SLAX to Python Cheat Sheet

SLAX Extensions/Functions	Python Extensions/Functions	Description
<xnm:warning> { <message> “string”; }	jcs.emit_warning(‘string’)	Emit warning from automation scripts.
<xnm:error> { <message> “string”; }	jcs.emit_error(‘string’)	Emit errors from automation scripts.
expr jcs:output(‘string’)	jcs.output(‘string’) (OR) print ‘string’	Generate unformatted output text that is immediately sent to the CLI session.
var $user-input = jcs:get-input(*string*);	user_input=jcs.get_input(‘string’) (OR) user_input = jcs.getinput(‘string’) (OR) user_input = raw_input(‘ ‘)	Invoke a CLI prompt and wait for user input.
var $user-input = jcs:get-secret(*string*);	input = jcs.get_secret(‘string’) (OR) input = jcs.getsecret(‘string’)	Invoke a CLI prompt and wait for user input, the input is not echoed back to the user.
expr jcs:printf(*expression*);	jcs.printf(expression)	Generate formatted output text. Most standard printf formats are supported.
expr jcs:progress('*string*');	jcs.progress(‘string’)	Issue a progress message containing the single argument immediately to the CLI session provided that the detail flag was specified when the script was invoked.
expr jcs:syslog(*priority*, *message*, <*message2*>…)	jcs.syslog(priority,<message>, <message2>…)	Log messages with the specified priority to the system log file. The priority can be expressed as a *facility*.*severity* string or as a

		calculated integer. The *message* argument is a string or variable that is written to the system log file.
expr jcs:trace('*expression*');	jcs.trace(expression)	Issue a trace message, which is sent to the trace file. You must configure traceoptions under the respective script type in the configuration hierarchy in order to output the jcs.trace message to the trace file. The output goes to configured trace file. If traceoptions is enabled, but no trace file is explicitly configured, the output goes to the default trace file for that script type.
var $name = jcs:hostname(*expression*);	name = jcs.hostname(expression)	Return the fully qualified domain name associated with a given IPv4 or IPv6 address. The DNS server must be configured on the device in order to resolve the domain name.
var $result = jcs:dampen(*tag-string*, *max*, *interval*);	retval = jcs.dampen((tag-string, max, interval)	Prevent the same operation from being repeatedly executed within a script. The dampen function returns false if the number of calls to the jcs.dampen() function exceeds a *max* number of calls in the time interval *interval*. Otherwise the function returns true. The function parameters include an arbitrary string that is used to distinguish different calls to the jcs.dampen() function. This tag is stored in the /var/run directory on the device.
var $value = jcs:sysctl(*sysctl-value*, "(i | s)");	ret = jcs.sysctl(sysctl-value, “i | s”)	Return the given sysctl value as a string or an integer. Use the "i" argument to specify an integer. Use the "s" argument to specify a string
call jcs:emit-change($dot=*expression*, $name = name($dot), $tag = "(change | transient-change)" {	jcs.emit_change(‘<xml-string/>’, “change”, “xml”) (OR)	JUNOS configuration in the form of xml string will get committed from Python commit scripts with ‘change’

with $content = {
...
}
	jcs.emit_change('<xml-string/>', "transient-change", "xml")	(or) 'transient-change' tags and 'xml' as the supported format.

var $connection = jcs:open(*remote-hostname*, <*username*>, <*passphrase*>, <*routing-instance-name*>);	dev=Device(host=$host, user="*username*", passwd="*password*") dev.open( )	User can execute Junos PyEZ methods using any user account that has access to the managed device running Junos OS. You can explicitly define the user and password when creating a new instance of the jnpr.junos.device.Device class, or if you do not specify a user in the parameter list, the user defaults to $USER. When using a password, we recommend that the Junos PyEZ code prompt for the password. If you do not provide a password when creating a Device instance, Junos PyEZ defaults to using SSH keys
expr jcs:close($connection)	dev.close( )	Close the connection with device
Needs to Run RPC's and fetch details.	dev.facts	Returns the device facts, which are automatically retrieved when the connection and NETCONF session are established.
var $connection = jcs:open(); use $connection as connection handle.	dev= Device(host='dc1a.example.com') cu = Config(dev)	Creates a device and connection handle cu
var $result = jcs:invoke(*"lock-configuration"*); (OR) var $result = jcs:execute($connection, "lock-configuration");	cu.lock( )	Lock the configuration database.
<load-configuration rescue="rescue"/> <load-configuration rollback="*index*"/> <load-configuration url="*url*" action="set" format="text" /> <load-configuration [action="(merge | override | replace | update)"] [format="xml"]>	cu.load(config_text, format="text",merge=True)	Load the configuration to database.(Supported formats: text/xml/set)

<configuration> <!-- *tag elements for configuration elements to load* --> </configuration> </load-configuration> <load-configuration [action="(merge | override | replace | update)"] format="text"> <configuration-text> <!-- *formatted ASCII configuration statements to load* --> </configuration-text> </load-configuration> <load-configuration action="set" format="text"> <configuration-set> <!-- *set configuration mode commands to load* --> </configuration-set> </load-configuration>	cu.load(config_xml, format="xml",merge=True) cu.load(config_set, format="set",merge=True)	
var $result = jcs:invoke(*"commit-configuration"*); (OR) var $result = jcs:execute($connection, "commit-configuration");	cu.commit( )	Commit to the configuration to database.
var $result = jcs:invoke("unlock-configuration"); (OR) var $result = jcs:execute($connection, "unlock-configuration");	cu.unlock( )	Unlock the configuration database.
expr jcs:sleep(*seconds*, <*milliseconds*>);	import time time.sleep(t) t -- This is the number of seconds execution to be suspended.	Cause the script to pause for a specified number of seconds and (optionally) milliseconds. You can use this function to help determine how a device component works over time.
var $lines = jcs:break-lines(*expression*);	str.splitlines()	Break a simple element into multiple elements, delimited by newlines.
var $result = jcs:parse-ip("ipaddress/(prefix-length | netmask)");	result = jcs.parse_ip(ipaddress/(prefix-length | netmask)")	Parse an IPV4 or IPV6 address.