

Strength of Materials Problem Exercises

Curated by James Lord, Jake Grohs, ...

2024-01-05

Table of contents

Welcome to Demo Site

Welcome to this demonstration site of the Strength of Materials Open Problem Exercises companion to the Strength of Materials Open Textbook.

At this time, we are simply using this site as a demonstration and shell for our ongoing work. The intent is to demonstrate a more traditional static style problem exercise pack along with dynamic versions which allow students to quickly check answers and receive basic feedback, and/or to input their math in an interactive interface which will provide them with targeted feedback based on their attempted solution.

This work is still very much in progress and you may find bugs. We would welcome any input or feedback you have about this. Thanks!

Part I

List of Complete Problems

Problem 2.1

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

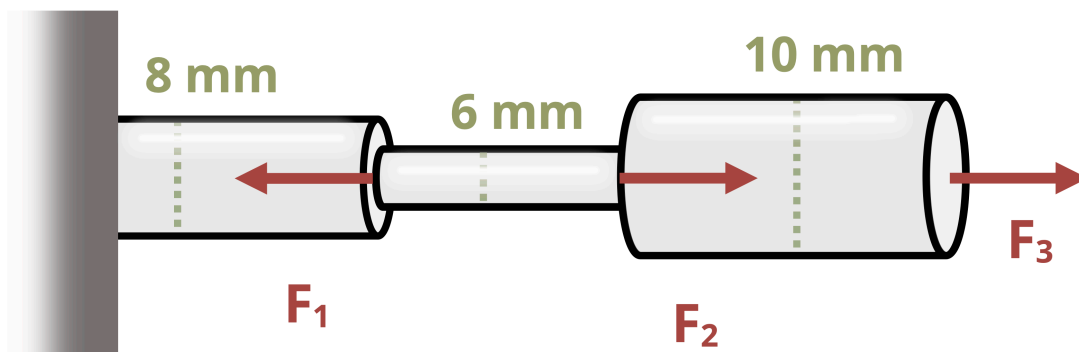


Figure 0.1: Figure 1: A series of solid circular bars are loaded with three loads

```
#!/ standalone: true
#!/ viewerHeight: 600
#!/ components: [viewer]

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
```



```

F2.set(random.randrange(10, 30, 1))
F3.set(F1()-F2())

@reactive.Effect
@reactive.event(input.submit)
def _():
    attempt_counter.set(attempt_counter() + 1) # Increment the attempt counter on each

    # Calculate the instructor's answer and determine if the user's answer is correct.
    instr= (F1()/(math.pi*(d2/(1000*2))**2))/10**6

    if math.isclose(float(input.answer()), instr, rel_tol=0.001):
        check = "*Correct*"
        correct_indicator = "JL"
    else:
        check = "*Not Correct.*"
        correct_indicator = "JG"

    # Generate random parts for the encoded attempt.
    random_start = generate_random_letters(4)
    random_middle = generate_random_letters(4)
    random_end = generate_random_letters(4)
    encoded_attempt = f"{random_start}{problem_ID}-{random_middle}{attempt_counter()}{correct_indicator}"

    # Store the most recent encoded attempt in a reactive value so it persists across sub
    session.encoded_attempt = reactive.Value(encoded_attempt)

    # Append the attempt data to the attempts list without the encoded attempt
    attempts.append(f"{datetime.now()}, {attempt_counter()}, {input.answer()}, {check}\n")

    # Show feedback to the user.
    feedback = ui.markdown(f"Your answer of {input.answer()} is {check}. For reference i
    m = ui.modal(
        feedback,
        title="Feedback",
        easy_close=True
    )
    ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():

```

```

# Start the CSV with the encoded attempt (without label)
final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else None
yield f"{final_encoded}\n\n"

# Write the header for the remaining CSV data once
yield "Timestamp,Attempt,Answer,Feedback\n"

# Write the attempts data, ensure that the header from the attempts list is not written
for attempt in attempts[1:]: # Skip the first element which is the header
    await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
    yield attempt

# App installation
app = App(app_ui, server)

```


Problem 2.2

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

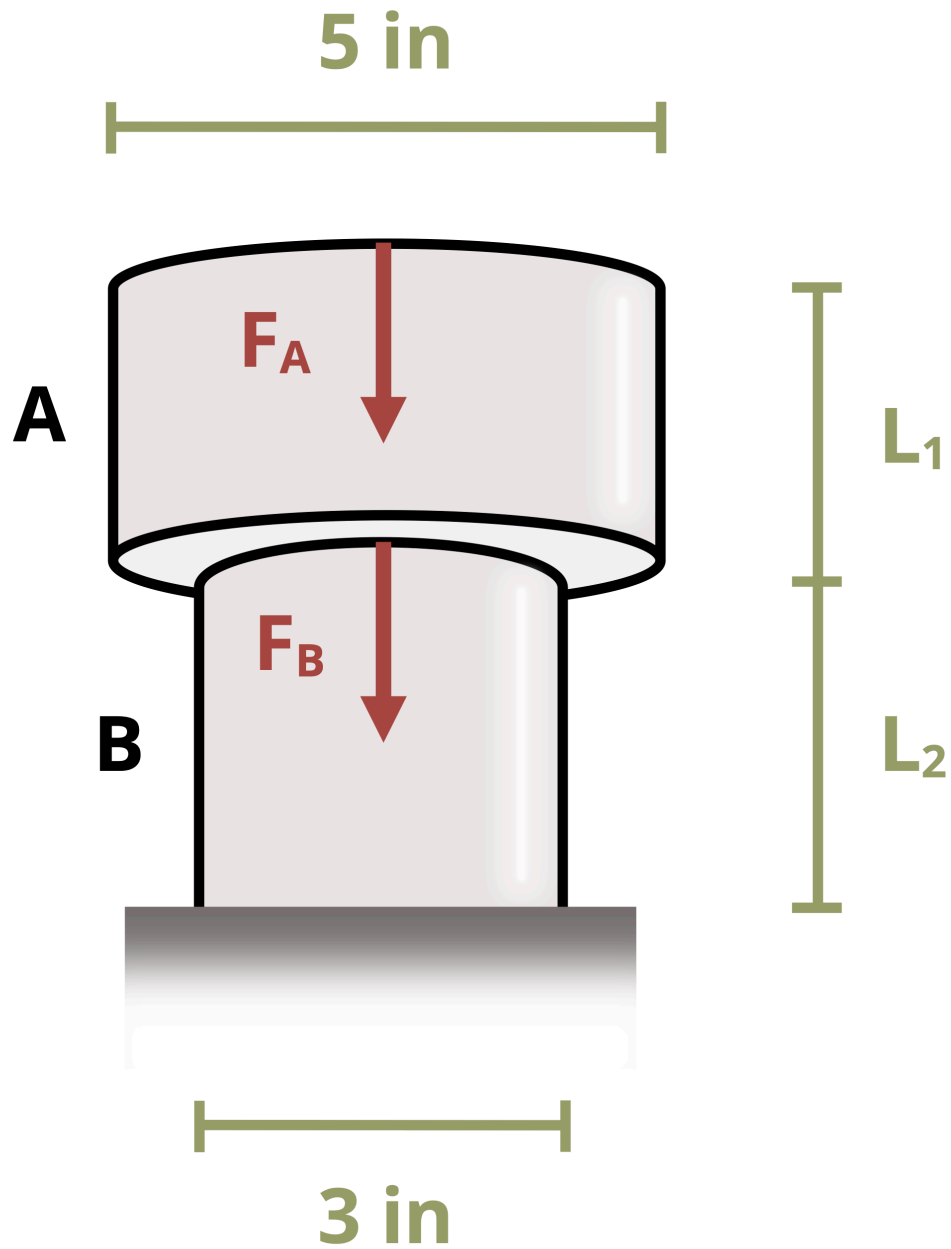


Figure 0.2: Figure 1: Two cylinders are stacked on top of each other.

```

#| standalone: true
#| viewerHeight: 600
#| components: [viewer]

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
from pathlib import Path

def generate_random_letters(length):
    # Generate a random string of letters of specified length
    return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))

problem_ID="139"
L1=reactive.Value("__")
L2=reactive.Value("__")
FA=reactive.Value("__")
FB=reactive.Value("__")
E = 30*10**6

attempts=["Timestamp,Attempt,Answer,Feedback\n"]

app_ui = ui.page_fluid(
    ui.markdown("**Please enter your ID number from your instructor and click to generate your problem**"),
    ui.input_text("ID","", placeholder="Enter ID Number Here"),
    ui.input_action_button("generate_problem", "Generate Problem", class_="btn-primary"),
    ui.markdown("**Problem Statement**"),
    ui.output_ui("ui_problem_statement"),
    ui.input_text("answer","Your Answer in units of inches", placeholder="Please enter your answer"),
    ui.input_action_button("submit", "Submit Answer", class_="btn-primary"),
    ui.download_button("download", "Download File to Submit", class_="btn-success"),
)

def server(input, output, session):
    # Initialize a counter for attempts
    attempt_counter = reactive.Value(0)

```

```

@output
@render.ui
def ui_problem_statement():
    return[ui.markdown(f"Two cylinders are stacked on top of one another and two forces s

@reactive.Effect
@reactive.event(input.generate_problem)
def randomize_vars():
    random.seed(input.ID())
    FA.set(random.randrange(300, 700, 10))
    FB.set(random.randrange(100, 300, 10))
    L1.set(random.randrange(2, 7, 1))
    L2.set(L1() * 1.3)

@reactive.Effect
@reactive.event(input.submit)
def _():
    attempt_counter.set(attempt_counter() + 1) # Increment the attempt counter on each s

    instr= (FA()*L1())/((math.pi*(5/2)**2)*E) + (FB()*L2())/((math.pi*(3/2)**2)*E)
    if math.isclose(float(input.answer()), instr, rel_tol=0.001):
        check = "*Correct*"
        correct_indicator = "JL"
    else:
        check = "*Not Correct.*"
        correct_indicator = "JG"

    # Generate random parts for the encoded attempt.
    random_start = generate_random_letters(4)
    random_middle = generate_random_letters(4)
    random_end = generate_random_letters(4)
    encoded_attempt = f"{random_start}{problem_ID}-{random_middle}{attempt_counter()}{co

    # Store the most recent encoded attempt in a reactive value so it persists across sul
    session.encoded_attempt = reactive.Value(encoded_attempt)

    # Append the attempt data to the attempts list without the encoded attempt
    attempts.append(f"{datetime.now()}, {attempt_counter()}, {input.answer()}, {check}\n

    # Show feedback to the user.
    feedback = ui.markdown(f"Your answer of {input.answer()} is {check}. For reference in

```

```

        m = ui.modal(
            feedback,
            title="Feedback",
            easy_close=True
        )
        ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():
    # Start the CSV with the encoded attempt (without label)
    final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else None
    yield f"{final_encoded}\n\n"

    # Write the header for the remaining CSV data once
    yield "Timestamp,Attempt,Answer,Feedback\n"

    # Write the attempts data, ensure that the header from the attempts list is not written
    for attempt in attempts[1:]: # Skip the first element which is the header
        await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
        yield attempt

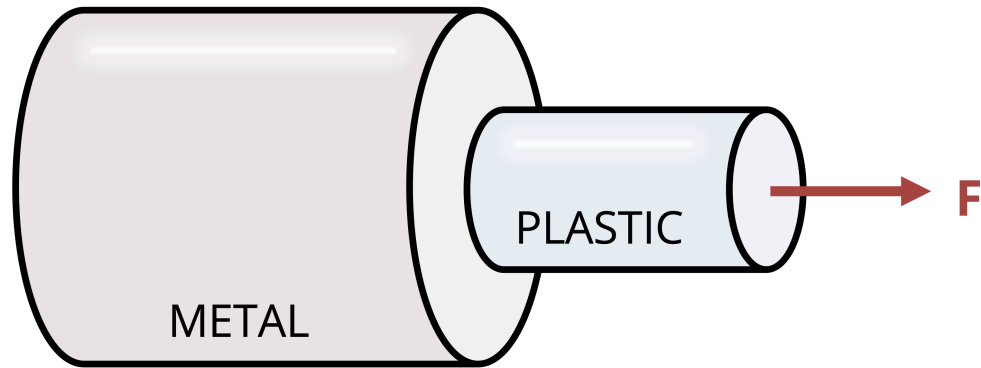
# App installation
app = App(app_ui, server)

```

Problem 2.3

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image



CROSS SECTION VIEW

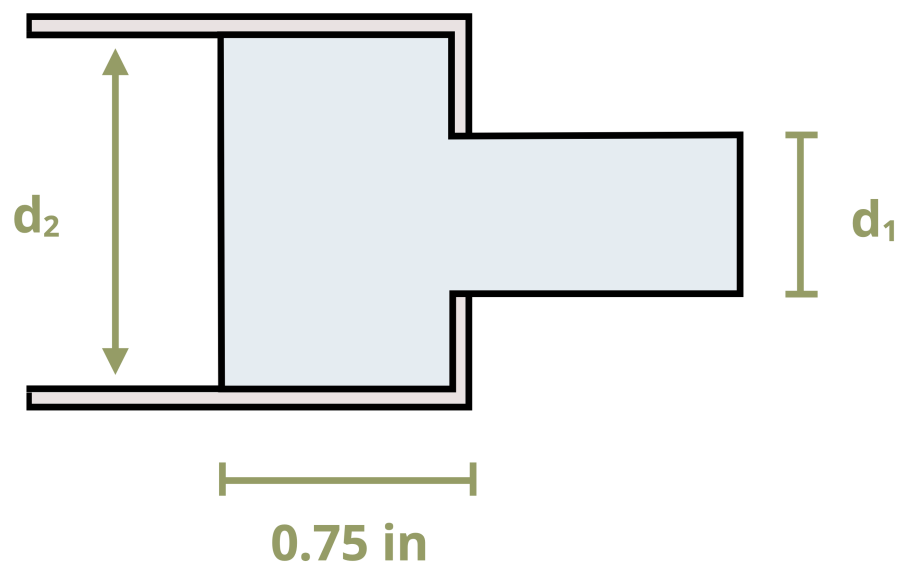


Figure 0.3: Figure 1: A plastic cylindrical peg is constrained by a metal cap

```

#| standalone: true
#| viewerHeight: 600
#| components: [viewer]

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
from pathlib import Path

def generate_random_letters(length):
    # Generate a random string of letters of specified length
    return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))

problem_ID="144"
F=reactive.Value("__")
d1=reactive.Value("__")
d2=reactive.Value("__")

attempts=["Timestamp,Attempt,Answer,Feedback\n"]

app_ui = ui.page_fluid(
    ui.markdown("**Please enter your ID number from your instructor and click to generate your problem**"),
    ui.input_text("ID","", placeholder="Enter ID Number Here"),
    ui.input_action_button("generate_problem", "Generate Problem", class_="btn-primary"),
    ui.markdown("**Problem Statement**"),
    ui.output_ui("ui_problem_statement"),
    ui.input_text("answer","Your Answer in units of psi", placeholder="Please enter your answer"),
    ui.input_action_button("submit", "Submit Answer", class_="btn-primary"),
    ui.download_button("download", "Download File to Submit", class_="btn-success"),
)

def server(input, output, session):
    # Initialize a counter for attempts
    attempt_counter = reactive.Value(0)

    @output

```



```

        feedback,
        title="Feedback",
        easy_close=True
    )
    ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():
    # Start the CSV with the encoded attempt (without label)
    final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else None
    yield f"{final_encoded}\n\n"

    # Write the header for the remaining CSV data once
    yield "Timestamp,Attempt,Answer,Feedback\n"

    # Write the attempts data, ensure that the header from the attempts list is not written
    for attempt in attempts[1:]: # Skip the first element which is the header
        await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
        yield attempt

# App installation
app = App(app_ui, server)

```

Problem 2.4

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

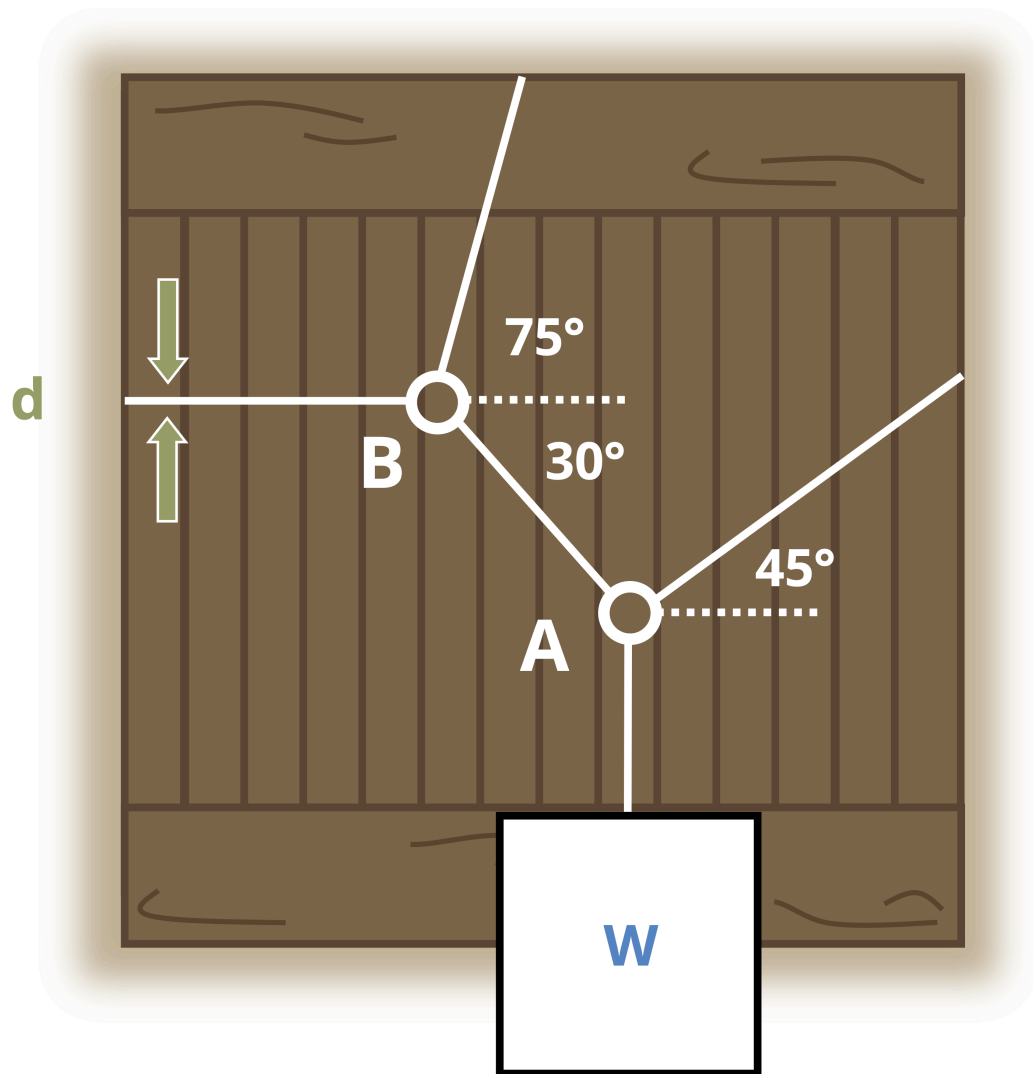


Figure 0.4: Figure 1: A crate is suspended by a set of cables

```
#| standalone: true
#| viewerHeight: 600
#| components: [viewer]
```

```

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
from pathlib import Path

def generate_random_letters(length):
    # Generate a random string of letters of specified length
    return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))

problem_ID="146"
W=reactive.Value("__")
d=reactive.Value("__")
angle1=math.radians(45)
angle2=math.radians(30)
angle3=math.radians(75)

attempts=["Timestamp,Attempt,Answer,Feedback\n"]

app_ui = ui.page_fluid(
    ui.markdown("**Please enter your ID number from your instructor and click to generate your problem**"),
    ui.input_text("ID","", placeholder="Enter ID Number Here"),
    ui.input_action_button("generate_problem", "Generate Problem", class_="btn-primary"),
    ui.markdown("**Problem Statement**"),
    ui.output_ui("ui_problem_statement"),
    ui.input_text("answer","Your Answer in units of GPa", placeholder="Please enter your answer"),
    ui.input_action_button("submit", "Submit Answer", class_="btn-primary"),
    ui.download_button("download", "Download File to Submit", class_="btn-success"),
)

def server(input, output, session):
    # Initialize a counter for attempts
    attempt_counter = reactive.Value(0)

    @output
    @render.ui
    def ui_problem_statement():

```

```

        return[ui.markdown(f"A crate weighing {W()} kN is suspended by a set of cables. The c

@reactive.Effect
@reactive.event(input.generate_problem)
def randomize_vars():
    random.seed(input.ID())
    W.set(random.randrange(30, 90, 1))
    d.set(random.randrange(20, 90, 1)/10)

@reactive.Effect
@reactive.event(input.submit)
def _():
    attempt_counter.set(attempt_counter() + 1) # Increment the attempt counter on each s

    R1 = W()/(((math.cos(angle1)/math.cos(angle2))*math.sin(angle2))+math.sin(angle1))
    instr= (R1*10**3/(math.pi*((d)/(1000*2))**2))/10**9
    if math.isclose(float(input.answer()), instr, rel_tol=0.001):
        check = "*Correct*"
        correct_indicator = "JL"
    else:
        check = "*Not Correct.*"
        correct_indicator = "JG"

    # Generate random parts for the encoded attempt.
    random_start = generate_random_letters(4)
    random_middle = generate_random_letters(4)
    random_end = generate_random_letters(4)
    encoded_attempt = f"{random_start}{problem_ID}-{random_middle}{attempt_counter()}{co

    # Store the most recent encoded attempt in a reactive value so it persists across sub
    session.encoded_attempt = reactive.Value(encoded_attempt)

    # Append the attempt data to the attempts list without the encoded attempt
    attempts.append(f"{datetime.now()}, {attempt_counter()}, {input.answer()}, {check}\n")

    # Show feedback to the user.
    feedback = ui.markdown(f"Your answer of {input.answer()} is {check}. For reference in
    m = ui.modal(
        feedback,
        title="Feedback",
        easy_close=True

```



```

    )
    ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():
    # Start the CSV with the encoded attempt (without label)
    final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else
    yield f"{final_encoded}\n\n"

    # Write the header for the remaining CSV data once
    yield "Timestamp,Attempt,Answer,Feedback\n"

    # Write the attempts data, ensure that the header from the attempts list is not writ
    for attempt in attempts[1:]: # Skip the first element which is the header
        await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
        yield attempt

# App installation
app = App(app_ui, server)

```

Problem 2.47

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

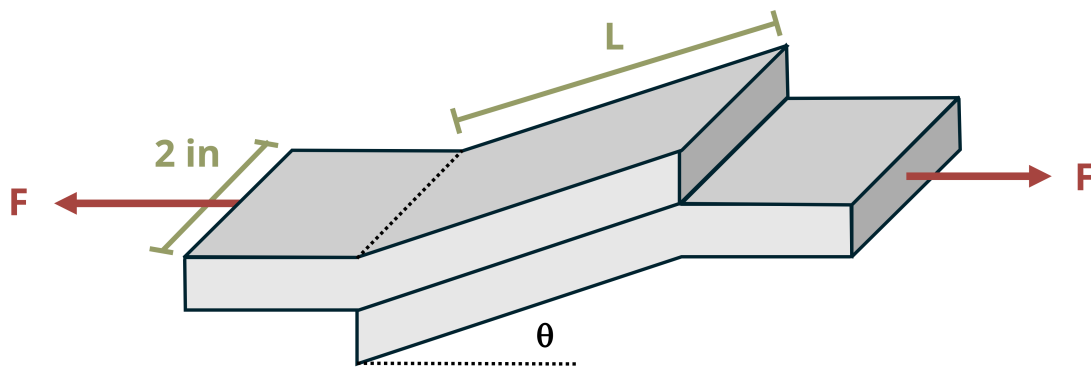


Figure 0.5: Figure 1: Two slanted brackets are glued together

```
#!/ standalone: true
#!/ viewerHeight: 600
#!/ components: [viewer]

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
from pathlib import Path
```

```

def generate_random_letters(length):
    # Generate a random string of letters of specified length
    return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))

problem_ID="153"
F=reactive.Value("__")
L=reactive.Value("__")
Θ=reactive.Value("__")

attempts=["Timestamp,Attempt,Answer,Feedback\n"]

app_ui = ui.page_fluid(
    ui.markdown("**Please enter your ID number from your instructor and click to generate your problem**"),
    ui.input_text("ID","", placeholder="Enter ID Number Here"),
    ui.input_action_button("generate_problem", "Generate Problem", class_="btn-primary"),
    ui.markdown("**Problem Statement**"),
    ui.output_ui("ui_problem_statement"),
    ui.input_text("answer","Your Answer in units of psi", placeholder="Please enter your answer"),
    ui.input_action_button("submit", "Submit Answer", class_="btn-primary"),
    ui.download_button("download", "Download File to Submit", class_="btn-success"),
)

def server(input, output, session):
    # Initialize a counter for attempts
    attempt_counter = reactive.Value(0)

    @output
    @render.ui
    def ui_problem_statement():
        return[ui.markdown(f"Two slanted brackets are glued together as shown. If  $F = \{F()\}$  then  $L = \{L()\}$  and  $\Theta = \{\Theta()\}$ ")]

    @reactive.Effect
    @reactive.event(input.generate_problem)
    def randomize_vars():
        random.seed(input.ID())
        F.set(random.randrange(200, 800, 10))
        L.set(random.randrange(20, 80, 1)/10)
        Θ.set(random.randrange(15, 30, 1))

    @reactive.Effect

```

```

@reactive.event(input.submit)
def _():
    attempt_counter.set(attempt_counter() + 1) # Increment the attempt counter on each s

    instr= (F()*math.sin(math.radians(Θ())))/(L()*2))
    if math.isclose(float(input.answer()), instr, rel_tol=0.001):
        check = "*Correct*"
        correct_indicator = "JL"
    else:
        check = "*Not Correct.*"
        correct_indicator = "JG"

    # Generate random parts for the encoded attempt.
    random_start = generate_random_letters(4)
    random_middle = generate_random_letters(4)
    random_end = generate_random_letters(4)
    encoded_attempt = f"{random_start}{problem_ID}-{random_middle}{attempt_counter()}{co

    # Store the most recent encoded attempt in a reactive value so it persists across su
    session.encoded_attempt = reactive.Value(encoded_attempt)

    # Append the attempt data to the attempts list without the encoded attempt
    attempts.append(f"{datetime.now()}, {attempt_counter()}, {input.answer()}, {check}\n

    # Show feedback to the user.
    feedback = ui.markdown(f"Your answer of {input.answer()} is {check}. For reference i
    m = ui.modal(
        feedback,
        title="Feedback",
        easy_close=True
    )
    ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():
    # Start the CSV with the encoded attempt (without label)
    final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else
    yield f"{final_encoded}\n\n"

    # Write the header for the remaining CSV data once
    yield "Timestamp,Attempt,Answer,Feedback\n"

```

```
        # Write the attempts data, ensure that the header from the attempts list is not writ
    for attempt in attempts[1:]: # Skip the first element which is the header
        await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
        yield attempt

# App installation
app = App(app_ui, server)
```

Problem 2.48

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

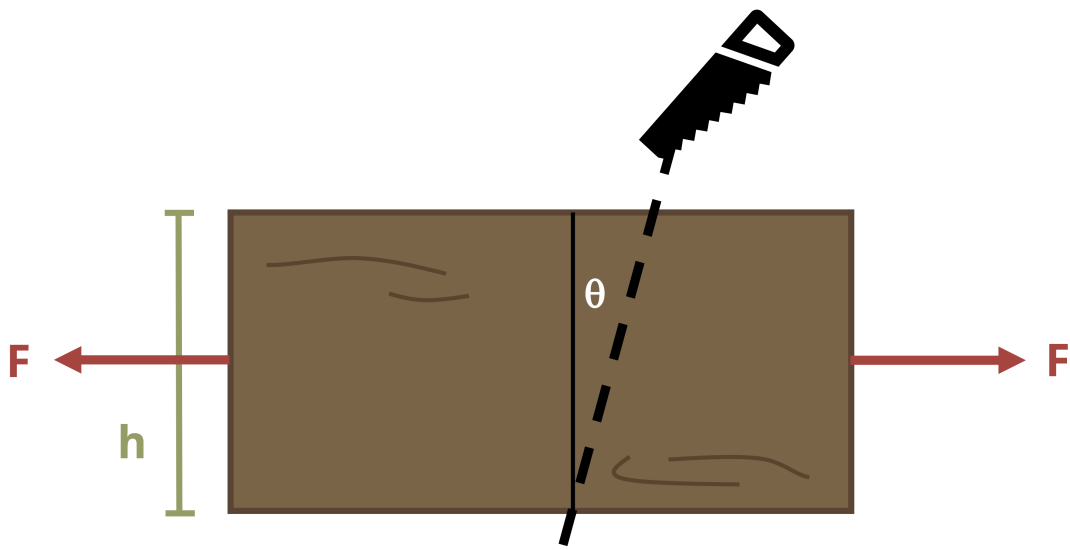


Figure 0.6: Figure 1: A 2 inch thick board is cut and then glued back together along a line

```
#| standalone: true
#| viewerHeight: 600
#| components: [viewer]

from shiny import App, render, ui, reactive
import random
```



```

random.seed(input.ID())
F.set(random.randrange(2000, 6000, 100))
h.set(random.randrange(50, 150, 1)/10)
θ.set(random.randrange(10, 20, 1))

@reactive.Effect
@reactive.event(input.submit)
def _():
    attempt_counter.set(attempt_counter() + 1) # Increment the attempt counter on each submit

    instr= (F()/(2*h()))*(math.cos(math.radians(θ()))**2)
    if math.isclose(float(input.answer()), instr, rel_tol=0.001):
        check = "*Correct*"
        correct_indicator = "JL"
    else:
        check = "*Not Correct.*"
        correct_indicator = "JG"

    # Generate random parts for the encoded attempt.
    random_start = generate_random_letters(4)
    random_middle = generate_random_letters(4)
    random_end = generate_random_letters(4)
    encoded_attempt = f"{random_start}{problem_ID}-{random_middle}{attempt_counter()}{correct_indicator}"

    # Store the most recent encoded attempt in a reactive value so it persists across subsequent
    session.encoded_attempt = reactive.Value(encoded_attempt)

    # Append the attempt data to the attempts list without the encoded attempt
    attempts.append(f"{datetime.now()}, {attempt_counter()}, {input.answer()}, {check}\n")

    # Show feedback to the user.
    feedback = ui.markdown(f"Your answer of {input.answer()} is {check}. For reference the correct answer is {instr}")
    m = ui.modal(
        feedback,
        title="Feedback",
        easy_close=True
    )
    ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():

```



```

# Start the CSV with the encoded attempt (without label)
final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else None
yield f"{final_encoded}\n\n"

# Write the header for the remaining CSV data once
yield "Timestamp,Attempt,Answer,Feedback\n"

# Write the attempts data, ensure that the header from the attempts list is not written
for attempt in attempts[1:]: # Skip the first element which is the header
    await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
    yield attempt

# App installation
app = App(app_ui, server)

```

Problem 4.37

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

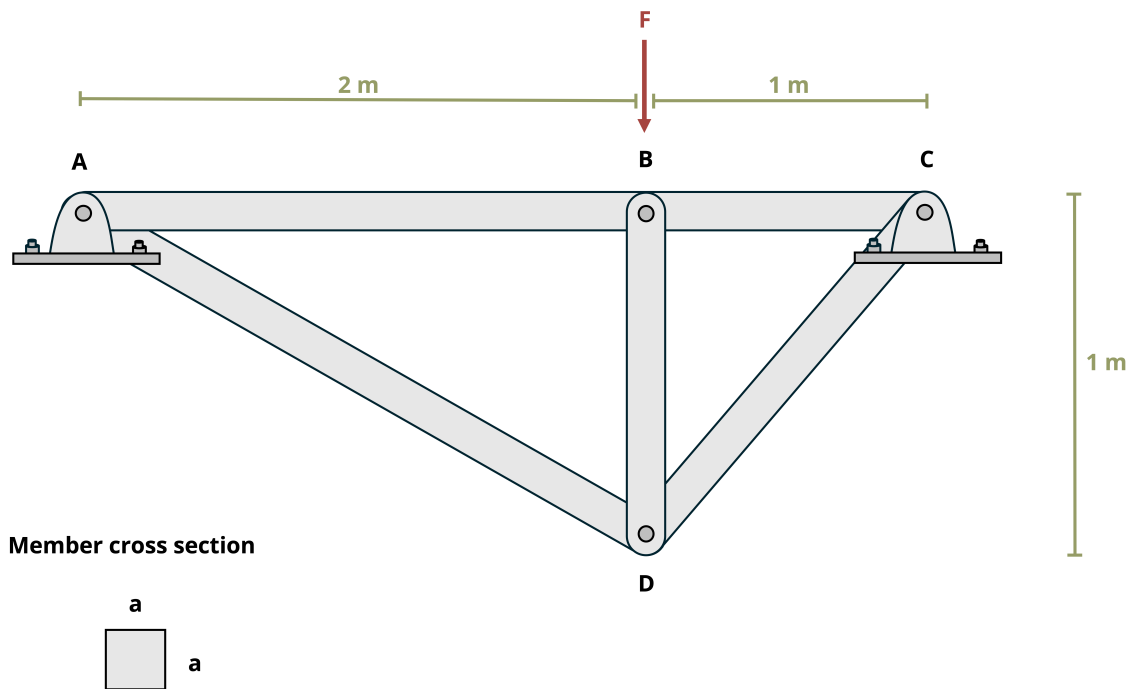


Figure 0.7: Figure 1: A small truss is constructed with solid square wood members

```
#| standalone: true
#| viewerHeight: 600
#| components: [viewer]
```

```

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
from pathlib import Path

def generate_random_letters(length):
    # Generate a random string of letters of specified length
    return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))

problem_ID="157"
F=reactive.Value("__")
FS=reactive.Value("__")
fail=reactive.Value("__")

attempts=["Timestamp,Attempt,Answer,Feedback\n"]

app_ui = ui.page_fluid(
    ui.markdown("**Please enter your ID number from your instructor and click to generate your problem statement**"),
    ui.input_text("ID","", placeholder="Enter ID Number Here"),
    ui.input_action_button("generate_problem", "Generate Problem", class_="btn-primary"),
    ui.markdown("**Problem Statement**"),
    ui.output_ui("ui_problem_statement"),
    ui.input_text("answer","Your Answer in units of centimeters", placeholder="Please enter your answer"),
    ui.input_action_button("submit", "Submit Answer", class_="btn-primary"),
    ui.download_button("download", "Download File to Submit", class_="btn-success"),
)

def server(input, output, session):
    # Initialize a counter for attempts
    attempt_counter = reactive.Value(0)

    @output
    @render.ui
    def ui_problem_statement():
        return[ui.markdown(f"A small truss is constructed with solid square wood members and")]

    @reactive.Effect

```

```

@reactive.event(input.generate_problem)
def randomize_vars():
    random.seed(input.ID())
    F.set(random.randrange(15, 50, 1))
    FS.set(random.randrange(15, 40, 1)/10)
    fail.set(random.randrange(40, 60, 1))

@reactive.Effect
@reactive.event(input.submit)
def _():
    attempt_counter.set(attempt_counter() + 1) # Increment the attempt counter on each

    dl = FS()*F()
    A = (dl/(fail()*10**3))*100*100
    instr= math.sqrt(A)
    if math.isclose(float(input.answer()), instr, rel_tol=0.001):
        check = "*Correct*"
        correct_indicator = "JL"
    else:
        check = "*Not Correct.*"
        correct_indicator = "JG"

    # Generate random parts for the encoded attempt.
    random_start = generate_random_letters(4)
    random_middle = generate_random_letters(4)
    random_end = generate_random_letters(4)
    encoded_attempt = f"{random_start}{problem_ID}-{random_middle}{attempt_counter()}{correct_indicator}"

    # Store the most recent encoded attempt in a reactive value so it persists across sessions.
    session.encoded_attempt = reactive.Value(encoded_attempt)

    # Append the attempt data to the attempts list without the encoded attempt
    attempts.append(f"{datetime.now()}, {attempt_counter()}, {input.answer()}, {check}\n")

    # Show feedback to the user.
    feedback = ui.markdown(f"Your answer of {input.answer()} is {check}. For reference it is {instr}")
    m = ui.modal(
        feedback,
        title="Feedback",
        easy_close=True
    )

```

```

        ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():
    # Start the CSV with the encoded attempt (without label)
    final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else None
    yield f"{final_encoded}\n\n"

    # Write the header for the remaining CSV data once
    yield "Timestamp,Attempt,Answer,Feedback\n"

    # Write the attempts data, ensure that the header from the attempts list is not written again
    for attempt in attempts[1:]: # Skip the first element which is the header
        await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
        yield attempt

# App installation
app = App(app_ui, server)

```

Problem 2.21

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

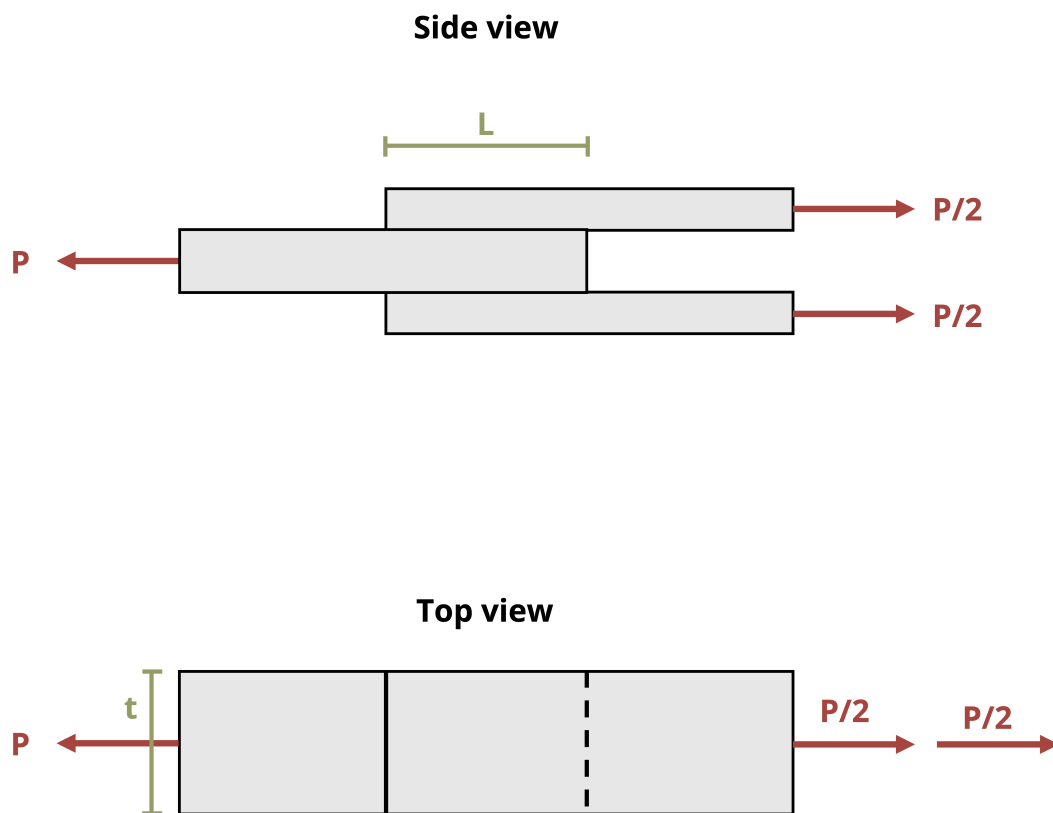


Figure 0.8: Figure 1: A double lap joint is glued together

```

#| standalone: true
#| viewerHeight: 600
#| components: [viewer]

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
from pathlib import Path

def generate_random_letters(length):
    # Generate a random string of letters of specified length
    return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))

problem_ID="164"
fail=reactive.Value("__")
L=reactive.Value("__")
t=reactive.Value("__")

attempts=["Timestamp,Attempt,Answer,Feedback\n"]

app_ui = ui.page_fluid(
    ui.markdown("**Please enter your ID number from your instructor and click to generate your problem**"),
    ui.input_text("ID","", placeholder="Enter ID Number Here"),
    ui.input_action_button("generate_problem", "Generate Problem", class_="btn-primary"),
    ui.markdown("**Problem Statement**"),
    ui.output_ui("ui_problem_statement"),
    ui.input_text("answer","Your Answer in units of kips", placeholder="Please enter your answer"),
    ui.input_action_button("submit", "Submit Answer", class_="btn-primary"),
    ui.download_button("download", "Download File to Submit", class_="btn-success"),
)

def server(input, output, session):
    # Initialize a counter for attempts
    attempt_counter = reactive.Value(0)

    @output

```



```

        feedback,
        title="Feedback",
        easy_close=True
    )
    ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():
    # Start the CSV with the encoded attempt (without label)
    final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else None
    yield f"{final_encoded}\n\n"

    # Write the header for the remaining CSV data once
    yield "Timestamp,Attempt,Answer,Feedback\n"

    # Write the attempts data, ensure that the header from the attempts list is not written
    for attempt in attempts[1:]: # Skip the first element which is the header
        await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
        yield attempt

# App installation
app = App(app_ui, server)

```

Problem 2.22

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

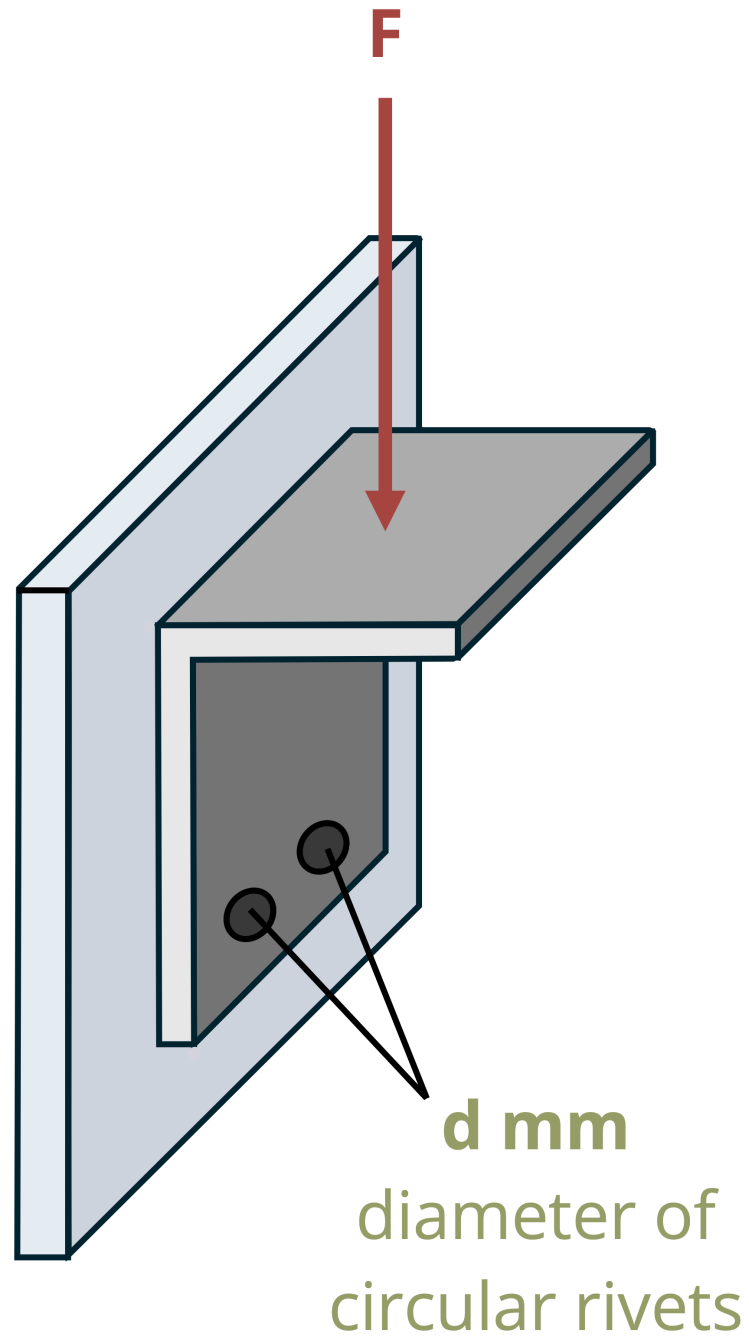


Figure 0.9: Figure 1: A bracket is attached to a wall with two circular rivets

```

#| standalone: true
#| viewerHeight: 600
#| components: [viewer]

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
from pathlib import Path

def generate_random_letters(length):
    # Generate a random string of letters of specified length
    return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))

problem_ID="165"
F=reactive.Value("__")
d=reactive.Value("__")

attempts=["Timestamp,Attempt,Answer,Feedback\n"]

app_ui = ui.page_fluid(
    ui.markdown("**Please enter your ID number from your instructor and click to generate your problem**"),
    ui.input_text("ID","", placeholder="Enter ID Number Here"),
    ui.input_action_button("generate_problem", "Generate Problem", class_="btn-primary"),
    ui.markdown("**Problem Statement**"),
    ui.output_ui("ui_problem_statement"),
    ui.input_text("answer","Your Answer in units of MPa", placeholder="Please enter your answer"),
    ui.input_action_button("submit", "Submit Answer", class_="btn-primary"),
    ui.download_button("download", "Download File to Submit", class_="btn-success"),
)

def server(input, output, session):
    # Initialize a counter for attempts
    attempt_counter = reactive.Value(0)

    @output

```

```

@render.ui
def ui_problem_statement():
    return[ui.markdown(f"A bracket is attached to a wall with two circular rivets of diameter {d()} cm. A force of {F()} N is applied to the bracket at an angle of {theta()} degrees to the horizontal. What is the area of the bracket? (Round your answer to two decimal places.)")]

@reactive.Effect
@reactive.event(input.generate_problem)
def randomize_vars():
    random.seed(input.ID())
    F.set(random.randrange(30, 100, 1))
    d.set(random.randrange(10, 40, 1))

@reactive.Effect
@reactive.event(input.submit)
def _():
    attempt_counter.set(attempt_counter() + 1) # Increment the attempt counter on each submission

    A = math.pi*(d()/(1000*2))**2
    instr= ((F()/2)/A)/1000
    if math.isclose(float(input.answer()), instr, abs_tol=0.001):
        check = "*Correct*"
        correct_indicator = "JL"
    else:
        check = "*Not Correct.*"
        correct_indicator = "JG"

    # Generate random parts for the encoded attempt.
    random_start = generate_random_letters(4)
    random_middle = generate_random_letters(4)
    random_end = generate_random_letters(4)
    encoded_attempt = f"{random_start}{problem_ID}-{random_middle}{attempt_counter()}{correct_indicator}"

    # Store the most recent encoded attempt in a reactive value so it persists across subsequent submissions.
    session.encoded_attempt = reactive.Value(encoded_attempt)

    # Append the attempt data to the attempts list without the encoded attempt
    attempts.append(f"{datetime.now()}, {attempt_counter()}, {input.answer()}, {check}\n")

    # Show feedback to the user.
    feedback = ui.markdown(f"Your answer of {input.answer()} is {check}. For reference, the correct answer is {instr}.")
    m = ui.modal(
        feedback,

```

```

        title="Feedback",
        easy_close=True
    )
    ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():
    # Start the CSV with the encoded attempt (without label)
    final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else None
    yield f"{final_encoded}\n\n"

    # Write the header for the remaining CSV data once
    yield "Timestamp,Attempt,Answer,Feedback\n"

    # Write the attempts data, ensure that the header from the attempts list is not written
    for attempt in attempts[1:]: # Skip the first element which is the header
        await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
        yield attempt

# App installation
app = App(app_ui, server)

```

Problem 2.37

This is a dynamic rendering of the problem with dynamic variables based on the username entered.

Problem Image

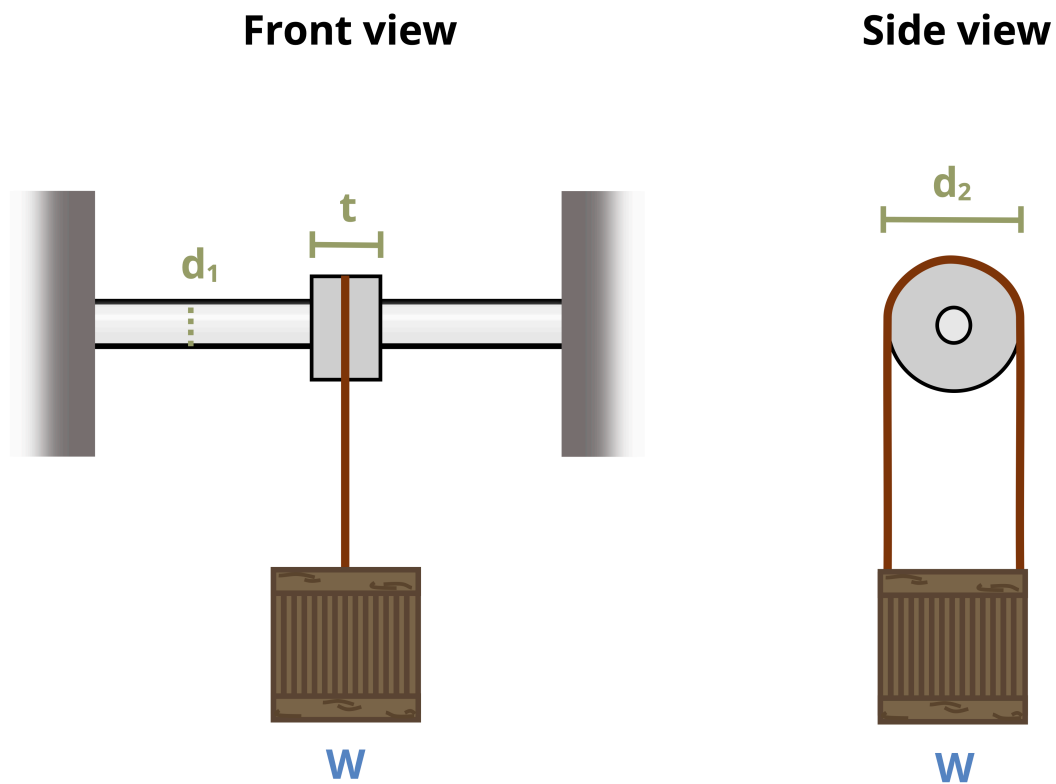


Figure 0.10: Figure 1: A crate is hanged on a circular solid metal rod.


```

#| standalone: true
#| viewerHeight: 600
#| components: [viewer]

from shiny import App, render, ui, reactive
import random
import asyncio
import io
import math
import string
from datetime import datetime
from pathlib import Path

problem_ID="166"
W=reactive.Value("__")
d1=reactive.Value("__")
d2=reactive.Value("__")
t=reactive.Value("__")

def generate_random_letters(length):
    # Generate a random string of letters of specified length
    return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))

attempts=["Timestamp,Attempt,Answer,Feedback\n"]

app_ui = ui.page_fluid(
    ui.markdown("**Please enter your ID number from your instructor and click to generate your problem**"),
    ui.input_text("ID","", placeholder="Enter ID Number Here"),
    ui.input_action_button("generate_problem", "Generate Problem", class_="btn-primary"),
    ui.markdown("**Problem Statement**"),
    ui.output_ui("ui_problem_statement"),
    ui.input_text("answer","Your Answer in units of ksi", placeholder="Please enter your answer"),
    ui.input_action_button("submit", "Submit Answer", class_="btn-primary"),
    ui.download_button("download", "Download File to Submit", class_="btn-success"),
)

def server(input, output, session):
    # Initialize a counter for attempts
    attempt_counter = reactive.Value(0)

```

```

@output
@render.ui
def ui_problem_statement():
    return[ui.markdown(f"A crate of weight W = {W()} lb hangs from a solid circular metal

@reactive.Effect
@reactive.event(input.generate_problem)
def randomize_vars():
    random.seed(input.ID())
    W.set(random.randrange(4000, 9000, 100))
    d1.set(random.randrange(5, 30, 1)/10)
    d2.set(round(d1() * 3, 2))
    t.set(round(d1() * 2, 2))

@reactive.Effect
@reactive.event(input.submit)
def _():
    attempt_counter.set(attempt_counter() + 1) # Increment the attempt counter on each s

    instr= (W()/(d1()*t()))/1000
    if math.isclose(float(input.answer()), instr, rel_tol=0.001):
        check = "*Correct*"
        correct_indicator = "JL"
    else:
        check = "*Not Correct.*"
        correct_indicator = "JG"

    # Generate random parts for the encoded attempt.
    random_start = generate_random_letters(4)
    random_middle = generate_random_letters(4)
    random_end = generate_random_letters(4)
    encoded_attempt = f"{random_start}{problem_ID}-{random_middle}{attempt_counter()}{cor

    # Store the most recent encoded attempt in a reactive value so it persists across sul
    session.encoded_attempt = reactive.Value(encoded_attempt)

    # Append the attempt data to the attempts list without the encoded attempt
    attempts.append(f"{datetime.now()}, {attempt_counter()}, {input.answer()}, {check}\n

    # Show feedback to the user.
    feedback = ui.markdown(f"Your answer of {input.answer()} is {check}. For reference in
    m = ui.modal(

```

```

        feedback,
        title="Feedback",
        easy_close=True
    )
    ui.modal_show(m)

@session.download(filename=lambda: f"Problem_Log-{problem_ID}-{input.ID()}.csv")
async def download():
    # Start the CSV with the encoded attempt (without label)
    final_encoded = session.encoded_attempt() if session.encoded_attempt is not None else None
    yield f"{final_encoded}\n\n"

    # Write the header for the remaining CSV data once
    yield "Timestamp,Attempt,Answer,Feedback\n"

    # Write the attempts data, ensure that the header from the attempts list is not written
    for attempt in attempts[1:]: # Skip the first element which is the header
        await asyncio.sleep(0.25) # This delay may not be necessary; adjust as needed
        yield attempt

# App installation
app = App(app_ui, server)

```

Part II

Interactive Interface Demo