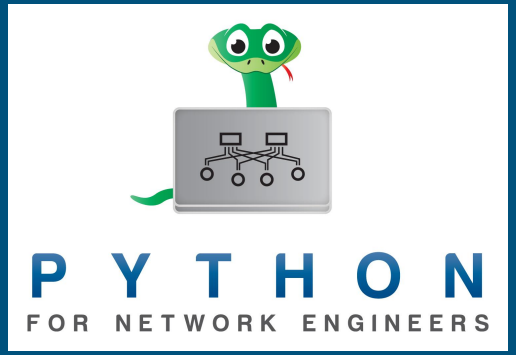
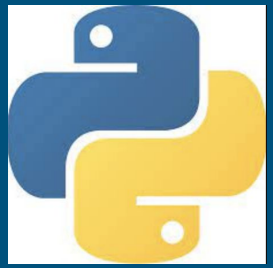




Training Session
November 2022 (Day5)

Day5 Presentation:

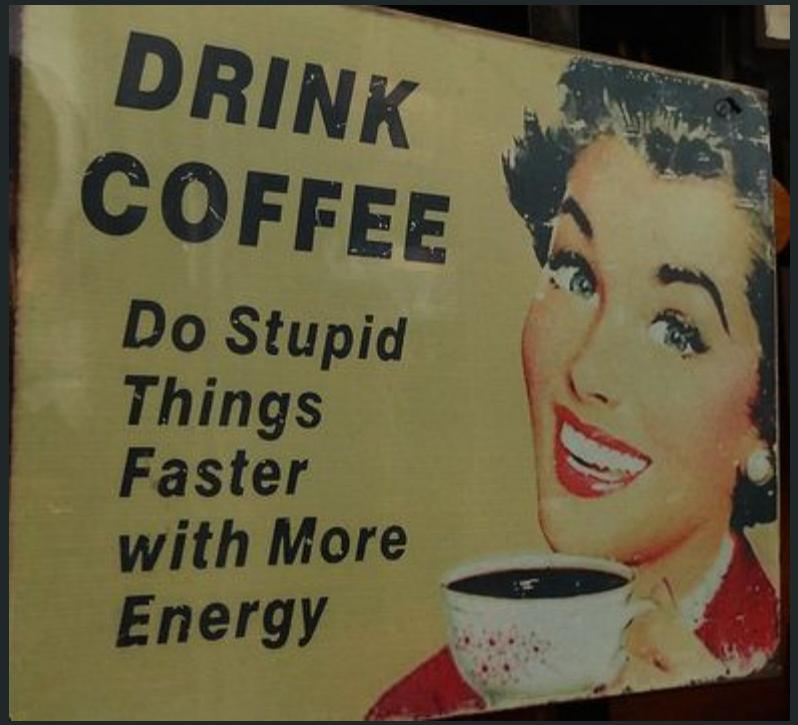
https://github.com/twin-bridges/pynet-ons-oct22/blob/main/python-training-day5.pdf

General:

Training Day
Nov 16th, Day5 (Wed) / 9AM - 5:00PM Central

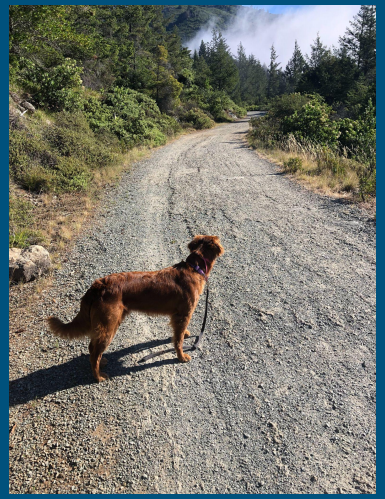
Focused/Minimize Distractions

The exercises are important.



Flickr: Ben Sutherland

Filtering Objects in the Aruba API



In the Aruba API, we can specify that the API should return ONLY some subset of fields using filters.

We accomplish this by adding a query string to the URL with the following the form:

```
&filter=[{"OBJECT": {"$eq": ["int_vlan.int_vlan_ip", "int_vlan.int_vlan_mtu"]}}]
```

Object filter - Meaning we are going to delimit based on parameter names.

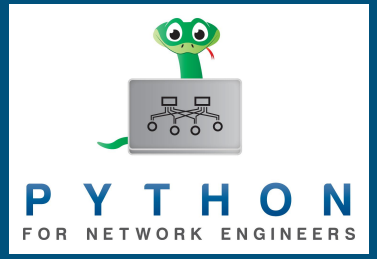
$eq or $neq

The set of fields to return in the output.

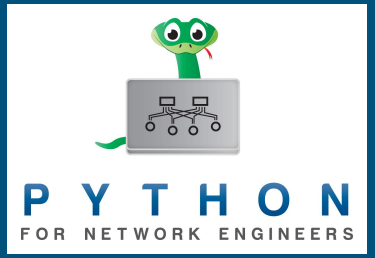
Example: The Normal Output (no filter)

This is from querying "object/int_vlan" with the shown CFG Path

```
CFG Path: ?config_path=/md/40Lab/VH/20:4c:03:39:5a:fc
{
    '_data': {
        'int_vlan': [
            {
                'id': 235,
                'int_vlan_ip': {'ipaddr': '10.5.235.16', 'ipparams': 'ipaddrmask', 'ipmask': '255.255.255.0'},
                'int_vlan_routing': {'_present': True, '_flags': {'default': True}},
                'int_vlan_ndra_hlimit': {'_flags': {'default': True}, 'value': 64},
                'int_vlan_ndra_interval': {'_flags': {'default': True}, 'value': 600},
                'int_vlan_ndra_ltime': {'_flags': {'default': True}, 'value': 1800},
                'int_vlan_ndra_mtu': {'_flags': {'default': True}, 'value': 1500},
                'int_vlan_nd_reachtime': {'_flags': {'default': True}, 'value': 0},
                'int_vlan_nd_rtrans_time': {'_flags': {'default': True}, 'value': 0},
                'int_vlan_mtu': {'_flags': {'default': True}, 'value': 1500},
                'int_vlan_suppress_arp': {'_present': True, '_flags': {'default': True}},
                'int_vlan_ip_ospf_cost': {'_flags': {'default': True}, 'value': 1},
                'int_vlan_ip_ospf_dead_interval': {'_flags': {'default': True}, 'value': 40},
                'int_vlan_ip_ospf_hello_interval': {'_flags': {'default': True}, 'value': 10},
                'int_vlan_ip_ospf_prior': {'_flags': {'default': True}, 'value': 1},
                'int_vlan_ip_ospf_retransmit_int': {'_flags': {'default': True}, 'value': 5},
                'int_vlan_ip_ospf_transmit_delay': {'_flags': {'default': True}, 'value': 1}
            }
        ]
    }
}
```



Now let's apply our filter:

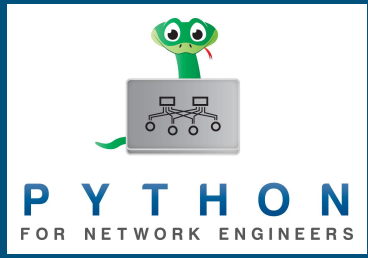



```
# Object filter
filter_object = [{"OBJECT": {"$eq": ["int_vlan.int_vlan_ip", "int_vlan.int_vlan_mtu"]}}]
filter_qs = f"filter={json.dumps(filter_object)}"
```

Note, I construct it as a Python data structure, but then use json.dumps to convert it to a JSON string.

```
config_path = "?config_path=/md/40Lab/VH/20:4c:03:39:5a:fc"
url_and_qs = f"{relative_url}{config_path}&{uid_aruba_qs}&{filter_qs}"
full_url = f"{base_url}{url_and_qs}"
response = session.get(full_url, verify=False)
```

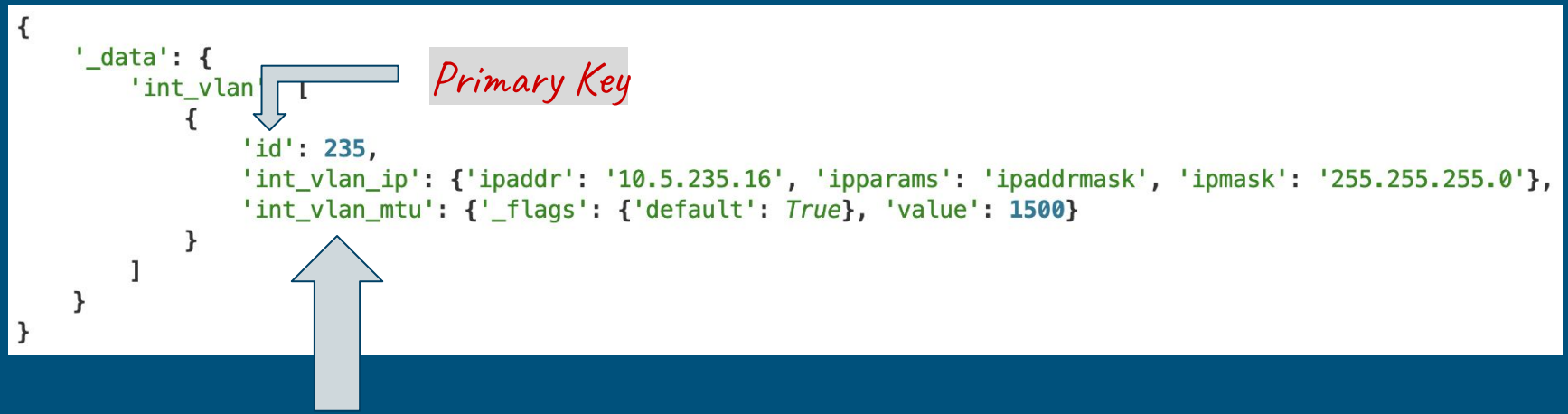
Applying the filter




```
config_path = "?config_path=/md/40Lab/VH/20:4c:03:39:5a:fc"
url_and_qs = f"{relative_url}{config_path}&{uid_aruba_qs}&{filter_qs}"
full_url = f"{base_url}{url_and_qs}"
response = session.get(full_url, verify=False)
```

Filter Applied Here

Updated Results



Reminder "filter" is

&filter=[{"OBJECT": {"$eq": ["int_vlan.int_vlan_ip", "int_vlan.int_vlan_mtu"]}}]

Exercise

Using the standard Aruba "auth" function located in "src", connect to the Aruba Controller.

Perform an HTTP Get using:

```
base_url = f"https://{host}:{api_port}/v1/configuration/"
relative_url = "object/int_gig"
```

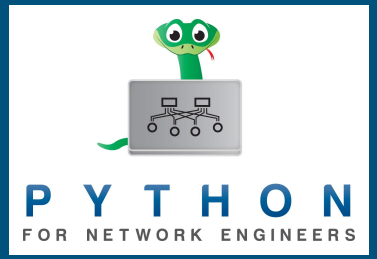
With a config_path of:

```
config_path = "?config_path=/md/40Lab/VH/20:4c:03:39:5a:fc"
```

rich.print the resulting data structure to the screen.

GitHub: {{ repo }}/day5/filtering/exercise1.txt

Exercise



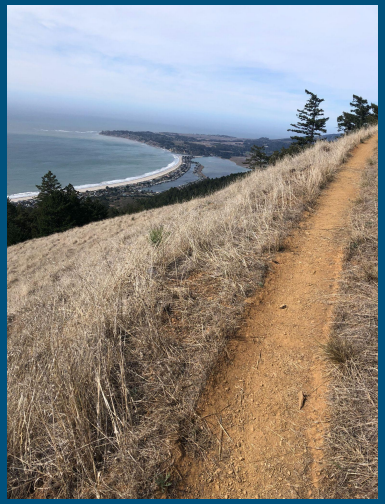
Now that you have viewed the full data-structure, create a filter that delimits the output to only the "slot/module/port" field.

Your filter should look as follows:

```
filter=[{"OBJECT": {"$eq": ["int_gig.slot/module/port"]}}]
```

And this gets applied to the end of the query string.

Filtering "Data" in the Aruba API



A second type of basic filter in the Aruba API is a "data" filter. With a data filter, you don't filter on field names in the API instead you filter on actual values.

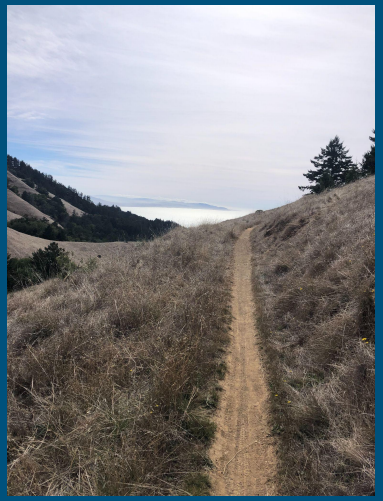
We accomplish this by adding a query string to the URL with the following the form:

filter=[{"vlan_name_id.name": {"$eq": ["user_3rd_floor"]}}]

Note, with a "Data" filter we do not specify "OBJECT" in the filter

Here when we query "object/vlan_name_id" we can limit what is returned to only those with the name="user_3rd_floor"

Filtering "Data" in the Aruba API



With Data filters we have additional comparison operators available to us:

$eq	$lt
$neq	$lte
$gt	$in
$gte	$nin

Note, the $in allows us to do string comparison i.e. is substring in larger_string (and similarly $nin allows us to do a not-in string comparison).

Filtering "Data" in the Aruba API

```
# Object filter
filter_data = [{"vlan_name_id.name" : { "$in" : ["user"]}}]
filter_qs = f"filter={json.dumps(filter_data)}"
```

Is "user" in the "vlan_name_id.name" field

Using $in comparison operator

```
{
    '_data': {
        'vlan_name_id': [
            {'name': 'user_5th_floor', 'vlan-ids': '905', '_flags': {'inherited': True}},
            {'name': 'user_4th_floor', 'vlan-ids': '904', '_flags': {'inherited': True}},
            {'name': 'user_7th_floor', 'vlan-ids': '907', '_flags': {'inherited': True}},
            {'name': 'user_6th_floor', 'vlan-ids': '906', '_flags': {'inherited': True}},
            {'name': 'user_3rd_floor', 'vlan-ids': '903', '_flags': {'inherited': True}},
            {'name': 'user_9th_floor', 'vlan-ids': '909', '_flags': {'inherited': True}},
            {'name': 'user_8th_floor', 'vlan-ids': '908', '_flags': {'inherited': True}}
        ]
    }
}
```

Exercise - Data Filtering

Using the standard Aruba "auth" function located in "src", connect to the Aruba Controller.

Perform an HTTP Get using:

```
base_url = f"https://{host}:{api_port}/v1/configuration/"
relative_url = "object/vlan_name_id"
```

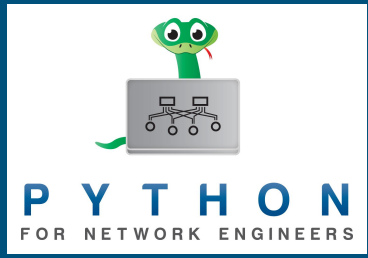
With a config_path of:

```
config_path = "?config_path=/md/40Lab/VH/"
```

rich.print the resulting data structure to the screen.

GitHub: {{ repo }}/day5/filtering/exercise2.txt

Exercise - Data Filtering

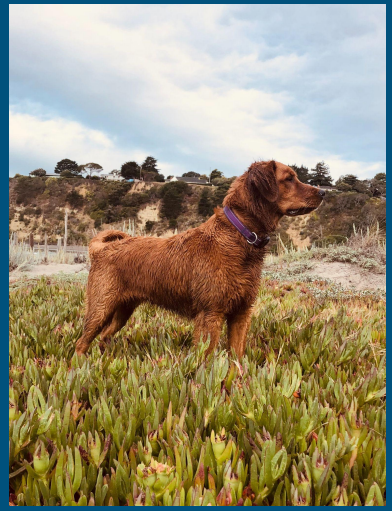


Now that you have viewed the full data-structure, create a filter that delimits the output to only VLANs with "guest" in their name

rich.print the data structure returned using this filter to standard output.

Creating a "get" function that supports filtering

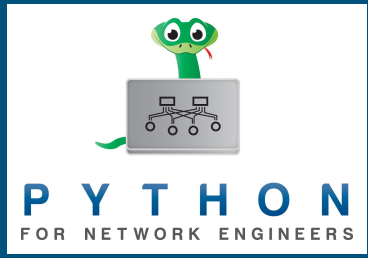
```
def get_request_filter(
    session,
    host,
    api_port=4343,
    relative_url="",
    config_path="",
    filter_str="",
    uid_aruba="",
):
```



```
filter_qs = ""
if filter_str:
    filter_qs = f"filter={json.dumps(filter_str)}"
```

```
if filter_qs:
    query_string += f"&{filter_qs}"
```

Ex3 and Ex4 - Using the new "get_request_filter" function



Create a new version of both filtering exercise1 and filtering exercise2 such that your new code uses the "get_request_filter" function.

GitHub: {{ repo }}/day5/filtering/exercise3.txt
GitHub: {{ repo }}/day5/filtering/exercise4.txt

Creating an Aruba Class

```
class ArubaAPI:
    def __init__(self, host, username, password, api_port="4343"):
        self.host = host
        self.api_port = api_port
        self.username = username
        self.password = password
```

```
def auth(self):
```

```
def show_command(self, command):
```

```
def get_request(self, relative_url, config_path="/mm/mynode"):
```

```
def logout(self):
```

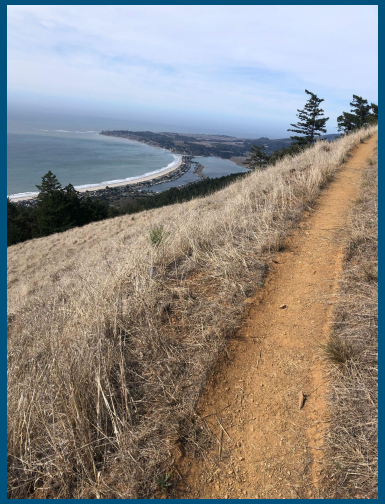
Exercise

Using the class located in src/aruba_class.py connect to the Aruba Controller.

Construct a query to retrieve "object/hostname" using the get_request() method.

Use rich.print to print the result to the screen.

Exercise1



Using the json library, read in "int_gig.json" and load it into your Python program.

```
import json
with open("int_gig.json") as f:
    data = json.load(f)
```

What type of data structure is returned?

From this data-structure, extract all of interface names ("slot/module/port" field) and the "int_gig_speed" and "int_gig_duplex" fields.

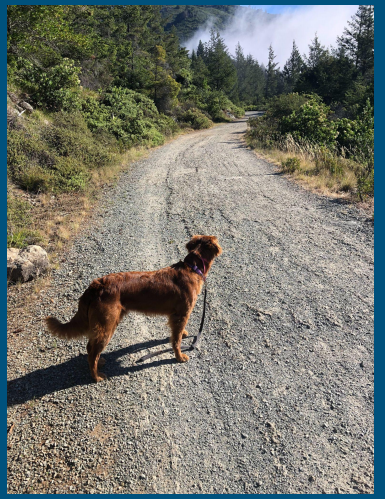
Print out the four interface names (slot/module/port) and their corresponding speed and duplex setting.

Exercise2

Using the ArubaAPI class located in aruba_class.py, connect to the API.

Use the show_command method to execute command="show+interface+counters".

For interface GE0/0/0, extract the InOctets and OutOctets and print those values to standard output.



Config using the API

```
base_url = f"https://{host}:{api_port}/v1/configuration/"
relative_url = "object/vlan_id"

# config_path = "?config_path=/md/40Lab/VH"
config_path = "?config_path=/md"
url_and_qs = f"{relative_url}{config_path}&{uid_aruba_qs}"
full_url = f"{base_url}{url_and_qs}"

# Create VLAN using an HTTP POST
cfg_payload = {"id": 910}
response = session.post(full_url, data=json.dumps(cfg_payload), verify=False)
print(f"\n{full_url}")
rich.print(response.json())
print()
```

Still point to a URL

Also there is still a config_path

The JSON payload for what we are configuring (from API docs)

HTTP Post

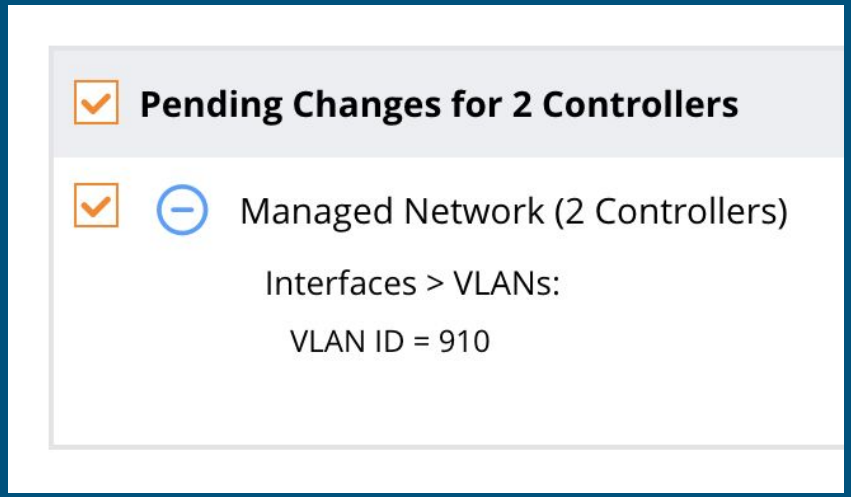
Pass in the payload.

Config: Successful Response

```
{
    'vlan_id': {'id': 910, '_result': {'status': 0, 'status_str': ''}},
    '_global_result': {'status': 0, 'status_str': 'Success', '_pending': 1}
}
```

Config: This will put the config into the pending state on the controller

We need to execute the API: object/write_memory

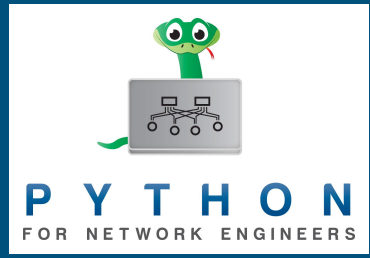


Config: Performing the "object/write_memory"

```python
# Performaing a write memory via the API
relative_url = "object/write_memory"
url_and_qs = f"{relative_url}{config_path}&{uid_aruba_qs}"
full_url = f"{base_url}{url_and_qs}"
response = session.post(full_url, verify=False)
print(f"\n{full_url}\n")
rich.print(response.json())
print()
```

```
{
    'write_memory': {'_result': {'status': 0, 'status_str': 'Success'}},
    '_global_result': {'status': 0, 'status_str': 'Success', '_pending': False}
}
```

Exercise - Configure SysContact



Use your standard auth function to authenticate to the Aruba Controller.

After authenticated, perform a get_request to retrieve "object/syscontact". Use a config_path of either "/md/40Lab/VH/20:4c:03:39:5a:fc" or "/md/40Lab/VH/20:4c:03:58:70:72".

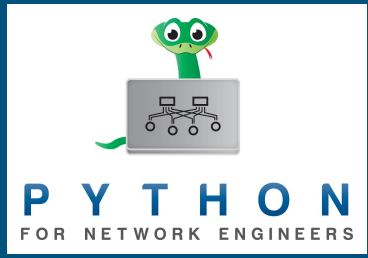
Print out the current "syscontact" to standard output.

Now use an HTTP Post and "object/syscontact" to configure the syscontact. Once again, use a config_path of either "/md/40Lab/VH/20:4c:03:39:5a:fc" or "/md/40Lab/VH/20:4c:03:58:70:72".

The syscontact HTTP Post payload needs to look as follows:

```
{"syscontact": "Some Contact Info"}
```

Exercise - Configure SysContact



Complete the configuration by performing an "object/write_memory".

Once again, this will require that you perform an HTTP Post and use a config_path of either "/md/40Lab/VH/20:4c:03:39:5a:fc" or "/md/40Lab/VH/20:4c:03:58:70:72".

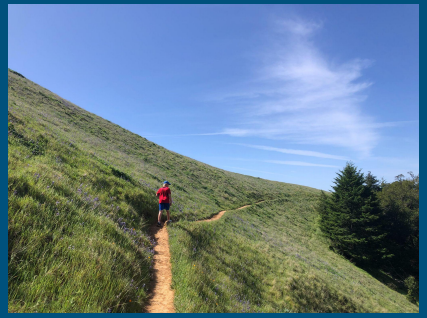
The config_path of the write_memory should match the config_path of your earlier configuration operation.

Deleting a Config object

```
# Delete SysContact using an HTTP POST
cfg_payload = {"syscontact": "Fake Contact", "_action": "delete"}
response = session.post(full_url, data=json.dumps(cfg_payload), verify=False)
```

Add "_action": "delete" to the payload

Constructing a Reusable Config Function



```python
def config_change(
    session,
    host,
    api_port=4343,
    relative_url="",
    config_path="",
    config_payload="",
    uid_aruba="",
):
```

```python
full_url = f"{base_url}{relative_url}{query_string}"
if config_payload:
    return session.post(full_url, data=json.dumps(config_payload), verify=False)
else:
    # Allow POSTs like "write memory" that have no payload
    return session.post(full_url, verify=False)
```

Exercise: Configure an IP Domain Name on the Controllers using the Newly Created Config Function.

Perform your standard authentication to the Mobility Master using your "auth" function.

Configure an IP-domain name of "lasthop.io" using the newly created config function and a config_path="/md/40Lab". The relevant API object is:

`/object/ip_domain_name`

The configuration payload is:

```
{"name": "lasthop.io"}
```

Complete the process by using the API to execute "write memory"

Let's Step Back and Review What We Have Covered.

Authentication

Key Points on Authentication:

- Authentication format i.e. structure of how to do the authentication.
- What you have to retain from the authentication (in your 8.6.X.X and earlier case), you need to retain the UIDARUBA.

Remember you are going to need to add UIDARUBA=<session-id> to the query string on your GET requests and on your POSTS

Let's Step Back and Review What We Have Covered.

More Authentication

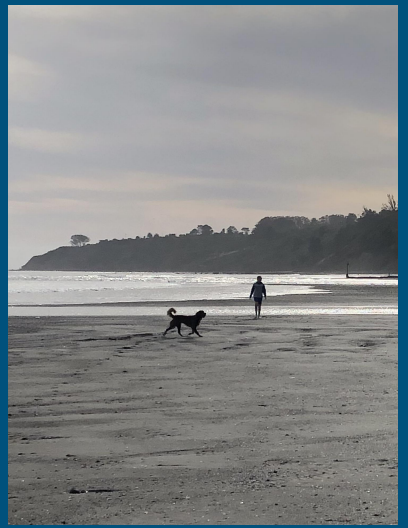
Key Points on Authentication:

- Look at the response code (you are looking for a 200 response).
- You will want to make a reusable function for authentication.
- Thinks about and handle the standard failure cases.
- Newer AOS is going to require that you start using X-CSRF-Token

Let's Step Back and Review What We Have Covered.

Data Retrieval (HTTP Get)

- The URL will keep changing to various "object/some_thing"; the HTTP action will continue to be a GET.
- Successful response will once again be a 200 OK.
- The returned payload will be in response.json()
- This potentially will be a large and complex data structure. Remember your process for handling large data structures.
- You will have to use the API documentation to figure out what to query.
- Once again you will want to build functions or other reusable building blocks around this.
- You can use "?config_path=" to qualify where in the Aruba hierarchy you are querying.



Let's Step Back and Review What We Have Covered.

You can also directly execute show commands and get JSON responses back with "showcommand?command="

Let's Step Back and Review What We Have Covered.

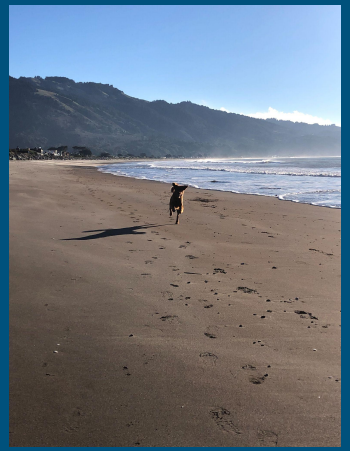
Filtering

We have two main ways of filtering: we can either filter on the fields that we want present in the output ("OBJECT" filtering) or we can filter on values or patterns we want in the output (data filtering).

Object filters let us do $eq or $neq

Data filters have more options including $in, and $nin providing us with simple pattern matching.

Let's Step Back and Review What We Have Covered.



Config

When we are making config changes, we will be using an HTTP Post. For the Aruba API, an HTTP Post is used for create, modify, and delete operations.

With config, you once again have to look at the API documentation to figure out which "object" you will be accessing. The API documentation will also specify the format of the payload.

In your HTTP Post request, you will pass in the URL (what you are modifying) and also a "data=json.dumps(data_struct)" where "data_struct" is the payload you created.

You can once again qualify where in the Aruba structure you are modifying by using config_path= as query string arguments in the URL.

Let's Step Back and Review What We Have Covered.

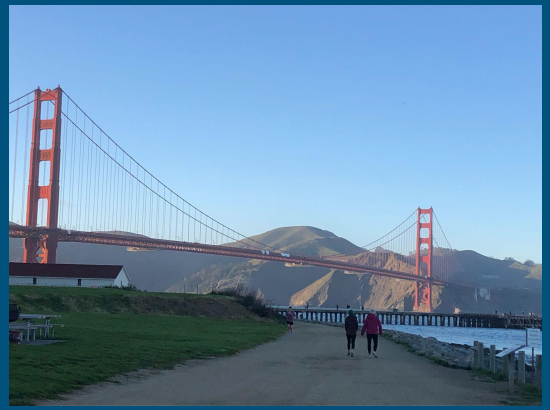
Config - Delete

You can add the following key-value pair

`"_action": "delete"`

to a configuration payload to perform a config delete operation.

Let's Step Back and Review What We Have Covered.



Write Memory

You will generally have to execute a "write memory" HTTP Post following a configuration change to deploy the configuration.

Update Our Class to Add a Config Method

```
def config_change(
    self,
    relative_url,
    config_payload="",
    config_path="",
):

    self.session.headers["Content-Type"] = "application/json"
    base_url = f"https://{self.host}:{self.api_port}/v1/configuration/"
```

Review Exercise1

Using entirely the functions located in the aruba_auth.py module, connect to the Aruba controllers and retrieve "object/ssid_prof" (use a config_path="/md/40Lab")

Use rich.print() to print the returned data structure to standard output.

Repeat this process except apply an "object" filter such that only the "profile-name" field and the "ssid_enable" fields are returned for each SSID Profile.

Make sure you call the "logout" function at the end of your program.

GitHub: {{ repo }}/day5/review_exercises/exercise1.txt

Review Exercise2

Using the functions located in the aruba_auth.py module, connect to the Aruba controllers.

Use the "show_command" to execute "show ap radio-database". Retrieve the JSON response from this and use rich.print() to print this to standard output.

Further process the response data, such that you create a new list. Each entry in the list should be a tuple/list consisting of (AP-Name, AP-IP_Address). Use rich.print() to print out this new data structure.

Your output should look similar to the following:

```
[('AP-01', '10.5.235.26'), ('AP-02', '10.5.235.30'),
 ('AP-03', '10.5.235.32'), ('AP-04', '10.5.235.31')]
```

Use the "logout" function and gracefully logout.

GitHub: {{ repo }}/day5/review_exercises/exercise2.txt

Review Exercise3

Using the config_change method in aruba_class.py, configure "object/ip_name_server" using a config_path="/md/40Lab" and a payload of:

```
payload = {"address": "8.8.8.8"}
```

Complete this change by executing "object/write_memory" via the API. Once again, use the config_change method in aruba_class.py to accomplish this. Your payload will be empty.

GitHub: {{ repo }}/day5/review_exercises/exercise3.txt