CIS 210
LAB 1: Working Environment

1. Introduction

Introduce VSCode and Python – Explore foundational Python concepts using the terminal and VSCode

(Note: We use VSCode because it's simpler, but you may gradually transition to the more comprehensive PyCharm IDE.)

1.1. Outcomes

- ☐ Become familiar with VSCode, an IDE (Integrated Development Environment), and the Python interpreter deployed using a terminal
- ☐ Interact with the Python interpreter (use the VSCode shell) to write and execute Python code
- ☐ Use Python built-in function help for general help, or help with a specific function
- ☐ Understand that two Python namespaces are available when Python starts up: __builtins__ (with Python built-in functions) and __main__ (where Python keeps track of user-defined Python objects)
- ☐ Use Python built-in function dir to inspect __main__ and __builtins__ namespaces
- ☐ Understand that the Python interpreter is a Read-Evaluate-Print Loop (REPL) – when we enter Python code, the Python interpreter:
  - *reads* the Python code (and checks that it is legal Python code)
  - *evaluates*/executes the code
  - returns the resulting value (if any), Python automatically *prints* it when it executes it.
  - waits for the next Python statement
- ☐ Execute Python expressions (built-in functions and operators), e.g., arithmetic operators, abs, pow, round
- ☐ Understand that expressions are evaluated and return a value
- ☐ Use assignment statements to define variables in the Python Shell; understand that variable names are identifiers for Python objects
- ☐ Understand that Python assignment statements are not expressions and do not return a value; they affect the namespace
- ☐ Use Python built-in function type to check the type of a Python object
- ☐ Define a function in the Python Shell
- ☐ Understand that function definitions are not expressions and do not return a value; they are assignments and affect the namespace
- ☐ Understand that functions are executable (callable) objects
- ☐ Understand that a function call is an expression – executing a function returns a value
- ☐ Execute user-defined functions in the Python interpreter
- ☐ Start becoming familiar with Python error messages
- ☐ Write, save, and execute (using Run Module) code written in the editor – variable assignments, including function definitions
- ☐ Write a short Python function according to CS 210 style guidelines, including the docstring
- ☐ More exposure to Python functions written according to CS 210 style guidelines (especially docstrings)

1.2. Important concepts

- ☐ Python is a language and a program interpreter.
- ☐ Python code we write is input to the Python interpreter program.
- ☐ We interact directly with the Python interpreter program when we use the Python Terminal in VSCode.
- ☐ Python executes well-formed expressions and statements; expressions return a value.
- ☐ The Python Terminal in VSCode is a REPL. When it executes an expression, it automatically prints the returned value.
- ☐ Variable assignments, including function definitions, are not expressions. They are not evaluated and do not

return a value. They associate an identifier (name) with a value (object); this information is stored in a Python namespace.

- ☐ Python objects have a type; objects of type function are executable/callable.
- ☐ Code written in the Editor window and saved to a .py file is persistent; code written in the Shell is not.
- ☐ Code written in the Editor window must be executed all at once (Run Module); in the terminal, running the Python interpreter, we can execute code snippets.

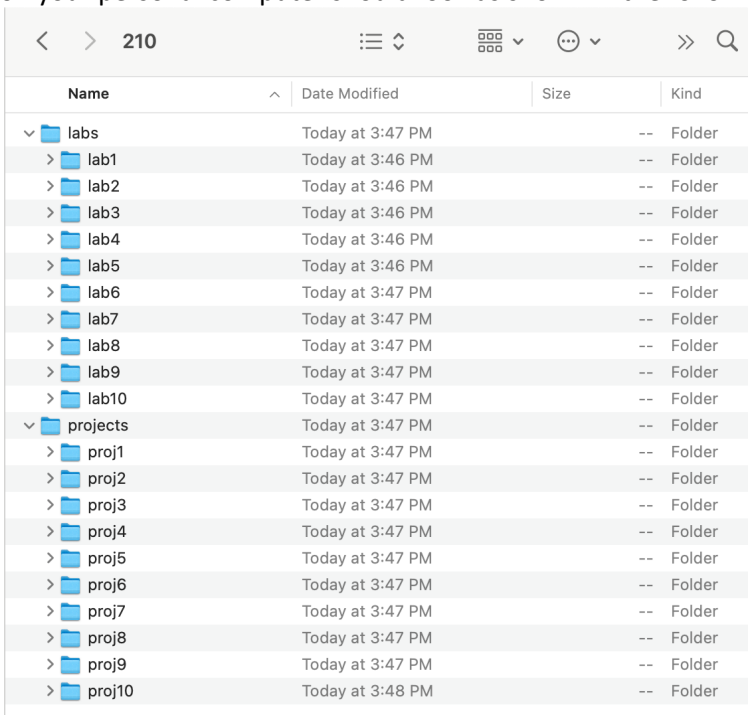
2. The 210 Local Folder

Where should your `210` folder live on your computer?

i. Create a folder named `210` in your `Documents` folder (or the desktop or any other place of your preference). This folder is your 210 Python workspace. Make sure you know where to find it without having to search.

ii. What structure should your `210` folder have? Create two subfolders called `labs` and `projects`. Within `labs` create subfolders named `lab1`, `lab2`, to `lab10`. Use exactly these names, completely in lowercase with no spaces. Within `projects` create subfolders named `proj1`, `proj2`, to `proj10`. Use exactly these names, completely in lowercase with no spaces.

Your 210 workspace on your personal computer should look as shown in the following image.

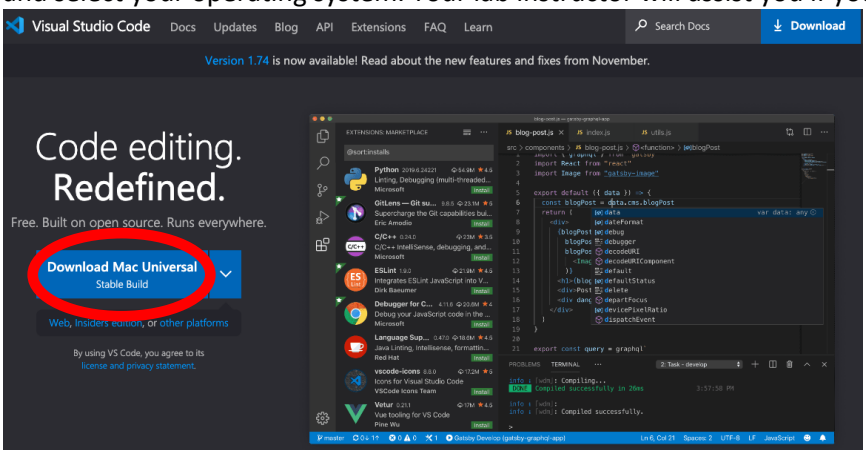


Name	Date Modified	Size	Kind
labs	Today at 3:47 PM	--	Folder
lab1	Today at 3:46 PM	--	Folder
lab2	Today at 3:46 PM	--	Folder
lab3	Today at 3:46 PM	--	Folder
lab4	Today at 3:46 PM	--	Folder
lab5	Today at 3:46 PM	--	Folder
lab6	Today at 3:47 PM	--	Folder
lab7	Today at 3:47 PM	--	Folder
lab8	Today at 3:47 PM	--	Folder
lab9	Today at 3:47 PM	--	Folder
lab10	Today at 3:47 PM	--	Folder
projects	Today at 3:47 PM	--	Folder
proj1	Today at 3:47 PM	--	Folder
proj2	Today at 3:47 PM	--	Folder
proj3	Today at 3:47 PM	--	Folder
proj4	Today at 3:47 PM	--	Folder
proj5	Today at 3:47 PM	--	Folder
proj6	Today at 3:47 PM	--	Folder
proj7	Today at 3:47 PM	--	Folder
proj8	Today at 3:47 PM	--	Folder
proj9	Today at 3:47 PM	--	Folder
proj10	Today at 3:48 PM	--	Folder

3. Visual Studio Code

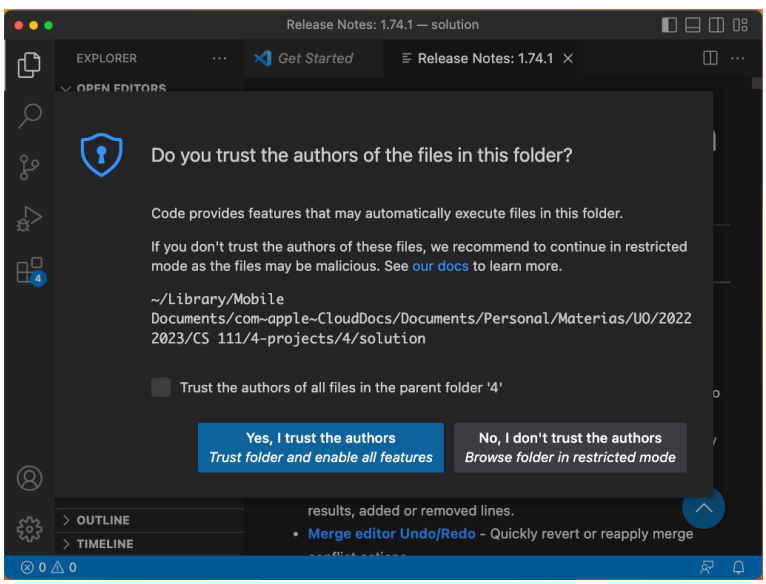
Visual Studio Code (VSC) is an Integrated Development Environment (IDE) that we will use in this course to edit the code we produce (HTML, CSS, and JavaScript). VSC is an easy-to-use IDE.
To use VSC, you must download and install it on your computer. Download Visual Studio Code from

https://code.visualstudio.com/. The VSC website will detect what computer and operating system you are using; just click on the Download button at the center left of the web page (highlighted with a red circle in the image below.) If it does not detect your operating system, click on the down arrow icon and select your operating system. Your lab instructor will assist you if you are in trouble.

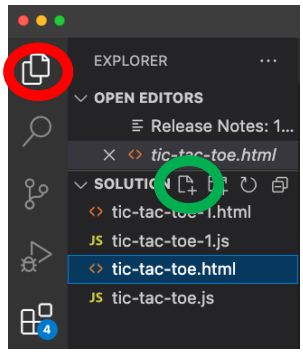


You downloaded a zip file. Double-click on the zip file to decompress it, then drag it to the applications folder. Drag it to the dock from the applications folder, so you have it readily available – you will be using it all the time this term. If you use Windows or Linux, your lab instructor will assist you during installation.

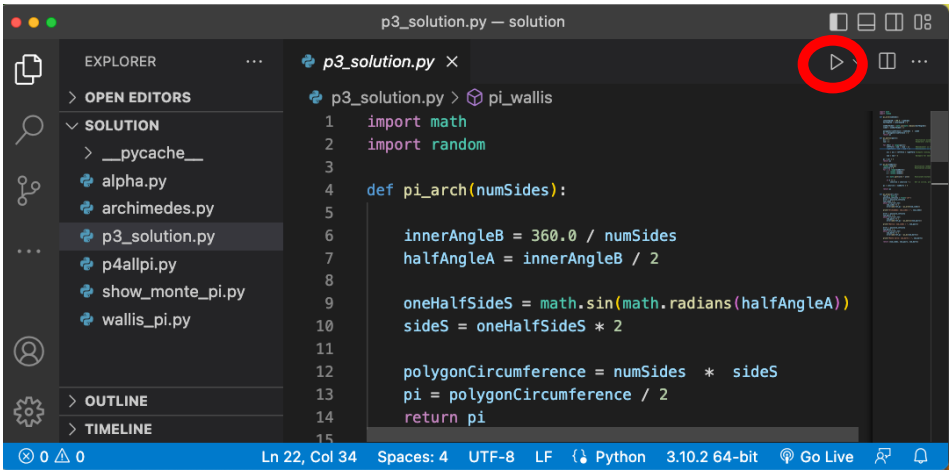
As indicated in the previous section, you must organize projects in folders. The best way to open a project is to drag that folder to the application icon. Alternatively, click on the **`File`** menu and select **`Open Folder`**. A popup window will ask if you trust the files inside that folder. Click on **`Yes …`** . Additionally, you may check the option **`Trust the authors of all files in the parent folder`**.



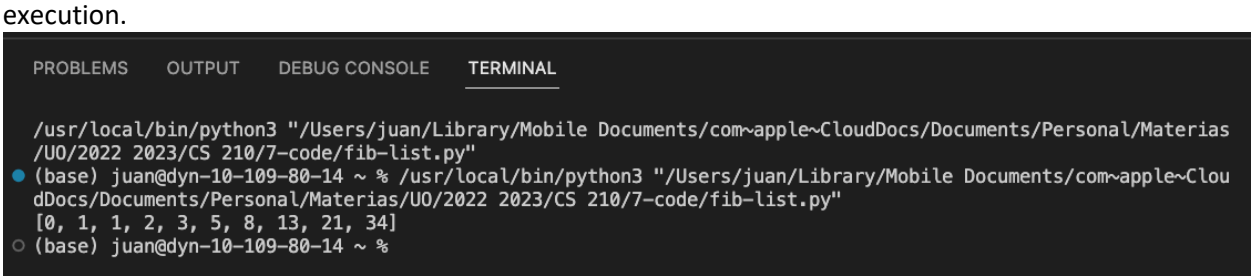
Once you open a project folder, click on the Files icon circled in red in the image below. You can modify the files inside that folder by clicking on the file from the file list and then clicking on the right pane to make the desired changes. You can also create new files by clicking on the New File icon, circled in green in the image below.



When editing a Python file, the last line of your window indicates if the file has errors, the cursor position, and other information.



Select **`Terminal`** from the application menu to execute a Python program, then select **`New`**. In the Terminal, type python and the name of the file, you want to execute. Alternatively, click on the Run Python Program icon, circled in red in the figure above. See the image below for an example of Python execution.



4. Exercises: Expressions, Assignment, and Error messages.

At the prompt (>>>), enter:	WHAT DO YOU THINK WILL HAPPEN?	WHAT HAPPENS?	QUESTIONS/NOTES
help()		Python built-in function helpexecutes; help program runs	
At help prompt (help>), enter different Python symbols			See the Python tutorial [1] and operators [2].
At help prompt (help>), enter			See the Python keywords table

At the prompt (>>>), enter:	WHAT DO YOU THINK WILL HAPPEN?	WHAT HAPPENS?	QUESTIONS/NOTES
keywords			[3].
At help prompt (help>), enter quit			
At the prompt (>>>), enter dir(__builtins__)		It produces a list of all the built-in Python functions provided when Python starts up.	We will mainly be using functions starting at abs.
help(abs)		Help on Python built-in function abs	
abs(99) **abs(-99)** **round(99.36)** **round(99.36, 1)** **pow(4, 3)**		Python evaluates those expressions; they return and print a value.	See extra exercises at the end of the worksheet for more on round.
4 ** 3 **Expressions using other Python numeric symbols.** **For example, 10 + 4, 10 - 4, 10 * 4, 10 / 4, 10 // 4, 10 ** 4, 10 % 4**			** symbol is a shortcut, or "syntactic sugar" for pow.
10%4			Python ignores in-line whitespaces, but the Python style guidelines advise using spaces on each side of non-unary operators, e.g., 10 % 4.
4 +- 3 **enter 4 + -3** **4 + (-3)** **4 %+ 3** **4 % +3** **4 % 3**			+ and – are also unary operators (they are "overloaded"). Python will execute all of them, but which style is better? Why?
4 +% 3 **4 $ 3**			Not well-formed expressions.
mypi			Well-formed expression, but Python does not recognize it.
mypi = 3 **mypi**			Now mypi is a defined symbol
Recall: namespaces are how Python keeps track of identifiers. A global namespace, __main__, is initialized when Python starts up (along with __builtins__). The names of variables created during the Python session are stored there.			
dir()		Python built-in function dir executes; it shows Python global namespace, with some special Python variables already there (including mypi).	
__name__		'__main__' (the name of the global namespace)	
def myfunc(i, j): ** return max(i, j)**			You will need to tab over.
dir()			myfunc has been added to the global namespace.
myfunc			
myfunc(99, 100)			Functions are an executable data type.
(Try to) execute mypi: **>>> mypi()**			mypi is not an executable data type.

At the prompt (>>>), enter:	WHAT DO YOU THINK WILL HAPPEN?	WHAT HAPPENS?	QUESTIONS/NOTES
type(mypi)			
type(myfunc)			

The Python Shell is convenient for testing code snippets – while developing a program or experimenting with and exploring Python.
But – what we enter in the Shell is not persistent. It is lost when we restart the Shell or quit VSCode. The Editor window allows us to write code that persists between Python sessions.

5. Exercises: Printing returned values, defining, and calling functions

TRY THIS At the prompt (>>>), enter:	WHAT DO YOU THINK WILL HAPPEN?	WHAT HAPPENS?	QUESTIONS/NOTES
From File Menu, choose New File		VSCode asks for a file name and location (let's say you call it greeting.py). After that, an Editor window appears.	
Type a file header into the untitled Editor window: ''' CS 210 Lab 1 – Lab 1 Exercises Author: [your name here] Credits: [acknowledgments - lab group, perhaps others] Lab exercises demonstrating how IDLE Editor and Shell interact. '''			All CS 210 .py files must include a file header like this
greeting = 'hello, Python'			(after leaving a blank line) Note the IDE support for Python code. E.g., the string is pink.
Save the file.			The combination of correct Python code and .py extension makes this a "Python program". Otherwise, it's just a plain text file.
From the VSCode terminal, run Python, then execute the line from greeting import *			Python executes all the module code, meaning that greeting is defined and added to the namespace.
At the Shell prompt, inspect the global namespace: dir()			

TRY THIS At the prompt (>>>), enter:	WHAT DO YOU THINK WILL HAPPEN?	WHAT HAPPENS?	QUESTIONS/NOTES
greeting type(greeting)			
Return to the editor window and change 'greeting' to 'welcome': welcome = 'hello, Python'			

Enter the following function definition:
`def is_even(n):`
`  '''`
`  (int) -> Boolean`
`  Returns True if n is an even number`
`  >>> is_even(100)`
`  True`
`  >>> is_even(101)`
`  False`
`  >>> is_even(0)`
`  True`
`  '''`
`  return (n % 2) == 0`
			notice color coding, auto indents, and other support for Python coding

Type again: from greeting import *			The global namespace is cleared, and the module's code is executed. This time, both welcome and is_even are added to the global namespace. (Note that greeting is no longer in the namespace.)
Inspect the global namespace; check the value and type of is_even: dir() is_even type(is_even)			
help(is_even) is_even(99)			

The Shell is an excellent tool for learning Python: testing bits of code and exploring with help. If anything goes wrong, just interrupt or restart the Shell (trashcan icon on the line above the terminal).

6. Optional (more on built-in function round)

TRY THIS At the prompt (>>>), enter:	WHAT DO YOU THINK WILL HAPPEN?	WHAT HAPPENS?	QUESTIONS/NOTES
round(123.4)			
round(123.6)			
round(123.5)			

TRY THIS At the prompt (>>>), enter:	WHAT DO YOU THINK WILL HAPPEN?	WHAT HAPPENS?	QUESTIONS/NOTES
round(124.5)			
round(2.5)			
round(3.5)			

Do you see what Python is doing? Explore further or confirm your hypothesis:
Do you have ideas about why Python 3 uses this method of rounding?
If you thought of trying >>> help(round) – great idea! It presents an excellent description of the function round, and it conveys this interesting/important detail: Documentation is important!
Explore the second parameter of the round function.

7. Lab Submission.

You will find an XL file on Canvas containing the same information as the tables in this document. Fill out that file with your results and submit it on Canvas->Assignemnts->Lab-1.

8. References

[1]. Python Tutorial. https://docs.python.org/3.9/tutorial/.
[2]. List of Python operators. https://techbeamers.com/python-operators-tutorial-beginners/.
[3]. List of Python keywords. https://www.w3schools.com/python/python_ref_keywords.asp.
[4]. How to run Python scripts. https://realpython.com/run-python-scripts/#:~:text=can%20continue%20reading.-,Using%20the%20python%20Command,python3%20hello.py%20Hello%20World.