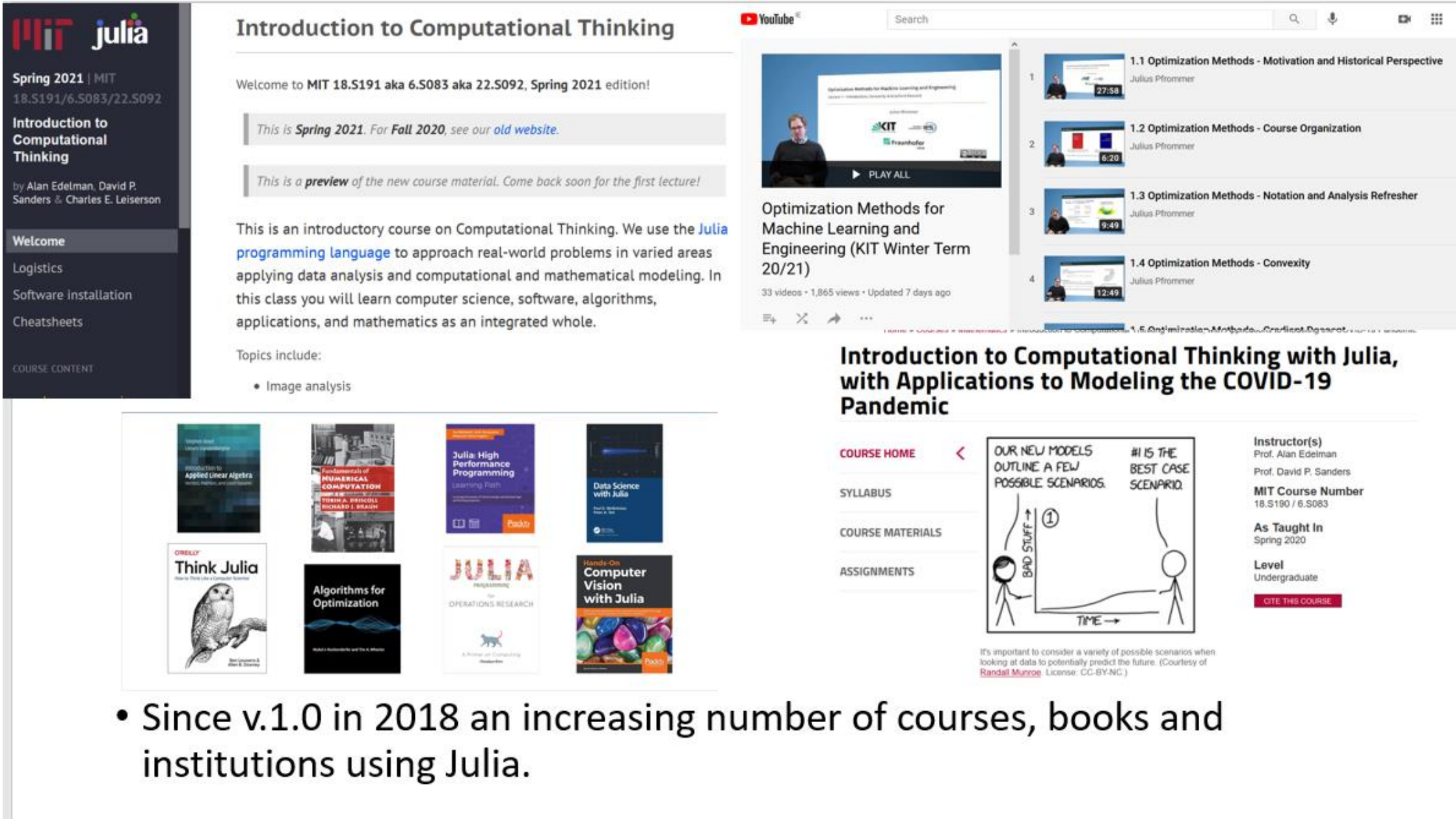
“Looks like Python, runs like C”

- a quick introduction to Julia for Python users

#SISTEM2021 Conference



- Since v.1.0 in 2018 an increasing number of courses, books and institutions using Julia.

My favorite slide…

A quick warning:

- Julia does not have classes, strictly speaking.
- Compared to Python, this means more use of functions that are not built as part of a class/object.
- E.g. some operations on strings: Julia vs Python

```julia
test_string = "Julia is a Cool Language"

findfirst("i", test_string)

lowercase(test_string)

occursin("Julia", test_string)
```

```python
test_string = "Python is a Cool Language"

test_string.find("y")

test_string.lower()

print("Python" in test_string)
```

Some primitive operations: Julia vs Python

```
parse(Int64, "42")
parse(Float64, "123.345")
string(123)
```

```
int("42")
float("123.345")
str(1210)
```

- Converting Strings to Integers or Floats is a little different to Python. We use parse() and pass the type we'd like to convert our string to. Conversion to string is much the same as Python.

```
println(typeof('a'))
println(typeof("a"))
```

```
Char
String
```

- **Note:** Single quotation marks are not strings in Julia, instead they are individual characters (or "chars").

Arrays and Dictionaries:

```
arr1 = [3, 5, 9]
length(arr1)
maximum(arr1)
```

- Some operations are much the same as in Python, i.e. max, len etc.

```
arr2 = ["Hello", 21, 4.2]
push!(arr2, "34")
println("Newly appended array: ", arr2)

arr3 = [3, 5, -1]
sort!(arr3)
println("Sorted array: ", arr3)
```

```
Newly appended array: Any["Hello", 21, 4.2, "34"]
Sorted array: [-1, 3, 5]
```

- However, lists are added to using push! Remember: there are no built-in methods like list.append(). Sorting behaves similarly: sorting the list in-situ.

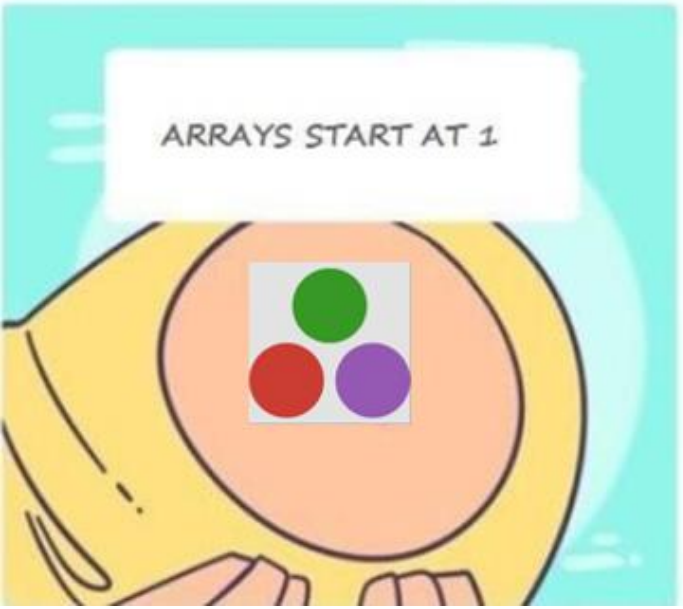
```
phonebook = Dict("John" => "555", "Alice" => "111", "Bob" => "333")

# We can retrieve keys using the keys() function. Ditto for values
println("Keys: ", keys(phonebook))
println("Values: ", values(phonebook))
```

- Tuples and dictionaries behave similarly, again using functions over methods.
- In some ways, the behavior is more consistently applied than in Python.

But most importantly....

Arrays start at 1!!!




```
letters    = ["a", "b", "c", "d"]
println(letters[1:end])
println(letters[2:3])
```

```
["a", "b", "c", "d"]
["b", "c"]
```

Functions & Loops:

```
function welcome(name)
    println("Hello $name")
end
welcome("Katya")

f(x, y) = println(x * y)
f(3, 5)
```

```
Hello Katya
15
```

```
for i = 1:5, j = 1:5
    println(i, " : ", j)
end

i = 1
while i <= 5
    println(i)
    i = i + 1
end
```

- Julia allows for very minimal one liner functions. More complex ones are similar to Python, but with an end block.
- Loops follow a similar convention of minimal syntax.

A study in Julia vs Python: a part of a statistical spelling corrector

```julia
words(text::String) = collect(m.match for m in eachmatch(r"\w+", text))

WORDS = countmap([word for word in words(x)])

"""Probability of a given 'word'."""
P(word::String, N::Float64 = sum(values(WORDS))) = WORDS[word] / N
```

- Julia

```python
def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open(x).read()))

def P(word, N=sum(WORDS.values())):
    """Probability of `word`."""
    return WORDS[word] / N
```

- Python

References:

- Getting started with Julia: https://julialang.org/learning/
- Optimisation methods for Machine Learning and Engineering: https://www.youtube.com/playlist?list=PLdkTDauaUnQpzuOCZyUUZc0lxf4-PXNR5
- MIT Introduction to Computational Thinking with Julia: https://computationalthinking.mit.edu/Fall20/

- GitHub repo for slides and some code: https://github.com/RobinsonLuzo/Talks/tree/master/SISTEM2021_Intro_to_Julia
- Dublin Julia Users Group: https://www.meetup.com/Dublin-Julia-Users-Group/