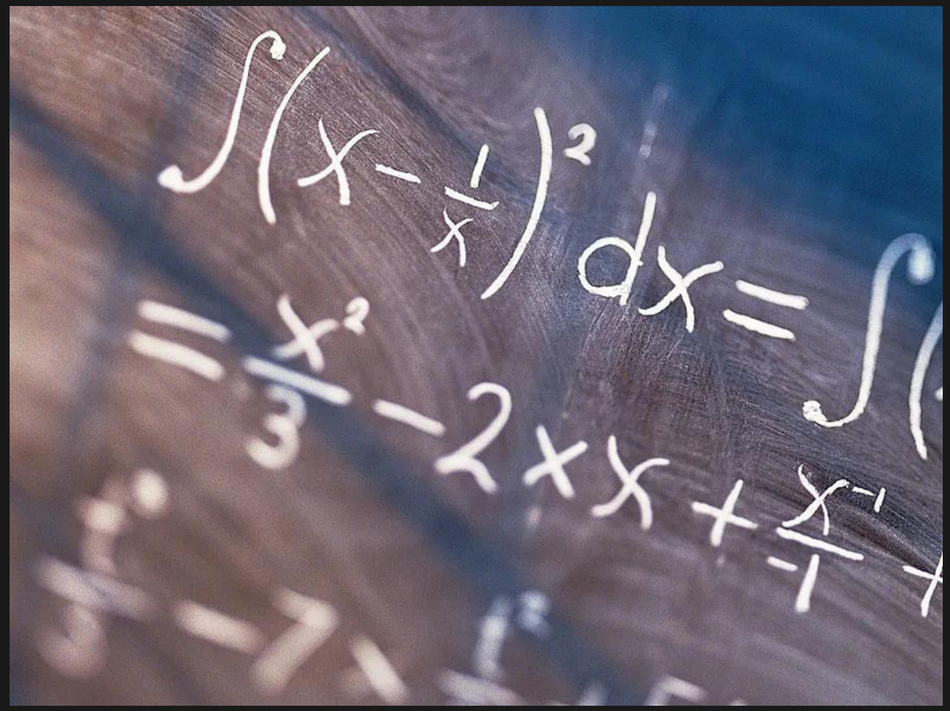
Terraform HCL
Advanced

HashiCorp
Terraform

Operators

- Terraform supports basic arithmetic operators
  - Addition, multiplication, division
  - Concatenating string by adding two strings
- Equality & Comparison operators
  - Compare the value of two variables, returning a boolean
  - a == b, returns true only if a & b have the same value and the same data type
  - a != b, returns true only if a & b does not have the same value, or the same data type
  - Usual comparison operators: >=, >, <, <=
- Logical operators
  - && (and) || (or) - behaves similar to other languages
  - Used to chain comparisons for conditional logic



Conditional Expressions

- Powerful expression to perform conditional logic, via ternary syntax
- If value a is true, then assign value a to argument, otherwise value b
- This is commonly used to conditionally create resources, or conditionally set values to arguments

```
resource "aws_instance" "example_bionic" {
  count                  = var.enable_new_ami ? 1 : 0
  instance_type          = "t2.micro"
  ami                    = data.aws_ami.ubuntu_bionic.id
  vpc_security_group_ids = [aws_security_group.instances.id]
  subnet_id              = aws_subnet.public.id
  tags = {
    Has_Toggle = var.enable_new_ami
  }
}
```

Conditional Expressions

- count - special keyword to create an array / list of that resource
  - Instead of just one aws_instance.example_bionic, you can create an array of them, accessible by index
- count allows you to conditionally create resources
  - If the variable var.enable_new_ami returns true, set count to 1, otherwise to 0
  - Approach is really common in feature flagging - only certain resources are created if conditions are met

```
resource "aws_instance" "example_bionic" {
  count                  = var.enable_new_ami ? 1 : 0
  instance_type          = "t2.micro"
  ami                    = data.aws_ami.ubuntu_bionic.id
  vpc_security_group_ids = [aws_security_group.instances.id]
  subnet_id              = aws_subnet.public.id
  tags = {
    Has_Toggle = var.enable_new_ami
  }
}
```

String Interpolation

- String interpolation is an expression which gets evaluated, and returns the result into a string
  - Denoted by ${....} brackets
  - Used inside a string to enable complex logic and evaluating expressions within the string
- "Hello, ${var.name}"
  - String interpolation takes whatever expression is inside ${....} and returns result into a string
  - Takes the value of variable var.name, returns it, and creates the string: "Hello, Bob"
- "The name is ${var.last_name}, ${var.first_name} ${var.last_name}"
  - If var.last_name = Bond, and var.first_name = James, what does this string evaluate to?
- "Hello, ${var.name != "" ? var.name : "Bob"}"
  - String interpolation can allow for powerful expressions
  - Here we utilize conditional expressions to dynamically create this string
  - If var.name has a value, then use it, otherwise assign a default value of Bob

Calling Functions

- Terraform comes in with suite of built-in functions, depending on version of Terraform you use
- Functions perform some set of logic, with the goal of achieving a specific result
  - max(...args): function which takes in any number of values, and returns the maximum value from its arguments
    - max(1,3,5) -> 5
  - merge(object, object, ...): takes arbitrary number of maps or objects, combines them to return a single map or object
- Built in functions cover a large array of usecases, from numeric, to string functions, encoding, etc.
  - Unfortunately, not able to create your own functions as of this time