Assignment 01: Using Variables

Due: Tuesday 7 September 11:59pm

Overview

In this first assignment you will write a Python program that determines the member tensions and support reactions in a simple truss. The problem should seem familiar to you from Introduction to Engineering Analysis (IEA). However, unlike in IEA, the angle of the applied force is initially unknown; various angles in degrees will be provided to your program when it is run by Submitty.

The geometry of the truss that you are to analyze is shown below in Figure 1. The load at joint C is at an unknown angle θ from the horizontal as shown. The following data is known:

30° - 60° - 90° three member simple truss	
Member AC angle from horizontal	30°
Joint-to-joint length of member BC:	0.6 meters
Magnitude of force **F** applied at joint B:	450 Newtons
Fixed support at joint A	
Rolling support at joint B	
20° ≤ θ ≤ 100°	

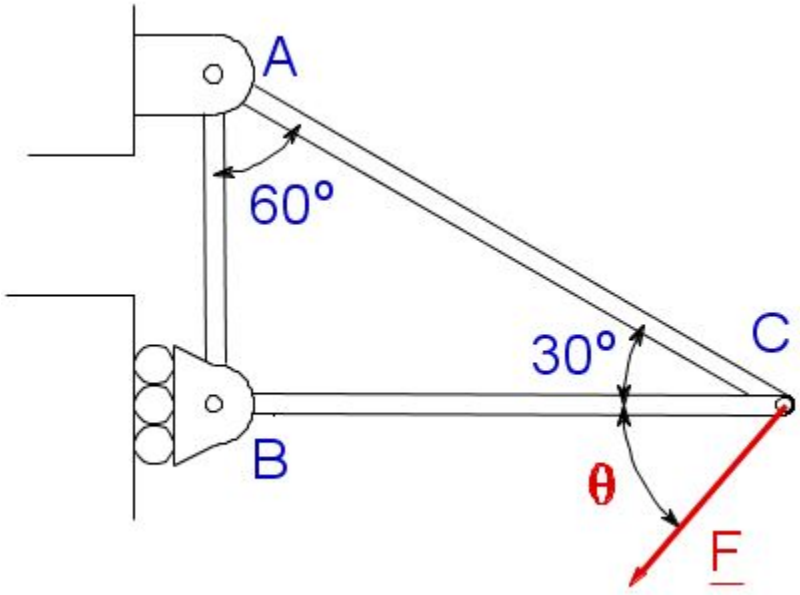


Figure 1: three bar truss with applied force **F** at angle θ counterclockwise from horizontal

For a given angle θ in degrees your task is to determine the tensions in members AB, BC, and AC, as well as reaction forces Ax, Ay, and Bx. The convention used here is that members in compression will have a negative value for their tension, and reaction forces are aligned with the positive x- and y-axes.

Deliverable(s)

Write Python code in a file named **hw01.py** that computes the member tensions in the truss shown in Figure 1. To define the angle theta (in degrees) you must include the following line at the top of your code:

```
from angles import theta
```

This line allows Submitty to provide a value for theta to your program.

Use variables to represent the tensions and reactions. Furthermore, you must ensure that these quantities are given the following variable names, and have the appropriate value, by the time your program finishes:

T_AB	the tension in member AB
T_AC	the tension in member AC
T_BC	the tension in member BC
Ax	the horizontal (+x) reaction force at the fixed support
Ay	the vertical (+y) reaction force at the fixed support
Bx	the horizontal (+x) reaction force at the rolling support

The above variables must be defined exactly as shown above (and recall that Python is case sensitive). You do not need to determine the values of these variables in the order shown above; it is only important that they have the correct value by the end of the program. Your code may (and should) include intermediate variables to simplify your mathematical expressions.

Output

When you run **hw01.py** for the last time before submitting to Submitty, there should be no output at all. Make sure you have commented out all of the `print` statements that you added to test your program.

In Python you can type the number sign (aka pound sign), #, in front of code that you want the Python interpreter to ignore. For example,

```
print(x)     # this print statement will run
#print(x)    # this statement will not run
```

In Spyder you can highlight a block of code and use menu “Edit” / “Comment/Uncomment” to toggle the presence or deletion of # at the start of the highlighted lines. The same toggle effect is accomplished using the <Control>+<1> key bind.

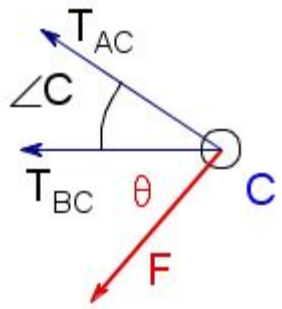
The above, the Lesson 01 Preliminaries, and the Lesson 01 Jupyter Notebook tutorials are sufficient for you to complete this assignment successfully. What follows are analysis hints specific to this assignment and programming tips that apply to many of the assignments.

Analysis Hints and Programming Tips

1. Don't abandon what you have already learned. Start by drawing the appropriate free body diagrams and then applying the 2D equations of static equilibrium per the method of joints.

For example, members AC and BC emerge from joint C as shown at right. Summing forces in the x direction at the illustrated joint C gives

$$\sum F_{x,C} = F\cos(180+\theta) + T_{BC}\cos(180) + T_{AC}\cos(180-30) = 0$$



In IEA you likely learned to derive this in terms of sines and cosines and to simply ignore force components perpendicular to the axis of interest. The advantage of the IEA way is that it is sometimes easier to simplify. The disadvantage is that you need to think about the sign of every term to make sure you get it right.

Using cosines of angles from the positive x axis for forces in the x direction (and cosines of angles from the positive y axis for forces in the y direction) will always give you the correct sign, at the cost of extra terms and more complicated angles. However, when you program your solution you should let your code do the computations for you.

At joints A and B it makes sense to keep the x and y axes aligned with the reaction forces so that they can be isolated and computed in your force balance (with one equation for each unknown). At joint C you may or may not find that rotating your local coordinate axes helps isolate a force so that it can be computed independently as a function only of **F** and θ and the geometry of the truss.

2. However you choose to derive your force balances you may choose to use functions like `cos` and `sin`, and you may also use the constant π to convert from degrees to radians as the `sin` and `cos` functions take their input in radians. These functions and constants can be imported into your program by adding the following line at the top of your code:

```
from math import sin, cos, pi
```

Then, to take the cosine of π, for example, you would use `cos(pi)`.

3. You can create your own **`angles.py`** file and use it for testing. The following content should be sufficient to make your program work when invoked by Python (though the comment on the line starting with # is optional).

```
# put these lines into your file angles.py
theta = 45.0
```

Of course, the value 45.0 degrees is arbitrary. Set it to whatever value you want during your testing. The file **`angles.py`** must be in the same directory as **`hw01.py`** to be found by Python.

4. You should test your code using multiple values of `theta` and think about whether the results you get make physical sense before confidently submitting your code.

The following are testing strategies that you should keep in mind throughout this course.

> Easy cases. You should know some of the member and/or support forces by inspection when $\theta = -30°$, $\theta = 0°$, $\theta = 90°$, $\theta = 150°$ and $\theta = 180°$. Note that most of these angles are excluded by the problem statement and may not even be physical, but they should still give mathematically correct answers if your mathematical model is correct. You should also be able to identify any zero-force members that are provided for structural stability even if they (theoretically) carry no load.
>
> Edge and iconic cases. Do the forces you compute make sense when:
> - the magnitude of the force is zero
> - the force is aligned with member AB or member AC
> - the force is vertical pointed down
> - the force is horizontal pointed left

While not necessarily part of testing, you might also consider "beyond design basis" cases. What forces do you compute when $150° < \theta < 330°$? Do they make sense? Should they make sense?

5. The code design is not difficult. You will likely take a *bottom-up* design approach where you build a small piece, test that piece, and continue building the code when that piece works. For example, you may need to convert degrees to radians. If you do not get that part working correctly then there is no hope of getting the rest of the program to give the right answer. Small pieces of code mean small places to look for bugs. Use print statements to display the numbers you want to check; add text to your print statements so you know what you are looking at. Include comments to explain what you are doing and how you are doing it. Make sure they are clear to anyone you might ask for help. Also have your written notes and sketches available if you expect anyone to be able to help you.

6. Your program will simply assign the results from a bunch of mathematical expressions to variables. What may be hard is keeping track of all of the angles in those expressions: angles that are given in degrees when the sine and cosine functions expect their arguments to be in radians. Think about how to organize the angles in degrees and the angles in radians and the variable names you use. Use pencil and paper well before you start to type any code. Also, resist the temptation to treat this exactly like an IEA problem and compute everything that you can. Write mathematical statements just as you have derived them; let Python do the computation work for you. You will usually make fewer mistakes that way.

Finally, what will Submitty do, and not do?

When you submit your `hw01.py` file to Submitty, it will move your code to a directory that already contains its own `angles.py` file. Submitty will then read from a `.json` file that we created to tell Submity what to do with the file you submitted. This will execute a testing program that we wrote in Python. We will learn about these programs in Lesson 08 of this course. For now, know that our testing program will itself execute your code and compare your answers with what we expect. In particular, it will check that the testing code ran with no errors, resulting in an output file something like the following:

```
......
----------------------------------------------------------------
Ran 6 tests in

OK
```

If your code contains errors, the test program output will include error messages that Submitty will allow you to see. However, these error messages are not necessarily for your benefit, or even intelligible. **Do not use Submitty to debug your code.**

You may submit your code to Submitty only 20 times. After that Submitty will either deduct points for each subsequent submission or will simply not allow you to make a subsequent submission. For maximum credit you can change which version of your code you want Submitty to assign you a grade for within those 20 submissions. Thus, if you try something new and lose points you can ask that Submitty grade a previous version for the higher grade.

Use the Python error messages that Submitty allows you to see to guide you to what and where the problems might be, but long before that you should have been continuously applying your own techniques to verify that your code is functioning correctly.