Julia basics
Using libraries

Using libraryname
Using libraries

Using libraryname

For example:

Using JuMP
Using Cbc
Output

Simplest form:

`println(expression, expression, expression, ...)`
Simplest form:

\texttt{println(expression, expression, expression, ...)}

Any of the \texttt{expressions} can be a constant string:

\texttt{println("the value of myvariable is: ", myvariable)}
Simplest form:

```java
println(expression, expression, expression, ...)
```

Any of the `expression`s can be a constant string:

```java
println("the value of myvariable is: ", myvariable)
```

C-style output:

**Using** `Printf`

```c
@printf(format, ...)
```
if condition
    ...
elseif condition
    ...
else
    ...
end
Loops

```plaintext
for i = set
    ...
end
```
Loops

for i in set
    ...
end
Loops

for i ∈ set
   ...
end
Loops

for i ∈ set
  ...
end

Typically, set would be a:b, yielding \{a, a + 1, \ldots, b - 1, b\}. 
Loops

```plaintext
for i ∈ set
    ...
end
```

Typically, `set` would be `a:b`, yielding `{a, a + 1, ..., b − 1, b}`.

For example `1:5` yields `{1, 2, 3, 4, 5}`.
Declaring model variables

With implicit indices
@variable(model, x[set, set, ...])

For example:
@variable(model, x[1:M, 1:N])

With explicit indices:
@variable(model, x[i in set, j in set, ...])

For example:
@variable(model, x[i in 1:M, j in 1:N])
Declaring model variables

With implicit indices
@variable(model, x[set, set, ...])

for example:
@variable(model, x[1:M, 1:N])
Declaring model variables

With implicit indices
@variable(model, x[set, set, ...])
for example:
@variable(model, x[1:M, 1:N])

With explicit indices:
@variable(model, x[i in set, j in set, ...])
Declaring model variables

With implicit indices
@variable(model, x[set, set, ...])

for example:
@variable(model, x[1:M, 1:N])

With explicit indices:
@variable(model, x[i in set, j in set, ...])

for example:
@variable(model, x[i in 1:M, j in 1:N])
Declaring model variables

When using explicit indices, we can add a condition:
@variable(model, x[i in set, j in set, ... ; condition])
Declaring model variables

When using explicit indices, we can add a condition:
@variable(model, x[i in set, j in set, ... ; condition])
for example:
@variable(model, x[i in 1:M, j in 1:N ; i < j])
Declaring model variables

Adding bounds directly with variable declaration:
@variable(model, lb <= x[...] <= ub)

Example: Lower bound:
@variable(model, x[i in 1:M] >= 0)

Example: Upper bound:
@variable(model, x[i in 1:M] <= 100)

Example: Lower and upper bounds:
@variable(model, 0 <= x[i in 1:M] <= 100)
Declaring model variables

Adding bounds directly with variable declaration:
@variable(model, lb <= x[...] <= ub)

Example: Lower bound:
@variable(model, x[i in 1:M] >= 0)

Example: Upper bound:
@variable(model, x[i in 1:M] <= 100)

Example: Lower and upper bounds:
@variable(model, 0 <= x[i in 1:M] <= 100)
Declaring model variables

Adding bounds directly with variable declaration:
```
@variable(model, lb <= x[...] <= ub)
```

Example: Lower bound:
```
@variable(model, x[i in 1:M] >= 0)
```

Example: Upper bound:
```
@variable(model, x[i in 1:M] <= 100)
```
Declaring model variables

Adding bounds directly with variable declaration:
@variable(model, lb <= x[... ] <= ub)

Example: Lower bound:
@variable(model, x[i in 1:M ] >= 0)

Example: Upper bound:
@variable(model, x[i in 1:M ] <= 100)

Example: Lower and upper bounds:
@variable(model, 0 <= x[i in 1:M ] <= 100)
Declaring model variables

Integer variables:
@variable(model, x[...], Int)

Binary (i.e. \{0, 1\}) variables:
@variable(model, x[...], Bin)

Example:
@variable(model, x[i in 1:M] >= 0, Int)
Note that @variable(model, 0 <= x[i in 1:M] <= 1, Int) is equivalent to @variable(model, x[i in 1:M], Bin)
Declaring model variables

Integer variables:
@variable(model, x[...], Int)

Binary (i.e. \{0, 1\}) variables:
@variable(model, x[...], Bin)
Declaring model variables

Integer variables:
@variable(model, x[...], Int)

Binary (i.e. \{0, 1\}) variables:
@variable(model, x[...], Bin)

Example:
@variable(model, x[i in 1:M] >= 0, Int)
Declaring model variables

Integer variables:
@variable(model, x[...], Int)

Binary (i.e. \{0, 1\}) variables:
@variable(model, x[...], Bin)

Example:
@variable(model, x[i in 1:M] >= 0, Int)

Note that
@variable(model, 0 <= x[i in 1:M] <= 1, Int)
Declaring model variables

Integer variables:
@variable(model, x[...], Int)

Binary (i.e. \{0, 1\}) variables:
@variable(model, x[...], Bin)

Example:
@variable(model, x[i in 1:M] >= 0, Int)

Note that
@variable(model, 0 <= x[i in 1:M] <= 1, Int)

is equivalent to
@variable(model, x[i in 1:M], Bin)
Declaring model constraints

With a for loop:

```plaintext
for i in set
    @constraint(model, expression)
end
```
Declaring model constraints

With a for loop:

```plaintext
for i in set
    @constraint(model, expression)
end
```

Example:

```plaintext
for i in 1:M
    for j in 1:N
        if i < j
            @constraint(model, x[i, j] <= i + j)
        end
    end
end
```
Declaring model constraints

All at once:
@constraint(model, [i in set, ...], expression)
Declaring model constraints

All at once:
@constraint(model, [i in set, ...], expression)

Example:
@constraint(model, [i in 1:M, j in 1:N ; i < j], x[i, j] <= i + j)
Custom sets

So far, all sets were a:b, but we can have arbitrary sets:

```python
myset = Set([expression for i in set if condition])
```
Custom sets

So far, all sets were $a:b$, but we can have arbitrary sets:

\[
\text{myset} = \text{Set}([\text{expression for i in set if condition}])
\]

Example:

\[
\text{myset} = \text{Set}([2 * i \text{ for i in 1:10}])
\]
So far, all sets were `a:b`, but we can have arbitrary sets:

```
myset = Set([expression for i in set if condition])
```

Example:

```
myset = Set([2 * i for i in 1:10])
```

```
Set([2, 4, 6, 8, 10, 12, 14, 16, 18, 20])
```
Custom sets

So far, all sets were `a:b`, but we can have arbitrary sets:

```python
myset = Set([expression for i in set if condition])
```

Example:

```python
myset = Set([2 * i for i in 1:10])
```

```python
Set([18, 4, 14, 10, 20, 2, 16, 8, 6, 12])
```
Custom sets

We can also have arbitrary *multidimensional* sets, for example:

```python
myset = Set([(i, j) for i in 1:3, j in 1:3 if j < i])
```

which we can use like this:

```python
for (i, j) in myset
    println(i, " ", j)
end
```

Output:

```
3 1
3 2
2 1
```
Custom sets

We can also have arbitrary multidimensional sets, for example:

```plaintext
myset = Set([[(i, j) for i in 1:3, j in 1:3 if j < i]])
```

which we can use like this:

```plaintext
for (i, j) in myset
    println(i, " ", j)
end
```

Output:
```
3 1
3 2
2 1
```
Custom sets

We can also have arbitrary **multidimensional** sets, for example:

```
myset = Set([(i, j) for i in 1:3, j in 1:3 if j < i])
```

which we can use like this:

```
for (i, j) in myset
    println(i, " ", j)
end
```

Output:

3 1
3 2
2 1
Custom sets

The point of using $\text{Set}(\ldots)$
Custom sets

The point of using \texttt{Set(...)} is that we can do

\begin{itemize}
  \item \texttt{union(set1, set2, ...)}
\end{itemize}
The point of using `Set(...)` is that we can do

- `union(set1, set2, ...)`
- `intersect(set1, set2, ...)`
Custom sets

The point of using `Set(...)` is that we can do

- `union(set1, set2, ...)`
- `intersect(set1, set2, ...)`
- `setdiff(set1, set2, ...)`

... and test for inclusion: `if value in set`
Custom sets

The point of using `Set(...)` is that we can do

- `union(set1, set2, ...)`
- `intersect(set1, set2, ...)`
- `setdiff(set1, set2, ...)`

and test for inclusion:

```python
if value in set
...
end
```
Read the documentation!

http://www.juliaopt.org/JuMP.jl/v0.19.0/