***LAB-08: CSPs: constraint satisfaction problems***

***Objective:***

* Learn how to implement a constraint satisfaction problem in Python.

***Activity Outcomes:***

An implementation of CSPs to solve the “The Australia map-coloring problem”:

* **Building a constraint-satisfaction problem framework**
* **Solving “The Australian map-coloring problem”**

***Lab Requirements***

Jupiter notebook (anaconda3)

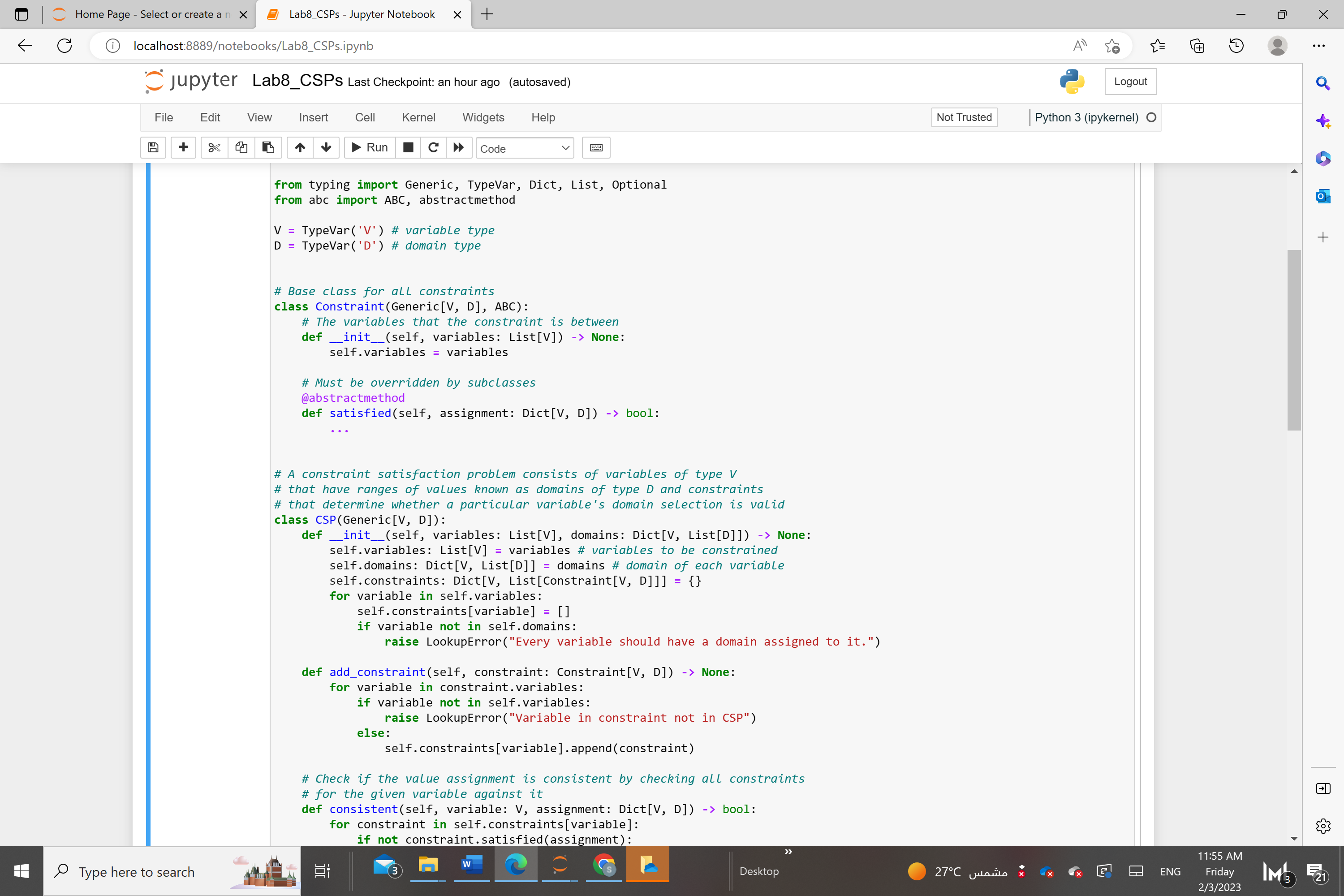
***Lab Description***

***CSPs:***

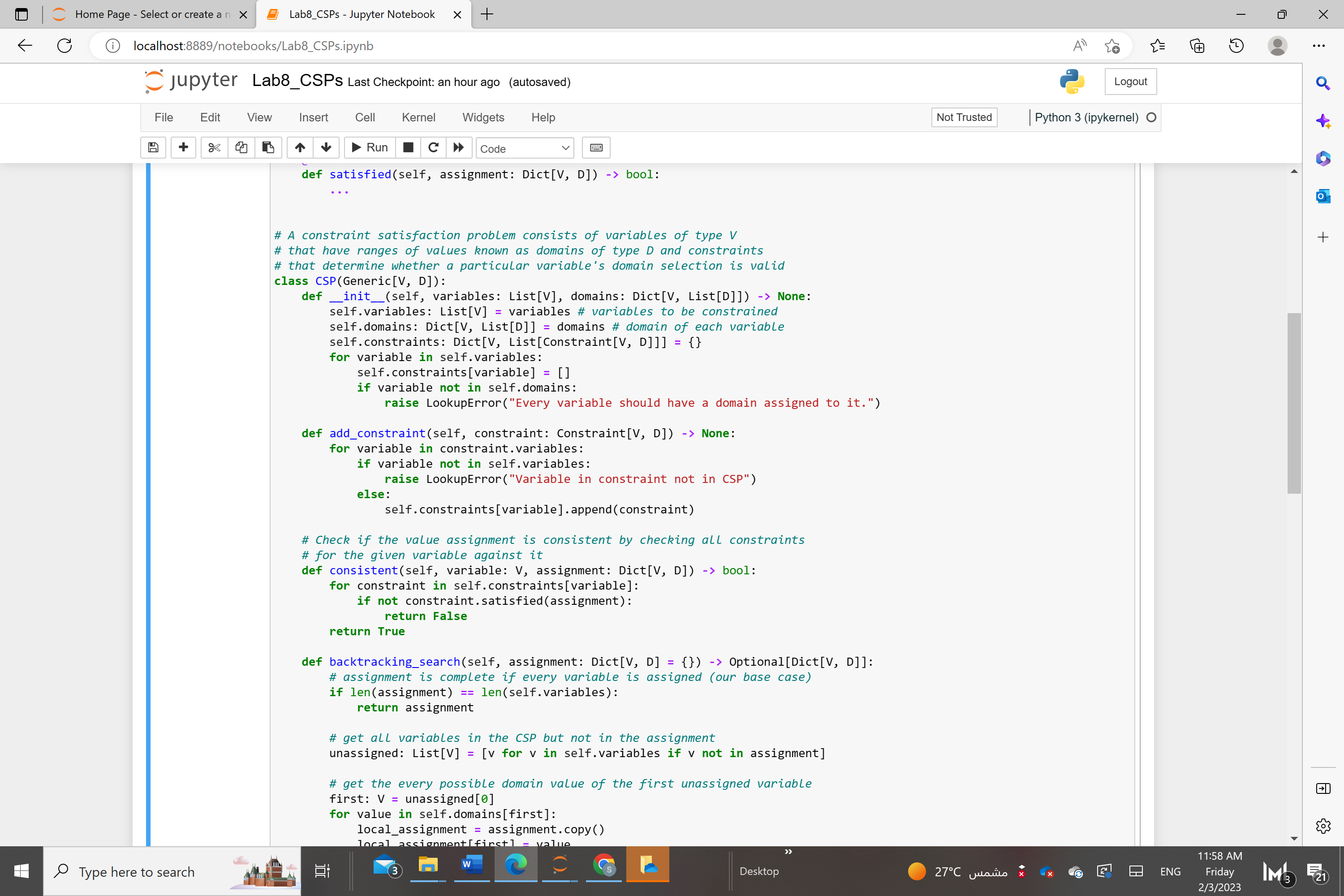
Many problems which computational tools solve can be broadly categorized as constraint-satisfaction problems (CSPs). CSPs are composed of variables with possible values which fall into ranges known as domains. Constraints between the variables must be satisfied in order for constraint-satisfaction problems to be solved. Those three core concepts—variables, domains, and constraints—are simple to understand, and their generality underlies the wide applicability of constraint-satisfaction problem solving.

**Building a constraint-satisfaction problem framework**

Constraints are defined using a Constraint class. Each Constraint consists of the variables it constrains and a method that checks whether it’s satisfied(). The determination of whether a constraint is satisfied is the main logic that goes into defining a specific constraint-satisfaction problem. The default implementation must be overridden because we’re defining our Constraint class as an abstract base class. Abstract base classes aren’t meant to be instantiated; only the subclasses that override and implement their @abstractmethods are used.

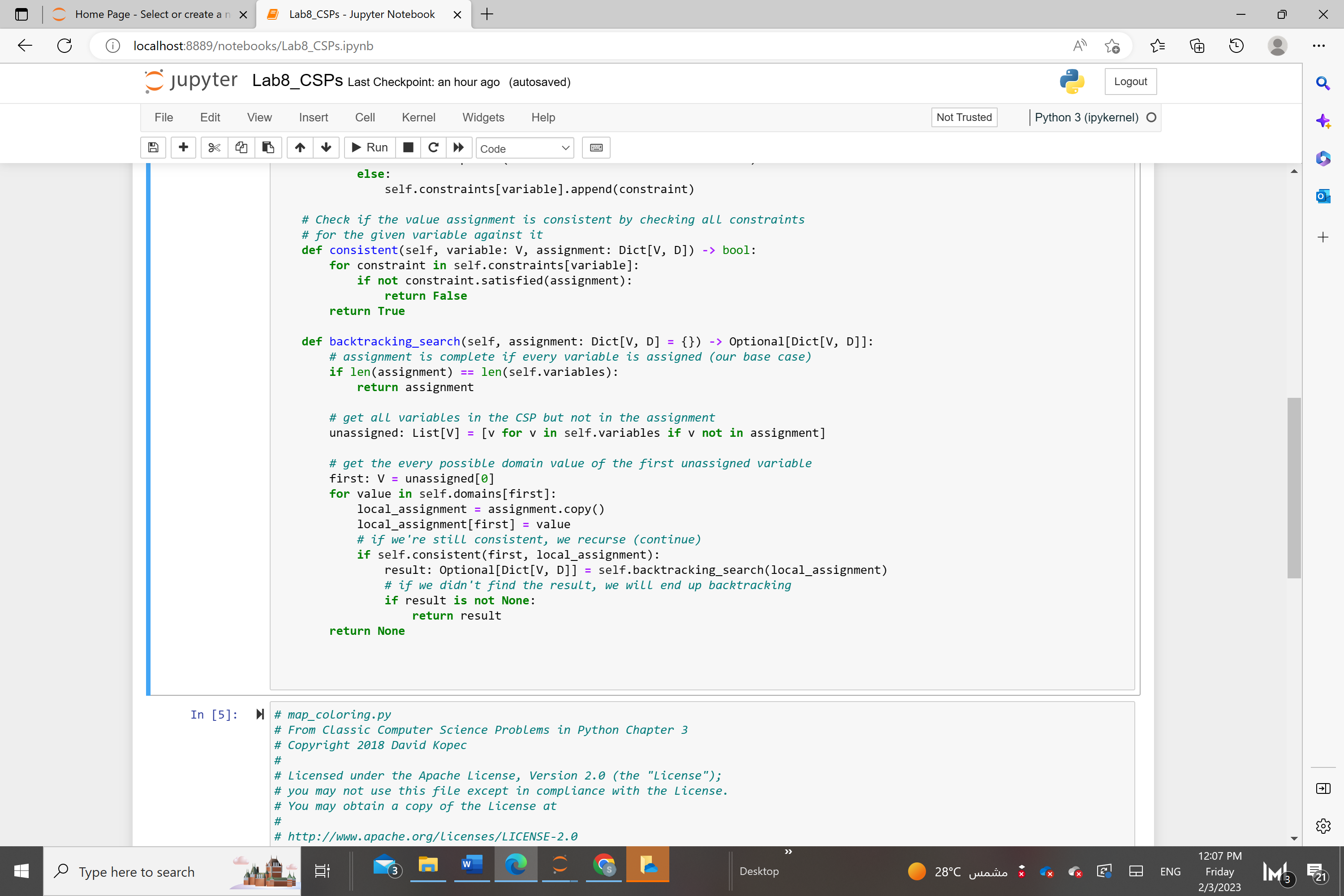


The centerpiece of our constraint-satisfaction framework is a class called CSP. CSP is the gathering point for variables, domains, and constraints. In terms of its type hints, it uses generics to make itself flexible enough to work with any kind of variables and domain values (V keys and D domain values). Within CSP, the definitions of the collections variables, domains, and constraints are of types that you’d expect. The variables collection is a list of variables, domains is a dict mapping variables to lists of possible values (the domains of those variables), and constraints is a dict that maps each variable to a list of the constraints imposed on it.



The \_\_init\_\_() initializer creates the constraints dict. The add\_constraint() method goes through all of the variables touched by a given constraint and adds itself to the constraints mapping for each of them. Both methods have basic error-checking in place, and raise an exception when a variable is missing a domain or a constraint is on a nonexistent variable.

How do we know if a given configuration of variables and selected domain values satisfy the constraints? We’ll call such a given configuration an “assignment.” We need a function that checks every constraint for a given variable against an assignment to see if the variable’s value in the assignment works for the constraints. Here we implement a consistent() function as a method on CSP.



consistent() goes through every constraint for a given variable (it’s always a variable that was newly added to the assignment) and checks if the constraint is satisfied, given the new assignment. If the assignment satisfies every constraint, True is returned. If any constraint imposed on the variable isn’t satisfied, False is returned.

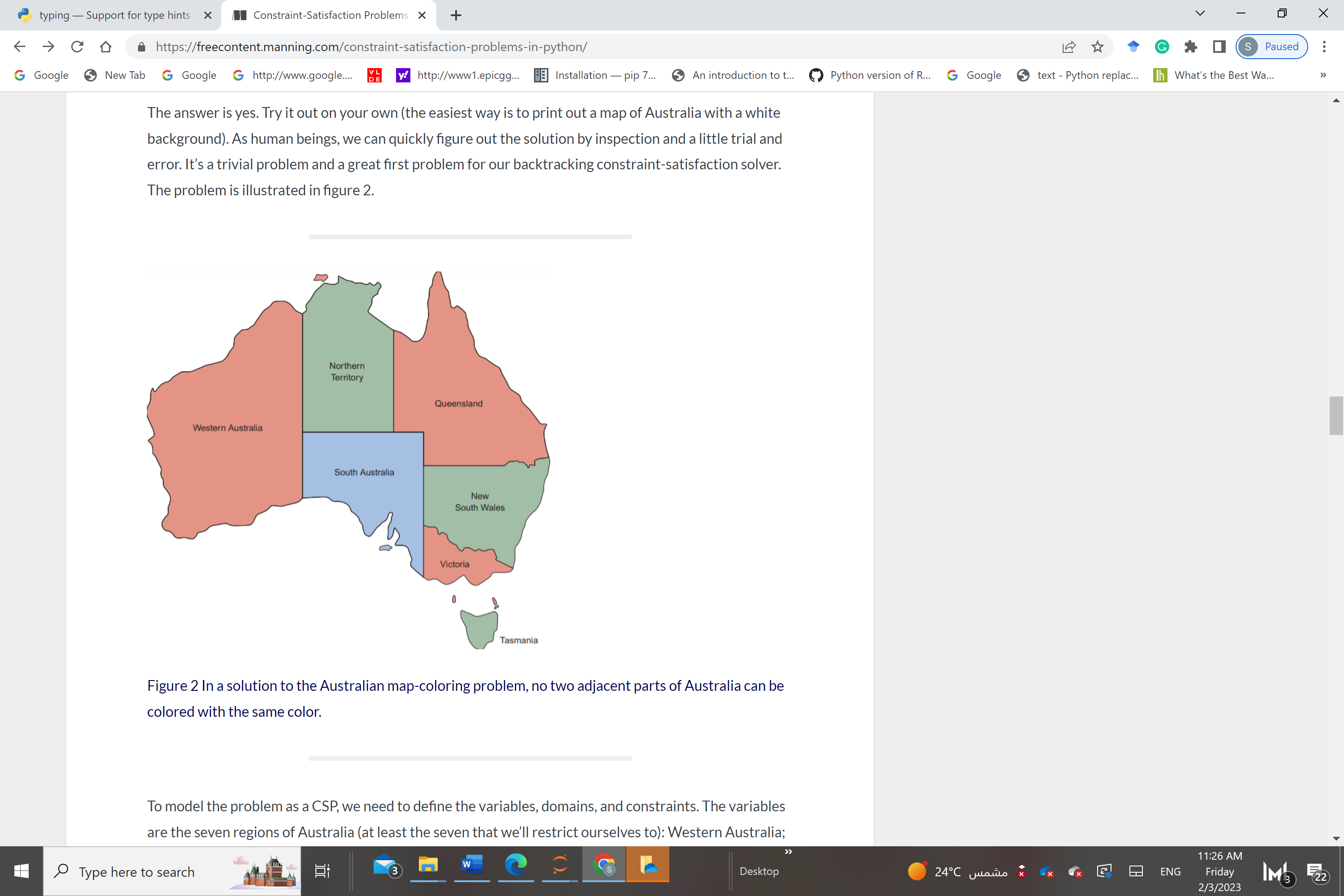
This constraint-satisfaction framework uses a simple backtracking search to find solutions to problems. *Backtracking* is the idea that once you hit a wall in your search, you go back to the last known point where you made a decision before the wall, and choose a different path. The backtracking search implemented in the following backtracking\_search() function is a kind of recursive depth-first search. This function’s added as a method to the CSP class.

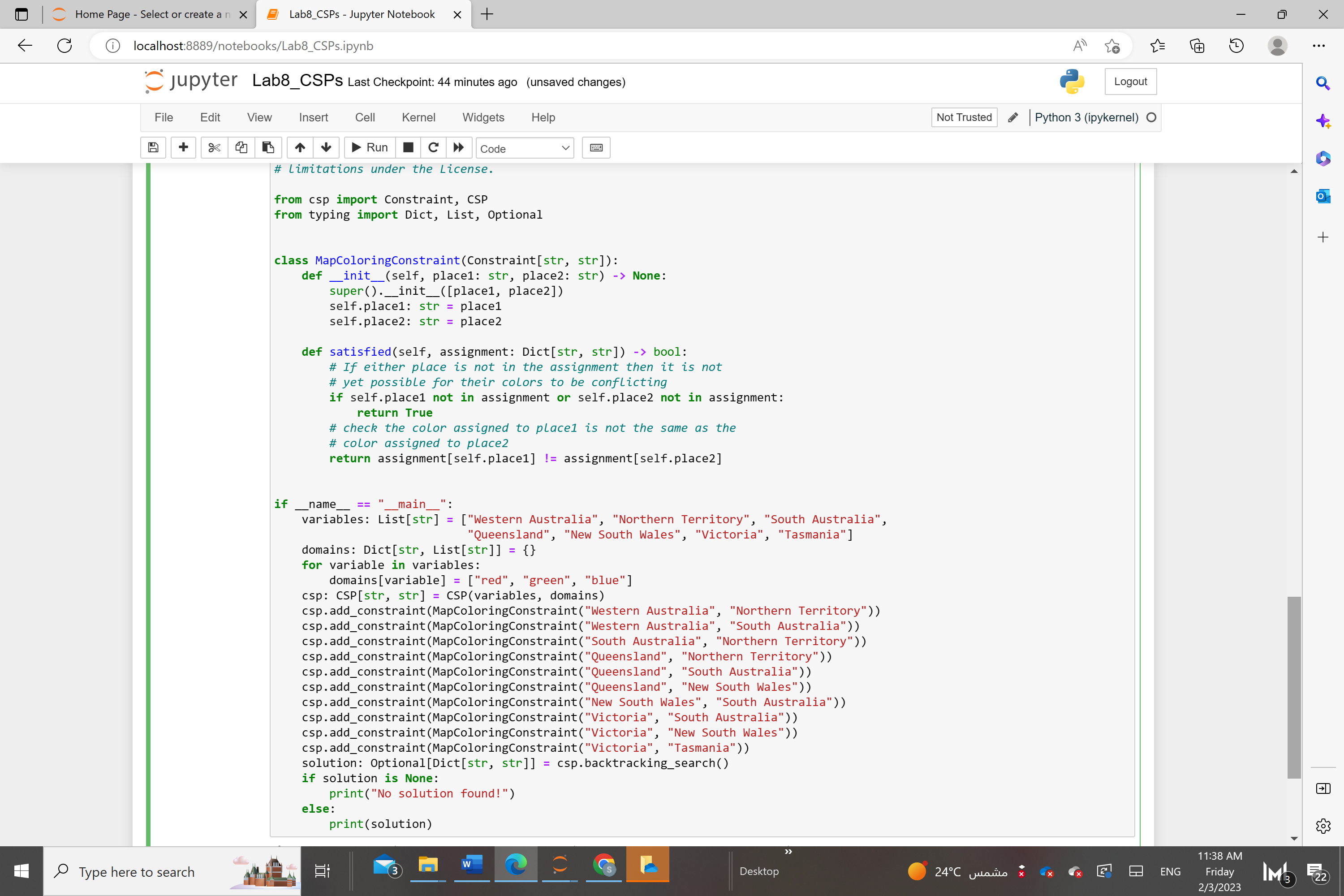
Graphical user interface, text

Description automatically generated

**The Australian map-coloring problem**

Imagine you have a map of Australia that you want to color by state/territory (which we’ll collectively call “regions”). No two adjacent regions should share a color. Can you color the regions with only three different colors?





Text

Description automatically generated

Text

Description automatically generated

**Assessment:**

You are required to implement this lab and submit your work showing the code and results.