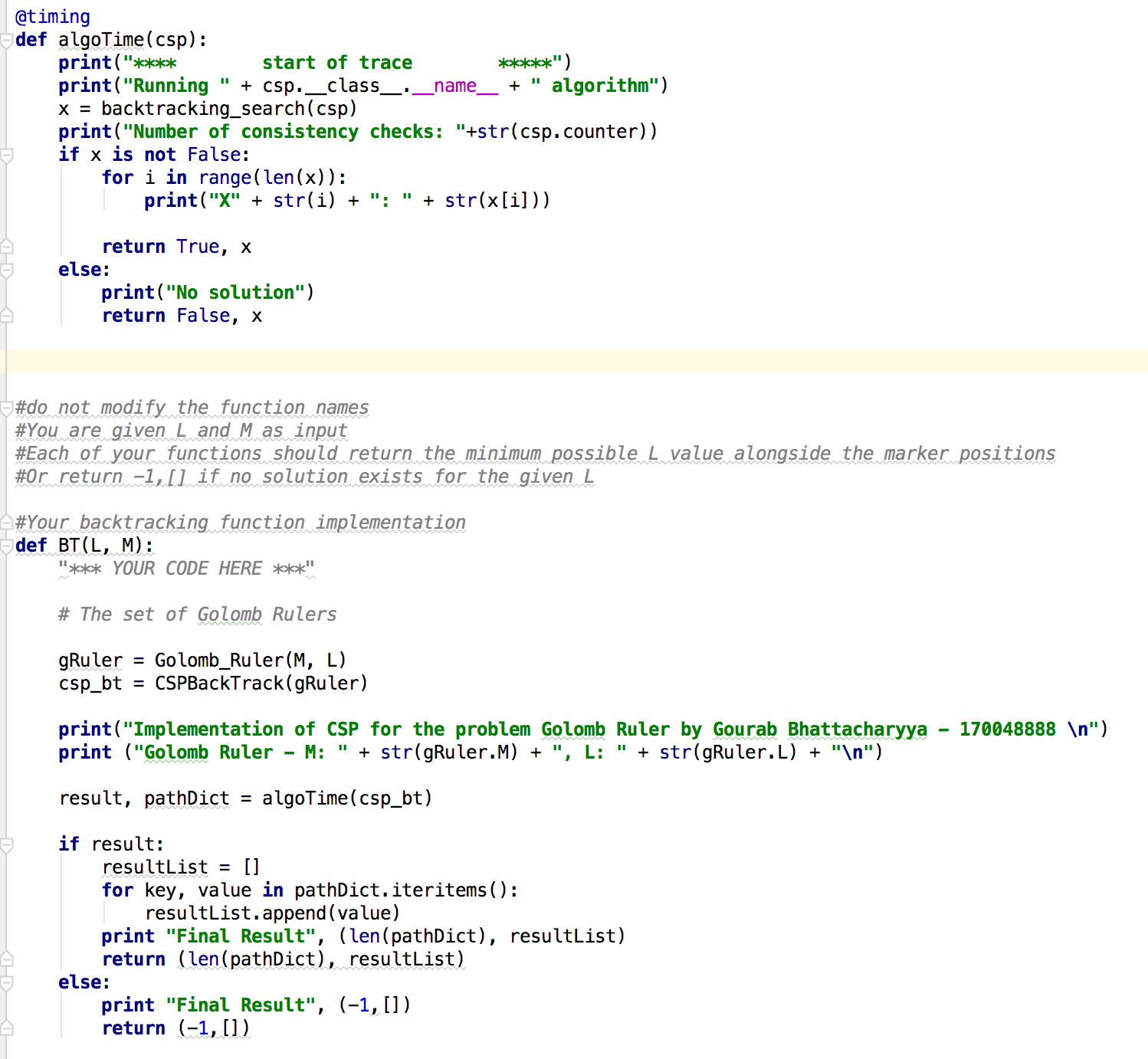
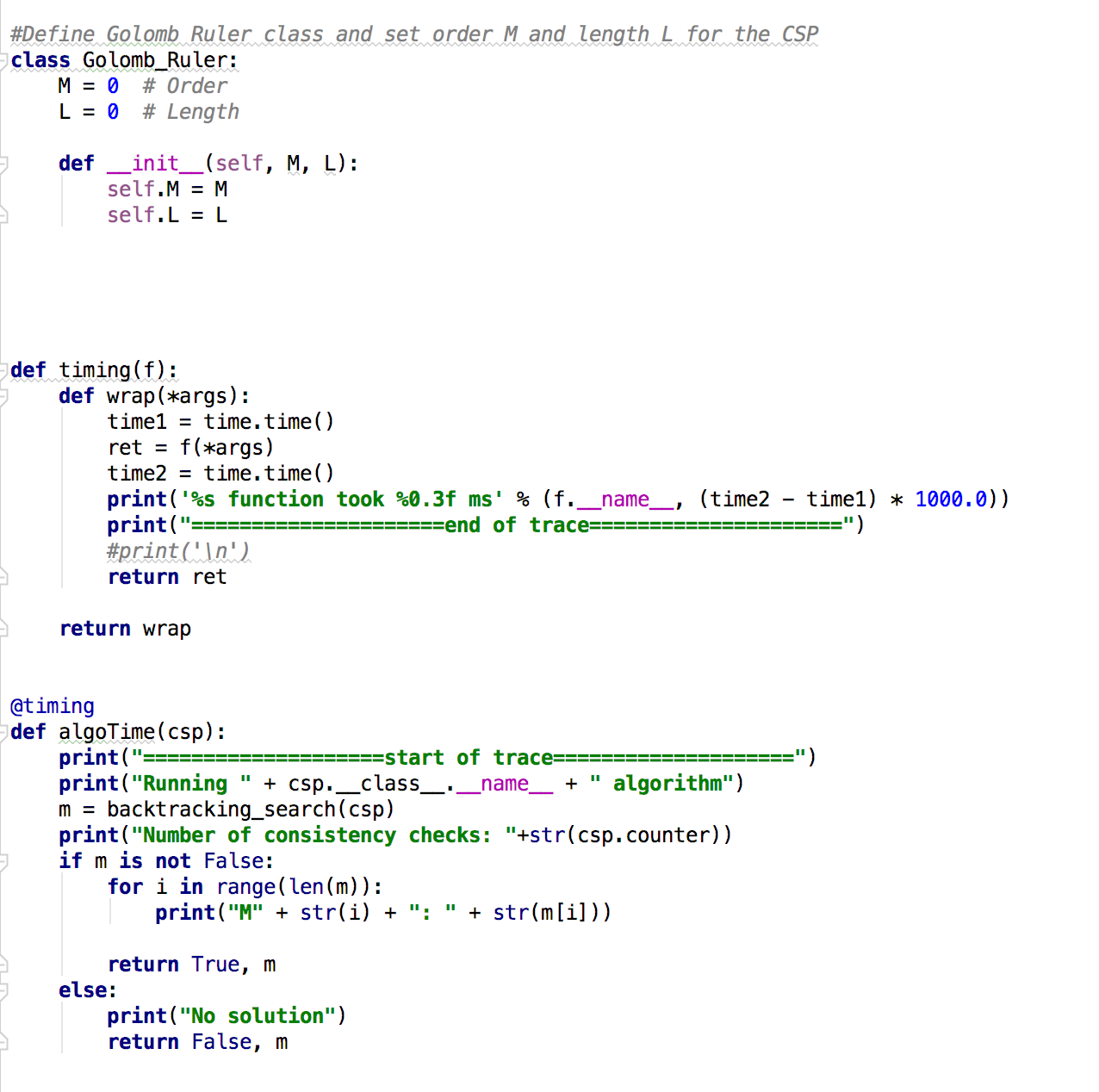
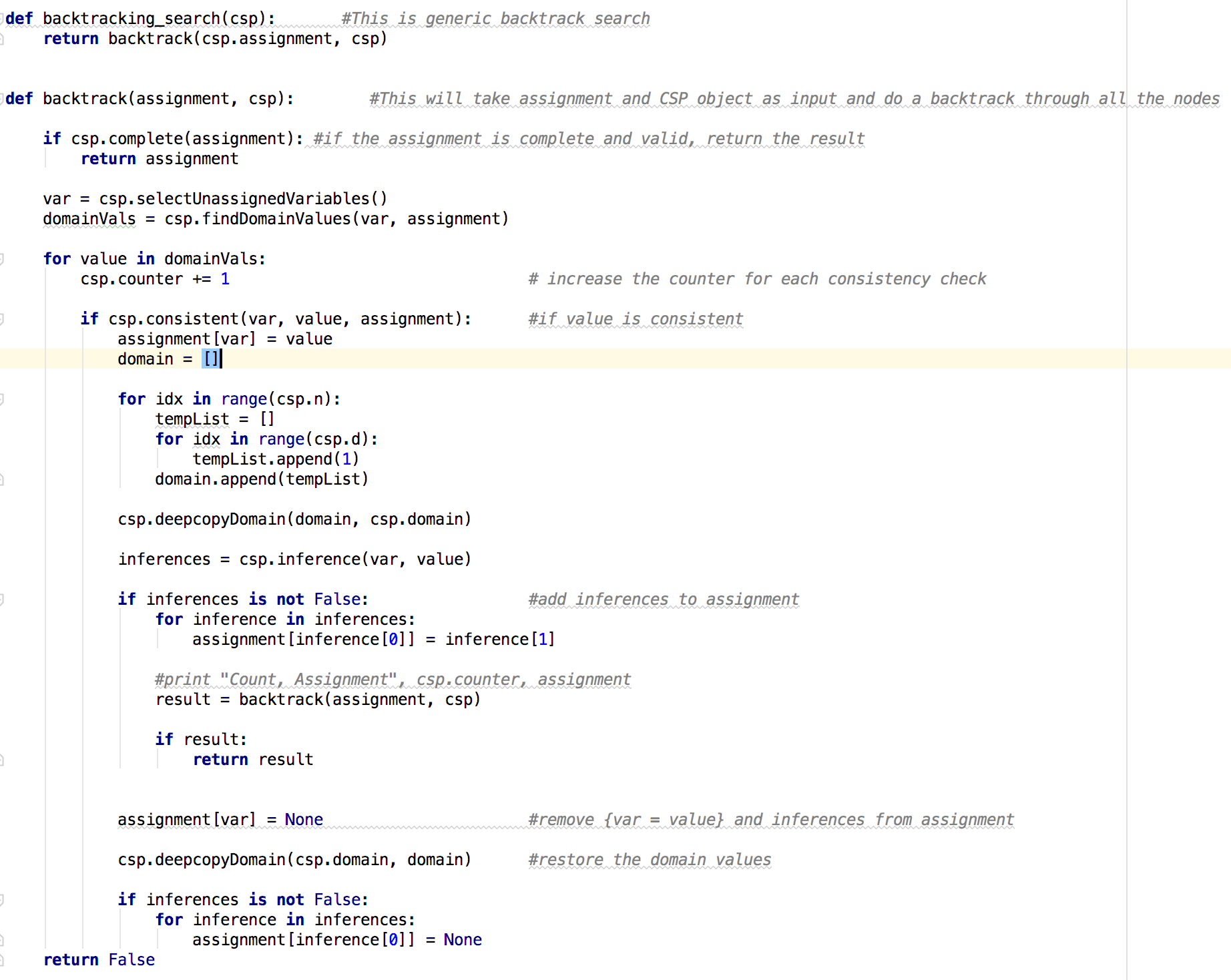
**HW3 Project Details Document**

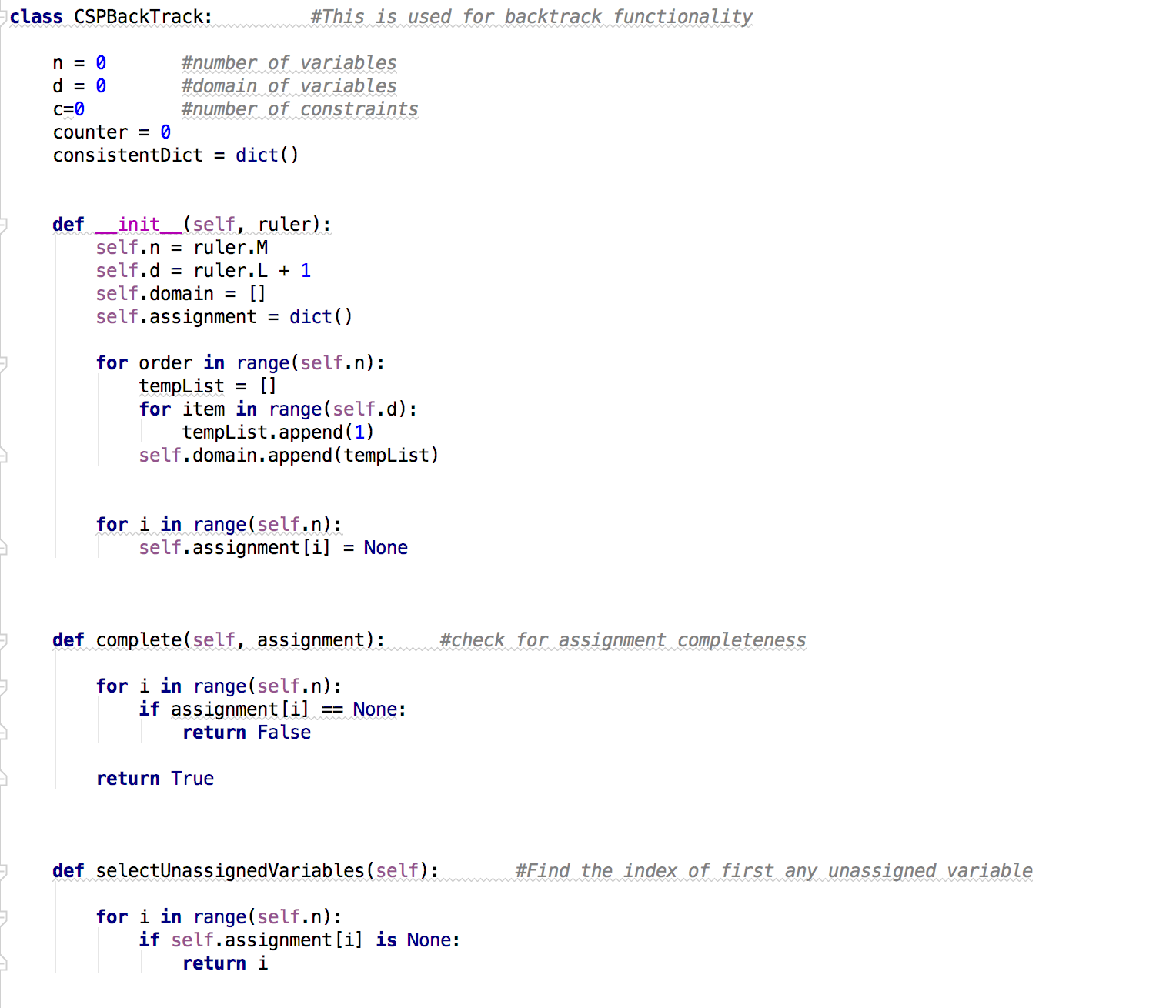
1. ***Question 1***Plain Backtracking (BT):

* **Code Details:**

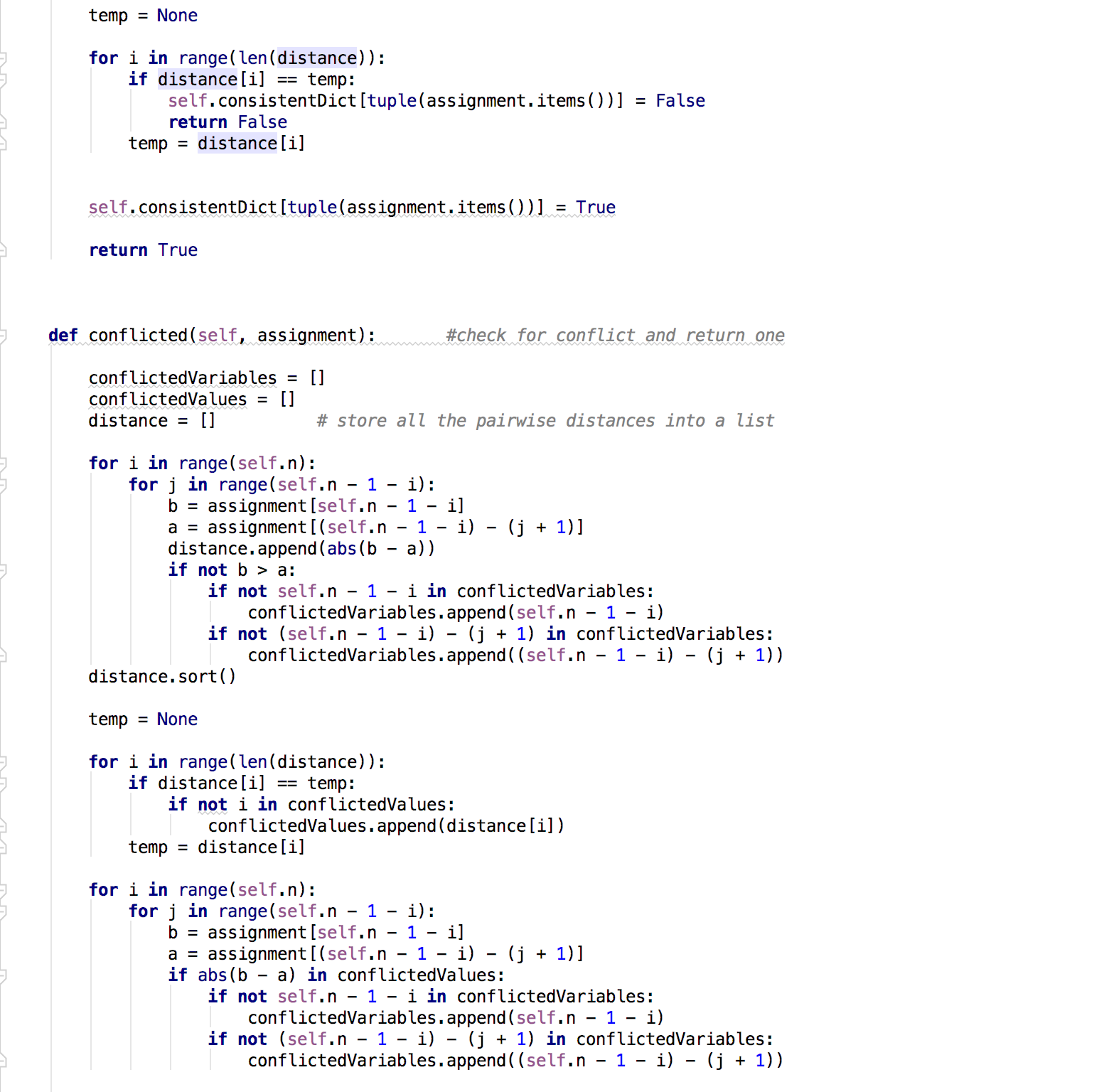


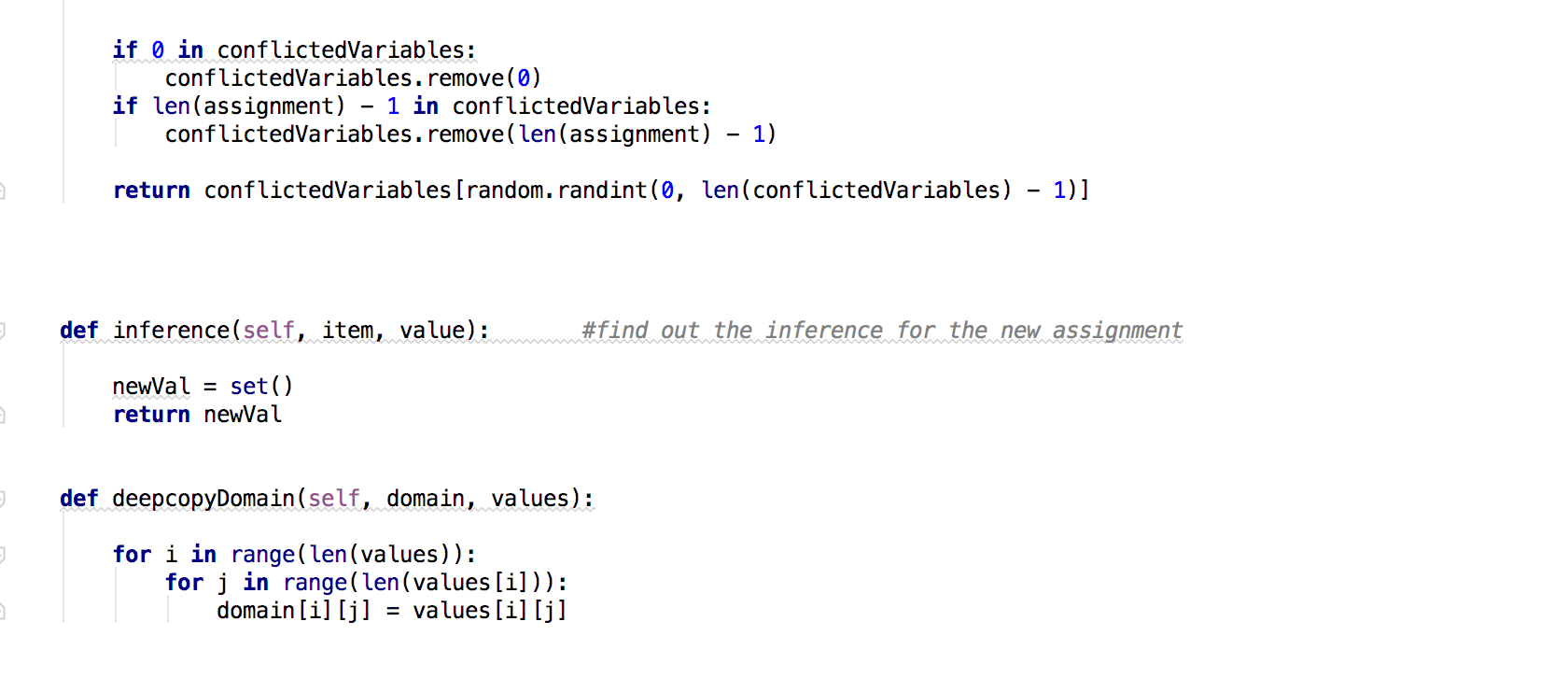








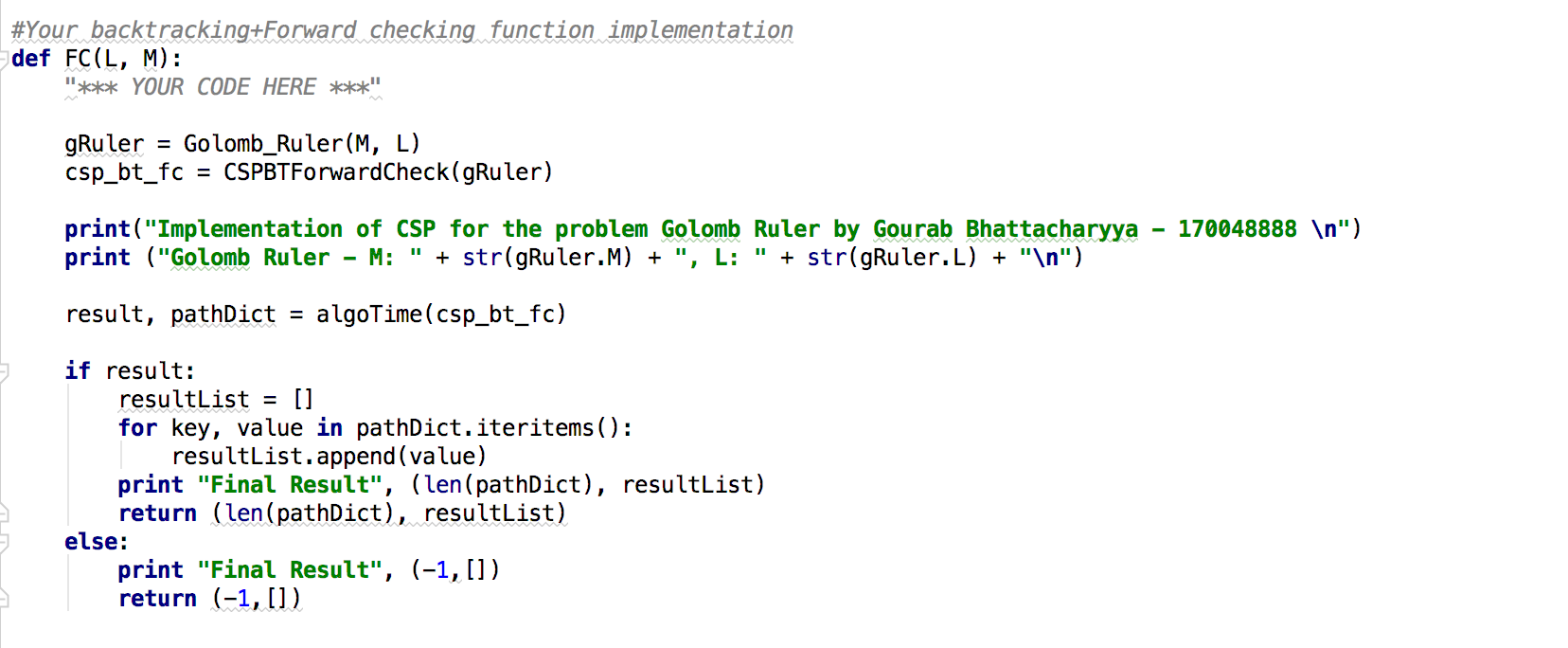


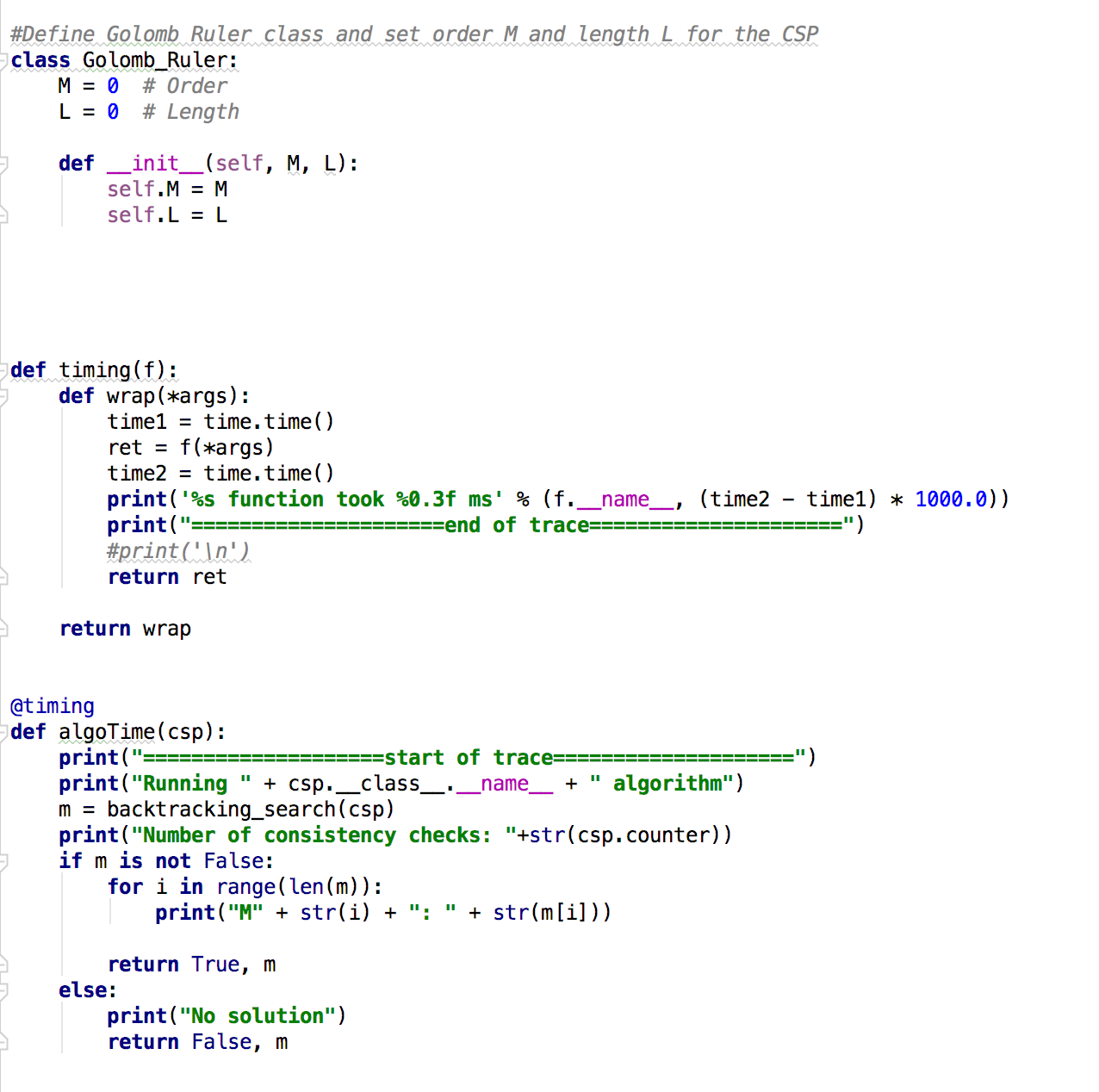


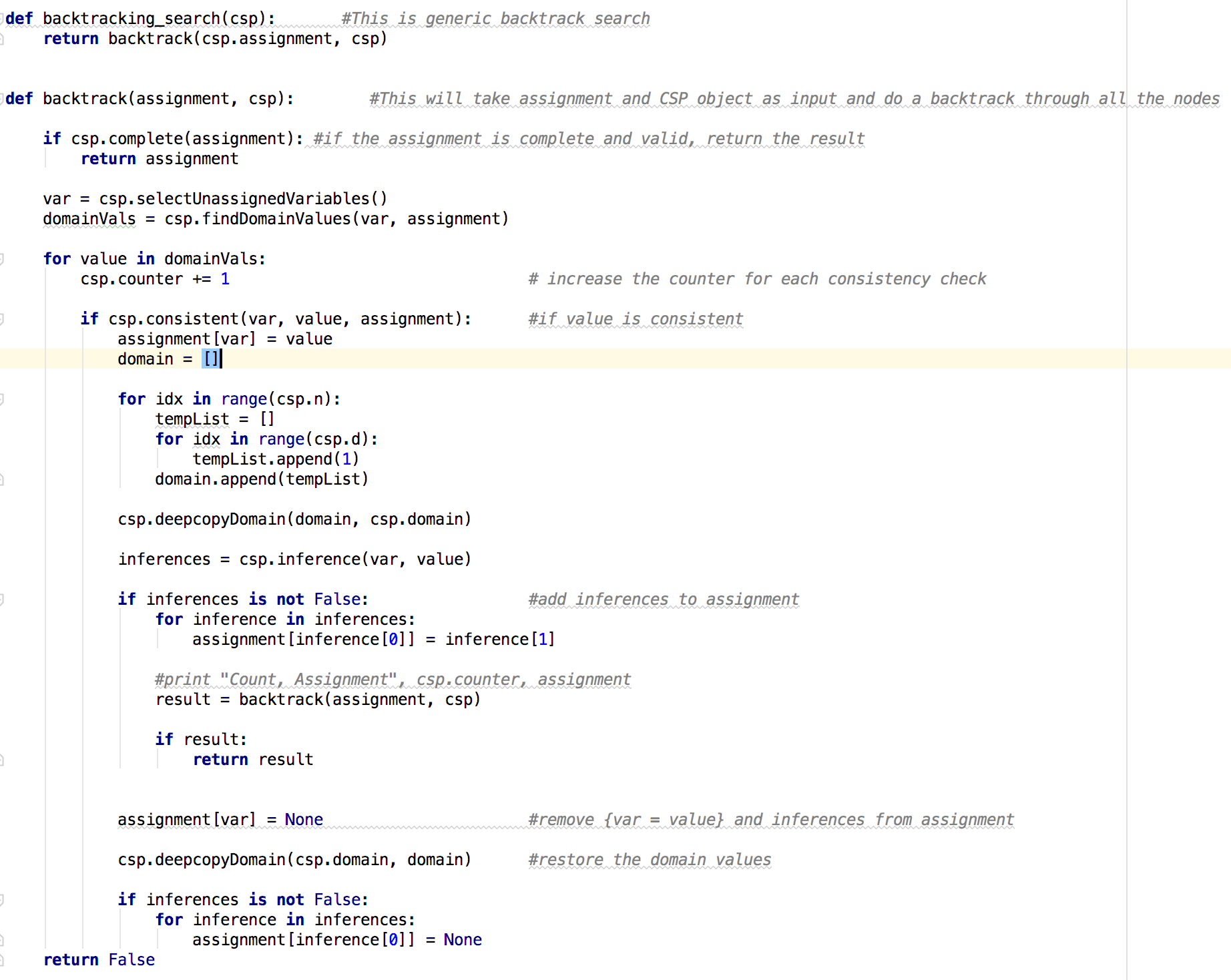
* The 3 main modules of this Backtracking implementation are as BT(), backtracking\_search() and CSPBackTrack.
* BT() module:
  + The first module BT() is the actual thread where the call will begin and move forward.
  + Here, I am first creating a golomb ruler object with the input parameters length and marks.
  + Then calling algoTime() module which in turn calls backtracking module.
  + This method returns pathDict dictionary which holdes the path from the initial variable to the end variable is there is no failure.
  + Return the output as expected format [(len(pathDict), resultList)]from this module.
* Backtracking\_search():
  + In this module I am using the CSP assignment and the CSP problem itself to identify the variables that can be assigned with legal values.
  + If CSP is complete and consistent then return the assignment else continue.
  + Get all the unassigned variables.
  + For each value in the domain check for consistency of the assignment.
  + If the new value is consistent then add the same in assignment dictionary.
  + Call the same module recursively to find out all the vaqlid assignments for the variables.
  + Once all the values are identified then set this new value as the domain of the values of the input variables.
  + At any point if the assignment is not consistent then return a failure message
* CSPBackTrack:
  + This is a class module which is getting instantiated with the golomb ruler object.
  + For all the new assignment, this module checks for the completeness of the values
  + Finds the index of the next unassigned variables
  + Find all the possible values that can be available in the domain of a variable
  + For a newly assigned variable value this module checks for the consistency and if it’s find any issue then returns a failure
  + This also check for conflicts of a variable value with its neighbor variable values
  + Finally, once a value is found then add that value in the domain

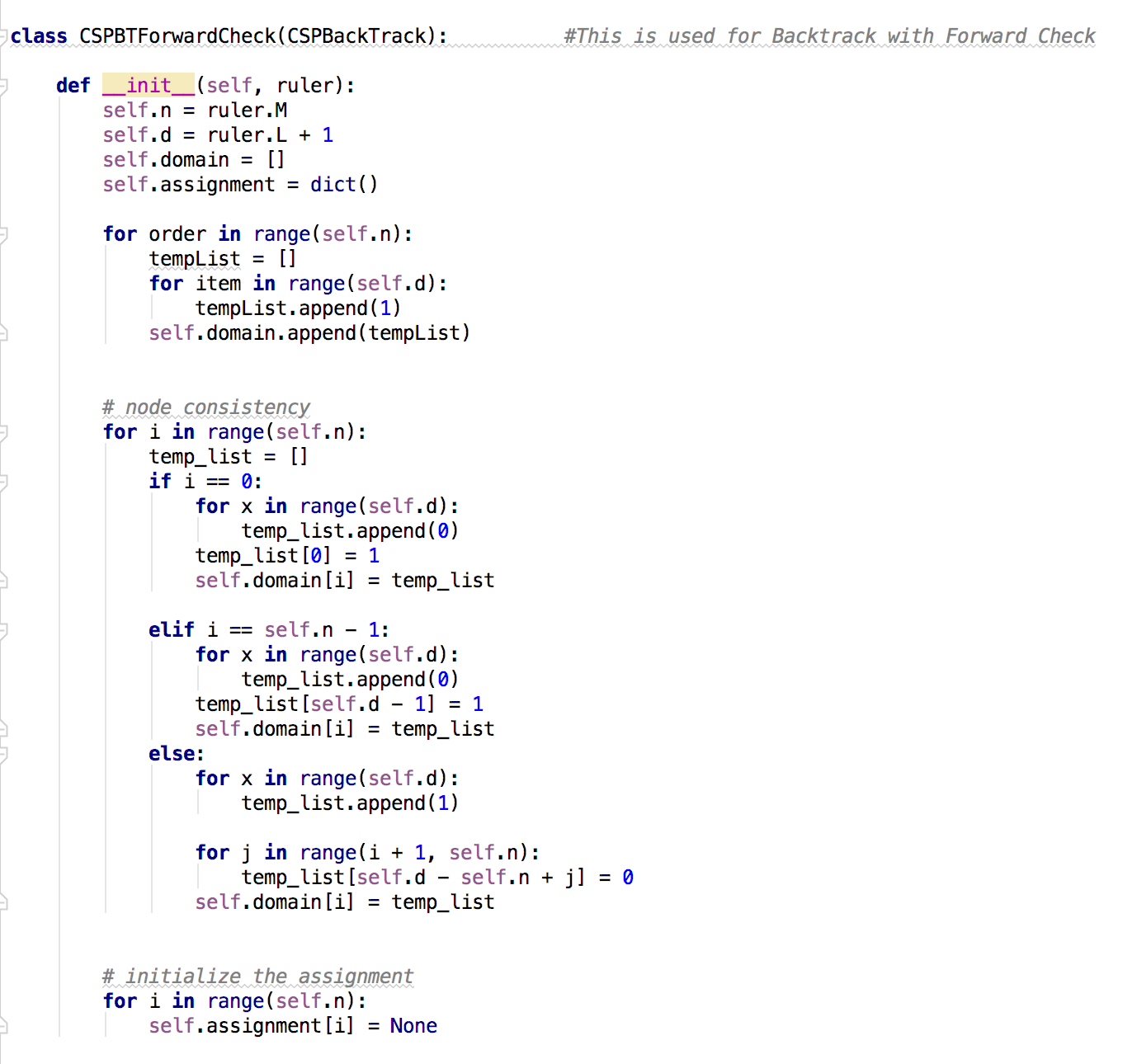
1. **Question 2**BT + Forward Checking (FC):

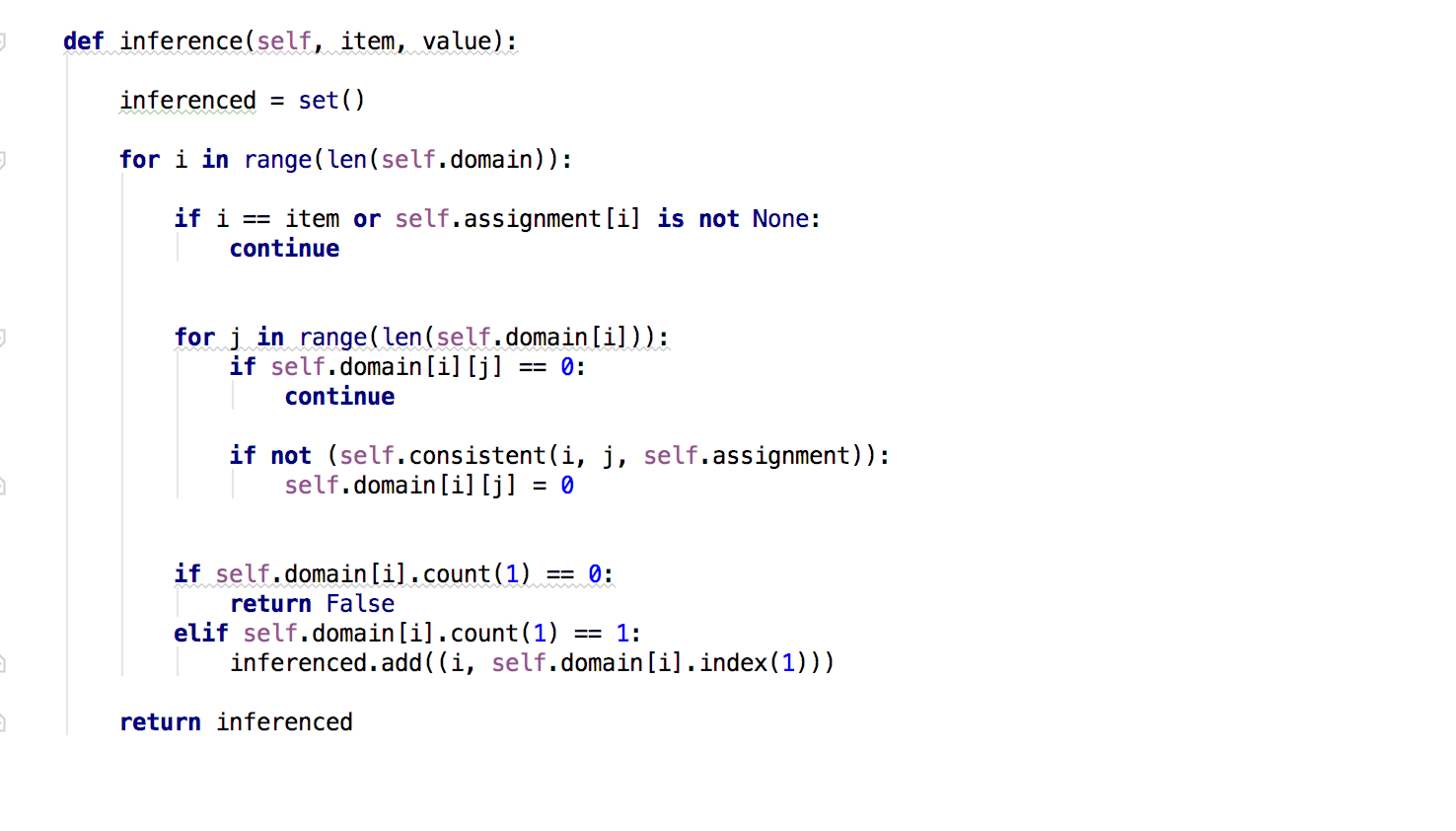
* **Code Details:**







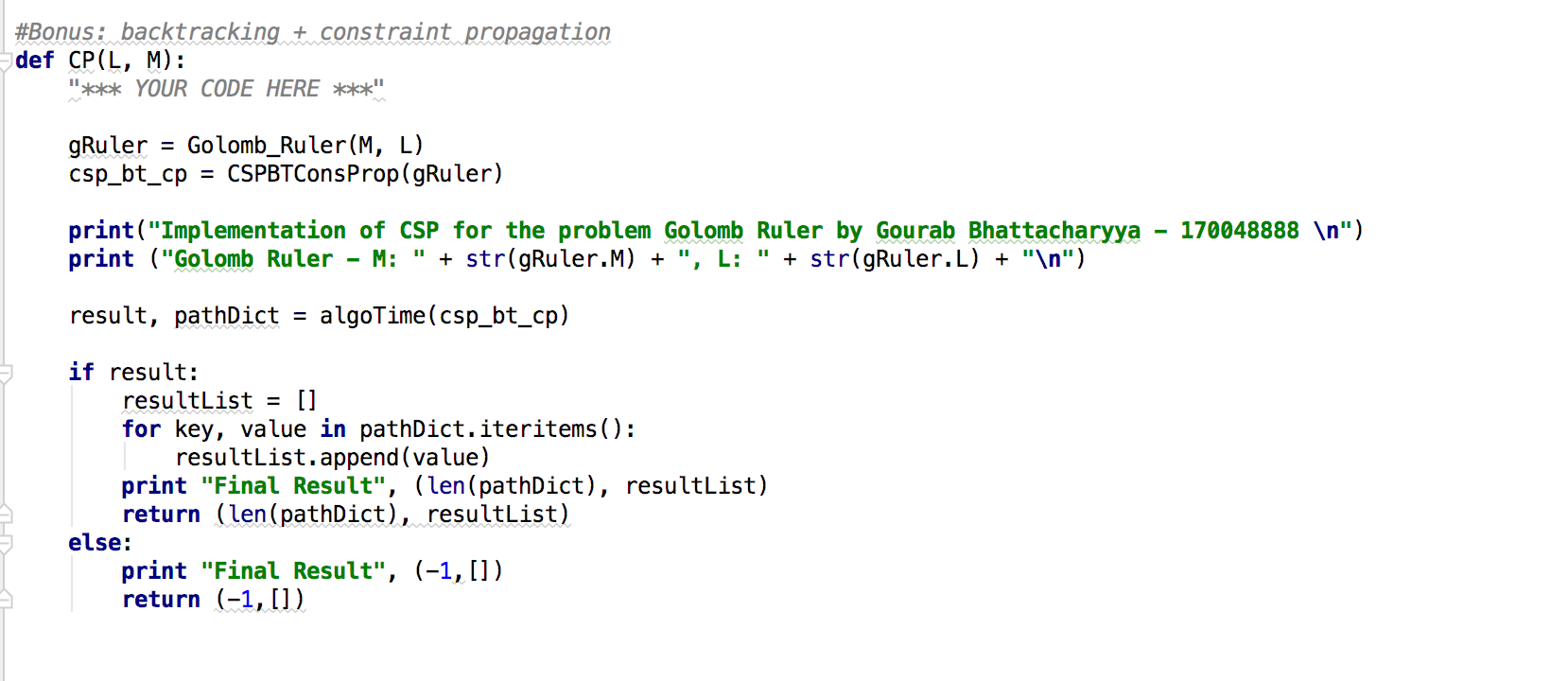


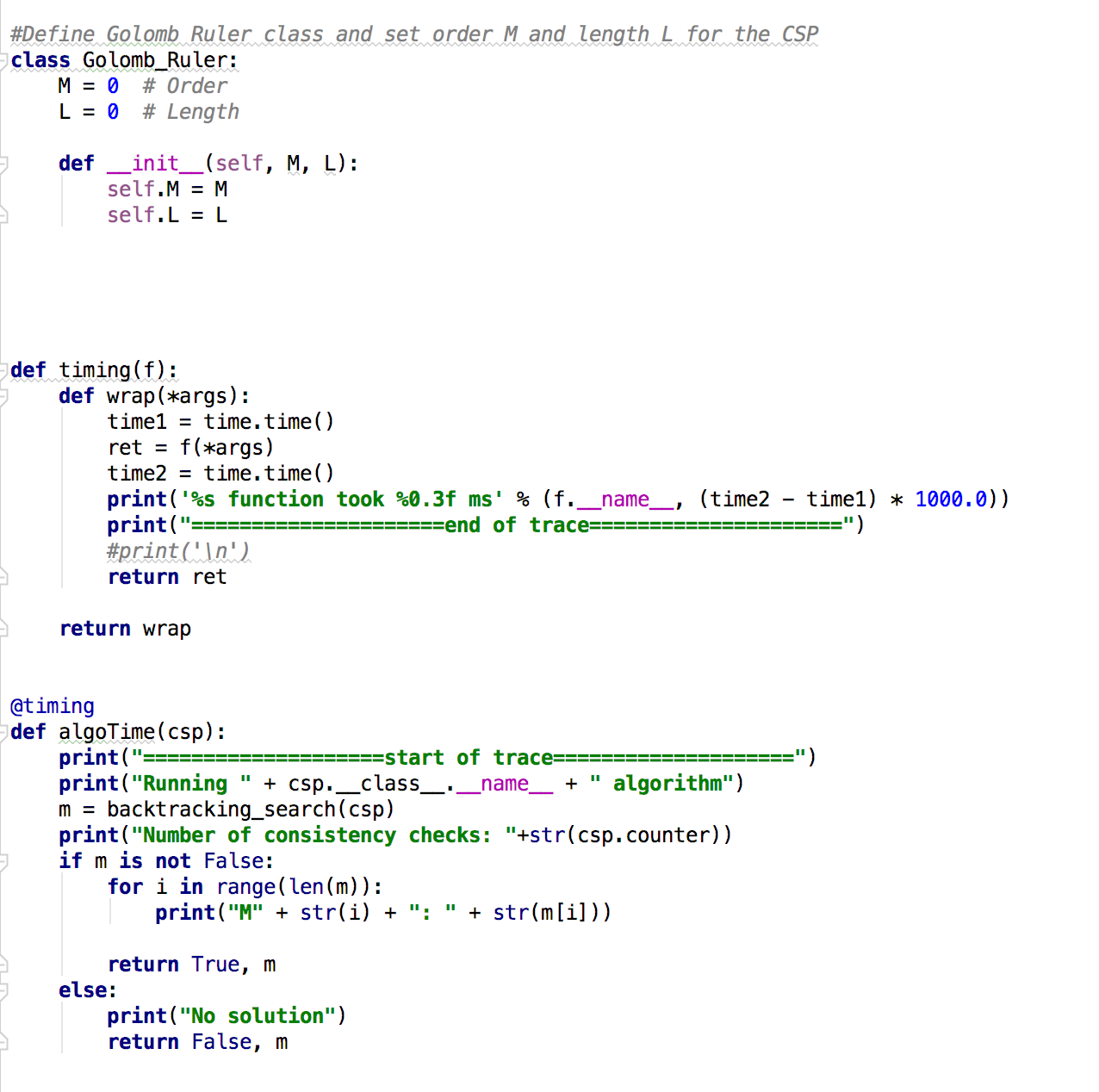


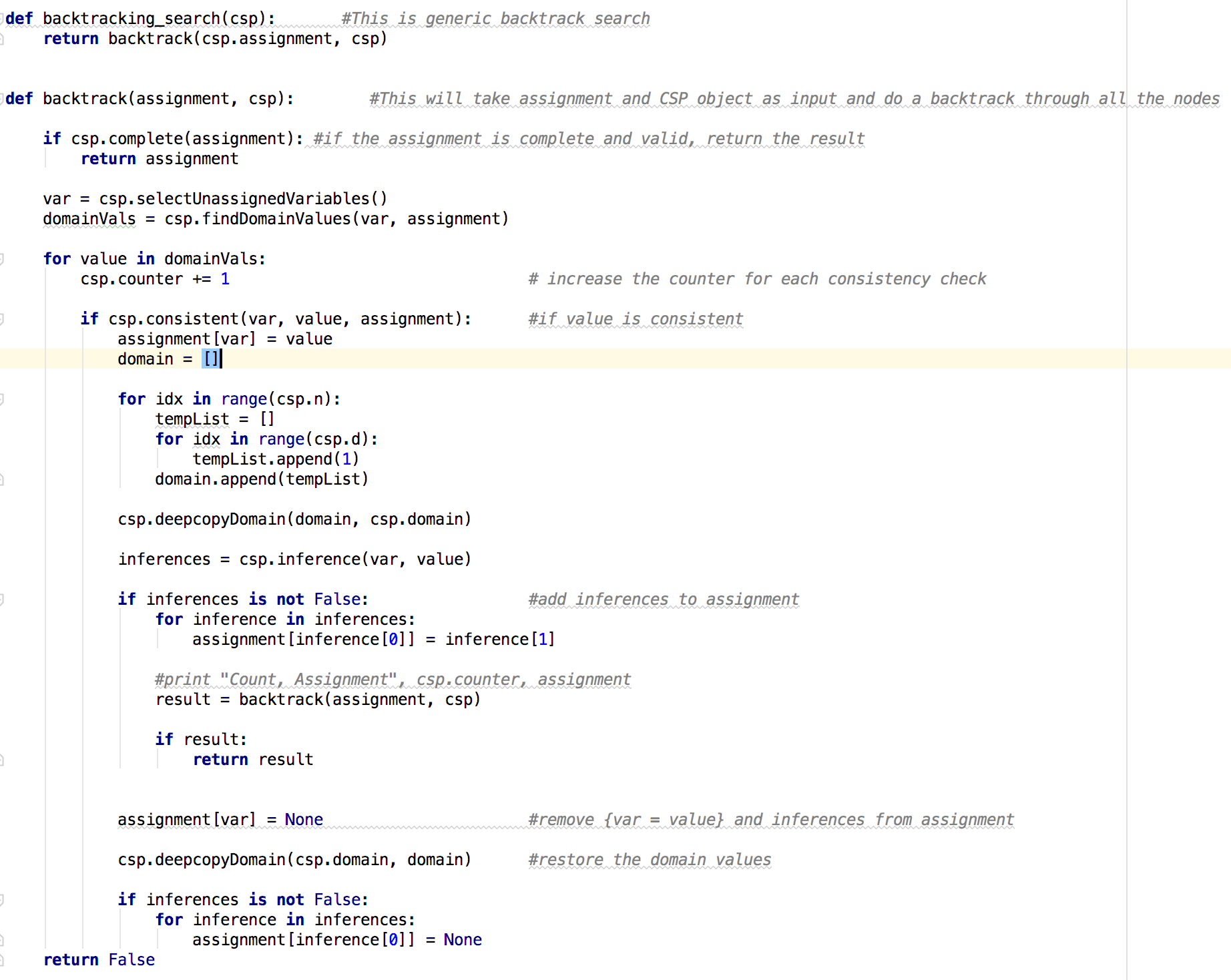
* The 3 main modules of this Backtracking implementation are as FC(), backtracking\_search() and CSPBTForwardCheck.
* FC() module:
  + The first module FC() is the actual thread where the call will begin and move forward.
  + Here, I am first creating a golomb ruler object with the input parameters length and marks.
  + Then calling algoTime() module which in turn calls backtracking module.
  + This method returns pathDict dictionary which holdes the path from the initial variable to the end variable is there is no failure.
  + Return the output as expected format [(len(pathDict), resultList)]from this module.
* Backtracking\_search():
  + In this module I am using the CSP assignment and the CSP problem itself to identify the variables that can be assigned with legal values.
  + If CSP is complete and consistent then return the assignment else continue.
  + Get all the unassigned variables.
  + For each value in the domain check for consistency of the assignment.
  + If the new value is consistent then add the same in assignment dictionary.
  + Call the same module recursively to find out all the vaqlid assignments for the variables.
  + Once all the values are identified then set this new value as the domain of the values of the input variables.
  + At any point if the assignment is not consistent then return a failure message
* CSPBTForwardCheck:
  + This is a class module which is getting instantiated with the golomb ruler object.
  + First, I am specifying a domain for the variables with all 1’s
  + Then checking for consistency and updating the domain accordingly.
  + Initializing the assignment dictionary.
  + Initialize a new set for inference values
  + If a new value found which is consistent then add the same in the assighnment and then in the domain of values.
  + This module will finally return the final domain of the valid variable values

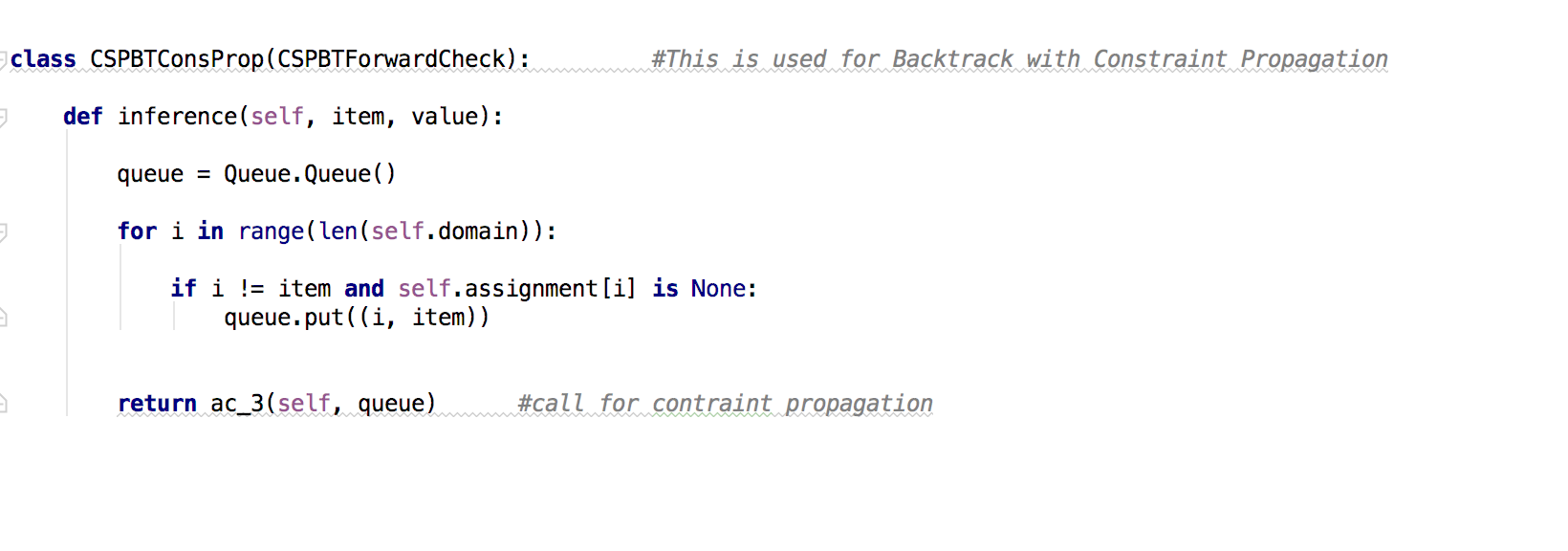
1. **Question 3** BT + Constraint Propagation (CP):

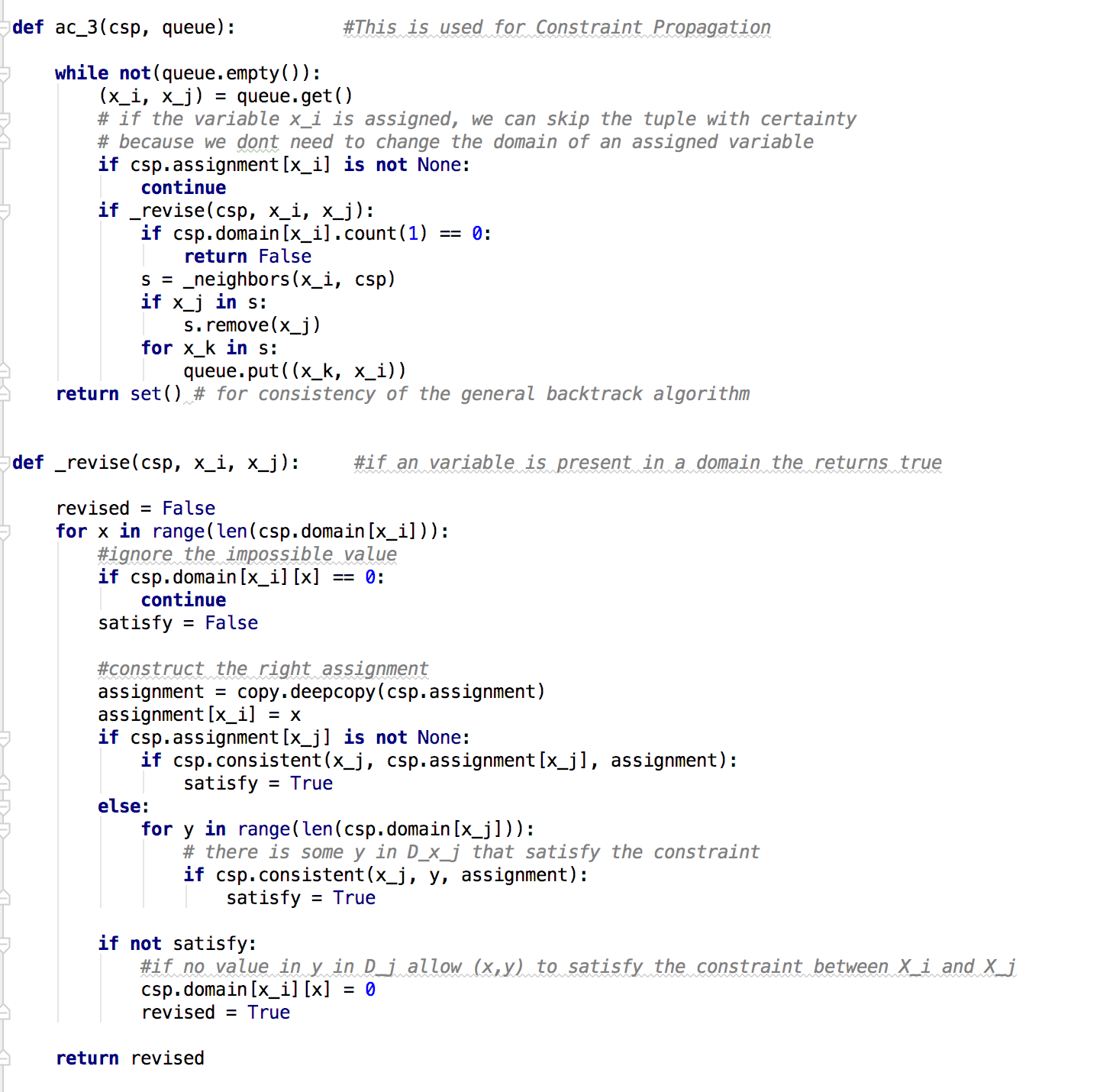
* **Code Details:**











* The 3 main modules of this Backtracking implementation are as CP(), backtracking\_search() and CSPBTConsProp.
* CP() module:
  + The first module CP() is the actual thread where the call will begin and move forward.
  + Here, I am first creating a golomb ruler object with the input parameters length and marks.
  + Then calling algoTime() module which in turn calls backtracking module.
  + This method returns pathDict dictionary which holdes the path from the initial variable to the end variable is there is no failure.
  + Return the output as expected format [(len(pathDict), resultList)]from this module.
* Backtracking\_search():
  + In this module I am using the CSP assignment and the CSP problem itself to identify the variables that can be assigned with legal values.
  + If CSP is complete and consistent then return the assignment else continue.
  + Get all the unassigned variables.
  + For each value in the domain check for consistency of the assignment.
  + If the new value is consistent then add the same in assignment dictionary.
  + Call the same module recursively to find out all the vaqlid assignments for the variables.
  + Once all the values are identified then set this new value as the domain of the values of the input variables.
  + At any point if the assignment is not consistent then return a failure message
* CSPBTConsProp:
  + This is a class module which is getting instantiated with the golomb ruler object.
  + Here I am using a FIFO queue to insert all the item in a queue first and then check for inconsistency and completeness for each variable value.
  + This is calling ac\_3 module which is actually performing this check.
  + ac\_3() module:
    - for each element in the queue if a value is already assigned to a variable then we will skip that variable and go to the next element.
    - For each variable, we will check the constraint satisfaction with its neighbor.
    - We will ignore the impossible values
    - Construct the correct assignment of variable value
    - If there is some value that satisfy all the constraint then add that value in the domain values.
    - If the value is inconsistent then remove that value from the queue.
    - Finally, this module as a whole will return a set of domain values which are the legal and valid values for the input variables.