Oliver Rothe
CS162 fall 2023

Create a program that designs, uses, and demonstrates an understanding of the following:

1. variables, conditionals, loops, and collections,
2. code organization (formatting, identifiers, placement of definitions),
3. code decomposition (functions, classes, methods, and modules),
4. an understanding of design (including hierarchy of aggregate objects and an inheritance tree),
5. an understanding of testing (test your methods and attributes, maybe have a whole automated example!),
6. user IO, file IO, and input validation,
7. recursion,
8. GUI components and event driven programming,
9. exceptions,
10. Inheritance.

Create a document that shows that your program completes the above topics and a screenshot of some source code that makes that example work.

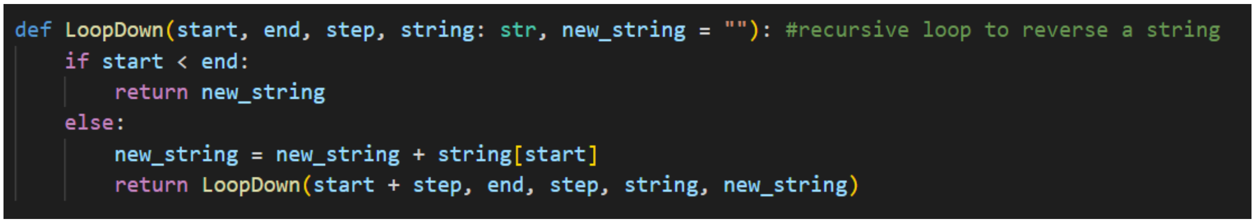
I couldnt think of any really good ‘big project’ ideas so I made a couple smaller programs to demonstrate my understanding of the material.

Program files include:

file_in.txt
file_reverser.py
ST_console_Main.py
ST_classes.py
ST_classes_testing.py

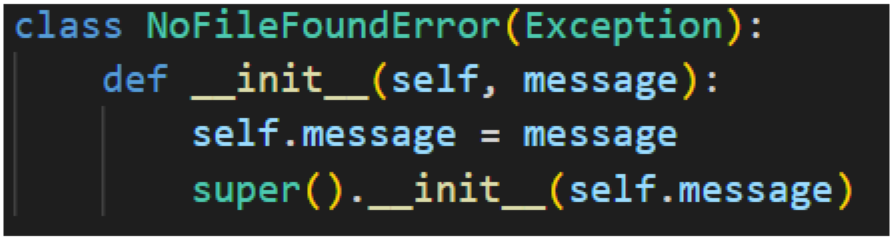
Criteria met examples:

Recursion: file_reverser.py in lines 16-21



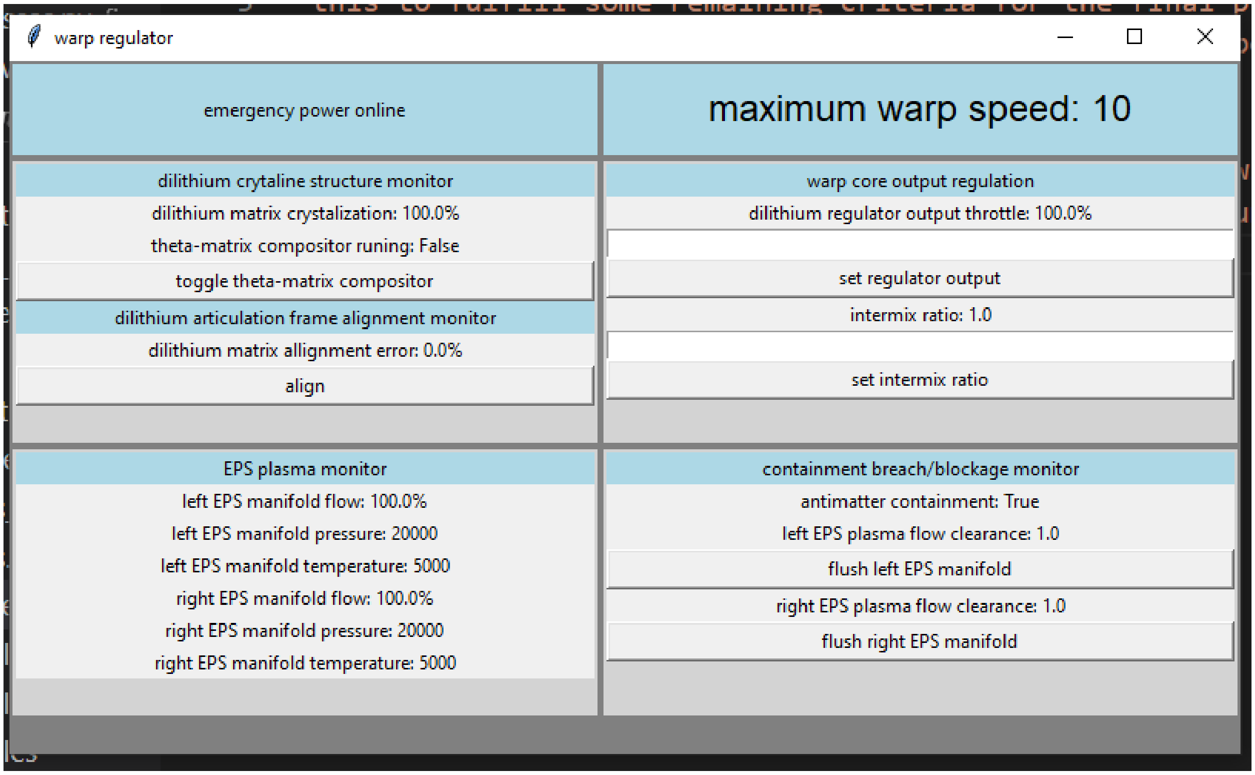
```
def LoopDown(start, end, step, string: str, new_string = ""): #recursive loop to reverse a string
    if start < end:
        return new_string
    else:
        new_string = new_string + string[start]
        return LoopDown(start + step, end, step, string, new_string)
```

Exceptions: file_reverser.py in lines 11-14, 31



```
class NoFileFoundError(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)
```

GUI components: all of ST_classes.py is designed to be a UI, runs in ST_console_Main.py.



warp regulator

emergency power online

maximum warp speed: 10

dilithium crytaline structure monitor

dilithium matrix crystalization: 100.0%

theta-matrix compositor runing: False

toggle theta-matrix compositor

dilithium articulation frame alignment monitor

dilithium matrix allignment error: 0.0%

align

warp core output regulation

dilithium regulator output throttle: 100.0%

set regulator output

intermix ratio: 1.0

set intermix ratio

EPS plasma monitor

left EPS manifold flow: 100.0%

left EPS manifold pressure: 20000

left EPS manifold temperature: 5000

right EPS manifold flow: 100.0%

right EPS manifold pressure: 20000

right EPS manifold temperature: 5000

containment breach/blockage monitor

antimatter containment: True

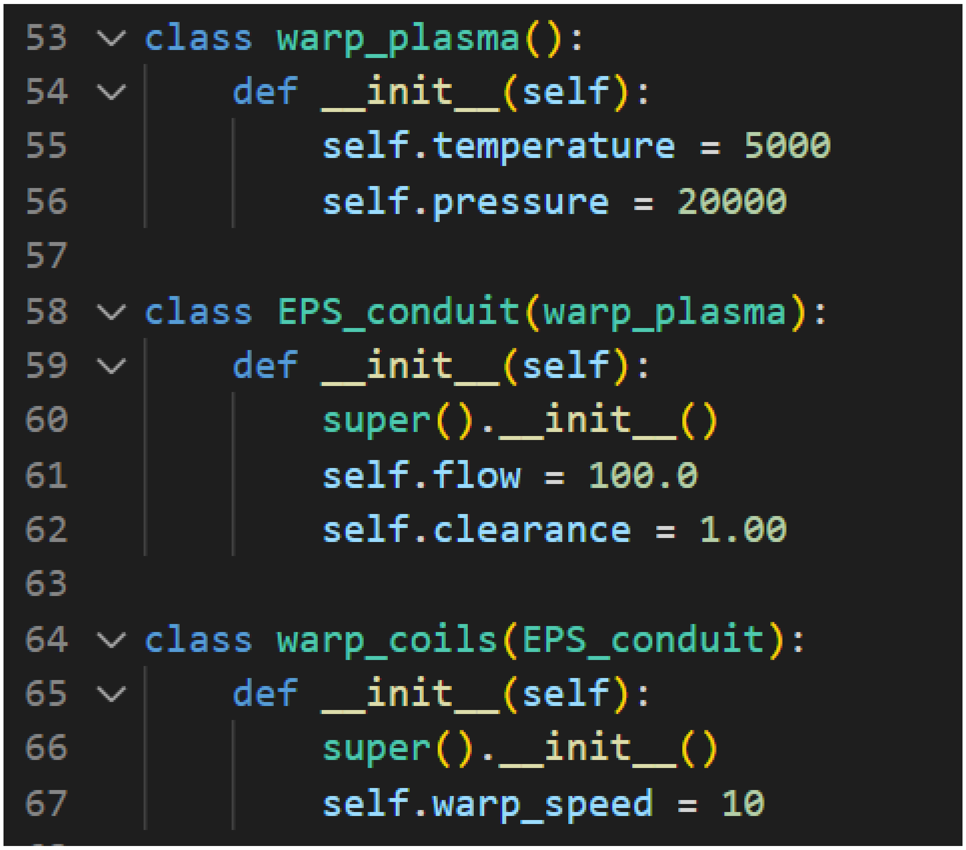
left EPS plasma flow clearance: 1.0

flush left EPS manifold

right EPS plasma flow clearance: 1.0

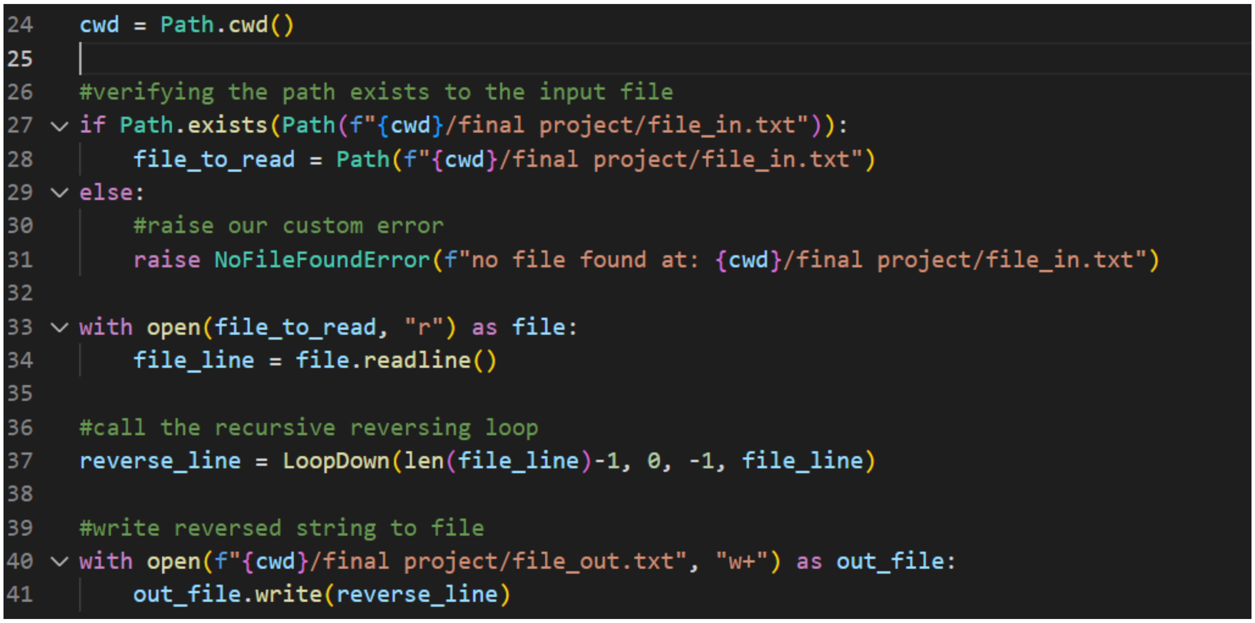
flush right EPS manifold

Inheritance: ST_classes.py in lines 53-67, also elsewhere in file are other examples of Inheritance



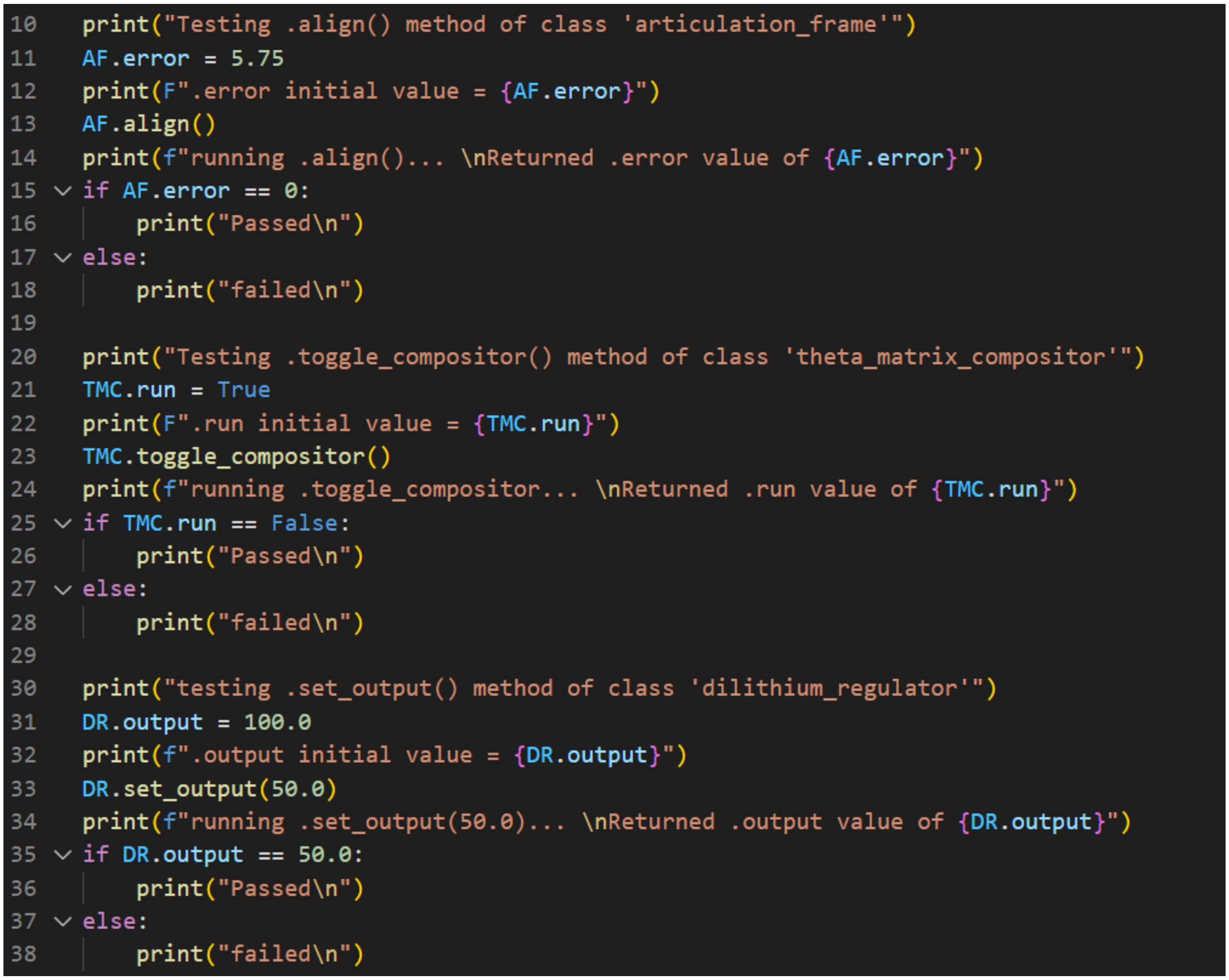
```python
class warp_plasma():
    def __init__(self):
        self.temperature = 5000
        self.pressure = 20000

class EPS_conduit(warp_plasma):
    def __init__(self):
        super().__init__()
        self.flow = 100.0
        self.clearance = 1.00

class warp_coils(EPS_conduit):
    def __init__(self):
        super().__init__()
        self.warp_speed = 10
```

file IO: file_reverser.py in line 24-41



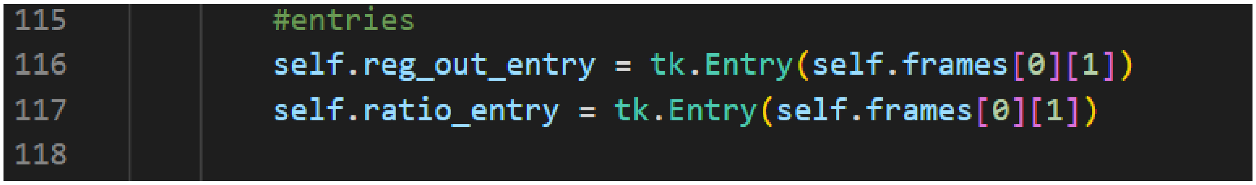
```python
24  cwd = Path.cwd()
25  
26  #verifying the path exists to the input file
27  if Path.exists(Path(f"{cwd}/final project/file_in.txt")):
28      file_to_read = Path(f"{cwd}/final project/file_in.txt")
29  else:
30      #raise our custom error
31      raise NoFileFoundError(f"no file found at: {cwd}/final project/file_in.txt")
32  
33  with open(file_to_read, "r") as file:
34      file_line = file.readline()
35  
36  #call the recursive reversing loop
37  reverse_line = LoopDown(len(file_line)-1, 0, -1, file_line)
38  
39  #write reversed string to file
40  with open(f"{cwd}/final project/file_out.txt", "w+") as out_file:
41      out_file.write(reverse_line)
```

Testing: Still could not get PyTest to work so I made some automated tests manually in file ST_classes_testing.py

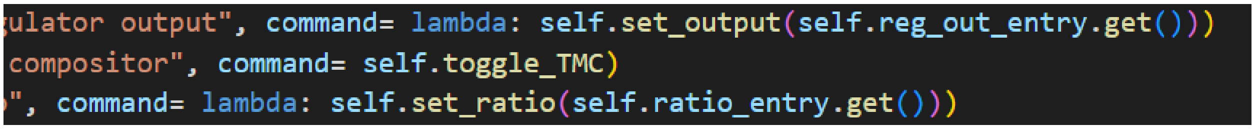


```python
print("Testing .align() method of class 'articulation_frame'")
AF.error = 5.75
print(F".error initial value = {AF.error}")
AF.align()
print(f"running .align()... \nReturned .error value of {AF.error}")
if AF.error == 0:
    print("Passed\n")
else:
    print("failed\n")

print("Testing .toggle_compositor() method of class 'theta_matrix_compositor'")
TMC.run = True
print(F".run initial value = {TMC.run}")
TMC.toggle_compositor()
print(f"running .toggle_compositor... \nReturned .run value of {TMC.run}")
if TMC.run == False:
    print("Passed\n")
else:
    print("failed\n")

print("testing .set_output() method of class 'dilithium_regulator'")
DR.output = 100.0
print(f".output initial value = {DR.output}")
DR.set_output(50.0)
print(f"running .set_output(50.0)... \nReturned .output value of {DR.output}")
if DR.output == 50.0:
    print("Passed\n")
else:
    print("failed\n")
```

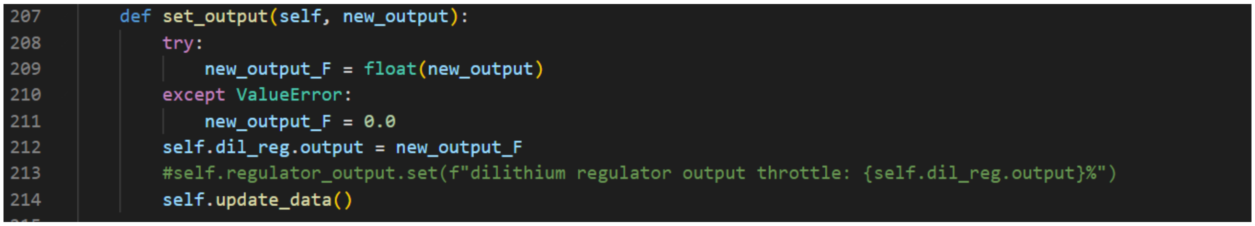
User I/O and input validation: ST_classes in lines 115-118,146-149, 207-214, 221-228. This is a little hard to show on a doc because the data is used in many places and is validated by simply setting it to 0 if it's not convertible to a float, and the user input is limited to button pushing and test entering in a box.



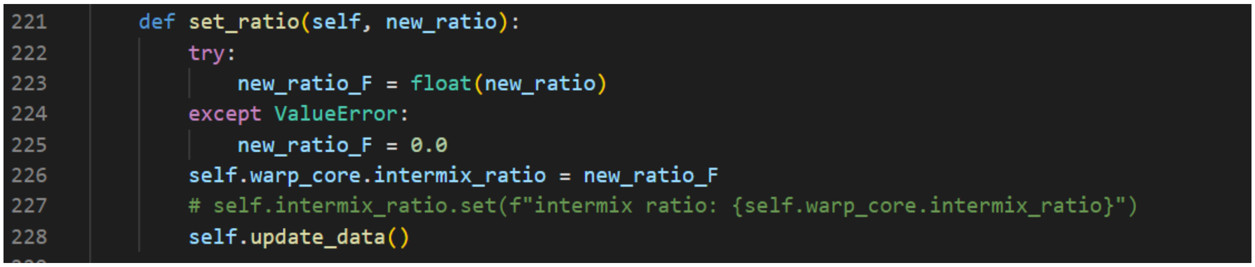
```python
115             #entries
116             self.reg_out_entry = tk.Entry(self.frames[0][1])
117             self.ratio_entry = tk.Entry(self.frames[0][1])
118
```



```python
ulator output", command= lambda: self.set_output(self.reg_out_entry.get()))
compositor", command= self.toggle_TMC)
", command= lambda: self.set_ratio(self.ratio_entry.get()))
```



```
    def set_output(self, new_output):
        try:
            new_output_F = float(new_output)
        except ValueError:
            new_output_F = 0.0
        self.dil_reg.output = new_output_F
        #self.regulator_output.set(f"dilithium regulator output throttle: {self.dil_reg.output}%")
        self.update_data()
```



```
    def set_ratio(self, new_ratio):
        try:
            new_ratio_F = float(new_ratio)
        except ValueError:
            new_ratio_F = 0.0
        self.warp_core.intermix_ratio = new_ratio_F
        # self.intermix_ratio.set(f"intermix ratio: {self.warp_core.intermix_ratio}")
        self.update_data()
```

variables, conditionals, loops, and collections,
code organization (formatting, identifiers, placement of definitions),
code decomposition (functions, classes, methods, and modules),
an understanding of design (including hierarchy of aggregate objects and an inheritance tree).

Can be found in all .py files, hard to give examples and pictures when it's everywhere and none stand out as especially good examples.

All programs have variables and conditionals. ST_classes.py uses an array to generate and place UI frames. File_reverser.py has a recursive loop. ST_classes if broken into many classes functions and methods, the aforementioned recursive loop in file_reverser.py is inherently a function by nature, definitions are places above code.