(a) List all of the input variables, including the state variables.							
Input variables:	template_name_or_list	**context	source	ex. flask.render_template("template_filter.html", value="abcd")			
State variables:							
(b) Define the characteristics of the input variables. Make sure you cover all input variables.							
Method	**Params**	**Returns**	**Values**	**Exception**	**Ch ID**		
render_template	None				C2		
	template_name	str	string		C4		
	template_name (with not exist template name)		string		C2		
	template_name, dictionary	str	string		C1		
	template_name, dictionary (with malformed template)				C3		
	template_name, dictionary (no matching variable between template and dictionary)				C5		
render_template_string	None			TypeError	C6		
	source	str	string		C8		
	source, dictionary	str	string		C1		
	source, dictionary (with malformed source)			TemplateSyntaxError	C7		
	source, dictionary (no matching variable between template and dictionary)				C9		
(c) Partition the characteristics into blocks. Designate one block in each partition as the "Base" block.							
ID	Characteristic	render_template()	render_template_string()				
C1	if dictionary is not empty	v	v				
C2	if templates not found	v					
C3	if template is malformed (invalid jinja2 syntax)	v					
C4	if template conatins no jinja syntax	v					
C5	if template variable don't match dictionary	v		inconsistent variable type			
C6	if string is None		v				
C7	if string is malformed (invalid jinja2 syntax)		v				
C8	if string conatins no jinja syntax		v				
C9	if string's variable don't match dictionary		v				
(d) Define values for each block.							
		Value	Partition				

C1	if dictionary is not empty	template_name=test.html, text=" hello";	{true,false}				
C2	if templates not found	template_name=null; template_name does not exists	{true,false}				
C3	if template is malformed (invalid jinja2 syntax)	{% if ... %} ...	{true,false}	expected: {% endif %}			
C4	if template conatins no jinja syntax	template_name=test.html;	{true,false}				
C5	if template variable don't match dictionary	template_name=test.html, txt=" hello";	{true,false}				
C6	if string is None	source=None	{true,false}				
C7	if string is malformed (invalid jinja2 syntax)	source="{text}", text="hello"	{true,false}				
C8	if string conatins no jinja syntax	source="hello"	{true,false}				
C9	if string's variable don't match dictionary	source="{text}", message="hello"	{true,false}				
(e) Define a test set that satisfies Base Choice Coverage (BCC). Write your tests with the values from the previous step. Be sure to include the test oracles.							
Method	characteristic	Test Requirements	Infeasible TRs	Revises TRs	#TRs		
render_template	C1,C2,C3,C4,C5	{TTTTT, FTTTT, TFTTT, TTFTT, TTTFT, TTTTF}	TTTTT, FTTTT, TFTTT, TTFTT, TTTFT, TTTTF	TTTTT->TFTTT->TFTFT, FTTTT->FFTTT->FFTFT TTFTT->TFFTT->TFFFT, TTTFT->TFTFT->TFTFF, TTTTF->TFTTF->TTFTF	5		
render_template_string()	C1,C6,C7,C8,C9	{TTTTT, FTTTT, TFTTT, TTFTT, TTTFT, TTTTF}	TTTTT, FTTTT, TFTTT, TTFTT, TTTFT, TTTTF	TTTTT->TFTTT->TFTFT, FTTTT->FFTTT->FFTFT TTFTT->TFFTT->TFFFT, TTTFT->TFTFT->TFTFF, TTTTF->TFTTF->TTFTF	5		
Testcases							
			render_template				
			TFTFT, render_template("template_invalid_syntax_mismatch.html", dict)				
			FFTFT, render_template("template_invalid_syntax_mismatch.html")				
			TFFFT, render_template("template_mismatch.html", dict)				
			TFTFF, render_template("template_invalid_syntax_match.html", dict)				
			TTFTF, render_template("not_found.html", dict)				
			render_template_string				
			TFTFT, render_template_string("content of template_invalid_syntax_mismatch.html", dict)				
			FFTFT, render_template_string("content of "template_invalid_syntax_mismatch.html".html")				
			TFFFT, render_template_string("content of template_mismatch.html", dict)				
			TFTFF, render_template_string("content of template_invalid_syntax_match.html", dict)				
			TTFTF, render_template_string(None, dict)				