



# Cheat sheet

<https://cloud.google.com/workflows/docs/reference/syntax/>

## RUNTIME ARGUMENTS

```
main:
  params: [args]
  steps:
    - read_runtime_args:
        assign:
          - arg1: ${args.arg1}
          - arg2: ${args.arg2}
```

## VARIABLES AND DATA TYPES

```
- data_types:
  assign:
    - my_integer: 1      # 64 bit, signed
    - my_double: 4.1    # 64 bit, signed floating point number
    - my_string: "hello" # unicode <= 256 KB length
    - my_boolean: true  # true/false, True/False, TRUE/FALSE
    - my_null: null
    - my_list: ["zero", "one", "two"]
    - my_map:
        name: Lila
        last_name: Barton
        birthYear: 1990
- conversion_functions:
  assign:
    - to_double: double("2.7") # string, integer to double
    - to_int: int(2.7)         # string, double to integer
    - to_string: string(1.7)   # int, double, boolean to string
```

## BOOLEAN

```
- logical_ops:
  assign:
    - my_true: true
    - my_false: false
    - my_false: ${my_true and my_false}
    - my_true: ${my_true or my_false}
    - my_false: ${not my_true}
```

## STRINGS

```
- string_ops:
  assign:
    - my_string: "hello"
    - string_len: ${len(my_string)}
    - string_plus_string: ${my_string+"+"world"}
    - string_plus_int: ${my_string+" "+string(my_integer)}
    - string_escaped: '${"a: " +my_string}'
```

## LISTS

```
- list_ops:
  assign:
    - my_list: ["zero", "one", "two"]
    - my_list_len: ${len(my_list)}
    - key_exists: ${"Key1" in my_list}
    - my_list[0]: 0
    - idx: 0
    - my_list[idx + 1]: 1
    - my_list[my_list_len - 1]: 2
    - my_list: ${list.concat(my_list, 3)}
    - my_multi_dimen_list: [[10, 11, 12], [20, 21, 22]]
    - my_multi_dimen_list[0][1]: "Value11"
```

## MAPS

```
- map_ops:
  assign:
    - my_map: {"Key1": "hello"}
    - map_len: ${len(my_map)}
    - key_exists: ${"Key1" in my_map}
    - key_list: ${keys(my_map)}
    - key_is_null: ${default(map.get(my_map, "Key1"), "Couldn't find key!")}
    - key_with_special_char: '${"foo" + var.key["special!key"]}'
    - key_str: "Key"
    - my_map.Key1: "Value1"
    - my_map["Key2"]: "Value2"
    - my_map[key_str + "3"]: "Value3"
    - my_nested_map: {"NestedMapKey": {"Key1": "Value1"}}
    - my_nested_map.NestedMapKey.Key2: "Value2"
```

## CONTROLLING FLOW

```
- step_with_next:
  assign:
    - foo: "bar"
  next: step_with_nested_steps
- step_with_end:
  assign:
    - foo: "bar"
  next: end
- step_with_nested_steps:
  steps:
    - nested_step_1:
        assign:
          - foo: "bar"
    - nested_step_2:
        assign:
          - foo: "bar"
```

## ITERATION

```
- for-in-list:
  steps:
    - assignList:
        assign:
          - list: [1, 2, 3, 4, 5]
          - sum: 0
    - loopList:
        for:
          value: v
          in: ${list}
          steps:
            - sumList:
                assign:
                  - sum: ${sum + v}
- for-in-map:
  steps:
    - assignMap:
        assign:
          - map: {1: 10, 2: 20, 3: 30}
          - sum: 0
    - loopMap:
        for:
          value: key
          in: ${keys(map)}
          steps:
            - sumMap:
                assign:
                  - sum: ${sum + map[key]}
- for-range:
  steps:
    - assignRange:
        assign:
          - sum: 0
    - loopRange:
        for:
          value: v
          range: [1, 9]
          steps:
            - sumRange:
                assign:
                  - sum: ${sum + v}
```

Continue on back



## PARALLEL BRANCHES

```
- parallel_branches:
  parallel:
    shared: [user, notification]
    branches:
      - getUser:
          steps:
            - getUserCall:
                call: http.get
                args:
                  url: ${"https://example.com/users/" + args.userId}
                result: user
      - getNotification:
          steps:
            - getNotificationCall:
                call: http.get
                args:
                  url: ${"https://example.com/notification/" + args.notificationId}
                result: notification
```

## PARALLEL ITERATION

```
- parallel_loop:
  parallel:
    shared: [total]
    for:
      value: postId
      in: ${args.posts}
      steps:
        - getPostCommentCount:
            call: http.get
            args:
              url: ${"https://example.com/postComments/" + postId}
            result: numComments
        - add:
            assign:
              - total: ${total + numComments}
```

## SUBWORKFLOWS

```
- call_subworkflow:
  call: subworkflow_name_message
  args:
    first_name: "Ada"
    last_name: "Lovelace"
  result: output
- call_subworkflow2:
  assign:
    - output2: ${subworkflow_name_message("Sherlock", "Holmes")}

subworkflow_name_message:
  params: [first_name, last_name, country: "England"]
  steps:
    - prepareMessage:
        return: ${"Hello " + first_name + " " + last_name + " from " + country + "."}
```

## CONDITIONS

```
- switch_basic:
  switch:
    - condition: ${my_integer < 10}
      next: switch_embedded_steps
    - condition: ${my_boolean}
      next: switch_embedded_steps
    - condition: true # optional, default condition
      next: switch_embedded_steps
  next: switch_embedded_steps
- switch_embedded_steps:
  switch:
    - condition: ${my_integer < 10}
      steps:
        - stepA:
            assign:
              - foo: "bar"
        - stepB:
            assign:
              - foo: "bar"
```

## CONNECTORS

```
# https://cloud.google.com/workflows/docs/reference/googleapis
## googleapis.compute.v1
- insert_machine:
  call: googleapis.compute.v1.instances.insert
  args:
    project: ${projectId}
    zone: europe-west1-b
    body:
      name: my-machine
      machineType: zones/europe-west1-b/e2-small
      disks:
        - initializeParams:
            sourceImage: "projects/debian-cloud/global/images/debian-10-buster-v123"
            boot: true
            autoDelete: true
          networkInterfaces:
            - network: "global/networks/default"
```

## STANDARD LIBRARY

```
# https://cloud.google.com/workflows/docs/reference/stdlib/overview
## http
- http_get:
  call: http.get
  args:
    url: https://en.wikipedia.org/w/api.php
    headers:
      Content-Type: "text/plain"
    query:
      action: opensearch
      search: monday
  result: wikiResult
- http_post:
  call: http.post
  args:
    url: https://us-central1-myproject.cloudfunctions.net/myfunc
    auth:
      type: OIDC
    body:
      some_val: "Hello World"
      another_val: 123
  result: the_message
## sys
- log:
  call: sys.log
  args:
    data: ${wikiResult}
- get_env_vars:
  assign:
    - projectNumber: ${sys.get_env("GOOGLE_CLOUD_PROJECT_NUMBER")}
    - projectId: ${sys.get_env("GOOGLE_CLOUD_PROJECT_ID")}
    - location: ${sys.get_env("GOOGLE_CLOUD_LOCATION")}
    - workflowId: ${sys.get_env("GOOGLE_CLOUD_WORKFLOW_ID")}
    - workflowRevisionId: ${sys.get_env("GOOGLE_CLOUD_WORKFLOW_REVISION_ID")}
    - workflowExecutionId: ${sys.get_env("GOOGLE_CLOUD_WORKFLOW_EXECUTION_ID")}
- get_now:
  assign:
    - now: ${sys.now()}
- wait:
  call: sys.sleep
  args:
    seconds: 10
```



<https://cloud.google.com/workflows/docs/reference/syntax/>

## RAISE ERRORS

```
- raise_custom_string_error:
  raise: "Something went wrong."
- raise_custom_map_error:
  raise:
    code: 55
    message: "Something went wrong."
```

## CATCH ERRORS

```
- try_retry_except:
  try:
    steps: # steps is only needed if multiple steps
    - step_a:
        call: http.get
        args:
          url: https://host.com/api
        result: api_response1
    - step_b:
        call: http.get
        args:
          url: https://host.com/api2
        result: api_response2
  # retry is optional
  # Either, you can use a retry with default policy
  # retry: ${http.default_retry}
  # Or, you can use a more fine-grained policy
  # retry:
  #   predicate: ${http.default_retry_predicate}
  #   max_retries: 10
  #   backoff:
  #     initial_delay: 1
  #     max_delay: 90
  #     multiplier: 3
  except:
    as: e
    steps:
      - known_errors:
          switch:
            - condition: ${not("HttpError" in e.tags)}
              return: "Connection problem."
            - condition: ${e.code == 404}
              return: "Sorry, URL wasn't found."
            - condition: ${e.code == 403}
              return: "Authentication error."
      - unhandled_exception:
          raise: ${e}
```

## RETURN FROM WORKFLOW

```
- return_multiple_values:
  return:
    my_integer: ${my_integer}
    my_string: ${my_string}
    my_true: ${my_true}
    my_false: ${my_false}
    my_list: ${my_list}
    my_multi_dimen_list: ${my_multi_dimen_list}
    my_map: ${my_map}
    my_nested_map: ${my_nested_map}
- return_single_value:
  return: ${my_integer}
```

