Making Flexible Python Protocols

Introduction

On a surface level, Protocol Designer and Python Protocols have very similar capabilities. However, we can use some of the features in Python to make our protocols more flexible and easier to edit. If you check out the protocols in Opentrons' Protocol Library, the website allows you to select a few options before generating a protocol file, which are then passed into the script. In this tutorial, we will learn how to implement a similar feature in our protocols, as well as a few other measures to increase protocol flexibility. We will start with a very simple serial dilution protocol:

```
from opentrons import protocol_api

metadata = {
    'protocolName': 'Serial Dilution',
    'author': 'Felipe Xavier Buson',
    'apiLevel': '2.10'
    }

def run(protocol: protocol_api.ProtocolContext):
    source_plate = protocol.load_labware('opentrons_24_aluminumblock_nest_1.5ml_snapcap',
'1')
    destination_plate = protocol.load_labware('biorad_96_wellplate_200ul_pcr', '2')
    tips_p300 = [protocol.load_labware('opentrons_96_tiprack_300ul', '3')]
    tips_p20 = [protocol.load_labware('opentrons_96_tiprack_20ul', '6')]

    p300 = protocol.load_instrument('p300_single_gen2', 'right', tip_racks = tips_p300)
    p20 = protocol.load_instrument('p20_single_gen2', 'left', tip_racks = tips_p20)

    # Transfer water to prepare for dilution
    p300.distribute(90, source_plate['A1'],
                    [destination_plate['A1'],
                     destination_plate['B1'],
                     destination_plate['C1'],
                     destination_plate['D1'],
                     destination_plate['E1']])

    # Perform each transfer for the dilution, always mixing 5x with 20uL after dispensing
    p20.transfer(10, source_plate['B1'], destination_plate['A1'], mix_after = (5, 20))
    p20.transfer(10, destination_plate['A1'], destination_plate['B1'], mix_after = (5, 20))
    p20.transfer(10, destination_plate['B1'], destination_plate['C1'], mix_after = (5, 20))
    p20.transfer(10, destination_plate['C1'], destination_plate['D1'], mix_after = (5, 20))
    p20.transfer(10, destination_plate['D1'], destination_plate['E1'], mix_after = (5, 20))
```

Here, we first distribute some water into wells of a 96-well plate (from "source_plate" A1 to wells in "destination_plate"). Then, we shift 10µL of our sample (on "source_plate" B1) from well to well, making sure to mix our dilutions after each step.

Notice how for every command to the robot, we have hard-coded the source and destination wells. If we want to change any of these, or the volumes we use, we would then need to change this information on every line of code. This is not the worst for this protocol, since it's only a handful of

lines long, but for longer protocols changing every line can be tedious and error-prone. We will go through some ways with which we can avoid this.

Flexible commands and variables

We will work backwards through the protocol, and the first thing to address is the number of times we write our transfer function. Considering we're doing the same operation every line, we can easily trim this down by using a loop, or a many-to-many transfer function, using a list to hold our source and destination wells:

```
# define wells to get liquid form and wells to send to
source_wells = [source_plate['B1'],
                destination_plate['A1'],
                destination_plate['B1'],
                destination_plate['C1'],
                destination_plate['D1']]

destination_wells = [destination_plate['A1'],
                     destination_plate['B1'],
                     destination_plate['C1'],
                     destination_plate['D1'],
                     destination_plate['E1']]

# Transfer water to prepare for dilution
p300.distribute(90, source_plate['A1'], destination_wells)

# perform transfer
for idx in range(len(source_wells)):
    p20.transfer(10, source_wells[idx], destination_wells[idx], mix_after = (5, 20))
```

Or, without looping through the lists, the script for performing the transfers can be substituted for:

```
# perform transfer
p20.transfer(10, source_wells, destination_wells, mix_after = (5, 20), new_tip = 'always')
```

The next aspect that we can do better is defining our source and destination wells. Instead of defining each well one by one, we can instead just define the row we want to use and the number of dilutions, then use the slicing functionality in Python to build our lists:

```
# define wells to get liquid form and wells to send to
col_to_use = 0
no_of_dilutions = 5
destination_wells = destination_plate.columns()[col_to_use][:no_of_dilutions]
source_wells = [source_plate['B1']] + destination_wells[:-1]
```

Now, we can solely use the "col_to_use" and "no_of_dilutions" variables to change which wells will be used.

Getting data from CSV files

Making interactive and editable Python protocols can make it easier for users with no programming training to use the OT2 robot, by just change a few variables in the code. However, an even more flexible and beginner-friendly way of passing information into the protocols is through comma-separated value (CSV) tables or other kinds of files that can be read in Python. Opentrons already

has some guidance on this, where they recommend inserting the csv text directly into your code or using your computer's Command Line and scp to upload the files into the robot.

Here, we will cover two other ways of using CSV files with your protocols, that won't require editing our script after we're set up, and won't require the use of the Command Line. The first option uses a separate python program to build a new protocol for every new run, and is very similar to just writing your csv text into your protocol. The second option makes use of the Jupyter Notebook environment in the OT2, and is by far the most flexible option, although coming with the drawbacks of working in that environment.

We will still use a dilution protocol for this, but a bit different. This time, we're tackling the situation where we want many different samples to be at the same molar concentration. Our example has three different samples, and they're defined right after the labware declaration block:

```
from opentrons import protocol_api

metadata = {
    'protocolName': 'Serial Dilution',
    'author': 'Felipe Xavier Buson',
    'apiLevel': '2.10'
    }

def run(protocol: protocol_api.ProtocolContext):
    source_plate = protocol.load_labware('opentrons_24_aluminumblock_nest_1.5ml_snapcap',
'1')
    destination_plate = protocol.load_labware('biorad_96_wellplate_200ul_pcr', '2')
    tips_p300 = [protocol.load_labware('opentrons_96_tiprack_300ul', '3')]
    tips_p20 = [protocol.load_labware('opentrons_96_tiprack_20ul', '6')]

    p300 = protocol.load_instrument('p300_single_gen2', 'right', tip_racks = tips_p300)
    p20 = protocol.load_instrument('p20_single_gen2', 'left', tip_racks = tips_p20)


    # define wells to get liquid form and wells to send to
    samples = [
        {'location':source_plate['A1'], 'concentration_ng/ul':100, 'size':4000},
        {'location':source_plate['A2'], 'concentration_ng/ul':140, 'size':2000},
        {'location':source_plate['A3'], 'concentration_ng/ul':20, 'size':500}
    ]

    water = source_plate.wells()[-1] # Last well on the plate, in this case D6
    row_to_use = 'A'
    destination_wells = destination_plate.rows_by_name()[row_to_use][:len(samples)] # uses wells
on a defined row. Number of wells defined by number of samples

    # Other variables
    final_concentration_nM = 15
    final_volume_uL = 100
    DNA_gpermol = 650

    protocol.comment('\n--- DILUTING SAMPLES TO ' + str(final_concentration_nM) + 'nM WITH
FINAL VOLUME ' + str(final_volume_uL) + 'uL ---')
```

```
    protocol.comment('\n Samples will be diluted in row ' + row_to_use + ' of your destination
plate')

    vols_to_dilute = []
    for sample in samples:
        current_concentration_nM = (sample['concentration_ng/ul'] * pow(10, 6)) / (DNA_gpermol *
sample['size'])
        vol = final_volume_uL * final_concentration_nM/current_concentration_nM
        vols_to_dilute.append(round(vol, 1))

    # check if concentrations are too high/low
    usable_samples = []
    for idx, sample in enumerate(samples):
        protocol.comment('')
        protocol.comment('Sample ' + str(idx+1))
        protocol.comment('Volume for dilution (uL): ' + str(vols_to_dilute[idx]))
        if vols_to_dilute[idx] > final_volume_uL:
            protocol.comment("!! WARNING: Concentration too low on sample " + str(idx+1))
        elif final_volume_uL - vols_to_dilute[idx] < 1:
            protocol.comment("!! WARNING: Too close to desired concentration on sample " +
str(idx+1))
        elif vols_to_dilute[idx] < 1:
            protocol.comment("!! WARNING: Concentration too high on sample " + str(idx+1))
        else:
            usable_samples.append(idx)

    # distribute water
    protocol.comment('\n--- DISTRIBUTING WATER ---')
    for idx, sample in enumerate(samples):
        water_to_use = final_volume_uL - vols_to_dilute[idx]
        if idx in usable_samples:
            protocol.comment('\nSample ' + str(idx+1) + ': Transferring ' + str(water_to_use) + 'uL of
water')
            if water_to_use > 20:
                p300.transfer(water_to_use, water, destination_wells[idx])
            else:
                p20.transfer(water_to_use, water, destination_wells[idx])

    # transfer samples
    protocol.comment('\n--- TRANSFERRING SAMPLES ---')
    for idx, sample in enumerate(samples):
        if idx in usable_samples:
            protocol.comment('\nSample ' + str(idx+1) + ': Transferring ' + str(vols_to_dilute[idx]) + 'uL
of sample')
            if vols_to_dilute[idx] > 20:
                p300.transfer(vols_to_dilute[idx], samples[idx]['location'], destination_wells[idx],
mix_after = (5, vols_to_dilute[idx] * 0.9))
            else:
                p20.transfer(vols_to_dilute[idx], samples[idx]['location'], destination_wells[idx],
mix_after = (5, vols_to_dilute[idx] * 0.9))
```

Here, we provide the concentration for our three samples located in tubes in our “source_plate”, a 24-well OT aluminium rack with 1.5mL tubes on it, which will be used to calculate how much sample and water to use in dilutions on a “destination plate”. Both of our approaches will only change how we get the “samples” variable, which defines our sample concentrations in a Python dictionary.

Using separate build program

This solution is as good as writing the csv directly into the python program, with the only difference being that you will only need to interact with CSV files instead of python ones. Our CSV file with concentrations (we’ll call it ”concentrations_data.csv") will look like this:

```
location,concentration_ng/ul,size
A1,100,4000
A2,140,2000
A3,20,500
```

We’ll have to make sure this file is in the same folder as our build file:

```
#build_protocol.py

import csv
import json

csv_file = open('concentrations_data.csv', 'r', encoding='utf-8')

with open('csv_dilutions_example.py','r', encoding='utf-8') as protocol_file:
    as_dict = [row for row in csv.DictReader(csv_file)]
    for sample in as_dict:
        sample['concentration_ng/ul'] = int(sample['concentration_ng/ul'])
        sample['size'] = int(sample['size'])
    new_protocol = protocol_file.read().replace('{{SAMPLES}}', json.dumps(as_dict))

with open('csv_dilutions_example.py', 'w', encoding='utf-8') as protocol_file:
    protocol_file.write(new_protocol)
```

As you may see, this build script simply checks our protocol Python script (here, calling it “csv_dilutions_example.py”) for the text “{{SAMPLES}}” and replaces it with a json version of whatever it finds in the CSV file. For that to work it’s important that the structure in the CSV file remains the same. In our protocol, we must define where this text will be placed, and pass it into our “samples” variable. Additionally, note that we only used “A1/A2/A3” as our locations for each sample. This means when we’re accessing that, we’ll have to specify which plate that location is from (the “source_plate”) In the end, the protocol should look like this:

```
def get_sample_data():
    import json
    return json.loads('{{SAMPLES}}', parse_int = int)

from opentrons import protocol_api

metadata = {
    'protocolName': 'Serial Dilution',
    'author': 'Felipe Xavier Buson',
    'apiLevel': '2.10'
    }
```

```
def run(protocol: protocol_api.ProtocolContext):
    source_plate = protocol.load_labware('opentrons_24_aluminumblock_nest_1.5ml_snapcap',
'1')
    destination_plate = protocol.load_labware('biorad_96_wellplate_200ul_pcr', '2')
    tips_p300 = [protocol.load_labware('opentrons_96_tiprack_300ul', '3')]
    tips_p20 = [protocol.load_labware('opentrons_96_tiprack_20ul', '6')]

    p300 = protocol.load_instrument('p300_single_gen2', 'right', tip_racks = tips_p300)
    p20 = protocol.load_instrument('p20_single_gen2', 'left', tip_racks = tips_p20)


    # define wells to get liquid form and wells to send to
    samples = get_sample_data()

    water = source_plate.wells()[-1] # Last well on the plate, in this case D6

    row_to_use = 'A'
    destination_wells = destination_plate.rows_by_name()[row_to_use][:len(samples)] # uses wells
on a defined row. Number of wells defined by number of samples

    # Other variables

    final_concentration_nM = 15
    final_volume_uL = 100
    DNA_gpermol = 650

    protocol.comment('\n--- DILUTING SAMPLES TO ' + str(final_concentration_nM) + 'nM WITH
FINAL VOLUME ' + str(final_volume_uL) + 'uL ---')
    protocol.comment('\n Samples will be diluted in row ' + row_to_use + ' of your destination
plate')

    vols_to_dilute = []
    for sample in samples:
        current_concentration_nM = (sample['concentration_ng/ul'] * pow(10, 6)) / (DNA_gpermol *
sample['size'])
        vol = final_volume_uL * final_concentration_nM/current_concentration_nM
        vols_to_dilute.append(round(vol, 1))

    # check if concentrations are too high/low
    usable_samples = []
    for idx, sample in enumerate(samples):
        protocol.comment('')
        protocol.comment('Sample ' + str(idx+1))
        protocol.comment('Volume for dilution (uL): ' + str(vols_to_dilute[idx]))
        if vols_to_dilute[idx] > final_volume_uL:
            protocol.comment("!! WARNING: Concentration too low on sample " + str(idx+1))
        elif final_volume_uL - vols_to_dilute[idx] < 1:
            protocol.comment("!! WARNING: Too close to desired concentration on sample " +
str(idx+1))
        elif vols_to_dilute[idx] < 1:
```

```
            protocol.comment("!! WARNING: Concentration too high on sample " + str(idx+1))
        else:
            usable_samples.append(idx)

    # distribute water
    protocol.comment('\n--- DISTRIBUTING WATER ---')
    for idx, sample in enumerate(samples):
        water_to_use = final_volume_uL - vols_to_dilute[idx]
        if idx in usable_samples:
            protocol.comment('\nSample ' + str(idx+1) + ': Transferring ' + str(water_to_use) + 'uL of
water')
            if water_to_use > 20:
                p300.transfer(water_to_use, water, destination_wells[idx])
            else:
                p20.transfer(water_to_use, water, destination_wells[idx])

    # transfer samples
    protocol.comment('\n--- TRANSFERRING SAMPLES ---')
    for idx, sample in enumerate(samples):
        if idx in usable_samples:
            protocol.comment('\nSample ' + str(idx+1) + ': Transferring ' + str(vols_to_dilute[idx]) + 'uL
of sample')
            if vols_to_dilute[idx] > 20:
                p300.transfer(vols_to_dilute[idx], source_plate[samples[idx]['location']],
destination_wells[idx], mix_after = (5, vols_to_dilute[idx] * 0.9))
            else:
                p20.transfer(vols_to_dilute[idx], source_plate[samples[idx]['location']],
destination_wells[idx], mix_after = (5, vols_to_dilute[idx] * 0.9))
```

To generate a usable script, simply place the CSV file, the build script and the protocol script on the same folder and run the build script. You should be able to see our protocol has changed and now includes our data.

Using Jupyter Notebooks

Our second approach uses the csv files more directly (without the use of a build script), but needs to use the OT2's Jupyter Notebook environment. Check out our Jupyter Notebooks on the OT2 tutorial for a guide on how to use that environment. After opening the environment, we recommend making a new folder called "data", which will hold our csv_file.

Inside our Jupyter Notebook protocol, to get the "samples" variable we will have to run a cell that is very similar to our build script above:

```
import csv

csv_file = open('data/concentrations_data.csv', 'r', encoding='utf-8')

with open('csv_dilutions_example.py','r', encoding='utf-8') as protocol_file:
    samples = [row for row in csv.DictReader(csv_file)]
    for sample in samples:
        sample['concentration_ng/ul'] = int(sample['concentration_ng/ul'])
        sample['size'] = int(sample['size'])
```

Then, we can run our protocol in the Jupyter Notebook as usual. We won't provide a Jupyter Notebook version of this protocol, but we hope our other tutorial in using the environment is enough for you to convert the protocol by yourself.

Tips and best practices

Communicating what is happening in the code

As you change your script to be adaptable to different inputs, it's easy to lose track of what the protocol is doing just from looking at the code, since there will be less indications of what volumes, wells, and instruments are being used. Therefore, it's especially important to comment your code and explain every step if you want other people to amend it. Using comments that will also go into runs and simulations with the ".comment()" command can also be extremely useful.

Further flexibility on code

Some protocols in the Protocol Library have the option of using single or multi-channel pipettes, or numbers for volumes and wells that might also change qualitatively how the protocol should behave. This is generally handled through "if" statements, checking for key values on the initial values and changing the script accordingly. It's worth it to inspect some protocols from the library and check how they handle different situations.