Code Style Requirements

If you have any questions about how to conform to any of these requirements, please feel free to post a question on Ed.

Code techniques

* All data members of a class should be private unless explicitly specified otherwise in the README.
* Do not use any global variables.
* Do not use any mutable default arguments. A mutable default argument is simply a default argument that is of a mutable type (such as a list, dictionary, or set). Mutable arguments are fine, and default arguments are fine - just not both at once. Instead of doing this:

def some\_func(my\_list=[]):  
 my\_list.append(1)  
 return my\_list

do this:

def some\_func(my\_list=None):  
 if my\_list is None:  
 my\_list = []  
 my\_list.append(1)  
 return my\_list

In both examples, my\_list is the parameter. In the first example, [] is the default argument. Since that's a list, and lists are mutable, that makes it a mutable default argument. In the second example, None is the default argument. Since None is not mutable, it is not a mutable default argument. Since integers, floats, Booleans, strings, and tuples are not mutable, you don't need to worry about using None instead, but lists, dictionaries and sets are mutable. [For a little more information, you can refer to the CS 161 exploration on mutability vs. immutability (or in the review course, if you're a transfer student).]

* Do not use "while True" as a loop condition. Instead, you should write a loop condition that will eventually become False, thus ending the loop. This makes the behavior of your loop simpler to read. Project 8a, which uses a generator function, will be an exception, since the loop is meant to be infinite. (There may be other exceptions in complex real world code, but not in this course.)

Comments

You should include comments at the top of every file, giving the file's author, date and description.

# Author:   
# GitHub username:  
# Date:  
# Description:

The "Description" comment should be a high-level description of what the code in the file **does**.

Note that the GitHub username is a **new** requirement starting from Winter 2022 term for this course.

The above should usually not be the only comments in your programs.  Comments are very valuable for both yourself (you forget faster than you might think) and for other programmers who have to understand and maintain your code.  Comments should describe what is happening, what parameters and variables mean, any restrictions or bugs, etc.  Other good places for comments include where the intention of a section of code might not be obvious, or where you had to make some kind of design decision. Short comments should be *what* comments, such as "compute mean value", rather than *how* comments such as "sum of values divided by n".

Docstrings

All functions and classes should have docstrings describing what the function or class does. The one exception is \_\_init\_\_() methods.

Naming Conventions

* All names of variables, constants, functions and classes should be descriptive of their purpose.
* Avoid using names that are very similar to each other.
* Variable names should start with a lower-case letter.  If the name is made of multiple words, the words should be separated with an underscore, e.g. min\_score, or client\_favorite\_food.
* Single letter names are not descriptive and should not be used except in list comprehensions or in the context of a well-known equation, such as the quadratic formula.
* Function names should follow the rules as variable names, e.g. binary\_search() or find\_max\_value().
* Class names should begin with a capital letter, with additional capital letters at the beginning of successive words that are part of the name, e.g. Student, or SalariedEmployee.

General Readability

* Lines should be kept to a decent length. Don't try to do too much in a single line.  (Don't worry if a line is long just because you used descriptive names for variables/functions/classes.)
* Functions also should be kept to a decent length - if a function is getting too long, you may want to break it up into multiple functions.
* **Always use proper indentation**to indicate the structure of your code.
* Use one blank line to separate logical chunks within a function.
* Separate functions/methods/classes with at least one blank line.

Note that you could lose up to 80% of points for that class/method if you don't follow these requirements.

Having trouble accepting or cloning a repository - Creating your own GitHub repo

Introduction

If you are having trouble accepting or cloning a repository from the class, you can create your own on the GitHub website.

Steps

1. Open an internet browser and navigate to [https://github.com/Links to an external site.](https://github.com/)

2. Select the green ‘New’ button on the sidebar to the left side of the screen. The button will be in the top right corner of the sidebar. This will take you to a new page where you can write the repository name and description

Graphical user interface, application

Description automatically generated

3. Create the repository. You should choose a logical name that will help you remember the project, like ‘cs161program3’ and you should choose the ‘private’ option. Remember the course rules and the Student Code of Conduct mandate that any GitHub repository you post assignment code in **must be private**. If you keep your repository public, you could be reported for academic dishonesty.

The description is optional, and you can edit it later. When you have typed a name, you can select the green ‘create repository’ button at the bottom of the screen. This will take you to a new page.

Graphical user interface, text, application, email

Description automatically generated

4. Upload the files

Graphical user interface, text, application, email

Description automatically generated

Click the linked text ‘uploading an existing file’ to go to the file upload page

5. Click on the linked text ‘choose your files’ and a window will pop-up where you can select your Python file. It will probably be in a folder called PyCharmProjects on your computer. You should see a screen like this,

Graphical user interface, text, application

Description automatically generated

6. After your file has uploaded, click the green ‘commit changes’ button at the bottom of the page. This should update your new repository with the python file, then you can submit it to GradeScope the same way you usually do.

GradeScope failed - What now?

If you've tested your code in PyCharm and are sure that it's working as expected, you might still be getting errors in Gradescope. Here are some common issues and how to fix them.

**Common Gradescope Problems**

You submit your code, which seems to work perfectly in PyCharm, and see something like "The autograder failed to execute correctly." These are the frustrating errors because they are problems in Gradescope, not your code. Before jumping on Slack or Piazza or emailing your TA, run through this checklist of possible errors.

* Is your file/class/function named correctly according to the README?
* Are the parameters for your functions the same as described in the README?
* Did you leave your test cases in the assignment?  Your submission should only contain the required class and function definitions, not your test code.  While you should be thoroughly testing your code, these test cases need to be removed for your function to work in Gradescope. Note that in cases like Assignment 2, the unit tests that you write are supposed to be in a file different from the main file and they are not tested by GradeScope.
* Did you push your latest updates to GitHub? It's easy to resubmit and realize that you didn't save your latest changes that were meant to fix an earlier problem.

**Is there a Python error message?**

Sometimes you will encounter Traceback Errors. These are your friend! You can search the web for the error message. Usually you want to grab the last line of the error message and paste that into your search bar with "python" at the beginning. Chances are very high that someone else on the internet has seen your error before.

**Infinite Loop**

If you accidentally created an infinite loop in your program, Gradescope will give you an error message about a “timeout”. You may think you didn't create an infinite loop, but sometimes one test will work fine, but another, that uses a slightly different path through your code, leads to an infinite loop. The Gradescope runs all tests when you submit, not just the visible test, so it's possible that one test case created an infinite loop.

**"The autograder failed to execute correctly. Please ensure that your submission is valid. Contact your course staff for help in debugging this issue. Make sure to include a link to this page so that they can help you most effectively."**

This almost always means that you included test code after your function and class definitions when only definitions were required.  Removing or commenting out the test code should fix this.

# Reference sheet for some common Python error messages

## File Contents

### **General Autograder Errors**

* Traceback - incorrect file name
* Traceback - end of file error
* Autograder failed to execute

### **Output Formatting Errors**

* List index out of range / Output doesn’t match
* Regex didn’t match: ‘<expected value>’ not found in ‘<found value>’
* ‘<your output>’ != ‘<expected output>’

### **Incorrect Value Errors**

* Unsupported operand type(s) for -: ‘<a type>’ and ‘<another type>’
* <an integer> != <another integer>
* <a float> != <another float> within 7 places
* Lists differ: <a list> != <another list>

### **Execution Errors**

* Name ‘<variable name>’ is not defined
* ‘<object name>’ has no attribute ‘<attribute name>’

## General Autograder Errors

### **Traceback - incorrect file name**

Test Failed: Failed to import test module: tests  
Traceback (most recent call last):  
  File "/usr/lib/python3.6/unittest/loader.py", line 428, in \_find\_test\_path  
module = self.\_get\_module\_from\_name(name)  
  File "/usr/lib/python3.6/unittest/loader.py", line 369, in \_get\_module\_from\_name  
\_\_import\_\_(name)  
  File "/autograder/source/tests/tests.py", line 3, in <module>  
from example\_file import example\_function  
ModuleNotFoundError: No module named ‘example\_file’

In this case, the student has submitted a file with the name ‘example\_file..py’. Because the auto-grader system is programmed to expect files with very specific names (in this case, ‘example\_file.py’, with only one dot), it raised this error message upon submission.

This error message might look like total gibberish at first, but it’s actually full of invaluable information for finding out what’s wrong with our code - we just have to know how to read it.

The first line - **‘Test Failed: Failed to import test module: tests’** - tells us that the program is failing to import the ‘tests’ module, which is just like what happens in Python when you import an outside library (also known as a module) into a file, but there is an error retrieving that library. This can happen for a number of reasons, so we need to keep reading.

The next part is what we call a **traceback**, which is what Python uses in an error message to direct the user to exactly where the error occurred. Everything indented is part of the traceback. Notice that it lists file paths like “/usr/lib/python3.6/unittest/loader.py” to indicate which file the error is coming from, and then even line numbers and python code from within that file to direct the user more precisely to where the error is coming from. Also note that in this situation, the error is occurring *inside the auto-grader system*, so these file names will look unfamiliar to the user, and are ultimately not very helpful in this situation. For this reason, we need to keep reading.

Finally, we see something useful in the last two lines of the error message:

* **“from example\_file import example\_function”** - This, being the last line of the traceback, denotes exactly where in the auto-grader’s code the error occurred. It is trying to import the student’s submitted file, and in this case is looking for the example function, from the example\_file.py file.
* **“ModuleNotFoundError: No module named 'example\_file'” - The final line of the error message tells us that the autograder cannot find what it’s looking for, indicating to the user that either the wrong file was submitted, the file was named improperly, or the file is missing.**

For more information on Tracebacks, see [thisLinks to an external site.](https://realpython.com/python-traceback/" \t "_blank) article.

### **Traceback – incorrect function name**

Test Failed: Failed to import test module: tests  
Traceback (most recent call last):  
  File "/usr/lib/python3.6/unittest/loader.py", line 428, in \_find\_test\_path  
module = self.\_get\_module\_from\_name(name)  
  File "/usr/lib/python3.6/unittest/loader.py", line 369, in \_get\_module\_from\_name  
\_\_import\_\_(name)  
  File "/autograder/source/tests/tests.py", line 3, in <module>  
from example\_file import example\_function  
ImportError: cannot import name ‘example\_file’

This is a similar error where the student has submitted a file with a function named ‘examplefile’. Because the auto-grader system is programmed to expect files with very specific names (in this case, ‘example\_file’, with an underscore), it raised this error message upon submission.

In the last two lines of the error message, we see:

* **“from example\_file import example\_function”** - This, being the last line of the traceback, denotes exactly where in the auto-grader’s code the error occurred. It is trying to import the student’s submitted file, and in this case is looking for the example function, from the example\_file.py file.
* **“ImportError: cannot import name 'example\_file'” - The final line of the error message tells us that the autograder cannot find what it’s looking for, indicating to the user that either the function was named wrong, or some necessary capitalization is either present or absent.**

### **Traceback – end of file error**

Test Failed: Failed to import test module: tests  
Traceback (most recent call last):  
  File "/usr/lib/python3.6/unittest/loader.py", line 428, in \_find\_test\_path  
= self.\_get\_module\_from\_name(name)  
  File "/usr/lib/python3.6/unittest/loader.py", line 369, in \_get\_module\_from\_name  
\_\_import\_\_(name)  
  File "/autograder/source/tests/tests.py", line 3, in <module>  
from example\_file import example\_function  
  File "/autograder/source/example\_file.py", line 12, in <module>  
val = input()  
EOFError: EOF when reading a line

This is another example of where leaving some testing code in the file is causing issues.

You will notice this error provides a traceback, which is a break-down of where in the code the issue is stemming from. It tells us “most recent call last”, which means that probably the most valuable information is going to be from the last section of the traceback.

The last part of the traceback says:

File “/autograder/source/example\_file.py”, line 12, in <module>  
val = input()

This is telling us that the issue is occurring in the file named “example\_file.py”, at line 12, and then it shows what is at that line in the file, “val = input()”.

Indeed, line 12 of this submission is “val = input()”, and it is the cause of the error. In this case, the submission has testing code that gets user input via input() and then uses that input to call a function for testing. When the autograder gets to that input line, it is expecting the file to be finished, but instead it is instructed to read input. This creates the EOF (end of file) error that is displayed.

To resolve this error, the testing code should either be removed or commented out, and the file should be resubmitted.

### **Autograder failed to execute**

Most common in assignments 4 - 10

The autograder failed to execute correctly. Please ensure that your submission is valid. Contact your course staff for help in debugging this issue. Make sure to include a link to this page so that they can help you most effectively.

The most common reason this error occurs is because students leave testing code in their submitted files, especially function calls. Before you submit your files, always make sure to remove or comment out the code you used to test your functions.

For example, starting in week 4, your submissions will be in the form of functions. The files for these submissions should include the function definitions(s) and code for the logic of the functions(s), but it should not include any testing code.

During development, you may (and should) be testing the functions by calling them in the file. This would look something like:

# function definition and logic  
def example\_function(some\_parameter):  
    new\_value = some\_paramter \* 2  
    return new\_value  
  
# testing code  
result = example\_function(4)  
print(result)

Leaving the “testing code” section in the file will result in the autograder error above. To resolve this error, the testing code should be removed, and the file should be resubmitted. Alternatively, you can comment out the tested code and the autograder will ignore it, like this:

# testing code  
# result = example\_function(4)  
# print(result)

## Output Formatting Errors

These errors present themselves in many ways. Sometimes it is a generic message like “list index out of range”, while other times the error will provide you more information about what is going wrong with the formatting. In the latter, it will show you what your file provided vs what the autograder was expecting. These types of error messages can be more helpful in directing you to where the issue is occurring.

In many of the following examples, we will be using generic example values, but the concepts of how to interpret the errors can be applied to many situations and values.

### **List index out of range / Output doesn’t match**

Most common in assignments 1 - 3

Test Failed: list index out of range

Test Failed: your output doesn’t quite match what I was expecting. Try checking your output against the expected string in the README file. Even spaces and newlines need to match, or I won’t get that it’s the same as the answer.

If you get either of the above errors from Gradescope, it usually means that there is an issue with the formatting of your code’s output. This can be from extra spaces, missing spaces, extra newlines, missing newlines, spelling errors, etc. This can also occur from misnamed files.

Why does it give you this error? Gradescope attempts to take the results of your program and look for an answer that matches what is in the README file. It does this by "splitting" the results so that it can compare the output string to strings that we provided for it. Even a very small difference like a space or a semicolon instead of a colon will trip the autograder up.

### **Regex didn’t match: ‘expected value’ not found in ‘found value’**

Test Failed: Regex didn’t match: ‘(13\\.2)’ not found in ‘ ‘

In this case, the autograder is expecting the solution of the program to be printed on its own line at the end of the file, but the submitted file has the solution printing on the same line as some other output message. These details need to be **exactly** as is shown in the README of an assignment.

What this error is showing is that when the test looked for that final line it found ‘ ‘ (nothing) instead of finding the solution to the assignment problem, ‘(13\\.2)’. Now, (13\\.2) may not look like the answer you are expecting, but the extra slashes in there have to do with the autograder using something called regular expressions (regex) to search the output.

Why does it give you this error? In these tests, Gradescope is using regular expressions to search for the correct answer. Regex is a sequence of characters that define a search pattern, and the autograder is using them to “find” the answer in the strings your program is outputting. In this case, the regex is broken up into sections based on newlines, and since the numeric part of the answer is supposed to be on its own line, it has its own regex test. However, since that line does not exist in the example solution, the regex test cannot “find” the answer, hence the error message.

### **‘<your output>’ != ‘<expected output>’**

Test Failed: ‘Num Dogs:  2\nNum Cats:  3\n’ != ‘Num Dogs: 2\nNum Cats: 3\n’  
- Num Dogs:  2  
? -  
+ Num Dogs: 2  
- Num Cats:  3  
? -  
+ Num Cats: 3

In this case, the first line of the error is showing the output of the submitted file vs the expected output of the autograder test, and it is showing that they are not equal (!=). Those two strings do look very similar (i.e. same categories and same values), but if you look closely, the number of spaces between the colons and numbers are different.

The rest of the error output is an attempt to show you what is specifically extra and/or missing. The first line, “- Num Dogs:  2” is saying the answer *should not* include this line (think “minus” this part). Then the next two lines, “?   -“ and “+ Num Dogs: 2” are saying the answer *should* include “Num Dogs: 2” (think “plus” this part). Again, these strings do look almost identical, but upon close inspection you will see again that the number of spaces is off.

Why does it give you this error? In the expected solution, there is a single space between the colon and the values, but in the submitted solution, there are two spaces between the colon and values. The most common reason this occurs is when students attempt to pass multiple arguments to the print() function (e.g. print(“Num Dogs: “, 2)). When using print() in this way, the output will automatically print a space between the items passed to print. Therefore, you should not include a space after the “Num Dogs:” to get the desired spacing.

## Incorrect Value Errors

These are going to be test failures that indicate there is something going wrong with your calculations or how they are being presented to the autograder.

### **Unsupported operand type(s) for -: ‘<a type>’ and ‘<another type>’**

Test Failed: unsupported operand type(s) for -: ‘int’ and ‘NoneType’

This failed test is saying that there is a TypeError in the testing. TypeErrors occur when an operation or function is applied to an object of an inappropriate type (e.g. if you try to add a string and an integer together).

In this case, the submission is returning a print statement instead of an integer (i.e. return print(answer)).

But, print() is a function call that does not return a value, so calling print() as a return results in returning nothing. In Python, this is represented with ‘None’ and is where the ‘NoneType’ in the error comes from.

Here, the autograder is expecting the function to return an integer value and it is testing that return against the correct integer to see if they are equal. Since the print() has returned None, the function returns None, and the test ends up comparing None to an integer. These two different types cannot be compared, so this results in the TypeError.

To resolve this error, the function should only return the integer. It should not use print() at all.

### **<an integer> != <another integer>**

Pass start (x, y, z); method\_a 1; method\_b 2

Test Failed: 3 != 5

This failed test is saying that the autograder has retrieved a value from your submission (an integer) and it does not equal the expected correct value (another integer). In the example, it is expecting to find 5 for a value, but it is finding 3 instead.

This indicates that there is an issue with the calculations being made somewhere in the submission. This error could be reflected in a return value of a method or function, or in a value being stored as a data member for a class.

These types of errors will also show what was being tested in the field above the error, beginning  with “Pass” (think, this is what is being passed to your submission for testing). This information can be very helpful in determining what is going wrong.

In this case, the submission being tested is a class and the test is shown as “Pass start (x, y, z); method\_a 1; method\_b 2”. We can break this information down and use it to track down where the error may be occurring. The first part (“start (x, y, z)”) is telling us that the class is being initialized with the values x, y, and z. Then the next part (“method\_a 1”) is telling us the class method, method\_a, is being called and passed a value of 1. Finally, the last part (“method\_b 2”) is telling us the class method, method\_b, is being called and passed a value of 2.

Now you know which parts of the submission were tested, and you can narrow down the areas to look for errors - the init method, method\_a, and method\_b. The error could be an issue with how the class is initialized, an error in calculations made in method\_a or method\_b, an error in how data is stored or returned in method\_a or method\_b, etc.

At this point, you can do the calculations yourself (with the values shown for this test) to see what the return values and/or data member values should be at each step and compare them to a run of your code with the same values. This should help you narrow down where the incorrect calculations have occurred.

### **<a float> != <another float> within 7 places**

Pass [1, 2, 3, 4, 5]

Test Failed: 1.0 != 2.0 within 7 places

This failed test is saying that the autograder has retrieved a value from your submission (a float) and it does not equal the expected correct value (another float). In the example, it is expecting to find 2.0 for a value, but it is finding 1.0 instead.

This is very similar to the previous example, except now the error includes the phrase “within 7 places”. This is included here because this submission is dealing with floating point values, and the tests to compare those values consider answers with 7 decimals places to be correct. This has to do with the way computers handle the precision of floating-point values in memory, and the extra places are to account for minor rounding differences in different runs of the program.

As with the previous example, the test case is provided to you so that you can use it to help figure out what is going wrong. The test is shown as “Pass [1, 2, 3, 4, 5]”. This means the submission is being tested with the values in this list in some fashion. Now you can use those values to step through your submission and try to narrow down where the calculations are going wrong.

### **Lists differ: <a list> != <another list>**

Pass [3, 4, 5, 6]

Test Failed: Lists differ: [9, 16, 25, 36] != [6, 8, 10, 12]  
First differing element 0:  
9  
6  
- [9, 16, 25, 36]  
+ [6, 8, 10, 12]

This failed test is saying that the autograder has retrieved a list from your submission and it does not equal the expected correct list. In the example, it is expecting to find [6, 8, 10, 12] for the result list, but it is finding [9, 16, 25, 36] instead.

The first line of the error message shows the entire found list and the entire expected list. The next part, “First differing element 0:” shows the first instance in the list that was found to be incorrect. In this case, the first difference occurs at index 0, where the test is expecting 6, but it found 9 instead.

The final part of the message is showing you what it found that does not belong and what it did not find that should be there. “- [9, 16, 25, 36]” is the part it found that does not belong (think “minus” this part). “+ [6, 8, 10, 12]” is the part it did not find that should be there (think “plus” this part).

Like in the other examples from this section, the test case is shown to you as “Pass [3, 4, 5, 6]”. This tells you that the submission is being tested with the values in this list in some fashion. Now you can use those values to step through your submission and try to narrow down where the calculations are going wrong.

## Execution Errors

### **Name ‘<variable name>’ is not defined**

Test Failed: name ‘val’ is not defined

This failed test is saying that there is a variable being used in this submission (named ‘val’) that is not defined. That means there is no value assigned to ‘val’ before it is used for the first time. Since there is no value stored in ‘val’, the program does not know what to do with it and a NameError occurs.

To resolve an issue like this, you should make sure your variables have a value assigned to them before they are used in a calculation.

### **‘<object name>’ object has no attribute ‘<attribute name>’**

Test Failed: ‘Item’ object has no attribute ‘\_price’

This failed test is saying that the autograder attempted to look up a data member  ‘\_price’ in a class object ‘Item’, but that field does not exist for the class Item objects. This is either because the class used a different spelling than what the autograder was expecting, or because the class does not have a price attribute at all.

To resolve an issue like this, make sure the data members for your class match what is expected in the README. Sometimes your class data members will be required to have specific names.

# Academic Integrity - Overview

## Overview to the Module

Welcome to the Student Academic Integrity Module! In this section, you will find the following pages:

### **Academic Integrity at OSU**

What is academic integrity and why is it important? What is classified as 'prohibited academic misconduct'? The [**Academic Integrity at OSU**](https://canvas.oregonstate.edu/courses/1915078/pages/academic-integrity-at-osu) page provides some brief answers to these questions, along with a link to the Code of Student Conduct.

### **Academic Integrity Policy Page**

The [**Academic Integrity Policy Page**](https://canvas.oregonstate.edu/courses/1915078/pages/academic-integrity-policy-for-students) is worth reading as it will change depending on the kind of course you are taking. Expectations for Student Conduct with regards to Academic Integrity, Code Sharing/Reusing policies, Discussion Netiquette, and the Handling of Plagiarism Offenses are among the topics covered.

### **Academic Integrity DOs and DON'Ts for CS 162**

The [Academic Integrity DOs and DON'Ts for CS 162](https://canvas.oregonstate.edu/courses/1915078/pages/academic-integrity-dos-and-donts-for-cs-162) page is a quick reference to what is permitted or not permitted in this course as far as academic integrity.

# Academic Integrity at OSU

## Understanding Academic Misconduct

Please review the following video and respond to the embedded questions on academic misconduct policies, procedures, and best practices.

## Why is academic integrity important?

Academic, research and scholarly integrity are of the utmost importance to Oregon State University, as an international public research university and the state’s Land Grant university. Faculty and students share responsibility in preserving the integrity of the academic experience at Oregon State. Academic misconduct damages the educational experience and ultimately hurts many parties, including faculty, other students, and the value of OSU credits and degrees. Students and faculty are encouraged to understand the expectations outlined by the [**Code of Student Conduct**](https://studentlife.oregonstate.edu/sites/studentlife.oregonstate.edu/files/student-conduct-community-standards/Code/code_of_conduct_comp.pdf)and professional standards of academic colleges and programs, to report suspected incidents of academic misconduct, and to hold each other to a high standard when it comes to the integrity of academic work.

Please utilize the information in the Student Code of Conduct located on the universities Student Conduct & Community Standards [**website**](https://studentlife.oregonstate.edu/studentconduct/academicmisconduct)to enhance your understanding of Academic Misconduct and the Academic Integrity Process at Oregon State. Here you will also find [**instructions**](https://studentlife.oregonstate.edu/studentconduct/academic-misconduct-students#am1)on what to do should you be involved in an academic integrity violation.

It is also important that you understand the guidelines for your course on what is permissible and what is a violation by reviewing the [**Academic Integrity Policy**](https://canvas.oregonstate.edu/courses/1915078/pages/academic-integrity-policy-for-students) posted in your course. Not all professors will have the same permissions or requirements.

## Prohibited Academic Misconduct

The Code of Student Conduct prohibits [**Academic Misconduct**](https://studentlife.oregonstate.edu/studentconduct/student-info)and defines it as:

###### **"Any action that misrepresents a student or group’s work, knowledge, or achievement, provides a potential or actual inequitable advantage, or compromises the integrity of the educational process."**

The Code further classifies and describes academic Misconduct examples to support understanding of what can be included in this definition.

Prohibited behaviors include, but are not limited to doing or attempting the following actions:

1. **Cheating.**  Unauthorized assistance, or access to or use of unauthorized materials, information, tools, or study aids. Examples include, but are not limited to, unauthorized collaboration or copying on a test or assignment, using prohibited materials and texts, unapproved use of cell phones, the internet, or other electronic devices, etc.
2. **Plagiarism.**  Representing the words or ideas of another person or presenting someone else's words, data expressed ideas, or artistry as one's own. Examples include, but are not limited to, presenting someone else's opinions and theories as one's own, using another person's work or words (including unpublished material) without appropriate source documentation or citation, working jointly on a project and then submitting it as one's own, etc.
3. **Falsification.**  Fabrication or invention of any information. Examples include, but are not limited to, falsifying research, inventing or falsely altering data, citing fictitious references, falsely recording or reporting attendance, hours, or engagement in activities such as internships, externships, field experiences, clinical activities, etc.
4. **Assisting.**  Any action that helps another engage in academic misconduct. Examples include, but are not limited to, providing materials or assistance without approval, altering someone's work, grades, or academic records, taking a test/doing an assignment for someone else, compelling acquisition, selling, bribing, paying, or accepting payment for academic work or assistance that contributes to academic misconduct, etc.
5. **Tampering.**  Interfering with an instructor’s evaluation of work by altering materials or documents, tampering with evaluation tools, or other means of interfering.
6. **Multiple submissions of work.**  Using or submitting work completed for another or previous class or requirement, without appropriate disclosure, citation, and instructor approval.
7. **Unauthorized recording and use.**  Recording and/or dissemination of instructional content without the express permission of the instructor(s), or an approved accommodation coordinated via Disability Access Services.

Academic Integrity Policy for Students

Expectations for Student Conduct with regards to Academic Integrity

Read through the entire document for course expectations and consequences. Students in academic studies are expected to demonstrate their own knowledge and capabilities. Work that is not created by yourself will be considered plagiarized material and may result in a failed submission and may result in administrative action if submitted.

* You may openly discuss the presented learning materials and participation category materials at any time with any party as long as they explicitly know that it is for an academic assignment.
* You may openly discuss the meaning of assignments, general approaches, and strategies with other students in the course; you may do this even before the grading date of the assignment has passed.

In this online program, we require collaboration and building upon the work of others in an honest, respectful way. This means that instead of strictly disallowing working with others (or their work), we encourage a collaboration process. We will primarily be using your assignments, collaboration, and projects to gauge your success in this course. Graders often check Chegg, Course Hero, and other similar sites. If you have posted an answer or have posted a submission from a site like Chegg, you will not receive credit! We expect you to do your own work.

If you are found in violation of any of the above policies, whether you are the giver or the receiver of non-cited help, you may be given a zero on the assignment, failed from the course, or receive higher administrative action. The academic dishonesty charge will be documented and sent to your school’s dean and the Office of Student Conduct.

* [**EECS Code Sharing Policy webpage**](https://eecs.oregonstate.edu/online-cs-students/current-students/class-resources/policies)
* [**OSU Student Academic Misconduct information**](https://studentlife.oregonstate.edu/studentconduct/academic-misconduct-students)
* [**Student code of conduct**](https://studentlife.oregonstate.edu/sites/studentlife.oregonstate.edu/files/student-conduct-community-standards/Code/code_of_conduct_comp.pdf)

How Academic Integrity Offenses will be handled

* There is no statute of limitation on checking for cheating, reports can be made post-term if cheating is suspected. or your materials are found shared on study-type websites.
* A note/comment will be made on your assignment similar to “…Your assignment is under review for a potential case of academic misconduct. The College of Engineering Hearing Officer (CHO) will be in touch soon regarding the incident, and any inquiries should go to them. Your grade will remain unassigned until the case has been reviewed and a determination has been made by the College Hearing Officer. For more information on the process, please check the [**Academic Misconduct Student Resource Page**](https://studentlife.oregonstate.edu/studentconduct/academic-misconduct-students)” Grade if found Not Responsible: X/Y (eg 0/30pts)
* Depending on term timing, you will be given a grade of I/X where X is the letter grade you would receive if you were found responsible for academic misconduct.
* If Student A copied Student B, you both will have reports filed. Instructors do not bear the burden of determining who cheated from whom, the CHO will attempt to sort out the details.
* Note: The 1st confirmed case of Academic Misconduct results in counseling and an assignment penalty. Any later cases can result in removal from the University.

Discussion Netiquette

For the group assignments, participating in the group discussions is required. You are expected to productively participate with your classmates. Your posts should facilitate meaningful discussion of the topic or assignment posted. It is expected, and often set up, in a manner where there will be opposing opinions. A discussion where we all agree can be quite boring and dead-ended. Challenging an idea or opinion is expected. The manner in which that happens must stay professional, courteous, and respectful. Posts that are enticing a fight, violent, name-calling, etc. will be deleted. Multiple violations will result in the removal of the opportunity to participate in discussion work. To maintain good netiquette, keep the following tips in mind:

1. No yelling (ALL CAP RESPONSES or video responses, \*with the exception of sharing video resources)
2. Avoid the use of sarcastic comments or mocking another individual
3. Sense of humor can be subjective, especially in print, stay aware of your tone, even in a joke.
4. Check your grammar and spelling
5. Be respectful in both tone and content
6. Read the full instructions before posting a response
7. Think and review before hitting post. Energized posts can often come across as rude and/or unprofessional.
8. Maintain professionalism
9. Be kind to each other
10. Stay on topic, try not to steer the conversation off the topic posted.
11. We all make mistakes, how we handle mistakes of our own and others speaks volumes. Stay respectful & be forgiving. We are all here to learn together.

Academic Integrity DOs and DON'Ts for CS 162

* Do feel free to discuss course content with each other, even including general discussion  
  of homework assignments and how to fix specific issues.
* Do feel free to post small snippets of non-working assignment code to Ed Discussion or the official course Microsoft Teams. If you have trouble narrowing the problem down to a small snippet, just describe the problem as well as you can and/or ask your ULA for help.
* Do feel free to ask conceptual questions related to assignments on Ed Discussion or the official course Microsoft Teams.
* Do feel free to post code for the exploration exercises on Ed Discussion or the official course Microsoft Teams.
* **Don’t** post any quiz questions or answers in any form.
* **Don’t** make your code publicly accessible, for example by posting it on Stack Overflow, Chegg, a public GitHub repository, etc.  Any GitHub repository you post assignment code on must be private. The Portfolio assignment can be made public after your final course grades have been posted to the Registrar.
* **Don’t** use AI such as GPT or GitHub Copilot to generate code for assignments.
* Do feel free to share and compare your assignment code with other students on Ed Discussion or the official course Microsoft Teams starting the Sunday after the assignment was due. Keep in mind that there's not just one correct way to write a program - there's almost always a variety of correct approaches.  As you compare with others' code, try to notice what seem to be advantages or disadvantages of a particular approach, and don't be afraid to ask questions about why someone made a certain design decision.
* **Don’t** copy someone else’s code, in whole or in part, whether someone else in the course, a previous student in the course, or code you found somewhere online. It’s easy for us to detect plagiarism, it will get you a zero on the assignment, and it goes on your academic record. This also includes detailed plans for code, such as pseudocode.