**Mesh types:**
- Mesh – the entire unstructured mesh.
- Cell – a 3D element composed of several faces.
- Face – a 2D element composed of several edges.
- Edge – a 1D element connecting two vertices.
- Vertex – a 0D element representing a single point.

**Set[T <: MeshObj]** – a set of vertices, edges, faces or cells (the MeshObj subclasses).

**Accessing the mesh:**
- Use the built-in mesh to reference the global mesh object.
- Accessing all vertices / edges / faces / cells in the mesh: vertices(mesh), edges(mesh), faces(mesh), cells(mesh)
- Accessing adjacent elements by vertex:
  - cells(vertex), faces(vertex), edges(vertex): Return the set of all cells, faces or edges containing the vertex.
  - vertices(vertex): Returns the set of all vertices sharing an edge with the vertex.
- Accessing elements adjacent to an edge:
  - cells(edge), faces(edge): Return the set of all cells or faces containing the edge.
  - vertices(edge): Return the two vertices that the edge contains.
  - facesCW(edge), facesCCW(edge): Return the set of all faces containing the edge, oriented such that when viewed from vertex head(edge), cell outside(f) is clock-wise / counter-clockwise from f.
  - head(edge), tail(edge): Return the vertex that the edge points towards / away from.
  - flip(edge): Returns the same edge, oriented in the opposite direction. (Note that head(edge) == tail(flip(edge))).
- Accessing elements adjacent to a face:
  - cells(face): Returns the set of cells containing a face.
  - edges(face), vertices(face): Return the set of edges/vertices contained by the face.
  - edgesCW(face), edgesCCW(face): Return the set of edges contained by the face oriented such clock-wise / counter-clockwise around the edge when observed from the cell outside(edge).
  - outside(face), inside(face): Return the cell on the face which is the dual of head / tail.
  - flip(face): Returns the same face, oriented in the opposite direction. (Note that outside(face) == inside(flip(face))).
- Accessing elements adjacent to a cell:
  - vertices(cell), edges(cell), faces(cell): Return the set of all vertices / edges / faces that are contained by the cell.
  - cells(cell): Returns the set of cells that share a face with the cell.
- Conditionally re-orienting elements:
  - towards(edge, vertex): Returns edge e such that head(e) == vertex. e is either edge or flip(edge).
  - towards(face, cell): Returns face f such that outside(f) == cell.f is either face or flip(face).

**Accessing adjacent mesh elements by index:**
Some topological operators allow indexing adjacent mesh objects.
- Ex: `vertex(cell, i)` accesses the 4th vertex of the cell.
  - `vertex: vertex(cell, i), vertex(face, i)`
  - `edge: edge(cell, i), edge(face, i)`
  - `face: face(cell, i), face(edge, i)`
  - `cell: cell(cell, i), cell(edge, i)`
**Liszt Cheat Sheet!**

**Vector and Matrix Construction:**
- creating vectors of n elements:
  ```
  var vec1: Vec[_1, Float] = Vec(1.f)
  var vec2: Vec[_2, Float] = Vec(1.f, 1.f)
  var vec3: Vec[_3, Float] = Vec(1.f, 1.f, 1.f)
  ...
  ```
- creating row x column matrices:
  ```
  var mat2x3 = Mat(vec3, vec3)
  var mat3x4 = Mat(vec4, vec4, vec4)
  ```

**Vector and Matrix Operators:**
- `<Vec[N,T]> += <Vec[N,T]>`
  Vector addition/subtraction.
  Matrix addition/subtraction.
  Scalar multiplication/division, returns a `Vec[N,T]`.
- `<Numeric> */ <Mat[R,C,T], <Mat[R,C,T]> */ <Numeric>`
  Scalar multiplication/division, returns a `Mat[R,C,T]`.
- `<Mat[R,C,T]> * Vec[R,T]`
  Matrix vector multiply, returns type `Vec[C,T]` if T is numeric.
- `<Vec[N,T]> min / max <Vec[N,T]>`
  Maps the min/max operator over the vectors, returns type `Vec[N,T]`.

**Vector Component Manipulation:**
- Use meta-integer literals (_0, _1, ...) to access vector and matrix components:
  ```
  v(_0), v(_2), v(_4) - accesses the 1st, 3rd and 5th element of vector v.
  m(_0, _2), m(_3, _1) - accesses element at row 0, col 2 and row 3, col 1 of matrix m.
  ```

**Constructing Fields and Boundary Sets:**
- `FieldWithConst[MO <: MeshObj, VT](s : VT) : Field[MO, VT]`
  Creates a new field over MO with value type VT and initial value s. This can only be called at object scope, not within any function.
  ```
  Ex: val pressure = FieldWithConst[Cell, Float](0.f)
  ```
- `FieldWithLabel[MO >: MeshObj, VT](url: String) : Field[MO, VT]`
  Creates a new field over MO with value type VT, and load the initial values using the locator string url. (The only currently supported locator is "position", which loads the positions of vertices in the mesh.) This can only be called at object scope.
  ```
  Ex: val position = FieldWithLabel[Vertex, Vec[_3,Float]]("position")
  ```
- `BoundarySet[MO <: MeshObj](name : String) : Set[MO]`
  Loads the set of mesh topology of the given identifier from the mesh filename.
  ```
  Ex: val wall = BoundarySet[Face]("wall")
  ```

**Associative Operators for Fields and Scalar Reduction Variables:**
- `+=` and/or `-=`
- `*=` and/or `/=`
- `min` (use form “a = a min b”)
- `max` (use form “a = a max b”)
- `&=`
- `|=`

Do not use different associative operators on the same reduction variable within a single for-comprehension unless they are in the same bullet point!

**Miscellaneous built-ins:**
- `Print(as: Any*) : Unit`
  Output all arguments, newline terminated.
  ```
  Ex: Print("Temperature is: ", num_degrees)
  ```
- `size[MO <: MeshObj](s:Set[MO])`
  Retrieve the size of a set.
  ```
  Ex: val num_cells = size(cells(mesh))
  ```
- `ID[MO >: MeshObj](m : MO) : Int`
  Return the unique ID of the mesh object as it was named in the input file.
  ```
  Ex: val id = ID(cell0)
  ```
- `wall_time()`
  Return time in seconds since the program started. Low resolution, accurate to 2ms.
- `processor_time()`
  Return a time in seconds according to the processor’s tick count. Accurate to 1ns, but will produce inconsistent results if the thread switches processors, so accurate only for short (<1s) events.