

Pre-Course Readings

Topics

1. Replit
2. Using the Shell
3. CSV files
4. Command Line Arguments

1. Replit

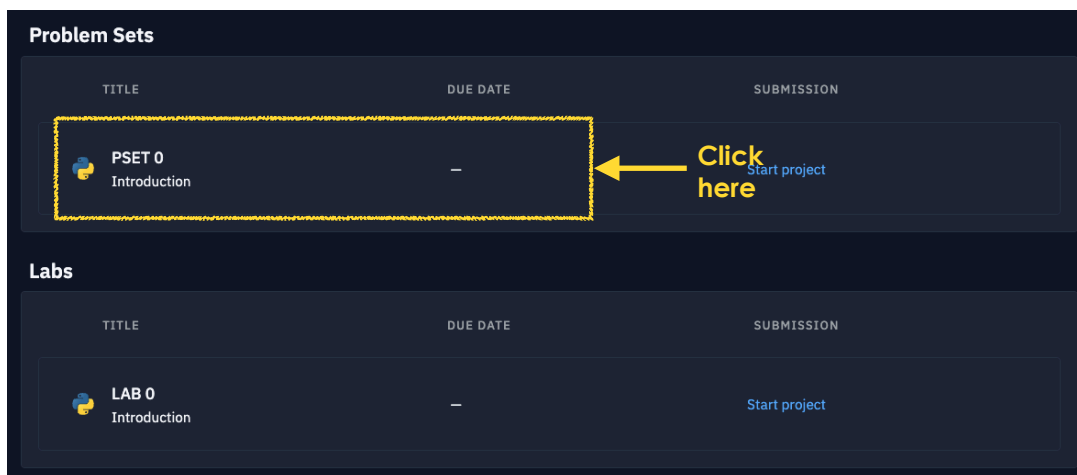
Replit is the Integrated Development Environment (IDE) that we will be using for this course. It is an online IDE that allows real time collaboration on projects.

Note: Replit is used such that students can monitor your work on lab problems & problem sets. If you wish to complete problem sets in your personal IDE, you may do so. However, if you would like your teacher mentor to take a look at your work, do remember to update your repl project!

1.1 Setting up Replit and trying out the interface

Step 1: To access the CodeIT replit team, please use the link sent to you on the course telegram group.

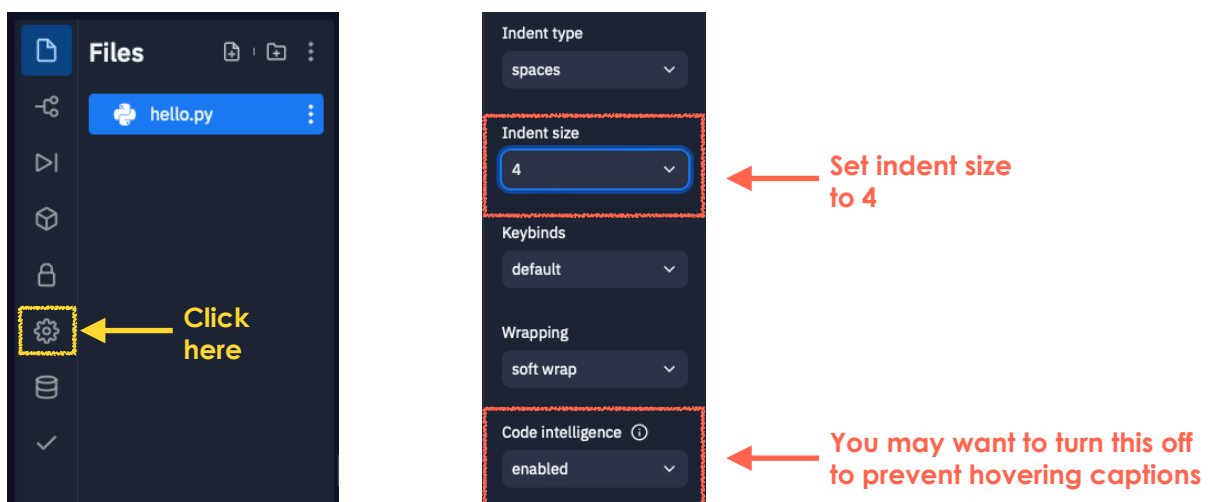
When you join, you should see the following:



To get to this page normally, you can click on the **Teams** tab in the menu of replit, and then click the team name

Step 2: Click on **LAB 0** to begin trying out the program.

Step 3: Change your indent size to 4. This is the standard we have set for this course.



2. Using the Shell


Lab problems and problem sets in this course often come with a starting set of files as well as a specific file for you to complete. These should not be renamed, or moved around as it may cause problems for the auto grader. While there is a **Run** button provided by repl, you should not use that, but instead use the **Shell** to run your python code.

Common Shell Commands

To demonstrate shell commands, we will continue with the LAB 0 project from earlier on. If you are familiar with Shell commands (executing .py scripts, navigating between directories), you may skip this section.

A. Executing a python script

To execute a python script, simply type `python <python file-name>`. If you followed the instructions accordingly on **hello.py**, and ran it, you should see an output like such:



The screenshot shows the Shell interface with the 'Shell' tab selected. The command prompt is `~/LAB-0-johnyeo10$`. The user has entered `python hello.py`, and the output is `name: John`, `age: 20`, and `Hello John, 20`. A red arrow points to the 'Shell' tab with the text 'Make sure you are using "Shell" and not "Console"'.

B. Navigating between directories

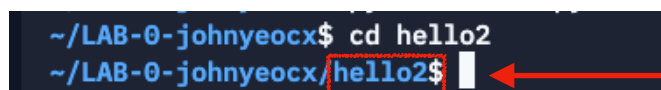
A directory represents which folder you are currently at on the Shell. Your current directory is important because you can **only execute python scripts contained in your current directory**. When you first start a project on repl, the Shell will always be at the root folder.



The screenshot shows the Shell interface with the 'Shell' tab selected. The command prompt is `~/LAB-0-johnyeocx$`. A yellow box highlights the prompt, and a yellow arrow points to it with the text 'Root Folder'.

``cd <folder-name>``

This first command allows you to enter any folder in your current directory. For instance, to navigate into the **hello2** folder:



The screenshot shows the Shell interface with the 'Shell' tab selected. The command prompt is `~/LAB-0-johnyeocx$`. The user has entered `cd hello2`, and the prompt has changed to `~/LAB-0-johnyeocx/hello2$`. A red arrow points to the new prompt with the text 'Now in the hello2 folder'.

``cd ..``

This command allows you to navigate to the parent folder. For instance, if we are currently in **hello2**, and want to navigate back to the root folder:



The screenshot shows the Shell interface with the 'Shell' tab selected. The command prompt is `~/LAB-0-johnyeocx/hello2$`. The user has entered `cd ..`, and the prompt has changed back to `~/LAB-0-johnyeocx$`. A red arrow points to the new prompt with the text 'Back to the root folder'.

3. CSV Module

In this course, we have opted to use the csv module when reading csv files. Imagine we have the following data:

signups.csv

```
Name, Age
John, 20
Alice, 25
Charlie, 18
```

3.1 Reading from a csv file with the csv module

```
import csv

with open('signups.csv', 'r') as csv_file:
    myReader = csv.reader(csv_file)

    lineCount = 0
    for row in myReader:
        if lineCount == 0:
            lineCount += 1
            continue
        print(row)
```

In the program above:

1. We imported the csv module
2. After opening **signups.csv** and storing the information in **csv_file**, we pass the variable to a function **csv.reader** from the csv module
3. This function returns an iterable **myReader**, which we can then iterate over to extract each row
4. We use **lineCount** to skip over the first row of our csv file, which is merely the names of the columns (**Name, Age**)

output

```
['John', '20']
['Alice', '25']
['Charlie', '18']
```

As you can see, each row is nicely extracted as a list, where each element is a column in our csv file (e.g. Name, Age)

Note: All data read is as strings. For instance, the age of each signup is a string

3.2 Writing to a csv file with the csv module

```
import csv

signups = []
with open('signups.csv', 'w') as csv_file:
    myWriter = csv.writer(csv_file)
    for i in range(3):
        name = input("name: ")
        age = input("age: ")
        myWriter.writerow([name, age])
```

Here, we create a for loop to take in **three names and ages** from the user

To write using the csv module, we:

1. Pass our `csv_file` variable into the **writer** function from the csv module
2. Use the method **writerow** which takes in a list as its argument
3. This will then write to the csv file in the format where each element in the list will be separated by a comma

output

```
name: Jacob
age: 22
name: Daniel
age: 24
name: Sarah
age: 32
```

signups.csv

```
Jacob,22
Daniel,24
Sarah,32
```

Note: To append to a csv file, simply change the mode to **'a'** when you `open()` the file

3.3 Writing multiple rows to a csv file with the csv module

In fact, csv makes the process of writing to csv files even easier by allowing us to write **multiple rows**. We do so using the `.writerows()` method:

```
import csv

data = [
    ["John", 20],
    ["Alice", 25],
    ["Charlie", 18]
]

with open("signups.csv", "w") as csv_file:
    myWriter = csv.writer(csv_file)
    myWriter.writerows(data)
```

4. Command Line Arguments

4.1 What are command line arguments

Command line arguments are simply just arguments that we pass when we **run our program**. For instance

```
python cla.py hello world
```

arg 0 arg 1 arg 2

To use command line arguments, we have to import the `sys` module. From there, we can then use the `argv` variable!

cla1.py

```
import sys

print(sys.argv[0])
print(sys.argv[1])
print(sys.argv[2])
```

output

```
python cla1.py hello world
cla1.py
hello
world
```

As you can see, **argv** is simply a list of our command line arguments, and if we index into it, we can retrieve the arguments passed when running the program.

4.2 Why command line arguments?

We often use command line arguments to pass information before running of the program. For instance, imagine we have files storing the information of different parties:

signups1.csv

```
John,20
Alice,25
Charlie,18
```

signups2.csv

```
Tobey,54
Andrew,35
Tom,25
```

signups3.csv

```
Jack,30  
Harry,22  
Peter,25
```

With command line arguments, we can target which file we want to read from like so:

cla2.py

```
import csv  
import sys  
  
with open(sys.argv[1], 'r') as csv_file:  
    myReader = csv.reader(csv_file)  
    for row in myReader:  
        print(row)
```

output

```
python cla2.py signups1.csv  
['John', '20']  
['Alice', '25']  
['Charlie', '18']  
  
python cla2.py signups2.csv  
['Tobey', '54']  
['Andrew', '35']  
['Tom', '25']  
  
python cla2.py signups3.csv  
['Jack', '30']  
['Harry', '22']  
['Peter', '25']
```

As you can see, we have now given ourselves the flexibility to read from any file we wish to!

Note: It is important to check that the user has passed in the correct number of command line arguments, or our program could create an error. To do so, we can simply check the **length of argv**

```
import csv  
import sys  
  
if len(sys.argv) < 2:  
    print("require file name")  
else:  
    with open(sys.argv[1], 'r') as csv_file:  
        myReader = csv.reader(csv_file)  
        for row in myReader:  
            print(row)
```

Note:

- To view the code examples, click on the L0 Materials project in the replit team
- You may try the above examples by yourself by clicking on the PSET 0 project