Rego Cheat Sheet

Rules The building blocks of Rego

Single-Value Rules

Single-value rules assign a single value. In older documentation, these are sometimes referred to as "complete rules". (Try It)

```
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
```

Multi-Value Set Rules

Multi-value set rules generate and assign a set of values to a variable. In older documentation these are sometimes referred to as "partial set rules". (Try It)

```
paths contains "/handbook/*"

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

Output

```
{
  "paths": ["/handbook/*", "/teams/owl/*", "/teams/tiger/*"]
}
```

Multi-Value Object Rules

Multi-value object rules generate and assign a set of keys and values to a variable. In older documentation these are sometimes referred to as "partial object rules". (Try It)

```
# Creates an object with sets as the values.
paths_by_prefix[prefix] contains path if {
  some path in input.paths
  parts := split(path, "/")
  prefix := parts[0]
}
```

Output

```
{
  "paths_by_prefix": {
    "a": ["a/123.txt", "a/456.txt"],
    "b": ["b/bar.txt", "b/foo.txt"],
    "c": ["c/x.txt"]
  }
}
```

Iteration Make quick work of collections

Some

Name local query variables. (Try It)

```
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)

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

Control Flow Handle different conditions

Logical AND

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

```
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)

```
# 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 {
  startswith(name, "a")
  count(name) > 2
}

allowed_firstname("joe") # if name == 'joe'

valid_name if allowed_firstname(input.name)

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

Output

```
{
  "email": "opa@example.com",
  "name": "anna",
  "method": "GET"
}
```

Testing Validate your policy's behavior

With

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

```
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)

```
allowed_users := {"alice", "bob"}

allow if {
  some user in allowed_users
  print(user)
  input.user == user
}
```

Output

```
// alice
// bob
```

Comprehensions

Rework and process collections

Arrays

Produce ordered collections, maintaining duplicates.
(Try It)

```
doubled := [m |
  some n in [1, 2, 3, 3]
  m := n * 2
]
```

Output

```
{ "doubled": [2, 4, 6, 6] }
```

Sets

Produce unordered collections without duplicates.
(Try It)

```
unique_doubled contains m if {
  some n in [10, 20, 30, 20, 10]
  m := n * 2
}
```

Output

```
{ "unique_doubled": [20, 40, 60] }
```

Objects

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

```
is_even[number] := is_even if {
  some number in [1, 2, 3, 4]
  is_even := (number % 2) == 0
}
```

Output

```
{
  "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)

```
example_string := "Build Policy as Code with OPA!"

check_match if regex.match(`\w+`, example_string)

check_replace := regex.replace(example_string, `\s+`, "_")
```

Output

```
{
  "check_match": true,
  "check_replace": "Build_Policy_as_Code_with_OPA!"
}
```

Strings

Check and transform strings. (Try It)

```
example_string := "Build Policy as Code with OPA!"

check_contains if contains(example_string, "OPA")
check_startswith if startswith(example_string, "Build")
check_endswith if endswith(example_string, "!")
check_replace := replace(example_string, "OPA", "OPA!")
check_sprintf := sprintf("OPA is %s!", ["awesome"])
```

Aggregates

Summarize data. (Try It)

```
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

```
{
  "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)

```
obj := {"userid": "18472", "roles": [{"name": "admin"}]}

# paths can contain array indexes too
val := object.get(obj, ["roles", 0, "name"], "missing")

defaulted_val := object.get(
  obj,
  ["roles", 0, "permissions"], # path
  "unknown", # default if path is missing
)

keys := object.keys(obj)
```

Output

```
{
  "val": "admin",
  "defaulted_val": "unknown",

  "keys": ["roles", "userid"]
}
```

Objects: Transforming Data

Manipulate and make checks on objects. (Try It)

```
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

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