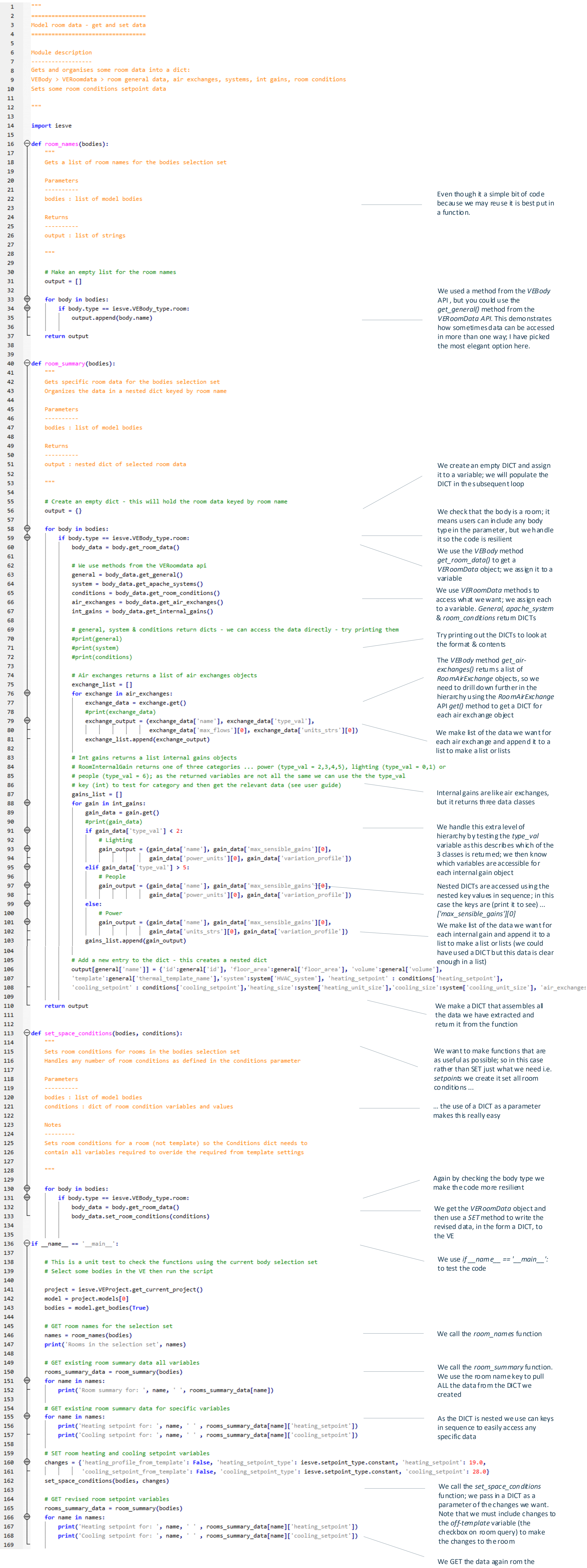
Python in the VE

VE Scripts – working with room data

What? We use GET and SET methods to access VE data. To use them you need to navigate the API hierarchy. The GET methods typically return data in DICT form, often nested, which we need to be able to access. Using DICTs is also a good way of organising data that we GET from the VE.

Why? In this example we navigate VEBody to VERoomdata and access room data via methods for room general data, air exchanges, systems, internal gains and room conditions. We use a DICT to organise the data we extract because DICT keys make sure we store what the values mean and give us an elegant way to access specific data. We also utilise a DICT to specify changes we make using a SET method.



```
"""
==================================
Model room data - get and set data
==================================

Module description
------------------
Gets and organises some room data into a dict:
VEBody > VERoomdata > room general data, air exchanges, systems, int gains, room conditions
Sets some room conditions setpoint data

"""

import iesve

def room_names(bodies):
    """
    Gets a list of room names for the bodies selection set

    Parameters
    ----------
    bodies : list of model bodies

    Returns
    ----------
    output : list of strings

    """

    # Make an empty list for the room names
    output = []

    for body in bodies:
        if body.type == iesve.VEBody_type.room:
            output.append(body.name)

    return output


def room_summary(bodies):
    """
    Gets specific room data for the bodies selection set
    Organizes the data in a nested dict keyed by room name

    Parameters
    ----------
    bodies : list of model bodies

    Returns
    ----------
    output : nested dict of selected room data

    """

    # Create an empty dict - this will hold the room data keyed by room name
    output = {}

    for body in bodies:
        if body.type == iesve.VEBody_type.room:
            body_data = body.get_room_data()

            # We use methods from the VERoomdata api
            general = body_data.get_general()
            system = body_data.get_apache_systems()
            conditions = body_data.get_room_conditions()
            air_exchanges = body_data.get_air_exchanges()
            int_gains = body_data.get_internal_gains()

            # general, system & conditions return dicts - we can access the data directly - try printing them
            #print(general)
            #print(system)
            #print(conditions)

            # Air exchanges returns a list of air exchanges objects
            exchange_list = []
            for exchange in air_exchanges:
                exchange_data = exchange.get()
                #print(exchange_data)
                exchange_output = (exchange_data['name'], exchange_data['type_val'],
                                   exchange_data['max_flows'][0], exchange_data['units_strs'][0])
                exchange_list.append(exchange_output)

            # Int gains returns a list internal gains objects
            # RoomInternalGain returns one of three categories ... power (type_val = 2,3,4,5), lighting (type_val = 0,1) or
            # people (type_val = 6); as the returned variables are not all the same we can use the the type_val
            # key (int) to test for category and then get the relevant data (see user guide)
            gains_list = []
            for gain in int_gains:
                gain_data = gain.get()
                #print(gain_data)
                if gain_data['type_val'] < 2:
                    # Lighting
                    gain_output = (gain_data['name'], gain_data['max_sensible_gains'][0],
                                   gain_data['power_units'][0], gain_data['variation_profile'])
                elif gain_data['type_val'] > 5:
                    # People
                    gain_output = (gain_data['name'], gain_data['max_sensible_gains'][0],
                                   gain_data['power_units'][0], gain_data['variation_profile'])
                else:
                    # Power
                    gain_output = (gain_data['name'], gain_data['max_sensible_gains'][0],
                                   gain_data['units_strs'][0], gain_data['variation_profile'])
                gains_list.append(gain_output)

            # Add a new entry to the dict - this creates a nested dict
            output[general['name']] = {'id':general['id'], 'floor_area':general['floor_area'], 'volume':general['volume'],
            'template':general['thermal_template_name'],'system':system['HVAC_system'], 'heating_setpoint' : conditions['heating_setpoint'],
            'cooling_setpoint' : conditions['cooling_setpoint'],'heating_size':system['heating_unit_size'],'cooling_size':system['cooling_unit_size'], 'air_exchanges

    return output


def set_space_conditions(bodies, conditions):
    """
    Sets room conditions for rooms in the bodies selection set
    Handles any number of room conditions as defined in the conditions parameter

    Parameters
    ----------
    bodies : list of model bodies
    conditions : dict of room condition variables and values

    Notes
    ---------
    Sets room conditions for a room (not template) so the Conditions dict needs to
    contain all variables required to overide the required from template settings

    """

    for body in bodies:
        if body.type == iesve.VEBody_type.room:
            body_data = body.get_room_data()
            body_data.set_room_conditions(conditions)


if __name__ == '__main__':

    # This is a unit test to check the functions using the current body selection set
    # Select some bodies in the VE then run the script

    project = iesve.VEProject.get_current_project()
    model = project.models[0]
    bodies = model.get_bodies(True)

    # GET room names for the selection set
    names = room_names(bodies)
    print('Rooms in the selection set', names)

    # GET existing room summary data all variables
    rooms_summary_data = room_summary(bodies)
    for name in names:
        print('Room summary for: ', name, ' ', rooms_summary_data[name])

    # GET existing room summary data for specific variables
    for name in names:
        print('Heating setpoint for: ', name, ' ' , rooms_summary_data[name]['heating_setpoint'])
        print('Cooling setpoint for: ', name, ' ' , rooms_summary_data[name]['cooling_setpoint'])

    # SET room heating and cooling setpoint variables
    changes = {'heating_profile_from_template': False, 'heating_setpoint_type': iesve.setpoint_type.constant, 'heating_setpoint': 19.0,
               'cooling_setpoint_from_template': False, 'cooling_setpoint_type': iesve.setpoint_type.constant, 'cooling_setpoint': 28.0}
    set_space_conditions(bodies, changes)

    # GET revised room setpoint variables
    rooms_summary_data = room_summary(bodies)
    for name in names:
        print('Heating setpoint for: ', name, ' ' , rooms_summary_data[name]['heating_setpoint'])
        print('Cooling setpoint for: ', name, ' ' , rooms_summary_data[name]['cooling_setpoint'])

```

Even though it a simple bit of code because we may reuse it is best put in a function.

We used a method from the *VEBody* API, but you could use the *get_general()* method from the *VERoomData API*. This demonstrates how sometimes data can be accessed in more than one way; I have picked the most elegant option here.

We create an empty DICT and assign it to a variable; we will populate the DICT in the subsequent loop

We check that the body is a room; it means users can include any body type in the parameter, but we handle it so the code is resilient

We use the *VEBody* method *get_room_data()* to get a *VERoomData* object; we assign it to a variable

We use *VERoomData* methods to access what we want; we assign each to a variable. *General, apache_system* & *room_conditions* return DICTs

Try printing out the DICTs to look at the format & contents

The *VEBody* method *get_air-exchanges()* returns a list of *RoomAirExchange* objects, so we need to drill down further in the hierarchy using the *RoomAirExchange* API *get()* method to get a DICT for each air exchange object

We make list of the data we want for each air exchange and append it to a list to make a list or lists

Internal gains are like air exchanges, but it returns three data classes

We handle this extra level of hierarchy by testing the *type_val* variable as this describes which of the 3 classes is returned; we then know which variables are accessible for each internal gain object

Nested DICTs are accessed using the nested key values in sequence; in this case the keys are (print it to see) … *['max_sensible_gains'][0]*

We make list of the data we want for each internal gain and append it to a list to make a list or lists (we could have used a DICT but this data is clear enough in a list)

We make a DICT that assembles all the data we have extracted and return it from the function

We want to make functions that are as useful as possible; so in this case rather than SET just what we need i.e. *setpoints* we create it set all room conditions …

… the use of a DICT as a parameter makes this really easy

Again by checking the body type we make the code more resilient

We get the *VERoomData* object and then use a *SET* method to write the revised data, in the form a DICT, to the VE

We use *if __name__ == '__main__':* to test the code

We call the *room_names* function

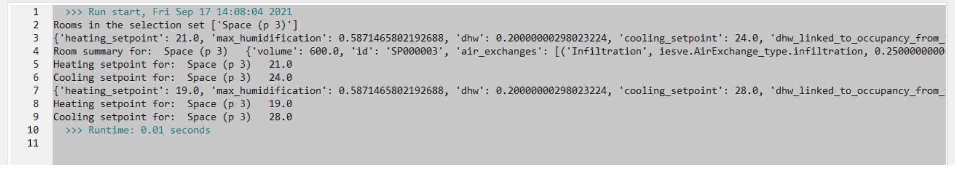
We call the *room_summary* function. We use the room name key to pull ALL the data from the DICT we created

As the DICT is nested we use can keys in sequence to easily access any specific data

We call the *set_space_conditions* function; we pass in a DICT as a parameter of the changes we want. Note that we must include changes to the *off-template* variable (the checkbox on room query) to make the changes to the room

We GET the data again rom the model to check the changes

Sample output:



```
  >>> Run start, Fri Sep 17 14:08:04 2021
Rooms in the selection set ['Space (p 3)']
{'heating_setpoint': 21.0, 'max_humidification': 0.5871465802192688, 'dhw': 0.20000000298023224, 'cooling_setpoint': 24.0, 'dhw_linked_to_occupancy_from_
Room summary for:  Space (p 3)   {'volume': 600.0, 'id': 'SP000003', 'air_exchanges': [('Infiltration', iesve.AirExchange_type.infiltration, 0.2500000000
Heating setpoint for:  Space (p 3)   21.0
Cooling setpoint for:  Space (p 3)   24.0
{'heating_setpoint': 19.0, 'max_humidification': 0.5871465802192688, 'dhw': 0.20000000298023224, 'cooling_setpoint': 28.0, 'dhw_linked_to_occupancy_from_
Heating setpoint for:  Space (p 3)   19.0
Cooling setpoint for:  Space (p 3)   28.0
  >>> Runtime: 0.01 seconds

```