

# Gremlin Cheat Sheet

## Read-Only Traversals

### Initial Lookups

| Steps  | Meaning  |
|--|--|
| <code>V()</code>                               | get all vertices in the graph  |
| <code>E()</code>                               | get all edges in the graph   |
| <code>V().hasLabel(label1, label2, ...)</code> | get all vertices with the specified labels   |
| <code>V().has(label, key, value)</code>        | get all vertices with the specified label and the property <b>key</b> matching the provided <b>value</b> |
| <code>V(1)</code>                              | get the vertex with the id <b>1</b>  |

### Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
=>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V()
==>v[1]
==>v[2]
==>v[3]
==>v[4]
==>v[5]
==>v[6]
gremlin> g.V().hasLabel("person")
==>v[1]
==>v[2]
==>v[4]
==>v[6]
gremlin> g.V().has("person","name","marko")
==>v[1]
gremlin> g.V(1)
==>v[1]
```

## Access Properties

| Steps                                    | Meaning  |
|--|--|
| <code>properties(key1, key2, ...)</code> | get all specified properties for the current element               |
| <code>values(key1, key2, ...)</code>     | get all specified property values for the current element          |
| <code>valueMap(key1, key2, ...)</code>   | get all specified property values for the current element as a map |

### Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
=>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V().hasLabel("person").properties("name")
==>vp[name->marko]
==>vp[name->vadas]
==>vp[name->josh]
==>vp[name->peter]
gremlin> g.V().hasLabel("person").values("name")
==>marko
==>vadas
==>josh
==>peter
gremlin> g.V().hasLabel("person").valueMap("name", "age")
==>[name:[marko],age:[29]]
==>[name:[vadas],age:[27]]
==>[name:[josh],age:[32]]
==>[name:[peter],age:[35]]
```

## Traversing the Graph

| Steps  | Meaning  |
|--|--|
| <code>out(label1, label2, ...)</code>            | get all adjacent vertices connected by outgoing edges with the specified labels                    |
| <code>in(label1, label2, ...)</code>             | get all adjacent vertices connected by incoming edges with the specified labels                    |
| <code>outE(label1, label2, ...)</code>           | get all outgoing edges with the specified labels   |
| <code>inE(label1, label2, ...)</code>            | get all incoming edges with the specified labels   |
| <code>both(label1, label2, ...)</code>           | get all adjacent vertices connected by an edge with the specified labels                           |
| <code>bothE(label1, label2, ...).otherV()</code> | traverse to all incident edges with the specified labels and then to the respective other vertices |

### Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
=>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V(1).outE("created")
==>e[9][1-created->3]
gremlin> g.V(1).out("created")
==>v[3]
gremlin> g.V().has("software","name","lop").in("created").values("name")
==>marko
==>josh
==>peter
```

## Filters

| Steps                            | Meaning  |
|----------------------------------|--|
| <code>has(key, value)</code>     | keep the current element if the specified property has the given value           |
| <code>has(key, predicate)</code> | keep the current element if the specified property matches the given predicate   |
| <code>filter(traversal)</code>   | keep the current element if the provided traversal emits a result                |
| <code>not(traversal)</code>      | keep the current element if the provided traversal doesn't emit a result         |
| <code>where(predicate)</code>    | keep the current element if it matches the predicate referencing another element |

**NOTE**      Predicates are static methods provided by [P](#) and [TextP](#).

### Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
=>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V().has("age",29).valueMap("name","age")
==>[name:[marko],age:[29]]
gremlin> g.V().has("age",gt(30)).valueMap("name","age")
==>[name:[josh],age:[32]]
==>[name:[peter],age:[35]]
gremlin> g.V().filter(outE())
==>v[1]
==>v[4]
==>v[6]
gremlin> g.V().not(outE())
==>v[2]
==>v[3]
==>v[5]
gremlin> g.V(1).as("other").
.....1>  out("knows").where(gt("other")).by("age").
.....2>  valueMap()
==>[name:[josh],age:[32]]
```

## Aggregations

| Steps                                     | Meaning  |
|---|--|
| <code>store(key)</code>                   | store the current element in the side-effect with the provided key   |
| <code>aggregate(key)</code>               | store all elements held by all current traversers in the side-effect with the provided key   |
| <code>group([key]).by(keySelector)</code> | group all current elements by the provided <b>keySelector</b> ; group into a side-effect if a side-effect <b>key</b> was provided, otherwise emit the result immediately |
| <code>fold()</code>                       | fold all current elements into a single list   |
| <code>unfold()</code>                     | unfold the incoming list and continue processing each element individually   |
| <code>count()</code>                      | count the number of current elements   |
| <code>min()/max()</code>                  | find the min/max value   |
| <code>sum()</code>                        | compute the sum of all current values  |
| <code>mean()</code>                       | compute the mean value of all current values   |

### Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
=>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V().hasLabel("person").store("x").select("x")
==>[v[1]]
==>[v[1],v[2]]
==>[v[1],v[2],v[4]]
==>[v[1],v[2],v[4],v[6]]
gremlin> g.V().hasLabel("person").aggregate("x").select("x")
==>[v[1],v[2],v[4],v[6]]
==>[v[1],v[2],v[4],v[6]]
==>[v[1],v[2],v[4],v[6]]
gremlin> g.V().group().by(label)
==>[software:[v[3],v[5]],person:[v[1],v[2],v[4],v[6]]]
gremlin> g.V().fold()
==>[v[1],v[2],v[3],v[4],v[5],v[6]]
gremlin> g.V().count()
==>6
gremlin> g.V().fold().count(local)
==>6
```

## Branches

| Steps   | Meaning  |
|---|--|
| <code>union(branch1, branch2, ...)</code>   | execute all branches and emit their results  |
| <code>choose(condition, true-branch, false-branch)</code>   | <b>if/then/else</b> -based traversal. If the condition matches (yields something), execute the <b>true-branch</b> , otherwise follow the <b>false-branch</b> . |
| <code>choose(selector).option(opt1, traversal).option(opt2, traversal).option(optN, traversal)</code> | value-based traversal; If an option value matches the value emitted by the <b>selector</b> traversal, the respective option traversal will be executed.        |

### Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
=>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V().hasLabel("person").union(out("knows"), count())
==>v[2]
==>v[4]
==>4
gremlin> g.V().hasLabel("person").
.....1>  choose(has("age",gt(30)), constant("senior"), constant("junior"))
==>junior
==>senior
==>senior
gremlin> g.V().hasLabel("person").values("age").
.....1>  union(min(), max(), sum(), mean(), count())
==>27
==>35
==>123
==>30.75
==>4
```

## Mutating Traversals

| Steps  | Meaning  |
|--|--|
| <code>addV(label)</code>                         | add a new vertex                                       |
| <code>addE(label).from(source).to(target)</code> | adds a new edge between the two given vertices         |
| <code>property(key, value)</code>                | adds or updates the property with the given <b>key</b> |

### Examples

```
gremlin> g = TinkerGraph.open().traversal()
=>graphtraversalsource[tinkergraph[vertices:0 edges:0], standard]
gremlin> g.addV('company').
.....1>  property('name','datastax').as('ds').
.....2>  addV('software').
.....3>  property('name','dse graph').as('dse').
.....4>  addV('software').
.....5>  property('name','tinkerpop').as('tp').
.....6>  addE('uses').from('ds').to('tp').
.....7>  addE('uses').from('dse').to('tp').
.....8>  addE('likes').from('ds').to('tp').iterate()
gremlin> g.V().outE().inV().path().by('name').by(label)
==>[datastax,develops,dse graph]
==>[datastax,likes,tinkerpop]
==>[dse graph,uses,tinkerpop]
```