perplexity

HashiCorp Certified: Terraform Associate Exam Sample Questions - HCL Code and Functions Focus

This comprehensive collection of sample questions is specifically designed to help you prepare for the HashiCorp Certified: Terraform Associate exam, with a particular emphasis on HashiCorp Configuration Language (HCL) syntax and Terraform's built-in functions[1][2].

Overview

The Terraform Associate certification exam tests your knowledge of Infrastructure as Code (IaC) concepts, Terraform fundamentals, and practical HCL usage[3][4]. These sample questions cover the essential areas you'll encounter on the actual exam, focusing on hands-on coding scenarios and function implementations that are commonly tested[1][5].

HCL Syntax Fundamentals

Variable Declaration and Reference

Understanding proper HCL syntax for variables is crucial for the exam[2][6]. The exam frequently tests knowledge of how to correctly define variables with default values and proper interpolation syntax.

Sample Question 1:
Which of the following is the correct HCL syntax for defining a variable with a default value?

A. `variable "instance_type" { default = "t2.micro" }`
B. `var instance_type = "t2.micro"`
C. `define variable instance_type default "t2.micro"`
D. `variable instance_type default = "t2.micro"`

Answer: A
In HCL, variables are defined using the 'variable' block with the variable name in quotes, and default values are specified using the 'default' argument[2][6].

Sample Question 2:
What is the correct interpolation syntax to reference a variable named 'vpc_id' in Terraform?

A. `${vpc_id}`
B. `var.vpc_id`
C. `${var.vpc_id}`
D. `@{var.vpc_id}`

Answer: B
In Terraform 0.12+, the preferred syntax is `var.vpc_id`. While `${var.vpc_id}` still works, the newer syntax without interpolation braces is recommended for simple references[7][8].

Map and Object Access Patterns

The exam tests your understanding of how to access nested data structures[1][9].

Sample Question 3:
Consider the following HCL code block:

```
variable "vpc_cidrs" {
  type = map(string)
  default = {
    us-east-1 = "10.0.0.0/16"
    us-west-2 = "10.1.0.0/16"
  }
}
```

How would you reference the CIDR block for us-east-1 in a resource?

A. `var.vpc_cidrs.us-east-1`
B. `var.vpc_cidrs["us-east-1"]`
C. `${var.vpc_cidrs.us-east-1}`
D. `var.vpc_cidrs`

Answer: B
For map variables, you access values using bracket notation with the key as a string: `var.vpc_cidrs["us-east-1"]`[1][9].

String Functions

String manipulation functions are heavily tested on the associate exam[10][11][12].

Join and Split Functions

Sample Question 4:
Which of the following functions would you use to combine a list of strings into a single string with a delimiter?

A. `concat()`
B. `join()`
C. `merge()`
D. `format()`

Answer: B
The `join()` function combines elements of a list into a single string with a specified separator. Syntax: `join(separator, list)`[10][13].

Sample Question 5:
What will be the output of the following HCL expression: `split(",", "apple,banana,cherry")`?

A. `"apple banana cherry"`
B. `["apple", "banana", "cherry"]`
C. `{"apple", "banana", "cherry"}`
D. `apple,banana,cherry`

Answer: B
The `split()` function divides a string into a list of strings based on the specified separator. It returns a list data type[10][13].

String Transformation Functions

Sample Question 6:
Which function would you use to convert a string to uppercase in Terraform?

A. `upper()`
B. `uppercase()`
C. `toupper()`
D. `capitalize()`

Answer: A
The `upper()` function converts all letters in a string to uppercase. Its counterpart `lower()` converts to lowercase[10][14].

Collection Functions

Collection functions help manipulate lists, maps, and sets[10][11].

Sample Question 7:
Which function would you use to get the number of elements in a list or map?

A. `count()`
B. `size()`
C. `length()`
D. `elements()`

Answer: C
The `length()` function returns the number of elements in a list, map, set, or string[10][11].

Sample Question 8:
What does the `lookup()` function do in Terraform?

A. Searches for files in the filesystem
B. Retrieves a value from a map given a key, with an optional default
C. Looks up DNS records
D. Finds resources in the state file

Answer: B
The `lookup()` function retrieves a value from a map using a key. If the key doesn't exist, it returns a default value if provided, or fails if no default is given[10][11].

Sample Question 9:
Which function would you use to merge two maps together in HCL?

A. `combine()`
B. `merge()`
C. `join()`
D. `concat()`

Answer: B
The `merge()` function combines multiple maps into a single map, with later arguments taking precedence for duplicate keys[10][11].

Meta-Arguments: for_each and count

Understanding meta-arguments is essential for the exam[15][16][17].

Sample Question 10:
What is the correct syntax for using `for_each` with a set of strings?

A. `for_each = ["web", "app", "db"]`
B. `for_each = toset(["web", "app", "db"])`
C. `for_each = {"web", "app", "db"}`
D. `for_each = set(["web", "app", "db"])`

Answer: B
The `for_each` meta-argument requires a map or set of strings. When using a list, you must convert it to a set using `toset()`[15][16].

Sample Question 11:
In a resource block using `for_each`, how do you reference the current key?

A. `current.key`
B. `each.key`
C. `for.key`
D. `this.key`

Answer: B
When using `for_each`, the special 'each' object provides access to `each.key` (current key) and `each.value` (current value)[15][16].

Type Conversion Functions

Type conversion is frequently tested[18][11].

Sample Question 12:
Which function converts a list to a set in Terraform?

A. `toset()`
B. `set()`
C. `list_to_set()`
D. `convert_set()`

Answer: A
The `toset()` function converts a list to a set, removing any duplicate values in the process[18][11].

Sample Question 13:
What happens when you use `tostring()` on a boolean value?

A. It returns an error
B. It converts true to "1" and false to "0"
C. It converts true to "true" and false to "false"
D. It returns null

Answer: C
The `tostring()` function converts boolean values to their string representations: true becomes "true" and false becomes "false"[18][11].

Advanced HCL Expressions

Conditional Expressions

Sample Question 14:
Which HCL expression correctly creates a conditional that sets instance_type to 't2.micro' if environment is 'dev', otherwise 't2.small'?

A. `var.environment == "dev" ? "t2.micro" : "t2.small"`
B. `if var.environment == "dev" then "t2.micro" else "t2.small"`
C. `var.environment == "dev" -> "t2.micro" | "t2.small"`
D. `case var.environment when "dev" then "t2.micro" else "t2.small"`

Answer: A
Terraform uses the ternary operator (condition ? true_value : false_value) for conditional expressions[19][20].

For Expressions

Sample Question 15:
Examine the following HCL code:

```
locals {
  environments = ["dev", "staging", "prod"]
  instance_configs = {
    for env in local.environments : env => {
      instance_type = env == "prod" ? "m5.large" : "t3.micro"
      count = env == "prod" ? 3 : 1
      }
    }
  }
```

What type of expression is being used to create the instance_configs map?

A. for_each expression
B. for expression
C. conditional expression
D. dynamic expression

Answer: B
This is a 'for' expression that iterates over the environments list to create a map. The syntax 'for env in local.environments : env ⇒ {...}' creates a map where keys are environment names and values are configuration objects[20][21].

Complex Function Combinations

Sample Question 16:
What does the following HCL expression evaluate to:

```
length(flatten([
  ["a", "b"],
  ["c", "d", "e"],
  ["f"]
]))
```

A. 3
B. 6
C. `["a", "b", "c", "d", "e", "f"]`
D. `[["a", "b"], ["c", "d", "e"], ["f"]]`

Answer: B
The `flatten()` function converts the nested list structure into a single flat list `["a", "b", "c", "d", "e", "f"]`, then `length()` returns the count of elements, which is 6[11][10].

Sample Question 17:
What will be the output of this HCL expression:

```
join("-", [for s in ["Hello", "World", "Terraform"] : lower(s)])
```

A. `"Hello-World-Terraform"`
B. `"hello-world-terraform"`
C. `["hello", "world", "terraform"]`
D. `"hello world terraform"`

Answer: B
The for expression converts each string to lowercase using `lower()`, creating `["hello", "world",`

`"terraform"]`, then `join()` combines them with hyphens[20][10].

File System Functions

Sample Question 18:
Which function would you use to read the contents of a file into a Terraform configuration?

A. `read_file()`
B. `file()`
C. `get_file()`
D. `load_file()`

Answer: B
The `file()` function reads the contents of a file at the given path and returns it as a string[10][11].

Sample Question 19:
What does the `fileexists()` function return?

A. The file contents if it exists, null otherwise
B. The file path if it exists, error otherwise
C. A boolean indicating whether the file exists
D. The file size if it exists

Answer: C
The `fileexists()` function returns true if the file exists at the given path, false otherwise[10][11].

Dynamic Blocks

Sample Question 20:
What is the purpose of dynamic blocks in HCL?

A. To create resources dynamically at runtime
B. To generate nested configuration blocks based on complex values
C. To import existing infrastructure
D. To create conditional resources

Answer: B
Dynamic blocks allow you to generate nested configuration blocks (like ingress rules in security groups) based on complex values like lists or maps[20][21].

Validation and Error Scenarios

Sample Question 21:
What does the `contains()` function return when checking if a list contains a specific value?

A. The index of the element
B. The element itself
C. A boolean value (true/false)
D. The number of occurrences

Answer: C
The `contains()` function returns a boolean value: true if the list contains the specified value, false otherwise[10][14].

Study Tips and Best Practices

Key Areas to Focus On

1. **String Functions**: Master `join()`, `split()`, `format()`, `upper()`, `lower()`, and `replace()` functions[10][12][13]
2. **Collection Functions**: Understand `length()`, `merge()`, `flatten()`, `contains()`, and `lookup()` [10][11]
3. **Type Conversion**: Practice with `toset()`, `tolist()`, `toString()`, `tonumber()`, and `tobool()`[18][11]
4. **Meta-Arguments**: Know when and how to use `for_each` vs `count`[15][16]
5. **For Expressions**: Practice creating lists and maps using for expressions[20][21]
6. **Conditional Logic**: Master ternary operators and conditional expressions[19]

Common Exam Patterns

The exam frequently tests[1][3][5]:

- Variable interpolation syntax and best practices
- Function combinations and nested expressions
- Error identification in HCL code
- Meta-argument usage scenarios
- Data structure manipulation

Practice Recommendations

1. Use `terraform console` to experiment with functions[10][11]
2. Practice writing complex for expressions[20][21]
3. Study the official Terraform documentation for function syntax[4][22]
4. Work through hands-on labs to reinforce concepts[23][24]
5. Take multiple practice exams to identify knowledge gaps[25][26][27]

This collection of sample questions provides a solid foundation for understanding the HCL syntax and function usage patterns you'll encounter on the HashiCorp Certified: Terraform Associate exam. Focus on understanding the underlying concepts rather than memorizing answers, as the exam tests practical application of Terraform knowledge in real-world scenarios[1][4].