

# CS410P

## ASSIGNMENT 1P

---

### Purpose

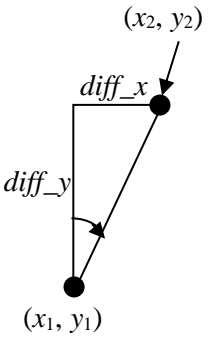
The purpose of this assignment is to give you practice with more expressions, more output formatting, conditional statements, exception handlers and simple loops

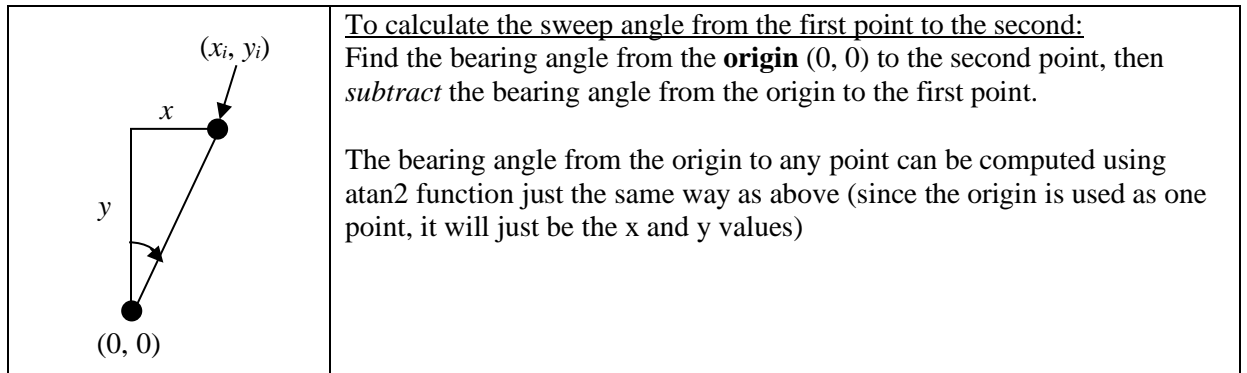
### Details

You will need to submit 2 programs in 2 separate files. Name the two files as indicated below so that it's clear which one is which (Mimir also expects specific file names otherwise your submission will come up with errors). You will need to submit both files together, you can certainly submit just one of them if you want to see how the test cases for that program works out but eventually you will need to submit both files for both of your programs to be graded.

#### 1) Filename: **measure.py**

- Write a program that prompts the user and accepts 2 pairs of coordinate points  $(x_1, y_1)$  and  $(x_2, y_2)$  which are stored in four different variables.
- We want to compute and print the following:
  - 1) The distance between them
  - 2) The bearing angle from the first point to the second point
  - 3) The angle (originating from the origin) swept from the first point to the second (called sweep angle)

	<p>Distance between two points <math>P(x_1, y_1)</math> and <math>Q(x_2, y_2)</math> is given by:</p> $d(P, Q) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
	<p><u>To calculate the bearing from the first point to the second:</u> For directional bearings, <math>0^\circ</math> is along the y-axis (instead of along the x-axis for math)—it's like the whole coordinate system is flipped around the diagonal line <math>y = x</math>. Therefore, <math>diff\_x</math> is the “opposite” side of the angle, and <math>diff\_y</math> is the “adjacent” side.</p> <p>You can use the <code>atan2</code> math function to find the bearing angle by providing the <math>diff\_x</math> and <math>diff\_y</math> explained above as argument i.e. <code>bearing_angle = atan2(diff_x, diff_y)</code></p> <p><i>Note: The bearing angle computed using <code>atan2</code> gives the angle in radians</i></p>



- Your program must print out 3 separate tables, each intended for audiences that have different requirements for precision, units, and space in which to print their tables. (To display an angle in degrees, convert the value from radians to degrees using an appropriate formula). Here are the specifications for those tables in terms of units, characters available for each item, and number of digits required for each item:

	Units distance/angle	Distance	Bearing	Sweep
Table 1	Meters / radians	25 characters, no decimal	25 chars, 3 decimal digits	25 chars, 3 decimal digits
Table 2	Meters / degrees	25 characters, 5 decimal digits	25 chars, 3 decimal digits	25 chars, 2 decimal digits
Table 3	Feet / degrees	25 characters, 2 decimal digits	25 chars, no decimal	25 chars, 1 digit

## Input

Two Pairs of (x, y) values – must be able to accept floating point numbers. Order of inputs: x1, y1, x2, y2

(x1, y1) is the first point; (x2, y2) is the second point

## Output

Sample output is available in the public folder for 1P, must conform to the specifications outlined in the table above. Your results may not match exactly with mine but it should be close (Use a constant for PI for the Math library so that your result is as close to mine as possible).

## Mimir Submission

There are 3 test cases set up on Mimir to check your results. Since we have a table of values to printed each with different formats or units these will be manually graded. These test cases will indicate 0 points but their corresponding rubric item

will carry the manual grade (when they are graded). You will need to make sure that the test cases are at least passing so that there are no compile errors. You should also click on each test case and see for yourself that the program output is what is expected which serves as a way to get the most points when it is graded.

## **2) Filename: leapyear.py**

### **Scenario**

There are approximately 365.2422 days in a solar year, our calendar has 365 days in a standard year and occasionally has a leap year of 366 days to account for the fractional part. In -46 B.C the Julian calendar established that those years that are divisible by 4 will be leap years. This scheme over-estimates the length of the solar year by approximately one day every 128 years. By 1753 we had adopted the Gregorian calendar with a more complicated leap year formula that corrects for this.

Definition of a Julian calendar leap year:

- For years between -46 B.C and 1752 A.D (both inclusive) a leap year is a year that is exactly divisible by 4

Definition of a Gregorian calendar leap year:

- For years 1753 A.D and after when Gregorian calendar is in effect, most years that are a multiple of 4 are leap years. However, if it is a multiple of 100, it is NOT a leap year unless it is ALSO a multiple of 400.

### **Details**

Write a program to read a year value from the input and determine whether it is a leap year or not. The process is the following: First prompt the user to enter a year and read in the value. Next, determine if the given year is a leap year or not and print out a suitable message which includes which calendar is in effect (Julian or Gregorian). You only need to check for years between -46 and 5000. Anything outside of this range is printed out as an Invalid year. Any value beyond -9999 and 9999 will stop the program.

### **Input**

- Your program continuously prompts the user for input (for Year) and stops only when the input is beyond -9999 or 9999.

- You must have an exception handler in your program to catch invalid input in terms input not being integers (only integers are allowed as input, strings and floating point numbers are invalid).
- Note that the exception handle can be used to check for invalid ‘data types’ but not for ensuring that the input is within a certain range – you will need to validate this by using if statements on your own after the exception handler check is done.

## Output

Sample output will be available in the public folder for 1P (I have just shown you a set of outputs all at once, I know that you will see just one line for every input year). Output format is very simple just make sure you use a print statement like the one indicated below where “year” is a variable that holds the value of year entered by the user:

```
print (“Year”, year, “Julian Leap Year”)
```

The following are string values to be printed based on the result:

Julian Leap Year

Julian Non Leap Year

Gregorian Leap Year

Gregorian Non Leap Year

Invalid Year

Your output for every year entered will be exactly like this format shown below:

Year 2000 Gregorian Leap Year

Year 101 Julian Non Leap Year

Year 5002 Invalid Year

Any invalid input (non-numeric, non-integer) will display an output:

Invalid input

and program continues.

## Mimir Submission

There are several “unit test cases” set up on Mimir to check your results of individual years. Some test cases actually show you what year is being tested, while others do not explicitly tell you this or might not indicate whether you passed the test or not (this is intended so that you are learning to test on your own as well). Once grading is completed you will see all the test cases and whether you passed or not.

*If you're not sure how to write a program that continuously prompts the user until a certain condition is reached, here's an example that uses a "while" statement (it keeps prompting user for a number until the user enters a negative number – so your entire code for checking for a leap year would be inside the body of the "else" statement in this case):*

```
done = False
while(done == False):
    x = eval(input("Enter X:"))
    if x < 0:
        done = True
    else:
        print("x is", x)

print("goodbye")
```

## Grade Key

<b>A</b>	Name, comments	<b>3</b>
<b>B</b>	Measurements - distance (values in first column, 2 points each)	<b>6</b>
<b>C</b>	Measurements – bearing angle (values in second column, 2 points each)	<b>6</b>
<b>D</b>	Measurements – sweep angle (values in third column, 2 points each)	<b>6</b>
<b>E</b>	Measurements – output formatting reasonable, as per specification	<b>8</b>
<b>F</b>	Leap Year: Input acceptable range -9999 to 9999, program runs continuously until a number outside this range is entered	<b>8</b>
<b>G</b>	Leap Year: Exception handler for non-integer input handled correctly, program should continue to prompt for user input after proper error message	<b>4</b>
<b>H</b>	Individual leap year tests – valid between -46 and 5000 (both inclusive), invalid outside this range but between -9999 and 9999 (2.5 or 3 points per test case)	<b>59</b>