# Exercises: Functions, Object Composition, Revealing Modules

Problems for exercises and homework for the [“JavaScript Advanced” course @ SoftUni](https://softuni.bg/courses/javascript-advanced). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/299/>.

## \* Euclid’s Algorithm

Write a program that receives **two numbers** as arguments and finds the **greatest common divisor** between them.

### Input

Input will be passed as two **numeric arguments** to your function.

### Output

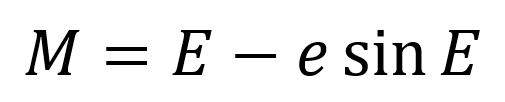
**Return** the greatest common divisor as a result of the function.

### Examples

|  |  |
| --- | --- |
| Sample Input | Sample Output |
| 252, 105 | 21 |

## \*\*\* Kepler’s Problem

Write a function that, given the mean anomaly and orbital eccentricity of a celestial body, calculates its eccentric anomaly. The eccentric anomaly ***E*** is related to the mean anomaly ***M*** by Kepler’s equation:



Where ***e*** is the eccentricity. Note this equation is transcendental, which means it cannot be solved for ***E*** algebraically. Use numerical analysis to approximate a root with accuracy 1x10-9. You can find information about Newton’s Method here: [https://en.wikipedia.org/wiki/Newton’s\_method](https://en.wikipedia.org/wiki/Newton's_method). Try to implement it recursively.

The **input** comes as **two number parameters**. The first parameter is the current mean anomaly in radians and the second is the orbital eccentricity of the body.

The **output** is an approximation of the eccentric anomaly and should be printed on the console. Display only the significant digits.

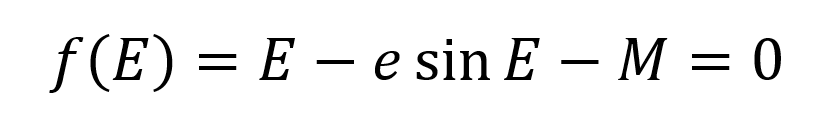
### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1, 0 | 1 | 3.1415926535, 0.75 | 3.141592654 |

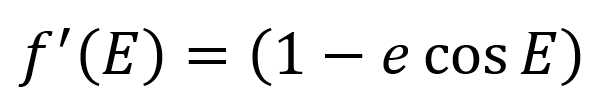
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 0.25, 0.99 | 1.156077258 | 4.8, 0.2 | 4.601234265 |

### Hints

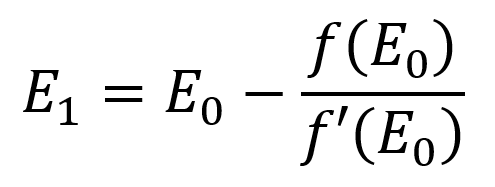
Newton’s method works with functions that equal zero. We shift the variables around to arrive at the following form:



Not coincidentally, this is also our progress check – as we look for a closer approximation for ***E***, the solution of this equation will be closer to zero. Once it’s within the aforementioned ***epsilon*** (required accuracy), we stop iterating and print the result. When implementing recursively, this condition will be the bottom of our recursion. The last bit we need is the first derivative of the function:



And to plug it all into Newton’s equation:



Where ***E0*** is the result of the previous iteration and ***E1*** will be the result of the current iteration. When beginning the iteration, pick an initial value for ***E0*** that might be close enough to our desired result (chose a value that is either zero or equal to the mean anomaly).