Classes

DanDirectedGraph
DanDirectedGraphBreadthFirstIterator

The class DanDirectedGraphBreadthFirstIterator implements GraphIterator interface

DanDirectedGraphDepthFirstIterator

The class DanDirectedGraphDepthFirstIterator implements GraphIterator interface

DanUndirectedGraph

The DanUndirectedGraph class handles undirected graphs

DanStack

DanStack is a simple class implementing Stackable interface

Members

ArcType

The directed arc type: incoming or outgoing

ArcType

The class DanDirectedGraph handles directed graphs

Functions

randomIntFromInterval(min, max) ⇒ `number`

Get a random integer number between min and max (included)

DanDirectedGraph

Kind: global class

- DanDirectedGraph
  - new DanDirectedGraph()
  - *instance*
    - ._getCopyOfInnerGraph() ⇒ `Map.<I, DanNodeAndDirectedArcs.<I, D>>`
    - .getNodeAndDirectedArcsFromNodeId(idNode) ⇒ `DanNodeAndDirectedArcs.<I, D>` | `undefined`
    - .addNode(node) ⇒ `boolean`
    - .addArcToNodeId(idNode, arcToAdd, arcType) ⇒ `boolean`
    - .addArcToNode(node, arcToAdd, arcType) ⇒ `boolean`
    - .removeNode(idNode) ⇒ `boolean`
    - .isNodeALeaf(idNode) ⇒ `boolean`
    - ._getALeaf() ⇒ `DanNode.<I, D>` | `undefined`

        - .toString(showDetails) ⇒ `string`
        - .countNodes() ⇒ `number`
        - .isEmpty() ⇒ `boolean`
        - ._getOutgoingNodesList(idNode) ⇒ `Array.<DanNode>`
        - ._isAcyclic() ⇒ `boolean`
        - .isAcyclic() ⇒ `boolean`
        - ._visitNodes(visitedNodes, nextNode)
        - .sourceConnectedToAllNodes(idNode) ⇒ `boolean`
        - .getDepthFirstIterator(startingNodeId) ⇒ `GraphIterator.<DanNode.<I, D>>`
        - .getBreadthFirstIterator(startingNodeId) ⇒ `GraphIterator.<DanNode.<I, D>>`
    - *static*
        - .generateConsecutiveNodeGraph(numOfNodes) ⇒ `DanDirectedGraph.<number, undefined>`
        - .generateRandomNodeGraph(numOfNodes) ⇒ `DanDirectedGraph.<number, undefined>`

new DanDirectedGraph()

the public class constructor

danDirectedGraph._getCopyOfInnerGraph() ⇒ `Map.<I, DanNodeAndDirectedArcs.<I, D>>`

Clone the inner graph structure

Kind: instance method of `DanDirectedGraph`
Returns: `Map.<I, DanNodeAndDirectedArcs.<I, D>>` -

a copy of the inner graph structure

danDirectedGraph.getNodeAndDirectedArcsFromNodeId(idNode) ⇒ `DanNodeAndDirectedArcs.<I, D>` | `undefined`

Get node and directed arcs structure (as DanNodeAndDirectedArcs) from nodeId

Kind: instance method of `DanDirectedGraph`
Returns: `DanNodeAndDirectedArcs.<I, D>` | `undefined` -

node and directed arcs structure or undefined if nodeId is not present in the graph

Param	Type	Description
idNode	`I`	the node identifier

danDirectedGraph.addNode(node) ⇒ `boolean`

Add a node to the graph

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if the node was correctly added to the graph; false if the node is already present

Param	Type	Description
node	`DanNode.<I, D>`	the node to add

danDirectedGraph.addArcToNodeId(idNode, arcToAdd, arcType) ⇒ `boolean`

Add arc to node given a node id

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if *arcToAdd* was correctly added to the icoming/ougoing arcs of idNode; false if the idNode does not exist or if *arcToAdd* was already present

Param	Type	Description
idNode	`I`	the id of the node receiving *arcToAdd*
arcToAdd	`DanArc.<I, D>`	the arc being added
arcType	`ArcType`	*arcToAdd* will be added among the incoming or outgoing arcs of idNode, based on the value of arcType

danDirectedGraph.addArcToNode(node, arcToAdd, arcType) ⇒ `boolean`

Add arc to node given a node structure

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if *arcToAdd* was correctly added to the icoming/ougoing arcs of node; false if *arcToAdd* was already present

Param	Type	Description
node	`DanNode.<I, D>`	the node receiving *arcToAdd*
arcToAdd	`DanArc.<I, D>`	the arc being added

Param	Type	Description
arcType	`ArcType`	*arcToAdd* will be added among the incoming or outgoing arcs of node, based on the value of arcType

danDirectedGraph.removeNode(idNode) ⇒ `boolean`

Remove a node given in input from the graph

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if the node is correctly removed

Param	Type	Description
idNode	`I`	the id of the node to remove

danDirectedGraph.isNodeALeaf(idNode) ⇒ `boolean`

Check if node is a leaf

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if the node is a leaf (no outgoing arcs)

Param	Type	Description
idNode	`I`	the id of the node to check

danDirectedGraph._getALeaf() ⇒ `DanNode.<I, D>` | `undefined`

Get a leaf in the graph or return undefined.

Kind: instance method of `DanDirectedGraph`
Returns: `DanNode.<I, D>` | `undefined` -

the first leaf found, or undefined if no leaf is found

danDirectedGraph.toString(showDetails) ⇒ `string`

The string representation of the directed graph

Kind: instance method of `DanDirectedGraph`
Returns: `string` -

the string representation of the directed graph

Param	Type	Description
showDetails	`boolean`	if this option is true, all the node and arc details will be included in the output string (default: false)

danDirectedGraph.countNodes() ⇒ `number`

Public method to retrieve the number of nodes in the graph

Kind: instance method of `DanDirectedGraph`
Returns: `number` -

the number of nodes in the graph

danDirectedGraph.isEmpty() ⇒ `boolean`

Check if the graph is empty

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if the graph does not contain any node

danDirectedGraph._getOutgoingNodesList(idNode) ⇒ `Array.<DanNode>`

Get all the outgoing nodes from a node identifier

Kind: instance method of `DanDirectedGraph`
Returns: `Array.<DanNode>` -

the list of outgoing nodes of idNode as array of DanNode<I, D>

Param	Type	Description
idNode	`I`	the id of the node to check

danDirectedGraph._isAcyclic() ⇒ `boolean`

Protected method to check if the graph is acyclic. Remove leaves iteratively from the graph: stop if the graph gets empty (acyclic graph), or if there are no more leaves to remove (cyclic graph).

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if the graph does not contain cycles, otherwise false

danDirectedGraph.isAcyclic() ⇒ `boolean`

Public method to check if the graph is acyclic. First we keep a copy of the current inner graph. Then we invoke the protected [this._isAcyclic](#DanDirectedGraph<I, D>+_isAcyclic) method to check if the graph is acyclic. Then we restore the inner state (memento pattern).

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if the graph does not contain cycles

danDirectedGraph._visitNodes(visitedNodes, nextNode)

Protected method to visit all the neighbours of a node, given in input the node id and a previous set of visitedNodes

Kind: instance method of `DanDirectedGraph`

Param	Type	Description
visitedNodes	`Set.<I>`	the nodes already visited
nextNode	`I`	the next node to visit

danDirectedGraph.sourceConnectedToAllNodes(idNode) ⇒ `boolean`

Public method to check if all nodes are connected to the source node in input

Kind: instance method of `DanDirectedGraph`
Returns: `boolean` -

true if source can reach all of the nodes in the graph

Param	Type	Description
idNode	`I`	source node to be checked

danDirectedGraph.getDepthFirstIterator(startingNodeId) ⇒ `GraphIterator.<DanNode.<I, D>>`

Get depth-first iterator

Kind: instance method of `DanDirectedGraph`
Returns: `GraphIterator.<DanNode.<I, D>>` -

the depth-first iterator

Param	Type	Description
startingNodeId	`I`	the starting node identifier for the iterator

danDirectedGraph.getBreadthFirstIterator(startingNodeId) ⇒ `GraphIterator.<DanNode.<I, D>>`

Get breadth-first iterator

Kind: instance method of `DanDirectedGraph`
Returns: `GraphIterator.<DanNode.<I, D>>` -

the breadth-first iterator

Param	Type	Description
startingNodeId	`I`	the starting node identifier for the iterator

DanDirectedGraph.generateConsecutiveNodeGraph(numOfNodes) ⇒ `DanDirectedGraph.<number, undefined>`

A utility public static method to generate a directed graph with a number of *numOfNodes* consecutive nodes

Kind: static method of `DanDirectedGraph`
Returns: `DanDirectedGraph.<number, undefined>` -

a directed graph witn *numOfNodes* nodes

Param	Type	Description
numOfNodes	`number`	the number of nodes of the output graph

DanDirectedGraph.generateRandomNodeGraph(numOfNodes) ⇒ `DanDirectedGraph.<number, undefined>`

A utility public static method to generate a directed graph with a number of *numOfNodes* random nodes

Kind: static method of `DanDirectedGraph`
Returns: `DanDirectedGraph.<number, undefined>` -

a directed graph witn *numOfNodes* nodes

Param	Type	Description

Param	Type	Description
numOfNodes	`number`	the number of nodes of the output graph

DanDirectedGraphBreadthFirstIterator

The class DanDirectedGraphBreadthFirstIterator implements GraphIterator interface

Kind: global class

- DanDirectedGraphBreadthFirstIterator
  - new DanDirectedGraphBreadthFirstIterator(collection, startingNodeId)
  - ._getNodeAndDirectedArcsFromNodeId(nodeId) ⇒ `DanNodeAndDirectedArcs.<I, D>`
  - ._initFields(startingNodeId)
  - .current() ⇒ `DanNode.<I, D>` | `undefined`
  - .next() ⇒ `DanNode.<I, D>` | `undefined`
  - ._addNextNodeOutgoingsToNextLevelQueueIfNotAlreadyVisitedOrQueued() ⇒ `boolean`
  - ._advance()
  - .hasNext() ⇒ `boolean`
  - .rewind()

new DanDirectedGraphBreadthFirstIterator(collection, startingNodeId)

The public class constructor

Throws:

- `Error`

  exception if startingNodeId was not found in graph

Param	Type	Description
collection	`DanDirectedGraph.<I, D>`	the directed graph
startingNodeId	`I`	the starting node identifier

danDirectedGraphBreadthFirstIterator._getNodeAndDirectedArcsFromNodeId(nodeId) ⇒ `DanNodeAndDirectedArcs.<I, D>`

Retrieve node and directed arcs structure from node identifier

Kind: instance method of `DanDirectedGraphBreadthFirstIterator`
Returns: `DanNodeAndDirectedArcs.<I, D>` -

the node and directed arcs structure DanNodeAndDirectedArcs

Throws:

- `Error`

  exception if nodeId was not found in graph

Param	Type	Description
nodeId	`I`	the node identifier

danDirectedGraphBreadthFirstIterator._initFields(startingNodeId)

Init the class fields with the starting node identifier passed in input

Kind: instance method of `DanDirectedGraphBreadthFirstIterator`
Throws:

- `Error`

  exception if startingNodeId was not found in graph

Param	Type	Description
startingNodeId	`I`	the starting node identifier

danDirectedGraphBreadthFirstIterator.current() ⇒ `DanNode.<I, D>` | `undefined`

Get the current node, or return undefined if iterator was not yet started

Kind: instance method of `DanDirectedGraphBreadthFirstIterator`
Throws:

- `Error`

  exception if this._currentNodeId was not found in graph

danDirectedGraphBreadthFirstIterator.next() ⇒ `DanNode.<I, D>` | `undefined`

Get the next node, or return undefined if the iterator's end was reached

Kind: instance method of `DanDirectedGraphBreadthFirstIterator`
Throws:

- `Error`

  exception if this._nextNodeId was not found in graph

danDirectedGraphBreadthFirstIterator._addNextNodeOutgoingsToNextLevelQueueIfNotAlreadyVisitedOrQueued() ⇒ `boolean`

Add the outgoing nodes of this._nextNodeId to the next-level-queue, if the node was not already visited, and if it's not already present inside the queue

Kind: instance method of `DanDirectedGraphBreadthFirstIterator`
Returns: `boolean` -

true if this._nextNodeId is not null, otherwise false

Throws:

- `Error`

  exception if this._nextNodeId was not found in graph

danDirectedGraphBreadthFirstIterator._advance()

Advance to the next node in a breadth first fashion

Kind: instance method of `DanDirectedGraphBreadthFirstIterator`

danDirectedGraphBreadthFirstIterator.hasNext() ⇒ `boolean`

Check if the iterator can step to a next node

Kind: instance method of `DanDirectedGraphBreadthFirstIterator`
Returns: `boolean` -

true if the iterator can step to a next node, false if there are no more nodes left to visit in the iterator

danDirectedGraphBreadthFirstIterator.rewind()

Restart the iterator

Kind: instance method of `DanDirectedGraphBreadthFirstIterator`

DanDirectedGraphDepthFirstIterator

The class DanDirectedGraphDepthFirstIterator implements GraphIterator interface

Kind: global class

- DanDirectedGraphDepthFirstIterator
  - new DanDirectedGraphDepthFirstIterator(collection, startingNodeId)
  - ._getNodeAndDirectedArcsFromNodeId(nodeId) ⇒ `DanNodeAndDirectedArcs.<I, D>`
  - ._initFields(startingNodeId)
  - .current() ⇒ `DanNode.<I, D>` | `undefined`
  - .next() ⇒ `DanNode.<I, D>` | `undefined`
  - ._advance()
  - .hasNext() ⇒ `boolean`
  - .rewind()

new DanDirectedGraphDepthFirstIterator(collection, startingNodeId)

The public class constructor

Throws:

- `Error`

  exception if startingNodeId was not found in graph

Param	Type	Description
collection	`DanDirectedGraph.<I, D>`	the directed graph
startingNodeId	`I`	the starting node identifier

danDirectedGraphDepthFirstIterator._getNodeAndDirectedArcsFromNodeId(nodeId) ⇒ `DanNodeAndDirectedArcs.<I, D>`

Retrieve node and directed arcs structure from node identifier

Kind: instance method of `DanDirectedGraphDepthFirstIterator`
Returns: `DanNodeAndDirectedArcs.<I, D>` -

the node and directed arcs structure DanNodeAndDirectedArcs

Throws:

- `Error`

  exception if nodeId was not found in graph

Param	Type	Description
nodeId	`I`	the node identifier

danDirectedGraphDepthFirstIterator._initFields(startingNodeId)

Init the class fields with the starting node identifier passed in input

Kind: instance method of `DanDirectedGraphDepthFirstIterator`
Throws:

- `Error`

  exception if startingNodeId was not found in graph

Param	Type	Description

Param	Type	Description
startingNodeId	`I`	the starting node identifier

danDirectedGraphDepthFirstIterator.current() ⇒ `DanNode.<I, D>` | `undefined`

Get the current node, or return undefined if iterator was not yet started

Kind: instance method of `DanDirectedGraphDepthFirstIterator`
Throws:

- `Error`

  exception if this._currentNodeId was not found in graph

danDirectedGraphDepthFirstIterator.next() ⇒ `DanNode.<I, D>` | `undefined`

Get the next node, or return undefined if the iterator's end was reached

Kind: instance method of `DanDirectedGraphDepthFirstIterator`
Throws:

- `Error`

  exception if this._nextNodeId was not found in graph

danDirectedGraphDepthFirstIterator._advance()

Advance to the next node in a depth first fashion

Kind: instance method of `DanDirectedGraphDepthFirstIterator`

danDirectedGraphDepthFirstIterator.hasNext() ⇒ `boolean`

Check if the iterator can step to a next node

Kind: instance method of `DanDirectedGraphDepthFirstIterator`
Returns: `boolean` -

true if the iterator can step to a next node, false if there are no more nodes left to visit in the iterator

danDirectedGraphDepthFirstIterator.rewind()

Restart the iterator

Kind: instance method of `DanDirectedGraphDepthFirstIterator`

DanUndirectedGraph

The DanUndirectedGraph class handles undirected graphs

Kind: global class

- DanUndirectedGraph
  - new DanUndirectedGraph()
  - *instance*
    - .addNode(node) ⇒ `boolean`
    - .addArcToNodeId(idNode, arcToAdd) ⇒ `boolean`
    - .addArcToNode(node, nodeToAdd) ⇒ `boolean`
    - .removeNode(idNode) ⇒ `boolean`
    - ._getAdjacentNodesList(idNode) ⇒ `Array.<DanNode>`
    - ._getANode() ⇒ `DanNode.<I, D>` | `undefined`
    - ._visitNodes(visitedNodes, nextNode)
    - ._checkForCycle(visitedNodes, nextNode, fromNode) ⇒ `boolean`
    - .isConnected() ⇒ `boolean`
    - .isAcyclic() ⇒ `boolean`
    - .countNodes() ⇒ `number`
    - .isEmpty() ⇒ `boolean`
    - .toString(showDetails) ⇒ `string`
    - .getInnerGraph() ⇒ `Map.<I, DanNodeAndUndirectedArcs.<I, D>>`
  - *static*
    - .generateConsecutiveNodeGraph(numOfNodes) ⇒ `DanUndirectedGraph.<number, undefined>`
    - .generateRandomNodeGraph(numOfNodes) ⇒ `DanUndirectedGraph.<number, undefined>`

new DanUndirectedGraph()

the public class constructor

danUndirectedGraph.addNode(node) ⇒ `boolean`

Add a node to the graph

Kind: instance method of `DanUndirectedGraph`
Returns: `boolean` -

true if the node was correctly added to the graph; false if the node is already present

Param	Type	Description
node	`DanNode.<I, D>`	the node to add

danUndirectedGraph.addArcToNodeId(idNode, arcToAdd) ⇒ `boolean`

Add an arc to a node, given the node identifier

Kind: instance method of `DanUndirectedGraph`
Returns: `boolean` -

true if *arcToAdd* was correctly added to the adjacent arcs of idNode; false if the idNode does not exist or if *arcToAdd* was already present

Param	Type	Description
idNode	`I`	the id of the node receiving *arcToAdd*
arcToAdd	`DanArc.<I, D>`	the arc being added

danUndirectedGraph.addArcToNode(node, nodeToAdd) ⇒ `boolean`

Add arc to node, given the node interface structure

Kind: instance method of `DanUndirectedGraph`
Returns: `boolean` -

true if *arcToAdd* was correctly added to the adjacent arcs of node; false if *arcToAdd* was already present

Param	Type	Description
node	`DanNode.<I, D>`	the node receiving *arcToAdd*
nodeToAdd	`DanArc.<I, D>`	the node being added

danUndirectedGraph.removeNode(idNode) ⇒ `boolean`

Remove the node given in input from the graph

Kind: instance method of `DanUndirectedGraph`
Returns: `boolean` -

true if the node is correctly removed

Param	Type	Description
idNode	`I`	the id of the node to remove

danUndirectedGraph._getAdjacentNodesList(idNode) ⇒ `Array.<DanNode>`

Get the list of adjacent nodes of a given node identifier

Kind: instance method of `DanUndirectedGraph`
Returns: `Array.<DanNode>` -

the list of adjacent nodes of idNode as array of DanNode<I, D>

Param	Type	Description
idNode	`I`	the id of the node to check

danUndirectedGraph._getANode() ⇒ `DanNode.<I, D>` | `undefined`

Get a node in the graph

Kind: instance method of `DanUndirectedGraph`
Returns: `DanNode.<I, D>` | `undefined` -

a node in the graph or undefined if the graph is empty

danUndirectedGraph._visitNodes(visitedNodes, nextNode)

Protected method to visit all the neighbours of a node, given in input the node id and a previous set of visitedNodes

Kind: instance method of `DanUndirectedGraph`

Param	Type	Description
visitedNodes	`Set.<I>`	the nodes already visited
nextNode	`I`	the next node to visit

danUndirectedGraph._checkForCycle(visitedNodes, nextNode, fromNode) ⇒ `boolean`

Recursively checks for the presence of a cycle starting from node id nextNode

Kind: instance method of `DanUndirectedGraph`
Returns: `boolean` -

true if a cycle is found, false if no cycle is found

Param	Type	Description
visitedNodes	`Set.<I>`	the nodes already visited
nextNode	`I`	the next node to visit
fromNode	`I`	`undefined`	the parent node (default: undefined)

danUndirectedGraph.isConnected() ⇒ `boolean`

Check if the graph is connected

Kind: instance method of `DanUndirectedGraph`
Returns: `boolean` -

true if the graph is connected

danUndirectedGraph.isAcyclic() ⇒ `boolean`

Public method to check if the graph is acyclic

Kind: instance method of `DanUndirectedGraph`
Returns: `boolean` -

true if the graph does not contain cycles

danUndirectedGraph.countNodes() ⇒ `number`

Public method to retrieve the number of nodes in the graph

Kind: instance method of `DanUndirectedGraph`
Returns: `number` -

the number of nodes in the graph

danUndirectedGraph.isEmpty() ⇒ `boolean`

Check if the graph is empty

Kind: instance method of `DanUndirectedGraph`
Returns: `boolean` -

true if the graph does not contain any node

danUndirectedGraph.toString(showDetails) ⇒ `string`

The string representation of the undirected graph

Kind: instance method of `DanUndirectedGraph`
Returns: `string` -

the string representation of the undirected graph

Param	Type	Description

Param	Type	Description
showDetails	`boolean`	if this option is true, all the node and arc details will be included in the output string (default: false)

danUndirectedGraph.getInnerGraph() ⇒ `Map.<I, DanNodeAndUndirectedArcs.<I, D>>`

Get the inner graph object

Kind: instance method of `DanUndirectedGraph`
Returns: `Map.<I, DanNodeAndUndirectedArcs.<I, D>>` -

the inner graph Map object

DanUndirectedGraph.generateConsecutiveNodeGraph(numOfNodes) ⇒ `DanUndirectedGraph.<number, undefined>`

A utility public static method to generate an undirected graph with a number of *numOfNodes* consecutive nodes

Kind: static method of `DanUndirectedGraph`
Returns: `DanUndirectedGraph.<number, undefined>` -

an undirected graph witn *numOfNodes* nodes

Param	Type	Description
numOfNodes	`number`	the number of nodes of the output graph

DanUndirectedGraph.generateRandomNodeGraph(numOfNodes) ⇒ `DanUndirectedGraph.<number, undefined>`

A utility public static method to generate an undirected graph with a number of *numOfNodes* random nodes

Kind: static method of `DanUndirectedGraph`
Returns: `DanUndirectedGraph.<number, undefined>` -

an undirected graph witn *numOfNodes* nodes

Param	Type	Description
numOfNodes	`number`	the number of nodes of the output graph

DanStack

DanStack is a simple class implementing Stackable interface

Kind: global class

- DanStack
  - new DanStack()
  - .push(val)
  - .pop() ⇒ `T` | `undefined`
  - .peek() ⇒ `T` | `undefined`
  - .isEmpty() ⇒ `boolean`
  - .clear()

new DanStack()

The public class constructor

danStack.push(val)

Insert a value to the top of the stack We use the 'push' method of the private member '_list'

Kind: instance method of `DanStack`

Param	Type	Description
val	`T`	the value to be pushed inside the stack

danStack.pop() ⇒ `T` | `undefined`

Get the value from the top of the stack and remove it from the stack itself We use the 'pop' method of the private member '_list'

Kind: instance method of `DanStack`
Returns: `T` | `undefined` -

the element on top of the stack if the stack is not empty; conversely it returns undefined

danStack.peek() ⇒ `T` | `undefined`

Get the value from the top of the stack but do not remove it from the stack itself

Kind: instance method of `DanStack`
Returns: `T` | `undefined` -

the element on top of the stack if the stack is not empty; conversely it returns undefined

danStack.isEmpty() ⇒ `boolean`

Check if the stack is empty We use the 'length' method of the private member '_list' to check the number of elements present: if the number is less than 1, the stack is empty

Kind: instance method of `DanStack`
Returns: `boolean` -

true if the stack is empty; it returns false is the stack is not empty

danStack.clear()

Clear all stack's elements We use the 'splice' method of the private member '_list'

Kind: instance method of `DanStack`

ArcType

The directed arc type: incoming or outgoing

Kind: global variable

ArcType

The class DanDirectedGraph handles directed graphs

Kind: global variable

randomIntFromInterval(min, max) ⇒ `number`

Get a random integer number between min and max (included)

Kind: global function
Returns: `number` -

an integer between min and max (included)

Param	Type	Description
min	`number`	minimum number
max	`number`	maximum number