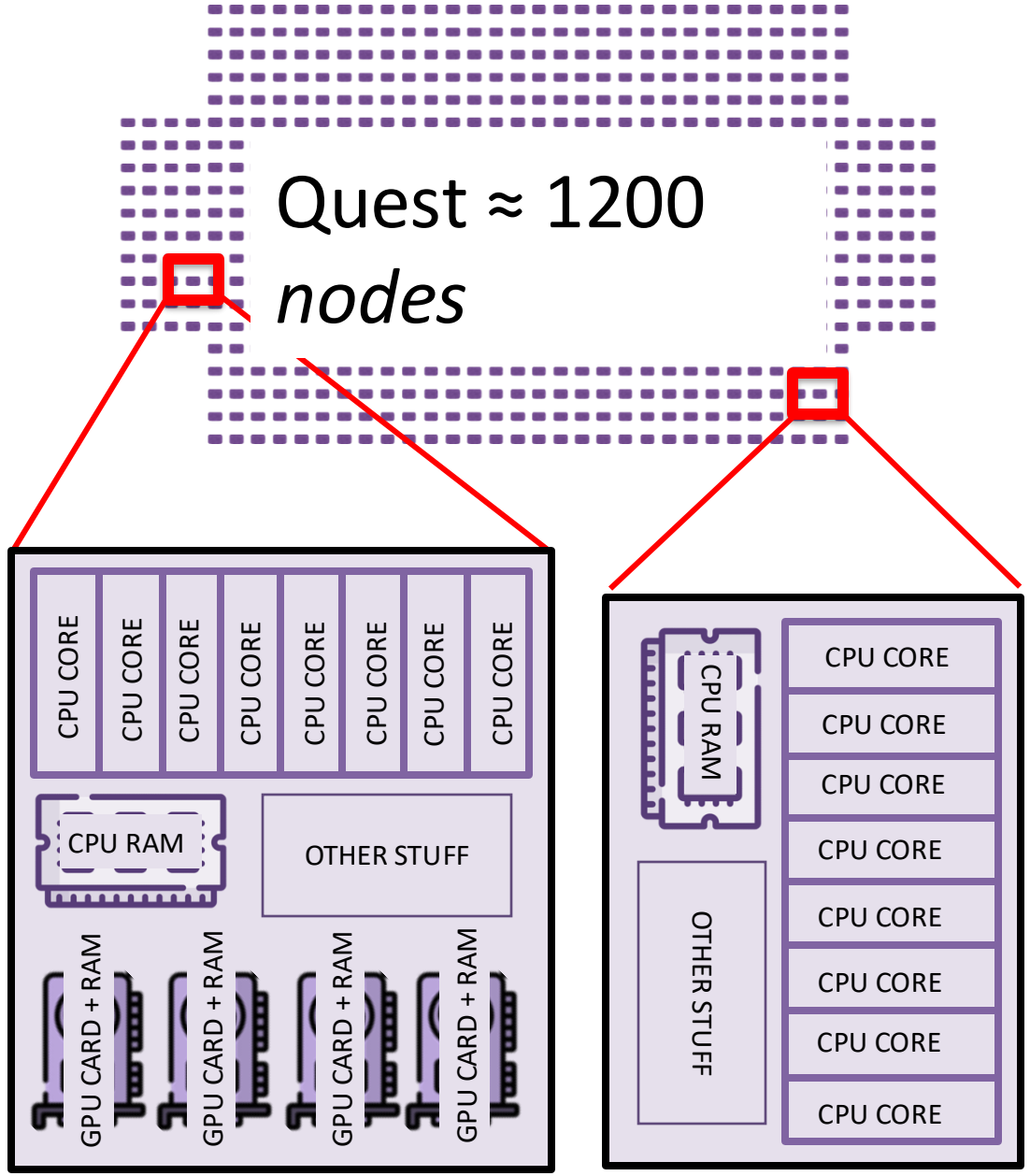
- High-performance computing (HPC) resources for compute and data intensive research
- Available to faculty, staff, and studens for research purposes
- Quest User Guide: https://rcdsdocs.it.northwestern.edu/

Computing Resources

- Nodes have:
  - Cores (CPUs and GPUs)
  - Memory
  - Networking (PCIe | NVM | SXM)
- There are different versions of nodes:
  - Quest10 | Quest11 | Quest12 | Quest13

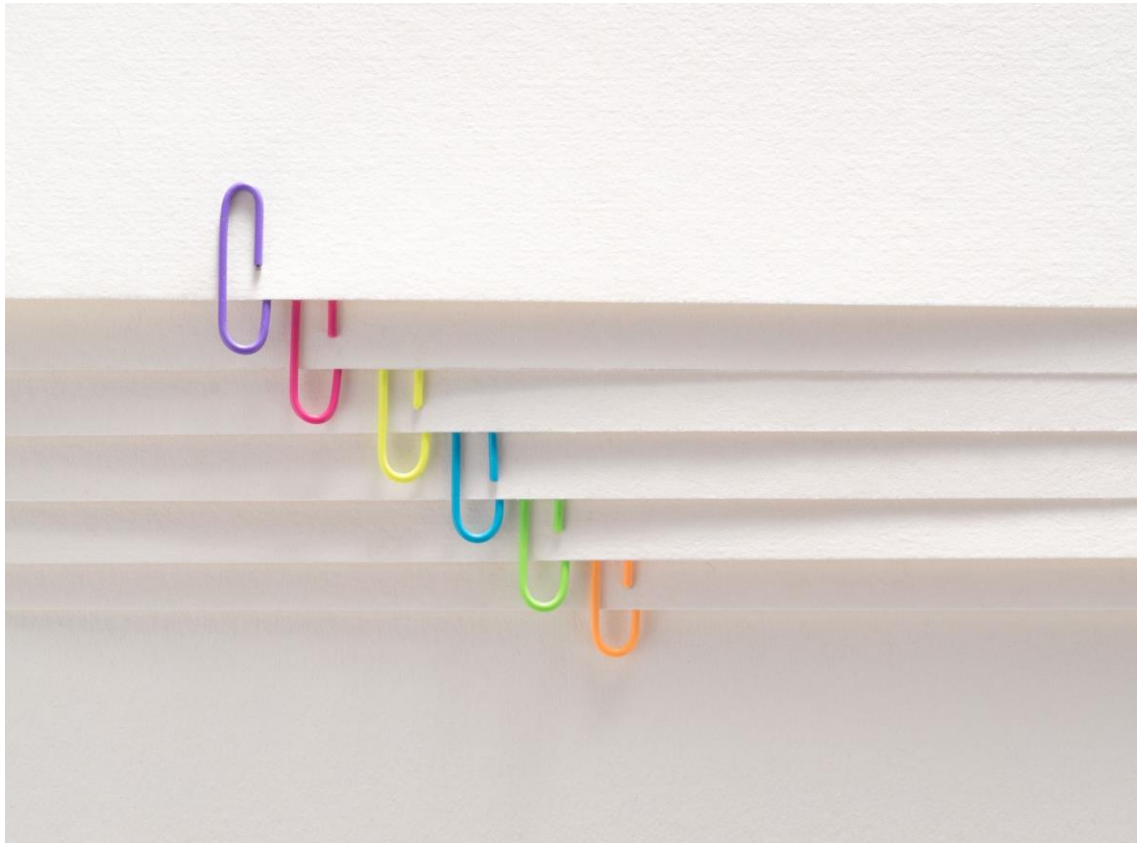
Computing Resources

- Each node has *schedulable resources:*
  - 40-128 *CPU* cores
  - 192-512 GB of *CPU RAM* (total)
  - 0-4 *GPU* cards



Working Directories

- /home - all users
  - 80 GB
  - Only accessible by the user
  - Backed up!
- /projects
  - General access and classrooms
  - /e33102
  - Not backed up!



Job Types

- Batch job
  - Jobs that are submitted via script
  - Good for: code that has a longer runtime/more resources
  - Example: Slurm
- Interactive job
  - Jobs that have a Graphical User Interface (GUI)
  - Good for: debugging and prototyping
  - Examples: RStudio, JupyterHub, SAS, Matlab

Connecting to Quest

- ssh
  - Done via terminal
  - ssh <netid>@login.quest.northwestern.edu

Julia

- Why Julia?
- Comparing Python to Julia
- Demo
- Intro to Quest
- **Starting up Julia on Quest**

Log into Quest and Load Julia

- ssh to Quest

```
[netid@quser]$ module load julia/1.11.4
[netid@quser]$ julia


 _       _ _(_)_     |  Documentation: https://docs.julialang.org
(_)     | (_) (_)    |
 _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
| | | | | | |/ _` |  |
| | |_| | | | (_| |  |  Version 1.11.4 (2025-03-10)
_/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |


julia>
```

Thank You!

Questions about Quest? Email us at: quest-help@northwestern.edu