

# APS106 – Lab #3

## Preamble

This week you will practice using conditional statements by writing a function to determine whether two circles overlap.

*Use appropriate variable names and place comments throughout your program.*

*The name of the source file must be “lab3.py”.*

## Deliverables

1. A Python file named lab3.py with your function.

Five test cases are provided on MarkUs to help you prepare your solution. **Passing all these test cases does not guarantee your code is correct.** You will need to develop your own test cases to verify your solution works correctly (see below for more details). Your programs will be graded using ten secret test cases. These test cases will be released after the assignment deadline.

### IMPORTANT:

- Do not change the file name or function names
- Do not use input() inside your program

## Problem

While developing solutions or new technologies, engineers are faced with balancing trade-offs in their design. For example, in designing a car, reducing the size could increase fuel efficiency but result in lower crash safety ratings. Computer scientists and software engineers typically need to balance the speed of their programs with the amount of memory they require to execute. One way to approach these trade-offs is to define some constraints and then optimize the design within those constraints. In our car design example, we could define a minimum safety rating and then design the car to maximize fuel efficiency while still meeting our minimum safety rating.

In this week's lab, you will be writing a function that will check whether a design meets a particular constraint. For our problem, we will imagine we are designing the layout of a [wind farm](#) and are trying to identify where to place hundreds of turbines<sup>1</sup>. We want to optimize turbine placement to maximize energy generation while adhering to land use constraints. These constraints define areas where turbines cannot be placed. These constraints come from land rights, proximity to other turbines, and regulations regarding turbine proximity to human housing and natural habitats.

You will write a function named `circle_overlap` which will analyze whether two circles overlap in two-dimensional space. In cases where the circles overlap, your function will determine the nature of the overlap (details below). In the context of our problem, these circles can be thought of as a restricted area and a proposed turbine site. The output of the function will tell us whether the proposed location meets the constraints.

Each of the circles input to the function will be defined by a **centre point** and a **radius** (figure 1).

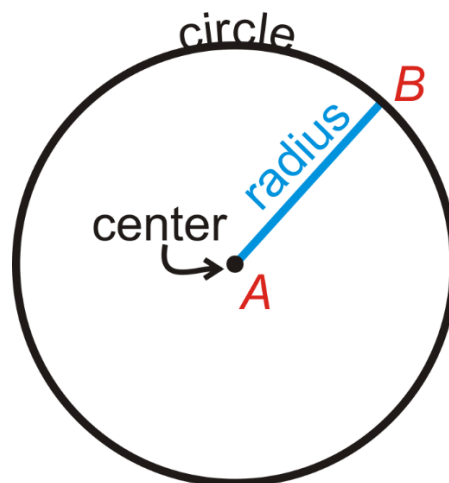


Figure 1. A circle can be defined with its centre coordinate and its radius.

---

<sup>1</sup> This is a real problem that has been previously investigated by Prof. Beck! See <https://www.sciencedirect.com/science/article/abs/pii/S0960148118303641?via=ihub>

The `circle_overlap` function will accept the following inputs:

Input Parameter Name	Description
<code>circ1_centre_x</code>	x coordinate of the centre coordinate of circle 1
<code>circ1_centre_y</code>	y coordinate of the centre coordinate of circle 1
<code>circ1_radius</code>	Radius of circle 1
<code>circ2_centre_x</code>	x coordinate of the centre coordinate of circle 2
<code>circ2_centre_y</code>	y coordinate of the centre coordinate of circle 2
<code>circ2_radius</code>	Radius of circle 2


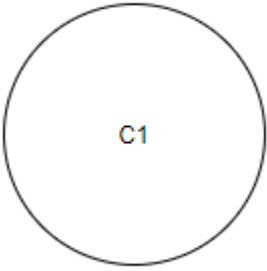
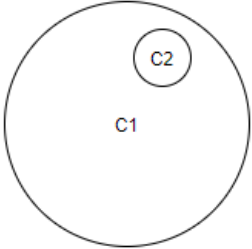
The function will output one of the five following **strings** describing the overlap of the two circles:

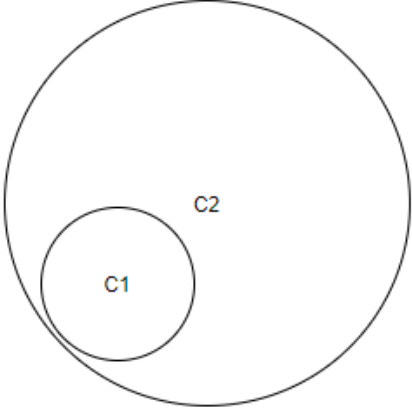
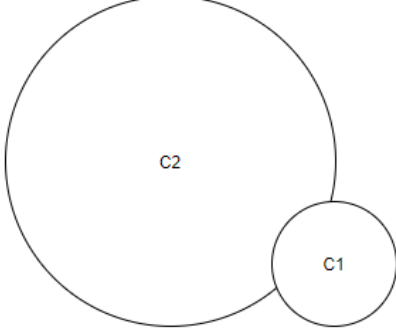
Function Output	Scenario
<code>'no overlap'</code>	The two circles do not have any overlapping area. For this problem we will define “overlap” as a non-zero area that is within both circles. This means the edges of circles can touch without there being any overlap.
<code>'identical circles'</code>	The two circles have the same centre and radius (i.e. the two circles are the same and perfectly overlap)
<code>'circle 2 is contained within circle 1'</code>	The entire area of circle 2 is contained within the area of circle 1 AND the two circles do <b>NOT</b> have the same centre and radius
<code>'circle 1 is contained within circle 2'</code>	The entire area of circle 1 is contained within the area of circle 2 AND the two circles do <b>NOT</b> have the same centre and radius
<code>'circles overlap'</code>	The circles share some overlapping area, but neither is contained completely within the other

**Hint #1:** When designing your algorithm plan, you may find it helpful to draw pictures or use this desmos tool to visualize different inputs: <https://www.desmos.com/calculator/ute1r4um7g>

**Hint #2:** To detect an overlap, you can compare the Euclidean distance between the circle centres to the sum of the two circle’s radii.

Each of these scenarios is presented below with a visual and sample inputs to help understanding. Note images are not to scale.

Scenario/Function Output	Visual Representation	Function Input Parameters
'no overlap'		<pre>circ1_centre_x: 0 circ1_centre_y: 1 circ1_radius: 3  circ2_centre_x: 6 circ2_centre_y: 4 circ2_radius: 1</pre>
'identical circles'	 <p>*C2 is hidden by C1, due to perfect overlap</p>	<pre>circ1_centre_x: 0 circ1_centre_y: 1 circ1_radius: 3  circ2_centre_x: 0 circ2_centre_y: 1 circ2_radius: 3</pre>
'circle 2 is contained within circle 1'		<pre>circ1_centre_x: 1 circ1_centre_y: 1 circ1_radius: 10  circ2_centre_x: 6 circ2_centre_y: 7 circ2_radius: 1</pre>

<p>'circle 1 is contained within circle 2'</p>		<pre> circ1_centre_x: -1 circ1_centre_y: -2 circ1_radius: 2  circ2_centre_x: 0 circ2_centre_y: 0 circ2_radius: 11 </pre>
<p>'circles overlap'</p>		<pre> circ1_centre_x: 1 circ1_centre_y: -2 circ1_radius: 2  circ2_centre_x: -4 circ2_centre_y: 0 circ2_radius: 5 </pre>

Additional notes and assumptions:

- We define “overlap” as a non-zero area that is within both circles. This means circles can share common borders without being considered as overlapping.
- Circle coordinates and radii will always be integers.
- Circle coordinates can be from any of the four quadrants of the two-dimensional x-y plane. That is, x and y coordinates may be zero, positive, or negative.
- The radii will always be positive integers