

Prep Sheet

Learn to Write a Python Protocol

OT-2 Edition

Authored by Emily Burghardt, Ph.D.

Opentrons

July 2026

Getting Started

In this lesson plan, students use the PCR protocol created with Protocol Designer in the first lab module to write their first Python protocol.

Additional Support and Resources

[Opentrons OT-2 Instruction Manual](#)

[Opentrons API Documentation](#)

If you need further support, please contact support@opentrons.com. Inform them that you are a part of the Opentrons for Education program and provide the date of your next laboratory class.

If you have questions related to the lesson plan, please reach out to the author, Emily Burghardt, at emily.burghardt@opentrons.com.

Educator Guide

Learn to Write a Python Protocol

OT-2 Edition

Authored by Emily Burghardt, Ph.D.

Opentrons

Contents

This educator guide includes the following sections:

- Purpose
- Background Knowledge
- Supplies
- Experimental Duration
- Basic Troubleshooting and Tips
- Procedure Guide
- Discussion Questions

Purpose

In this lab, students will use their PCR protocols (created using Protocol Designer in the previous lab module) to write a first Python protocol.

Protocol Designer exports protocols as Python files. However, there are key differences between the way Protocol Designer's software writes a protocol *for you* and how Opentrons' [Python Protocol API](#) allows you to write your own. This gives your students a solid foundation to start from, but still requires that they do their own work to write a new protocol.

We recommend that students use this lesson plan, their Protocol Designer protocol (open in a [Protocol Designer](#) window), and the Opentrons [API documentation](#) to create their Python protocol. They can use an [example protocol](#) written for this lesson plan for extra help.

*This educator guide lists other resources to help students as they write. To encourage students to complete their own work, the **Student Guide** doesn't mention OpentronsAI or the [example protocol](#).*



Students will gain experience with:

- Using API documentation
- Writing Python code for Opentrons robots
- Simulating and analyzing a new protocol for errors

Core Competencies

Laboratory Skills:

- Experimental design

Automation Skills

- Automation considerations for speed, efficiency, and reliability
- Using API documentation to write, test, and edit a Python protocol

Background Knowledge

This lesson plan was developed for students with an understanding of basic lab tasks, like reagent use and pipetting. No Python experience is required for this lab. However, students may need extra guidance working with:

-
- **The command line:** an (optional) way to test Opentrons Python protocols through simulation. You can find basic instructions [here](#).
 - **Python syntax:** Protocol Designer protocols created in the previous lab module will be helpful for students that have never worked with Python syntax before. You can also use a simple online tutorial or our example protocols (in the [API documentation](#) or [Protocol Library](#)) to see examples of code indentation, command structure, and more.

The **Student Guide** contains a pre-lab reading with a brief overview to help students get started writing an Opentrons Python protocol.

Supplies

Opentrons Equipment

- ☐ Opentrons OT-2 liquid handling robot
- ☐ OT-2 [Single-Channel GEN2 pipette](#)
- ☐ [OT-2 Thermocycler Module](#) (*optional*)
- ☐ [OT-2 Temperature Module](#) (*optional*)

Opentrons Tools

- ☐ Opentrons [Protocol Designer](#) (must be used in Google Chrome)
- ☐ [Learn Python](#) protocol

Non-Opentrons Equipment

- ☐ Benchtop thermocycler (*optional*)
- ☐ Benchtop microcentrifuge or vortex (*optional*)

Labware

- ☐ Opentrons pipette tips compatible with your chosen pipette
- ☐ PCR well plates, like [Opentrons Tough 96 Well Plate 200 \$\mu\$ L PCR Full Skirt](#)
- ☐ Snapcap tubes (like [Opentrons 24 Well Aluminum Block with NEST 2 mL Snapcap](#)) or generic PCR strip tubes (like [Opentrons 96 Well Aluminum Block with Generic PCR Strip 200 \$\mu\$ L](#))

Reagents and Other Materials

PCR reagents of your choice, including:

- ☐ Master mix (or individual polymerase, buffers, dNTPs mix, and cofactor like MgCl_2)
- ☐ DNA templates for amplification
- ☐ Forward and reverse primers
- ☐ Molecular grade water

Experimental Duration

This lesson plan was prepared for a traditional laboratory class time of 80–90 minutes. Instructors can choose to split this lesson plan in two to save time in class:

- Students use the API documentation to begin writing their Python protocol, which can be finished as a first draft outside of class.
- Students simulate, analyze, and edit their protocol, and complete a test run on the OT-2.

Basic Troubleshooting and Tips

- After analyzing for errors, student protocols can be tested using a *dry run* on the OT-2. Run the protocol without tips or liquid on the deck to check for expected movements and behavior. Be sure to create [labware offsets](#) for labware used in the protocol first.
- Instructors can choose the best fit for their lab's needs from the full list of Opentrons supported labware. Check

our [Labware Library](#) to see compatible well plates, reservoirs, and more.

- In the [example protocol](#), the OT-2 will pipette to combine buffers, dNTPs, and cofactor in the master mix, then combines your DNA template, primers, and water with the master mix. You and your students can decide which components should be assembled on- and off-deck.
- Your students can write their Python protocols in any code editor you prefer. We like Microsoft's [VSCode](#).
- In this lab, students should write their own Python protocol based on the Protocol Designer protocol created in the previous lab module. Here's a few tips to help your students use the Opentrons [Python Protocol API](#):
 - Labware lids have special restrictions for placement on the deck and on labware. Read more in [Loading Lids](#).
 - [Defining and loading liquids](#) are optional in an Opentrons Python protocol! We recommend adding this extra step so your students can visually compare



the two protocols. The [example protocol](#) includes color codes to match the liquids created in Protocol Designer.

- Concurrent commands can be used to perform module actions (for the [Temperature](#) and [Thermocycler](#) Modules) at the same time as other commands, like pipetting actions. This is an experimental design element your students can optionally use to decrease protocol run time.
- If your students used a single-channel pipette in Protocol Designer, they might have needed to add multiple transfer steps to accomplish a liquid transfer. When using the same [complex commands](#) in the Python Protocol API, [tip refilling](#) uses tip and transfer volume to calculate and add any additional steps needed. Your students can see this in action using the [example protocol](#) and its run preview in the Opentrons App.
- It's easy to add multiple advanced settings, like a blow out or air gap, to a transfer or mix step with a single click in Protocol Designer. Your students can add the same settings using [complex command parameters](#).
- There are several ways to list the wells the OT-2 should transfer liquid to and from in labware. See

[accessing wells in labware](#) to read more about list and dictionary options.

- The Thermocycler Module uses specific commands to program a [thermocycler profile](#) and [temperature holds](#).
- See the relevant sections below and in the **Student Guide** for instructions on analyzing, simulating, and troubleshooting Python protocols.
- Students can make a free account to use [OpentronsAI](#) if they're stuck. OpentronsAI can simulate protocols, fix errors, and answer questions about the Python Protocol API.

Procedure Guide

Lab Introduction and Experimental Planning

1. You might want to introduce the lab with a look at an example protocol to demonstrate Python syntax. You can find OT-2 protocols on the [Opentrons Protocol Library](#) or

use an example Protocol Designer protocol from the previous lab module.

2. Students should use the attached worksheet in the **Student Guide** to explore the Opentrons Python API and the differences they'll need to consider when writing their Python protocol.

Writing Python Protocols

When your students are ready to start writing, they can use the [Python API tutorial](#) to write their first few lines.

The resources listed below can be helpful as students write:

- **Protocol Designer protocols:** students can open the Python file they exported from Protocol Designer in the previous lab module. This can be used as a starting point, or simply as a helpful resource when they're stuck.
- **API reference:** the [API reference](#) includes an entry for every class and command in the Python Protocol API. For example, see this [entry](#) for the Temperature Module's `load_labware()` command, its behavior, and the parameters it accepts.
- **Labware library:** includes Python load names for all compatible labware.

- **Example protocol:** this lab module includes an example Python protocol that can be downloaded [here](#).
- **Opentrons code suggestions:** Students can install `pip` through the command line and should see suggestions as they write in their preferred code editor. They should install this as a part of [simulating their protocols](#).

As your students write Python to automate the same steps they designed in Protocol Designer, the following tips can be helpful:

- It can be helpful to add additional volume to your starting and dispense volumes to help account for dead volume (the amount of liquid that remains in the pipette tip after dispensing). You can customize parameters like [disposal volume](#) and [blow out](#) to ensure all liquid leaves the tip in the appropriate location.
- Completing a run of your protocol with *only water* can help your students visualize robot behavior and optimize their Python protocol without using valuable samples and resources.
- The Opentrons Python Protocol API is open-source. You and your students will find that there are multiple ways to complete the same task to account for considerations like time or contamination risk.

- You can customize aspirate and dispense height within and above your labware. [This section](#) in the API documentation takes a look at dispensing relative to the well top or bottom, or liquid meniscus.
- Different options can be used to control the pipette's order of operations. For example, the pipette can complete 10 dispenses into 10 wells, then mix in all 10 wells, or it can complete 10 dispenses, mixing after each one. These different options affect the amount of time it takes the OT-2 to complete your protocol.
- Depending on your students' familiarity with Python, this lesson plan offers flexibility in challenges and learning outcomes. This lesson plan and example protocol are written for absolute beginners, but the Opentrons Python Protocol API and OT-2 support:
 - More advanced Python, like lists and concatenate lists or dictionaries. Students can try this out with different methods to [access wells](#) in labware or by creating a [custom liquid class definition](#) for more control over liquid handling behavior.
 - Advanced control methods outside of the Opentrons OT-2 App, like [Jupyter notebook](#) and [command line](#) control.



You may want to have students analyze or simulate their protocols as they work to find and correct errors. See the section below for details.

Analyze and Troubleshoot Protocols

Students will need to test their protocols for errors—finished protocols containing errors can't be run on the OT-2. We recommend using the following methods in the order listed below:

- **Command line:** If your students can use the command line, they can [simulate](#) their protocol for errors. They'll need to install the [Opentrons Python module](#), but this won't require a connected robot.
 - If the protocol contains errors, this returns the first error encountered, including a line number.
Students can fix the current error and simulate again to continue checking the protocol.
 - If the protocol contains no errors, this returns a log of the commands in the protocol.
- **Opentrons App:** Download the [Opentrons OT-2 App](#) to analyze your protocol for errors. Because the OT-2 can't

run protocols that contain errors, the app completes an analysis every time a protocol is imported.

- Like simulating from the command line, click **error details** to view the error message and line number.
- When the protocol contains no errors, it can be run by the OT-2.
- **OpentronsAI**: In the chat, provide the Python file and ask OpentronsAI to simulate the protocol. Note that OpentronsAI is capable of fixing the errors for students, but can make mistakes. *You should always review OpentronsAI protocols yourself before running on the robot.*

When the protocol is complete, students can use [protocol visualization](#) as a final check:

1. Import the final, error-free protocol into the Opentrons OT-2 App.
2. Click the protocol to open its details page, then click **Visualize**.
3. On the setup page, click the **run preview** tab. This provides a full list of the actions the OT-2 will perform during the protocol, which students can match to their written Python protocol and check to be sure they see the intended behavior.

Protocol Run

A finished protocol can be used to complete a dry run or perform the PCR reaction on the OT-2.

Discussion Questions

Direct students to discuss the lab activities with one another. Example prompts might include:

- List two changes you needed to make in order to write your Python protocol. How were the commands different from your original Protocol Designer protocol?
- The Opentrons Python Protocol API often provides multiple ways to achieve the same goal. What was one command you wrote in your Python protocol that you could have written differently (either with parameters, an alternate method, or by using an entirely different command)?
- Were there any difficult errors to solve when analyzing and editing your protocol? Name one error and describe the changes you needed to make in your protocol to fix it.

Student Guide

Learn to Write a Python Protocol

OT-2 Edition

Authored by Emily Burghardt, Ph.D.

Opentrons

Pre-lab Reading

In last week's class, you created a Protocol Designer protocol to automate a polymerase chain reaction (PCR) on the OT-2. This week, you'll explore writing Python protocols using the [Opentrons Python Protocol API](#), a Python framework designed by Opentrons to make it easy for anyone to write their own protocols.

Your goal is to write a Python protocol that automates the *same steps* as your Protocol Designer protocol. This pre-lab reading will give you an introduction to Python protocols for the OT-2 so you can get started.

In a Python protocol, you'll be able to control the OT-2, its pipettes, and hardware modules (like a Temperature Module). Just like in your Protocol Designer protocol, you'll specify actions for the OT-2 to perform, like transferring and mixing liquid.

Before class, you should get started by reviewing the Opentrons Python Protocol API [tutorial](#) and reading the examples included on the following pages.

You'll start your protocol by providing information about the robot you'll use, the protocol name, and API level, like in the example below.

```
3  from opentrons import protocol_api, types
4
5  metadata = {
6      "protocolName": "Learn Python with the OT-2",
7      "author": "Emily Burghardt",
8      "description": "PCR master mix assembly, reagents assembly, and reaction in the
9                      Thermocycler Module. Paired with the 'Learn Python with Protocol Designer: OT-2' protocol."
10 }
11
12 # requirements
13 requirements = {
14     "robotType": "OT-2",
15     "apiLevel": "2.28"
16 }
```

Next, you'll load pipettes (and tips), labware, and any modules your protocol will use. From here on out, every line in your protocol should be inside the run function, as shown in the example below.

```
def run(protocol: protocol_api.ProtocolContext):
    tip_rack = protocol.load_labware(
        load_name="opentrons_96_tiprack_300ul",
        location="3"
    )
    pipette = protocol.load_instrument(
        instrument_name="p300_single_gen2",
        mount="left",
        tip_racks=[tip_rack])

    reservoir = protocol.load_labware(
        load_name="nest_12_reservoir_15ml",
        location="6"
    )
    plate = protocol.load_labware(
        load_name="nest_96_wellplate_200ul_flat",
        location="8"
    )
```

Once you've loaded all hardware, instruments (like pipettes), and labware you'll use in the protocol, it's time to tell the OT-2 *how* to use them. Take a look below at an example `transfer()` command, which tells the OT-2 to perform behavior similar to the transfer steps you added in Protocol Designer.

```
pipette.transfer(  
    volume=5,  
    source=tubes["A1"],  
    dest=tubes["A3"],  
)
```

Here, a pipette you've already defined in your protocol aspirates 5 μ L from well A1 and dispenses that same volume into the well A3 of a labware named `tubes` (defined earlier in your protocol, like in these [loading labware](#) examples). This command includes required *parameters* (`volume`, `source`, and `dest`, or destination). There are many optional parameters you can use to customize your transfers, just like you might have done in Protocol Designer. You can read more about them [here](#).

In today's lab, you'll write your own Python protocol using the basics covered in this pre-lab reading, but will need to dive a bit deeper. Although we think the best way to learn is to start writing, the pages listed below from the [Opentrons Python Protocol API](#) can help to review before class.

- [Introduction](#) to the Protocol API
- [Tutorial](#) for writing a first Python protocol
- Take a look at the API reference for a few commands you're likely to use in your protocol:

- [`load\_labware\(\)`](#)
- The Thermocycler Module commands [`execute\_profile\(\)`](#) and [`start\_execute\_profile\(\)`](#)
- [`mix\(\)`](#)

Purpose

In today's lab, you'll use the PCR protocol (created with Protocol Designer in the previous class) to write your first Python protocol. Your protocol should automate the same steps you added in Protocol Designer!

You might have noticed that Protocol Designer exported your completed protocol from last class as a Python (.py) file. Although this file contains Python commands, there are several differences in the way Protocol Designer writes a protocol *for you* and how you'll use Opentrons' API, or *application programming interface*, to write *your own*.

This means that you can use your Protocol Designer protocol for hints, or if you get stuck—but you'll still need to use the API to write your own work.

Learning Outcomes

- Learn about the Opentrons Python Protocol API
- Write a Python protocol to automate a PCR experiment on the OT-2
- Analyze, edit, and troubleshoot your Python protocol

Supplies

Opentrons Equipment

- ☐ Opentrons OT-2 liquid handling robot
- ☐ OT-2 [Single-Channel GEN2 pipette](#)
- ☐ [OT-2 Thermocycler Module](#) (*optional*)
- ☐ [OT-2 Temperature Module](#) (*optional*)

Opentrons Tools

- ☐ Opentrons [Protocol Designer](#) (must be used in Google Chrome)

- ☐ Opentrons [Python API documentation](#)

Non-Opentrons Equipment

- ☐ Benchtop thermocycler (*optional*)
- ☐ Benchtop microcentrifuge or vortex (*optional*)

Labware

- ☐ Opentrons pipette tips compatible with your chosen pipette
- ☐ PCR well plates, like [Opentrons Tough 96 Well Plate 200 \$\mu\$ L PCR Full Skirt](#)
- ☐ Snapcap tubes (like [Opentrons 24 Well Aluminum Block with NEST 2 mL Snapcap](#)) or generic PCR strip tubes (like [Opentrons 96 Well Aluminum Block with Generic PCR Strip 200 \$\mu\$ L](#))

Reagents and Other Materials

PCR reagents of your choice, including:

- ☐ Master mix (or individual polymerase, buffers, dNTPs mix, and cofactor like MgCl_2)

- ☐ DNA templates for amplification
- ☐ Forward and reverse primers
- ☐ Molecular grade water

Procedure Guide

Before Class

1. Complete the pre-lab reading, and review other materials as directed by your instructor.
2. If you'd like to see more Python examples before class, you can explore Opentrons protocols on the [Protocol Library](#). On the left hand side, choose your robot model (OT-2) and editability (Python).

Keep in mind that some protocols are more complex than others, making use of `for` loops or runtime parameters to decrease the amount of code written.

Experimental Planning

1. Open your Protocol Designer protocol from the last class. We recommend importing the Python file back into [Protocol Designer](#).
2. Use the attached worksheet and the [Opentrons API documentation](#) to plan your Python protocol.

Writing your Python Protocol

Start a new file in your preferred editor, like [Microsoft VSCode](#).

Use the attached worksheet to translate your Protocol Designer steps into a Python protocol.

The tips below might be helpful as you write:

- The OT-2's trash bin must *always* be loaded in slot 12, so you don't need to specify this in your protocol.
- In Protocol Designer, you needed to add starting liquid volumes to labware. In the Python API, [defining and loading liquids](#) are optional, but we still recommend you do it here. It's good practice, and you'll see the same liquid details in your two finished protocols.

- Find load names for the [labware](#) and [pipettes](#) you'll use at the links provided here.
- Read about [complex liquid handling commands](#) like `transfer()`, `distribute()`, and `consolidate()`. You might want to use `transfer()` and `distribute()` in your protocol.
- Click on each command to view its definition in the [API reference](#). This page contains clear definitions for each command in the API, including required and optional *parameters*.
- Labware lids have special restrictions for placement on the deck and labware. Read more in [Loading Lids](#).
- If your Protocol Designer protocol includes liquid class settings, you'll need to stick with legacy commands like `transfer()`. [Liquid class definitions](#) are one difference between Protocol Designer and the Python Protocol API: these commands aren't supported by the API (for an OT-2) and will cause an error in your protocol.
- It's easy to add multiple advanced settings, like a blow out or air gap, to a transfer or mix step with a single click in Protocol Designer. You're able to add the same settings in your Python protocol using [complex command parameters](#).

- The Python API provides multiple ways you can tell the OT-2 to perform the same action. One example of this is in module commands, of which there are two types:
 - **Blocking** module commands tell the OT-2 to perform a module action, like heating the Temperature Module, and wait for that action to complete. In other words, your protocol won't move to the next step until the Temperature Module reaches the temperature you asked it to.
 - **Concurrent** module commands tell the OT-2 to perform the same action and *continue* performing protocol steps while it completes.

Read more about the available command options for [Temperature](#) and [Thermocycler](#) modules.

- Read about [tip refilling](#) in the Protocol API. How can you use this when writing your protocol?

Stuck? Try the resources listed here:

- Check your work on the first parts of your protocol (metadata and requirements) in the OT-2 [protocol template](#).
- Find API load names for labware in the [Labware Library](#).



You can also open your Protocol Designer protocol in your code editor to see the commands and structure that Protocol Designer uses. Remember that Protocol Designer can use commands that are different from those in the Python protocol API.

Analyze and Troubleshoot your Python Protocol

Your completed protocol should be simulated and analyzed for errors before completing a test run on the OT-2. Follow the steps below to make sure your protocol is error-free:

1. [Simulate your protocol](#) from the command line. First, install the Opentrons Python package using `pip install opentrons`. Then, run `opentrons simulate path/to/my/protocol.py`, replacing with your own path and .py file name. If your protocol contains errors, you'll see error messages to help you fix them.
2. Import your protocol into the Opentrons OT-2 App. Only protocols that pass analysis can be run on the OT-2, and similar error messages should help you make required changes.



When your protocol is complete, use [protocol visualization](#) as a final check:

1. Import the final, error-free protocol into the Opentrons OT-2 App.
2. Click the protocol to open its details page, then click **Visualize**.
3. Match the protocol steps on the left, the deck view in the middle, and the labware and liquid changes on the right to your written Python protocol.
 - a. Is anything missing?
 - b. Will the OT-2 complete an action differently than you intended?

Protocol Run

1. Import your finished protocol into the Opentrons OT-2 App.
2. Complete deck setup and a [labware position check](#) for your protocol.
3. Run your protocol to automate PCR on the OT-2!

Discussion Questions

Follow your instructor's directions to discuss the following prompts, either with a neighbor or in a lab report:

- List two changes you needed to make in order to write your Python protocol. How were the commands different from your original Protocol Designer protocol?
- The Opentrons Python Protocol API often provides multiple ways to achieve the same goal. What was one command you wrote in your Python protocol that you could have written differently (either with parameters, an alternate method, or by using an entirely different command)?
- Were there any difficult errors to solve when analyzing and editing your protocol? Name one error and describe the changes you needed to make in your protocol to fix it.

Worksheet: Writing a Python Protocol

This worksheet will guide you in writing a Python protocol for the OT-2 using the [Opentrons Python Protocol API documentation](#).

Planning your Python protocol

Import your PCR protocol from last week back into Protocol Designer. You'll want to see your labware, hardware, and protocol steps on a second screen while you write.

Remember that there will be differences in the way Protocol Designer wrote your protocol and the way you'll write *your own* using the Python Protocol API. This could mean that you'll write *less* Python than Protocol Designer did!

Use the table and space provided below to gather information from the [Python Protocol API](#)—in other words, how will you create the same protocol steps by writing your own Python commands?

Check the list in **Writing your Python protocol** for hints and resources that might be helpful as you work.



Protocol Designer step details	API documentation section	Commands	Notes
Instruments:			
Modules:			
Liquids:			







Use the rest of the space provided for extra notes as needed.

—