Rego Style Guide

Rules

Complete

```
import future.keywords

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

```
import future.keywords

paths contains path if {
  path := "/handbook/*"
}

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

(Output)

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

Iteration

Some

```
import future.keywords

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

```
import future.keywords

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

Control Flow

Logical And

```
import future.keywords

valid_staff_email if {
  regex.match(`^\S+@\S+\.\S+$`, input.email)

  endswith(input.email, "example.com")
}
```

Logical Or

```
import future.keywords

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

# using functions
allowed_firstname(name) if name == "joe"
allowed_firstname(name) if name == "jane"
valid_name if {
  allowed_firstname(input.name)
}

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

Testing

With

```
import future.keywords

allow if {
  input.admin == true
}

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

Debugging

Print

```
import future.keywords

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

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

(Output)

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

Builtins

Aggregates

```
package play

import future.keywords.if

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

```
package play

import future.keywords.if

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

val := object.get(obj, ["roles", 0, "name"], "missing")
defaulted_val := object.get(obj, ["roles", 0, "
    permissions"], "unknown")

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

(Output)

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

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

Strings

```
package play

import future.keywords.if

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", "Styra")
check_sprintf := sprintf("OPA is %s!", ["awesome"])
```

(Output)

```
{
  "check_contains": true,
  "check_startswith": true,
  "check_endswith": true,
  "check_replace": "Build Policy as Code with Styra!",
  "check_sprintf": "OPA is awesome!"
}
```

Regex

```
package play

import future.keywords.if

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!"
}
```

Comprehensions

(Shared Code)

```
letters := ["q", "w", "i", "e", "r", "t", "y", "u", "i",
    " e", "y"]
vowels := ["a", "e", "i", "o", "u", "y"]
```

Arrays

```
array_match_vowels := [match |
  some letter in letters
  some vowel in vowels
  letter == vowel
  match := letter
]
```

(Output)

```
{
  "array_match_vowels": [
    "i", "e", "y", "u", "i", "e", "y"
  ]
}
```

Sets

```
set_match_vowels := {match |
  some letter in letters
  some vowel in vowels
  letter == vowel
  match := letter
}
```

(Output)

```
{
  "set_match_vowels": [
    "e", "i", "u", "y"
  ]
}
```

Objects

```
object_check_vowels := {letter: is_vowel |
  some letter in letters
  is_vowel := letter in vowels
}
```

(Output)

```
{
  "object_check_vowels": {
    "e": true, "i": true, "q": false, "r": false, "t":
    false, "u": true, "w": false, "y": true
  }
}
```