### Rego Cheat Sheet

#### Rules

**Complete Rules**

Complete rules assign a single value. *(Try It)*

```rego
default allow := false
allow if {
  input.user.role == "admin"
  input.user.internal
}
default request_quota := 100
request_quota := 1000 if input.user.internal
request_quota := 50 if input.user.plan.trial
```

**Partial Rules**

Partial rules generate and assign a set of values to a variable. *(Try It)*

```rego
paths contains "/handbook/"*
paths contains path if {
  some team in input.user.teams
  path := sprintf("/teams/%v/", [team])
}

(\Output)
```

```
paths: [ "/handbook/", "/teams/ow1/", "/teams/tiger/"
]
```

#### Control Flow

**Handle different conditions**

**Logical AND**

Statements in rules are joined with logical AND. *(Try It)*

```rego
valid_staff_email if {
  regex.match("^\S+@\S+\.(\S+)$", input.email)
  # and
  endswith(input.email, "example.com")
}
```

**Logical OR**

Express OR with multiple rules, functions or the in keyword. *(Try It)*

```rego
# using multiple rules
valid_email if endswith(input.email, "@example.com")
valid_email if endswith(input.email, "@example.org")
valid_email if endswith(input.email, "@example.net")

# using functions
allowed_firstname(name) if {
  count(name) > 2
} allowed_firstname("joe") if name == 'joe'
valid_name if {
  allowed_firstname(input.name)
}

# using 'in'
valid_request if {
  input.method in ("GET", "POST")
}
```

#### Iteration

**Make quick work of collections**

**Some**

Name local query variables. *(Try It)*

```rego
all_regions := {
  "emea": ["west", "east"],
  "na": ["west", "east", "central"],
  "latam": ["west", "east"],
  "apac": ["north", "south"],
}
allowed_regions contains region_id if {
  some area, regions in all_regions
  some region in regions
  region_id := sprintf("%s_%s", [area, region])
}

(\Output)
```

```
allowed_regions: [ 
  "apac_north", "apac_south", "emea_east", ...
]
```

**Every**

Check conditions on many elements. *(Try It)*

```rego
allow if {
  prefix := sprintf("/docs/%s/", [input.userID])
  every path in input.paths {
    startswith(path, prefix)
  }
}
```

#### Testing

**Validate your policy’s behavior**

**With**

Override input and data using the with keyword. *(Try It)*

```rego
allow if {
  input.admin == true
}

test_allow_when_admin if {
  allow with input as ["admin": true]
}
```

#### Debugging

**Find and fix problems**

**Print**

Use print in rules to inspect values at runtime. *(Try It)*

```rego
allowed_users := ["alice", "bob", "charlie"]
allow if {
  some user in allowed_users
  print (user)
}
```

```
// alice
// bob
// charlie
```
### Comprehensions
Rework and process collections

#### Arrays
Produce ordered collections, maintaining duplicates. (Try It)

```rego
doubled := [n |
    some n in [1, 2, 3, 3]
    n := n + 2 |
]
```

(Output)

```rego
[
    "doubled": [2, 4, 6, 6]
]
```

#### Sets
Produce unordered collections without duplicates. (Try It)

```rego
unique_doubled := [n |
    some n in [10, 20, 30, 20, 10]
    n := n + 2 |
]
```

(Output)

```rego
[
    "unique_doubled": [20, 40, 60]
]
```

#### Objects
Produce key:value data. Note, keys must be unique. (Try It)

```rego
is_even := (number: is_even |
    some number in [1, 2, 3, 4]
    is_even := (number % 2) == 0 |
)
```

(Output)

```rego
[
    "is_even": {
        "1": false, "2": true, "3": false, "4": true
    }
]
```

### Builtins
Handy functions for common tasks

#### Regex
Pattern match and replace string data. (Try It)

```rego
example_string := "Build Policy as Code with OPA!"
check_match if regex.match("\w+", example_string)
check_replace := regex.replace(example_string, '\s+', ".")
```

(Output)

```rego
["check_match": true,
"check_replace": "Build Policy as Code with OPA!"
]
```

#### Strings
Check and transform strings. (Try It)

```rego
example_string := "Build Policy as Code with OPA!"
check_contains if contains(example_string, "OPA")
check_starts_with if starts_with(example_string, "Build")
check_endswith if endswith(example_string, "!")
check_replace := replace(example_string, "OPA", "Styra")
check_sprintf := sprintf("OPA is %s!", ["awesome"])
```

(Output)

```rego
["check_sprintf": "OPA is awesome!
"
```

### Aggregates
Summarize data. (Try It)

```rego
vals := [5, 1, 4, 2, 3]
vals_count := count(vals)
vals_max := max(vals)
vals_min := min(vals)
vals_sorted := sort(vals)
vals_sum := sum(vals)
```

(Output)

```rego
[
    "vals_count": 5,
    "vals_max": 5,
    "vals_min": 1,
    "vals_sorted": [1, 2, 3, 4, 5],
    "vals_sum": 15
]
```

### Objects: Extracting Data
Work with key value and nested data. (Try It)

```rego
obj := {"userid": "18472", "roles": [{"name": "admin"}]}
# paths can contain array indexes too
defaulted_val := object.get(obj, ["roles", 0, "name"], "missing")
keys := object.keys(obj)
```

(Output)

```rego
[
    "val": "admin",
    "defaulted_val": "unknown",
    "keys": ["roles", "userid"]
]
```

### Objects: Transforming Data
Manipulate and make checks on objects. (Try It)

```rego
unioned := object.union({"foo": true}, {"bar": false})
subset := object.subset(
    {"foo": true, "bar": false},
    {"foo": true}, # subset object
)
removed := object.remove(
    {"foo": true, "bar": false},
    {"bar"}, # remove keys
)
```

(Output)

```rego
["removed": {
    "foo": true
},
"subset": true,
"unioned": {
    "bar": false,
    "foo": true
}
```

The Rego Cheat Sheet is maintained by Styra, the creators of OPA, and the Styra community. If you have any questions, suggestions, or would like to get involved, please join us on our Slack.