Terraform Function

OW

Functions

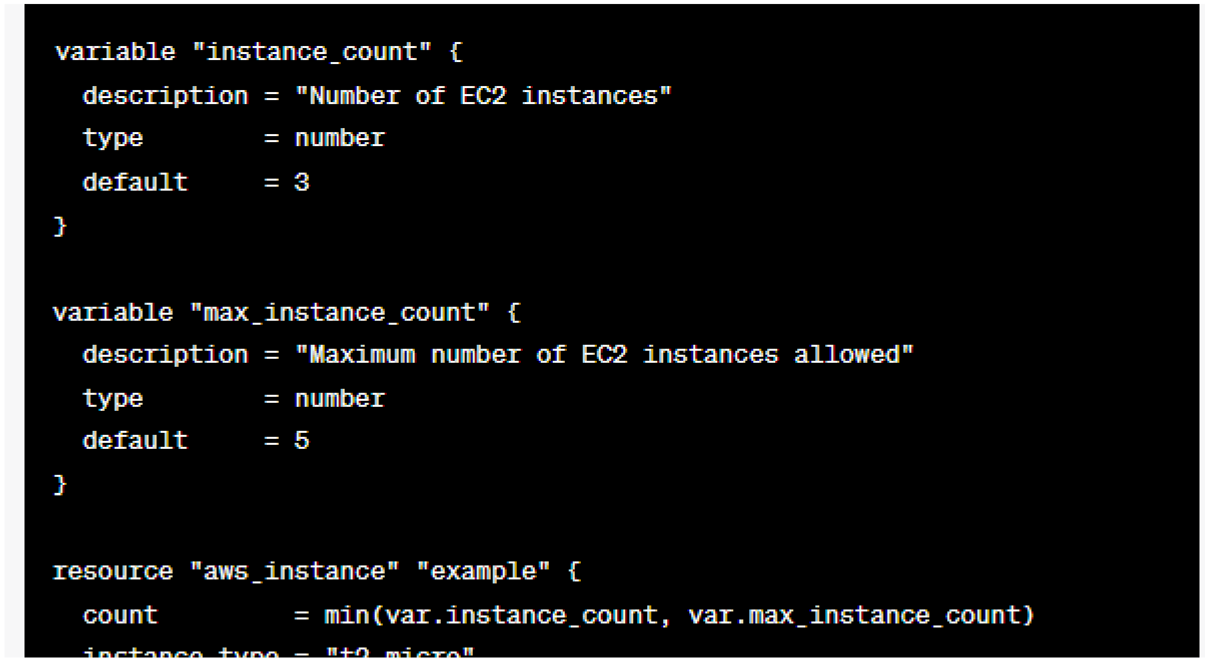
- Terraform includes a number of built-in functions that you can call from with in expression to transform and combine values.
- The general syntax for functions call is a function name followed by comma-separated in parantheses

```
max(5, 12, 9)
```

- Terraform doesn’t support user-defined functions.

Numeric Functions

- Max
- Min



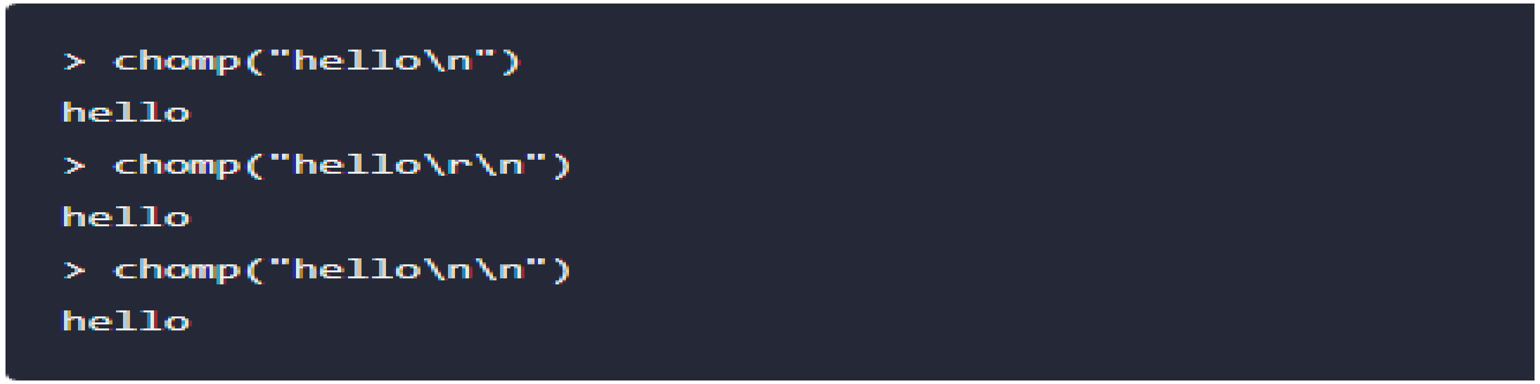
```
variable "instance_count" {
  description = "Number of EC2 instances"
  type        = number
  default     = 3
}

variable "max_instance_count" {
  description = "Maximum number of EC2 instances allowed"
  type        = number
  default     = 5
}

resource "aws_instance" "example" {
  count         = min(var.instance_count, var.max_instance_count)
  instance_type = "t2.micro"
```

string

- Chomp
- Format
- Join
- Regex
- Replace
- Split

chomp

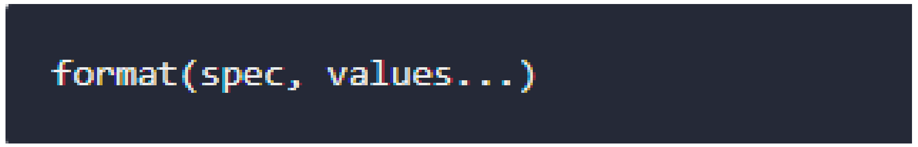
- chomp removes newline characters at the end of a string.
- This can be useful if, for example, the string was read from a file that has a newline character at the end.



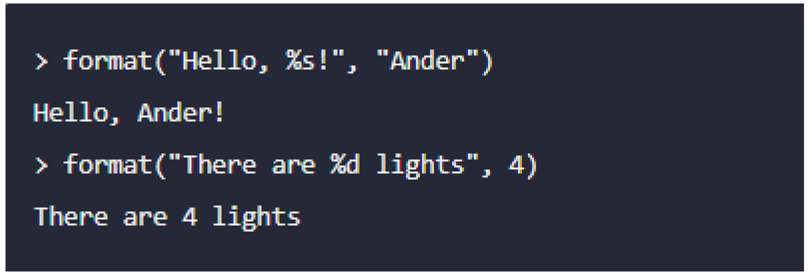
```
> chomp("hello\n")
hello
> chomp("hello\r\n")
hello
> chomp("hello\n\n")
hello
```

format

- The format function produces a string by formatting a number of other values according to a specification string. It is similar to the printf function in C, and other similar functions in other programming languages.



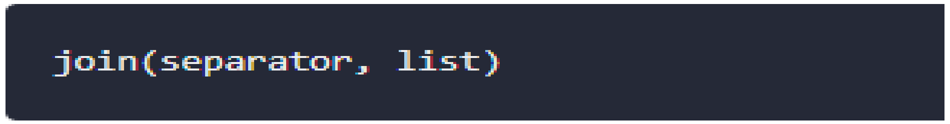
```
format(spec, values...)
```



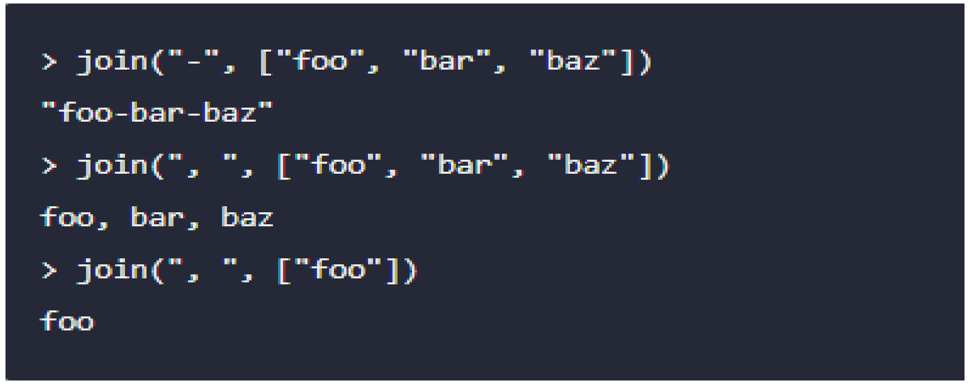
```
> format("Hello, %s!", "Ander")
Hello, Ander!
> format("There are %d lights", 4)
There are 4 lights
```

Join

- join produces a string by concatenating all of the elements of the specified list of strings with the specified separator.



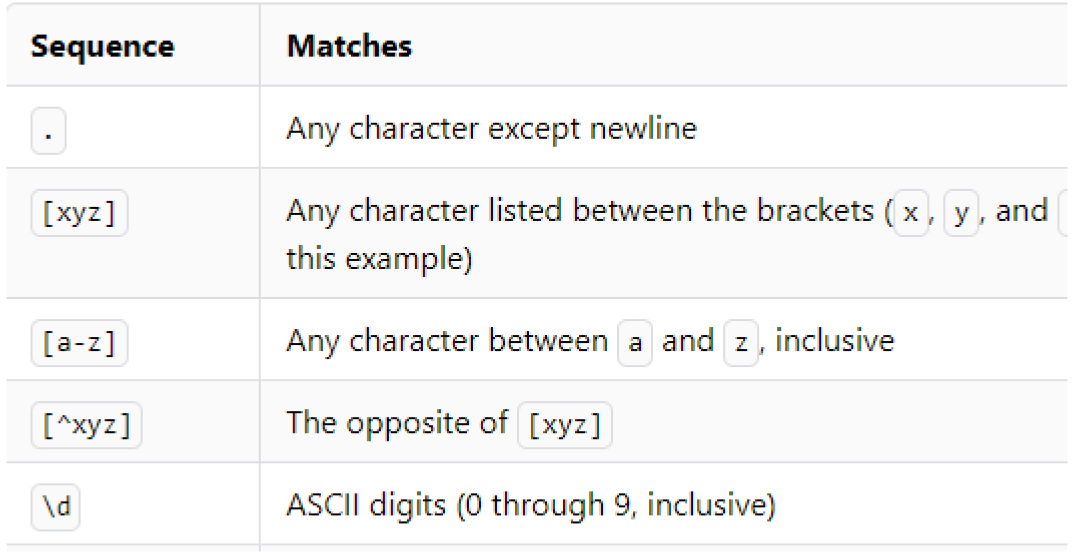
```
join(separator, list)
```



```
> join("-", ["foo", "bar", "baz"])
"foo-bar-baz"
> join(", ", ["foo", "bar", "baz"])
foo, bar, baz
> join(", ", ["foo"])
foo
```

regex

- regex applies a regular expression to a string and returns the matching substrings.



Sequence	Matches
`.`	Any character except newline
`[xyz]`	Any character listed between the brackets (`x` , `y` , and this example)
`[a-z]`	Any character between `a` and `z` , inclusive
`[^xyz]`	The opposite of `[xyz]`
`\d`	ASCII digits (0 through 9, inclusive)

replace

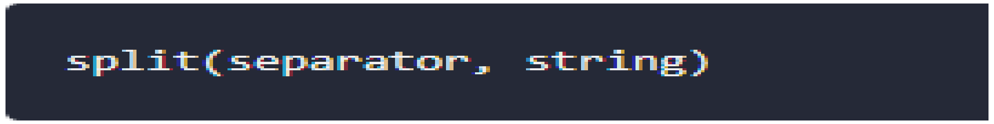
- replace searches a given string for another given substring, and replaces each occurrence with a given replacement string.

```
replace(string, substring, replacement)
```

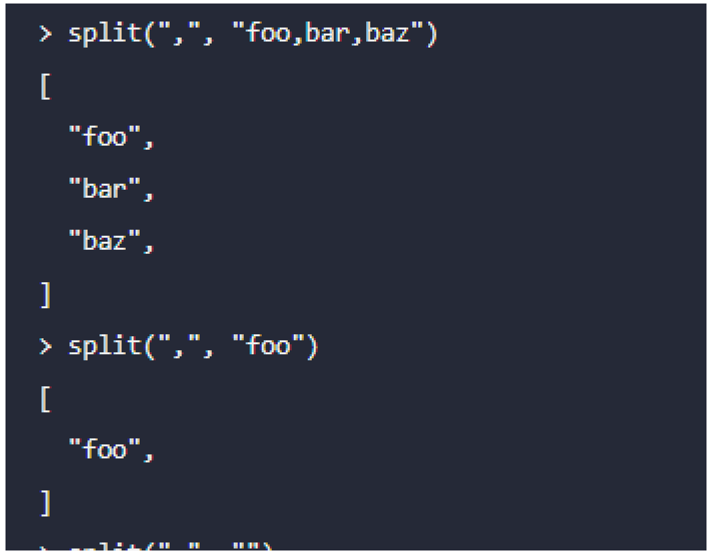
```
> replace("1 + 2 + 3", "+", "-")
1 - 2 - 3

> replace("hello world", "/w.*d/", "everybody")
hello everybody
```

split

- split produces a list by dividing a given string at all occurrences of a given separator.

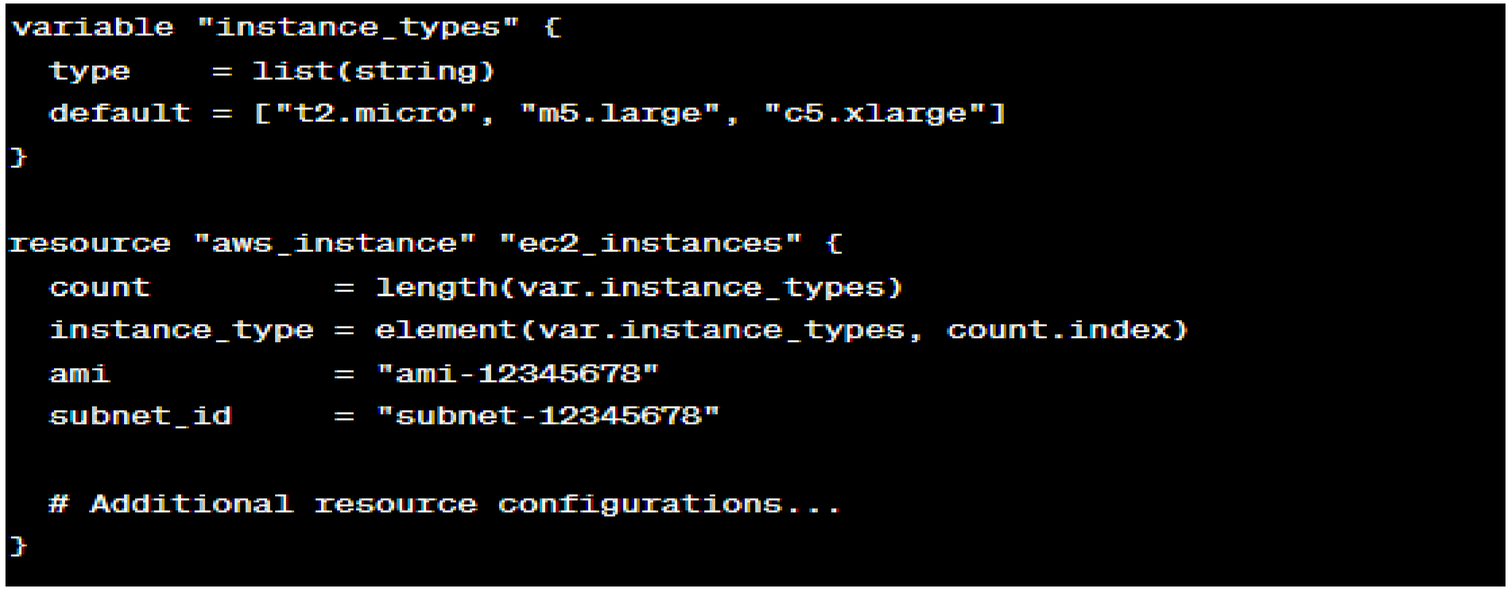


```
split(separator, string)
```



```
> split(",", "foo,bar,baz")
[
  "foo",
  "bar",
  "baz",
]
> split(",", "foo")
[
  "foo",
]
```

example



```
variable "instance_types" {
  type    = list(string)
  default = ["t2.micro", "m5.large", "c5.xlarge"]
}

resource "aws_instance" "ec2_instances" {
  count         = length(var.instance_types)
  instance_type = element(var.instance_types, count.index)
  ami           = "ami-12345678"
  subnet_id     = "subnet-12345678"

  # Additional resource configurations...
}
```

In the aws_instance resource block, we use the count argument and set it to the length of the instance_types list. This allows us to dynamically create multiple EC2 instances based on the length of the list.

The instance_type argument is set using the element() function. We pass in the instance_types list and the current index count.index to retrieve the corresponding instance type for each iteration of the resource creation.