tf.make_template

```
make_template(
    name_,
    func_,
    create_scope_now_=False,
    unique_name_=None,
    custom_getter_=None,
    **kwargs
)
```

Defined in `tensorflow/python/ops/template.py`.

See the guide: Variables > Sharing Variables

Given an arbitrary function, wrap it so that it does variable sharing.

This wraps `func_` in a Template and partially evaluates it. Templates are functions that create variables the first time they are called and reuse them thereafter. In order for `func_` to be compatible with a `Template` it must have the following properties:

- The function should create all trainable variables and any variables that should be reused by calling `tf.get_variable`. If a trainable variable is created using `tf.Variable`, then a ValueError will be thrown. Variables that are intended to be locals can be created by specifying `tf.Variable(..., trainable=false)`.
- The function may use variable scopes and other templates internally to create and reuse variables, but it shouldn't use `tf.global_variables` to capture variables that are defined outside of the scope of the function.
- Internal scopes and variable names should not depend on any arguments that are not supplied to `make_template`. In general you will get a ValueError telling you that you are trying to reuse a variable that doesn't exist if you make a mistake.

In the following example, both `z` and `w` will be scaled by the same `y`. It is important to note that if we didn't assign `scalar_name` and used a different name for z and w that a `ValueError` would be thrown because it couldn't reuse the variable.

```
def my_op(x, scalar_name):
  var1 = tf.get_variable(scalar_name,
                         shape=[],
                         initializer=tf.constant_initializer(1))
  return x * var1

scale_by_y = tf.make_template('scale_by_y', my_op, scalar_name='y')

z = scale_by_y(input1)
w = scale_by_y(input2)
```

As a safe-guard, the returned function will raise a `ValueError` after the first call if trainable variables are created by calling `tf.Variable`.

If all of these are true, then 2 properties are enforced by the template:

1. Calling the same template multiple times will share all non-local variables.
2. Two different templates are guaranteed to be unique, unless you reenter the same variable scope as the initial

definition of a template and redefine it. An examples of this exception:

```
def my_op(x, scalar_name):
  var1 = tf.get_variable(scalar_name,
                         shape=[],
                         initializer=tf.constant_initializer(1))
  return x * var1

with tf.variable_scope('scope') as vs:
  scale_by_y = tf.make_template('scale_by_y', my_op, scalar_name='y')
  z = scale_by_y(input1)
  w = scale_by_y(input2)

# Creates a template that reuses the variables above.
with tf.variable_scope(vs, reuse=True):
  scale_by_y2 = tf.make_template('scale_by_y', my_op, scalar_name='y')
  z2 = scale_by_y2(input1)
  w2 = scale_by_y2(input2)
```

Depending on the value of `create_scope_now_`, the full variable scope may be captured either at the time of first call or at the time of construction. If this option is set to True, then all Tensors created by repeated calls to the template will have an extra trailing _N+1 to their name, as the first time the scope is entered in the Template constructor no Tensors are created.

★ **Note:** name_, `func_` and `create_scope_now_` have a trailing underscore to reduce the likelihood of collisions with kwargs.

Args:

- `name_`: A name for the scope created by this template. If necessary, the name will be made unique by appending `_N` to the name.
- `func_`: The function to wrap.
- `create_scope_now_`: Boolean controlling whether the scope should be created when the template is constructed or when the template is called. Default is False, meaning the scope is created when the template is called.
- `unique_name_`: When used, it overrides name_ and is not made unique. If a template of the same scope/unique_name already exists and reuse is false, an error is raised. Defaults to None.
- `custom_getter_`: Optional custom getter for variables used in `func_`. See the `tf.get_variable` `custom_getter` documentation for more information.
- `**kwargs`: Keyword arguments to apply to `func_`.

Returns:

A function to encapsulate a set of variables which should be created once and reused. An enclosing scope will created, either where `make_template` is called, or wherever the result is called, depending on the value of `create_scope_now_`. Regardless of the value, the first time the template is called it will enter the scope with no reuse, and call `func_` to create variables, which are guaranteed to be unique. All subsequent calls will re-enter the scope and reuse those variables.

Raises:

- `ValueError`: if the name is None.