# Lab: Defining Classes

## Rhombus of Stars

Create a program that reads a **positive** **integer N** as input and prints on the console a **rhombus** with size **n**:

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 | \* |
| 2 | \*  \* \*  \* |
| 3 | \*  \* \*  \* \* \*  \* \*  \* |
| 4 | \*  \* \*  \* \* \*  \* \* \* \*  \* \* \*  \* \*  \* |

## Class Book

Create a class called Book. It should have an \_\_init\_\_() method that should receive a name, author and pages (number).

Submit only the class in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| book = Book("My Book", "Me", 200)  print(book.name)  print(book.author)  print(book.pages) | My Book  Me  200 |

## Scope Mess

Fix the code below, so it gives the expected output. Submit the fixed code in the judge system.

x = **"global"  
  
def** outer():  
 x = **"local"  
  
 def** inner():  
 x = **"nonlocal"** print(**"inner:"**, x)  
  
 **def** change\_global():  
 x = **"global: changed!"** print(**"outer:"**, x)  
 inner()  
 print(**"outer:"**, x)  
 change\_global()  
  
print(x)  
outer()  
print(x)

### Examples

|  |  |
| --- | --- |
| **Current Output** | **Expected Output** |
| global  outer: local  inner: nonlocal  outer: local  global | global  outer: local  inner: nonlocal  outer: nonlocal  global: changed! |

## Music

Create class named Music that receives title, artist and lyrics upon initialization. The class should also have methods get\_info() and play(). The print\_info() method should return the following:   
'This is "{title}" from "{artist}"'. The play() method should **return** the lyrics. Submit only the class in the judge system. Test your code with your own examples.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| song = Music("Title", "Artist", "Lyrics")  print(song.get\_info())  print(song.play()) | This is "Title" from "Artist"  Lyrics |

## Cup

Create a class called Cup. Upon initialization it should receive size and quantity (a number representing **how much liquid** is in it). The class should also have a method called fill(milliliters) which will **increase** the amount of liquid in the cup with the given **milliliters** (**if** there is **space** in the cup, **otherwise ignore**). The cup should also have a status() method which will **return** the **amount** of **free space** left in the cup. Submit only the class in the judge system. Don't forget to test your code.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| cup = Cup(100, 50)  cup.fill(50)  cup.fill(10)  print(cup.status()) | 0 |

## Flower

Create a class called Flower. Upon initialization, the class should receive name and water\_requirements. The flower should also have an attribute called is\_happy (False by default) and a method called water(quantity), which will water the flower. If the water is **greater than or equal** of the requirements of the flower, it becomes happy. (set is\_happy to True). The last method should be called status() and it should return "{name} is happy" if the flower **is happy**, otherwise it should return **"**{name} is not happy**"**. Submit only the class in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| flower = Flower("Lilly", 100)  flower.water(50)  print(flower.status())  flower.water(100)  print(flower.status()) | Lilly is not happy  Lilly is happy |