Started on	Thursday, 14 September 2023, 1:06 PM
State	Finished
Completed on	Thursday, 14 September 2023, 8:30 PM
Time taken	7 hours 23 mins
Marks	7.00/7.00
Grade	**1.00** out of 1.00 (**100**%)

Information

Introduction

This quiz is on constraint satisfaction problems (CSPs). You will need the csp module. Read the module code and documentation (doc strings). For all the questions in this quiz, the csp module is available on the quiz server–that is, you can import the module in your code; just remember to include the import statement in your answer.

In this quiz, we use Python dictionaries to represent assignments. The keys represent variable names (of type string) and the values are the values of the variables. For example `{'a': 3}` is an assignment in which variable *a* has taken the value 3.

The function `satisfies` from the `csp` module tests whether an assignment satisfies a constraint. For example `satisfies({'a':1, 'b':2}, lambda a: a > 0)` return `True`.

The function `scope` in the `csp` module returns the set of the names of formal parameters of a function. For example, `scope(lambda a, b: a + b)` returns `{'a', 'b'}`.

Optional activity:

The csp applet may help you better understand the arc consistency and domain splitting algorithms. An extensive tutorial is available here: http://aispace.org/constraint/. Note that this activity is strictly optional—you won't be using the applet in any of the questions.

Question 1

Correct

Mark 1.00 out of 1.00

Write a function `generate_and_test` that takes a CSP object and returns an iterable (e.g. list, tuple, set, generator, ...) of solutions. A solution is a complete assignment that satisfies all the constraints.

Notes:

1. `itertools.product` may be useful here.
2. If you wish, you can build your solution on the partial code preloaded in the answer box.
3. Remember to include all the required import statements (including csp).
4. The second test case is a crossword puzzle which is visualised here.

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>from csp import *
from student_answer import generate_and_test

simple_csp = CSP(
    var_domains={x: set(range(1, 5)) for x in 'abc'},
    constraints={
        lambda a, b: a < b,
        lambda b, c: b < c,
        })

solutions = sorted(str(sorted(solution.items())) for solution
                   in generate_and_test(simple_csp))
print("\n".join(solutions))</pre></td><td><pre>[('a', 1), ('b', 2), ('c', 3)]
[('a', 1), ('b', 2), ('c', 4)]
[('a', 1), ('b', 3), ('c', 4)]
[('a', 2), ('b', 3), ('c', 4)]</pre></td></tr>
<tr><td><pre>from csp import *
from student_answer import generate_and_test
import itertools

crossword_puzzle = CSP(
    var_domains={
        # read across:
        'a1': set("ant,big,bus,car".split(',')),
        'a3': set("book,buys,hold,lane,year".split(',')),
        'a4': set("ant,big,bus,car,has".split(',')),
        # read down:
        'd1': set("book,buys,hold,lane,year".split(',')),
        'd2': set("ginger,search,symbol,syntax".split(',')),
        },
    constraints={
        lambda a1, d1: a1[0] == d1[0],
        lambda d1, a3: d1[2] == a3[0],
        lambda a1, d2: a1[2] == d2[0],
        lambda d2, a3: d2[2] == a3[2],
        lambda d2, a4: d2[4] == a4[0],
        })

solution = next(iter(generate_and_test(crossword_puzzle)))

# printing the puzzle similar to the way it actually looks
pretty_puzzle = ["".join(line) for line in itertools.zip_longest(
    solution['d1'], "", solution['d2'], fillvalue=" ")]
pretty_puzzle[0:5:2] = solution['a1'], solution['a3'], " " + solution['a4']
print("\n".join(pretty_puzzle))</pre></td><td><pre>bus
u e
year
s r
  car
  h</pre></td></tr>
</table>

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```python
from csp import *
import itertools

def generate_and_test(csp):
    names, domains = zip(*csp.var_domains.items())
    for values in itertools.product(*domains):
        assignment = {names[i]: values[i] for i in range(len(names))}
        if all([satisfies(assignment, constraint) for constraint in csp.constraint
            yield assignment
```

	Test	Expected	Got	

✓	from csp import *
	from student_answer import generate_and_test
	
	simple_csp = CSP(
	    var_domains={x: set(range(1, 5)) for x in 'abc'},
	    constraints={
	        lambda a, b: a < b,
	        lambda b, c: b < c,
	        })
	
	solutions = sorted(str(sorted(solution.items())) for solution
	                   in generate_and_test(simple_csp))
	print("\n".join(solutions))
		[('a', 1), ('b', 2), ('c', 3)]
		[('a', 1), ('b', 2), ('c', 4)]
		[('a', 1), ('b', 3), ('c', 4)]
		[('a', 2), ('b', 3), ('c', 4)]
			[('a', 1), ('b', 2), ('c', 3)]
			[('a', 1), ('b', 2), ('c', 4)]
			[('a', 1), ('b', 3), ('c', 4)]
			[('a', 2), ('b', 3), ('c', 4)]
				✓

✓	from csp import *
	from student_answer import generate_and_test
	import itertools
	
	crossword_puzzle = CSP(
	    var_domains={
	        # read across:
	        'a1': set("ant,big,bus,car".split(',')),
	        'a3': set("book,buys,hold,lane,year".split(',')),
	        'a4': set("ant,big,bus,car,has".split(',')),
	        # read down:
	        'd1': set("book,buys,hold,lane,year".split(',')),
	        'd2': set("ginger,search,symbol,syntax".split(',')),
	        },
	    constraints={
	        lambda a1, d1: a1[0] == d1[0],
	        lambda d1, a3: d1[2] == a3[0],
	        lambda a1, d2: a1[2] == d2[0],
	        lambda d2, a3: d2[2] == a3[2],
	        lambda d2, a4: d2[4] == a4[0],
	        })
	
	solution = next(iter(generate_and_test(crossword_puzzle)))
	
	# printing the puzzle similar to the way it actually looks
	pretty_puzzle = ["".join(line) for line in itertools.zip_longest(
	    solution['d1'], "", solution['d2'], fillvalue=" ")]
	pretty_puzzle[0:5:2] = solution['a1'], solution['a3'], " " + solution['a4']
	print("\n".join(pretty_puzzle))
		bus
		u e
		year
		s r
		  car
		  h
			bus
			u e
			year
			s r
			  car
			  h
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Information

An instance of the constraint satisfaction problem can be specified with a set of variables, their domains, and a set of constraints. Here we use the class `CSP` which is available in the csp module. The class has two attributes as the following:

- `var_domains`: this is a dictionary of items of the form *var:domain* where *var* is a variable name (a string) and *domain* is a set of values that can be taken by the variable.
- `constraints`: a set of functions or lambda expressions (anonymous functions) such that each function takes a subset of variables, evaluates a condition, and returns true or false accordingly.

Question 2

Correct

Mark 1.00 out of 1.00

Make the following CSP *arc consistent* by modifying the code (if necessary) and pasting the result in the answer box.

```
from csp import CSP

crossword_puzzle = CSP(
    var_domains={
        # read across:
        'across1': set("ant big bus car has".split()),
        'across3': set("book buys hold lane year".split()),
        'across4': set("ant big bus car has".split()),
        # read down:
        'down1': set("book buys hold lane year".split()),
        'down2': set("ginger search symbol syntax".split()),
        },
    constraints={
        lambda across1, down1: across1[0] == down1[0],
        lambda down1, across3: down1[2] == across3[0],
        lambda across1, down2: across1[2] == down2[0],
        lambda down2, across3: down2[2] == across3[2],
        lambda down2, across4: down2[4] == across4[0],
        })
```

Notes:

1. This CSP instance is for a crossword puzzle (visualised here). In the instance presented above, the domains contain only words that have the same length as the corresponding field in the puzzle. Another approach would be to have all the given words in all the domains and provide additional length constraints for variables (e.g. `lambda across1: len(across1) == 3`).
2. Start by copying the above program into your editor. Modify the domains and then paste in the result. Remember to include the import statement in your answer.
3. The expression `set("ant big bus car has".split())` is equivalent to `{"ant", "big", "bus", "car", "has"}`. However, it emphasises the fact that we can use any expression that evaluates to a set of values and that these values need not be hard-coded. For example we could use an expression to fetch the words from a file, database, web, or other sources. In your answer, you are free to use any expression you wish as long as it evaluates to the correct set of values.

For example:

Test	Result

`from student_answer import crossword_puzzle`
 	
`print(sorted(crossword_puzzle.var_domains['across1']))`
	`['bus', 'has']`

Answer: (penalty regime: 0, 20, ... %)

```
from csp import CSP

crossword_puzzle = CSP(
    var_domains={
        # read across:
        'across1': set("bus has".split()),
        'across3': set("lane year".split()),
        'across4': set("ant car".split()),
        # read down:
        'down1': set("buys hold".split()),
        'down2': set("search syntax".split()),
        },
    constraints={
        lambda across1, down1: across1[0] == down1[0],
        lambda down1, across3: down1[2] == across3[0],
        lambda across1, down2: across1[2] == down2[0],
        lambda down2, across3: down2[2] == across3[2],
        lambda down2, across4: down2[4] == across4[0],
        })
```

	Test	Expected	Got	

✓	`from student_answer import crossword_puzzle`
	
	`print(sorted(crossword_puzzle.var_domains['across1']))`
		`['bus', 'has']`	`['bus', 'has']`	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 3

Correct

Mark 1.00 out of 1.00

Make the following CSP *arc consistent* by modifying the code (if necessary) and pasting the result in the answer box.

```
from csp import CSP
canterbury_colouring = CSP(
    var_domains={
        'christchurch': {'red', 'green'},
        'selwyn': {'red', 'green'},
        'waimakariri': {'red', 'green'},
        },
    constraints={
        lambda christchurch, waimakariri: christchurch != waimakariri,
        lambda christchurch, selwyn: christchurch != selwyn,
        lambda selwyn, waimakariri: selwyn != waimakariri,
        })
```

Notes and remarks

- Remember to include the import statement in your answer.
- All the example (test) cases of this question are hidden.
- This instance of CSP is also an instance of the *graph colouring* problem with only two colours which can be solved (or decided) in linear time. So a generic CSP algorithm is not really the best choice to solve this problem. Nevertheless, this is a good exercise.
- Before making the network arc-consistent, think whether it could have any solution. Then think what would you expect the arc-consistency outcome to be. Finally make the network arc-consistent. The end-result may surprise you!
- Think what would happen if instead of three separate constraints, we had one constraint that is the conjunction of these three; that is, `christchurch != waimakariri and christchurch != selwyn and selwyn != waimakariri`.

Answer: (penalty regime: 0, 20, ... %)

```
from csp import CSP
canterbury_colouring = CSP(
    var_domains={
        'christchurch': {'red', 'green'},
        'selwyn': {'red', 'green'},
        'waimakariri': {'red', 'green'},
        },
    constraints={
        lambda christchurch, waimakariri: christchurch != waimakariri,
        lambda christchurch, selwyn: christchurch != selwyn,
        lambda selwyn, waimakariri: selwyn != waimakariri,
        })
```

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 4

Correct

Mark 1.00 out of 1.00

Write a function `arc_consistent` that takes a CSP object and returns a new CSP object that is arc consistent (and also consequently domain consistent). The general arc consistency (GAC) algorithm is in the lecture notes and also available in section 4.4 of the textbook.

If you wish, you can build your solution upon the partial code preloaded in the answer box. The code closely follows the pseudocode including the name of the variables.

For example:

Test	Result

`from csp import *`
`from student_answer import arc_consistent`
 	
`simple_csp = CSP(`
`    var_domains={x: set(range(1, 5)) for x in 'abc'},`
`    constraints={`
`        lambda a, b: a < b,`
`        lambda b, c: b < c,`
`        })`
 	
`csp = arc_consistent(simple_csp)`
`for var in sorted(csp.var_domains.keys()):`
`    print("{}: {}".format(var, sorted(csp.var_domains[var])))`
	`a: [1, 2]`
	`b: [2, 3]`
	`c: [3, 4]`

`from csp import *`
`from student_answer import arc_consistent`
 	
`csp = CSP(var_domains={x:set(range(10)) for x in 'abc'},`
`          constraints={lambda a,b,c: 2*a+b+2*c==10})`
 	
`csp = arc_consistent(csp)`
`for var in sorted(csp.var_domains.keys()):`
`    print("{}: {}".format(var, sorted(csp.var_domains[var])))`
	`a: [0, 1, 2, 3, 4, 5]`
	`b: [0, 2, 4, 6, 8]`
	`c: [0, 1, 2, 3, 4, 5]`

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```
import itertools, copy
from csp import *

def arc_consistent(csp):
    csp = copy.deepcopy(csp)
    to_do = {(x, c) for c in csp.constraints for x in csp.var_domains.keys()} #
    while to_do:
        x, c = to_do.pop()
        ys = scope(c) - {x}
        new_domain = set()
        for xval in csp.var_domains[x]: # COMPLETE
            assignment = {x: xval}
            for yvals in itertools.product(*[csp.var_domains[y] for y in ys]):
                assignment.update({y: yval for y, yval in zip(ys, yvals)})
                if satisfies(assignment, c): # COMPLETE
                    new_domain.add(xval) # COMPLETE
                    break
        if csp.var_domains[x] != new_domain:
            for cprime in set(csp.constraints) - {c}:
                if x in scope(cprime):
                    for z in scope(cprime): # COMPLETE
                        if x != z: # COMPLETE
                            to_do.add((z, cprime))
            csp.var_domains[x] = new_domain     #COMPLETE
    return csp
```

	Test	Expected	Got	

✓	`from csp import *`
	`from student_answer import arc_consistent`
	
	`simple_csp = CSP(`
	`    var_domains={x: set(range(1, 5)) for x in 'abc'},`
	`    constraints={`
	`        lambda a, b: a < b,`
	`        lambda b, c: b < c,`
	`        })`
	
	`csp = arc_consistent(simple_csp)`
	`for var in sorted(csp.var_domains.keys()):`
	`    print("{}: {}".format(var, sorted(csp.var_domains[var])))`
		`a: [1, 2]`
		`b: [2, 3]`
		`c: [3, 4]`
			`a: [1, 2]`
			`b: [2, 3]`
			`c: [3, 4]`
				✓

✓	`from csp import *`
	`from student_answer import arc_consistent`
	
	`csp = CSP(var_domains={x:set(range(10)) for x in 'abc'},`
	`          constraints={lambda a,b,c: 2*a+b+2*c==10})`
	
	`csp = arc_consistent(csp)`
	`for var in sorted(csp.var_domains.keys()):`
	`    print("{}: {}".format(var, sorted(csp.var_domains[var])))`
		`a: [0, 1, 2, 3, 4, 5]`
		`b: [0, 2, 4, 6, 8]`
		`c: [0, 1, 2, 3, 4, 5]`
			`a: [0, 1, 2, 3, 4, 5]`
			`b: [0, 2, 4, 6, 8]`
			`c: [0, 1, 2, 3, 4, 5]`
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Information

Another way of representing a CSP is by a collection of *relations*. There will be one relation per constraint in the problem. A relation is basically a table that holds zero or more rows. The columns of the table are the variables that are in the scope of the constraint. See the documentation of the `Relation` class in the `csp module.`

For example, the following instance of CSP

```
CSP(
   var_domains = {var:{0,1,2} for var in 'ab'},
   constraints = {
      lambda a, b:  a > b,
      lambda b: b > 0,
   })
```

can be represented in relational form as the following:

```
[
    Relation(header=['a', 'b'],
             tuples={(2, 0),
                     (1, 0),
                     (2, 1)}),

    Relation(header=['b'],
             tuples={(2,),
                     (1,)})
]
```

Notes

- Since the variable names and the effect of their domains appear in the relations, there is no need to specify them separately.
- Here we are using a list for the collection of relations instead of a set to avoid the the need for creating hashable objects (required for Python sets). From an algorithmic perspective, the order of relation objects in the collection does not matter, therefore, a set would have been a more natural choice.
- Single (one-element) tuples in Python require an extra comma at the end, for example, `(2,)`.
- You can write a short function that takes a CSP object and returns a collection of Relations equivalent to the object.

Question 5

Correct

Mark 1.00 out of 1.00

Convert the following instance of CSP to an equivalent list of relations called `relations`. In each relation, the variables must appear in the alphabetic order. The order of relations in the outer list is not important.

```
csp = CSP(
   var_domains = {var:{0,1,2} for var in 'abcd'},
   constraints = {
      lambda a, b, c: a > b + c,
      lambda c, d: c > d
      }
   )
```

For example:

Test	Result

`from student_answer import relations`
`from csp import Relation`
 	
`print(len(relations))`
`print(all(type(r) is Relation for r in relations))`
	2
	True

Answer: (penalty regime: 0, 25, ... %)

Reset answer

```
from csp import Relation

relations = [
    Relation(header=['a', 'b', 'c'],
             tuples={
                 (1, 0, 0),
                 (2, 0, 0),
                 (2, 1, 0),
                 (2, 0, 1)
             }
    ),

    Relation(header=['c', 'd'],
             tuples={
                 (1, 0),
                 (2, 0),
                 (2, 1)
             }
    ),
]
```

	Test	Expected	Got	

✓	`from student_answer import relations`
	`from csp import Relation`
	
	`print(len(relations))`
	`print(all(type(r) is Relation for r in relations))`
		2
		True
			2
			True
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 6

Correct

Mark 1.00 out of 1.00

Convert the following instance of CSP to an equivalent list of relations called `relations`. In each relation, the variables must appear in alphabetic order. The order of relations in the outer list is not important. Then use the *variable elimination* algorithm to eliminate variable ***a*** and produce a new list of relations called `relations_after_elimination`.

```
csp = CSP(
   var_domains = {var:{-1,0,1} for var in 'abcd'},
   constraints = {
      lambda a, b: a == abs(b),
      lambda c, d: c > d,
      lambda a, b, c: a * b > c + 1
      }
   )
```

For example:

Test	Result

`from student_answer import relations`
`from csp import Relation`
 	
`print(len(relations))`
`print(all(type(r) is Relation for r in relations))`
	`3`
	`True`

Answer: (penalty regime: 0, 25, ... %)

Reset answer

```
from csp import Relation

relations = [
    Relation(header=['a', 'b'],
             tuples={
                 (0, 0),
                 (1, -1),
                 (1, 1)
             }
    ),

    Relation(header=['c', 'd'],
             tuples={
                 (0, -1),
                 (1, -1),
                 (1, 0)
             }
    ),

    Relation(header=['a', 'b', 'c'],
             tuples={
                 (-1, -1, -1),
                 (1, 1, -1)
             }
    )

      ]

relations_after_elimination = [
    Relation(header=['c', 'd'],
             tuples={
                 (0, -1),
                 (1, -1),
                 (1, 0)
             }
    ),

    Relation(header=['b', 'c'],
             tuples={
                 (1, -1)
             }
    )

    ]

```

	Test	Expected	Got	

✓	`from student_answer import relations`
	`from csp import Relation`
	
	`print(len(relations))`
	`print(all(type(r) is Relation for r in relations))`
		3
		True
			3
			True
				✓

Passed all tests! ✓

Correct
Marks for this submission: 1.00/1.00.

Question 7

Correct

Mark 1.00 out of 1.00

Provide an instance of CSP class named `cryptic_puzzle` that represents the following cryptarithmetic problem:

```
  two
+ two
------
 four
```

The objective is to find what digit each letter can represent. Each letter is associated to exactly one digit and each digit is associated to up to one letter. The letters on the left (t and f) cannot be zero (if they were they wouldn't be there). This problem is depicted in the lecture notes (as an additional example)

Important: Your solution must also contain the solution to two earlier questions: the functions `generate_and_test` and `arc_consistent`.

Notes

1. If you wish, you can use the code template provided in the answer box.
2. Use lower case letters for variable names.
3. The domains of *t, w, o, f, u*, and *r* must be the set of integers from 0 to 9 (inclusive). Enforce the requirements of the problem by adding appropriate constraints. The arc consistency algorithm will automatically trim the domains.
4. You need to define some auxiliary variables to take care of the carry overs. In the template provided, these are called `c1` and `c2`.
5. The CSP object must contain a number of constraints describing the given problem. One of the constraints, should require that the variable `t, w, o, f, u, r` must have different values. One way of implementing this is to use a set. For example, consider what the expression `len({a, b})` evaluates to if a and b have the same value.
6. All the constraints in this problem can be added as a number of lambda expression (one line each). A lambda expression evaluates to a function object without a name (an anonymous function). If you think for some of the constraints you need to write more code, you can define a named function outside and then include the name of the function in the set of constraints.
7. The answer must include all the required import statements.
8. Notice that we first make an arc-consistent version of the network (using the function you provide) and then solve the instance using generate-and-test. Without making the network arc-consistent, the generate-and-test algorithm takes a long time (about 30 seconds on a mid-range desktop computer) to find all the solutions. which is not allowed on the server. However, once the network is made arc-consistent, all the solutions can be found in a few seconds.

For example:

Test	Result

`from csp import *`
`from student_answer import cryptic_puzzle, arc_consistent, generate_and_test`
 	
`print(set("twofur") <= set(cryptic_puzzle.var_domains.keys()))`
`print(all(len(cryptic_puzzle.var_domains[var]) == 10 for var in "twofur"))`
	`True`
	`True`

`from csp import *`
`from student_answer import cryptic_puzzle, arc_consistent, generate_and_test`
 	
`new_csp = arc_consistent(cryptic_puzzle)`
`print(sorted(new_csp.var_domains['r']))`
	`[0, 2, 4, 6, 8]`

`from csp import *`
`from student_answer import cryptic_puzzle, arc_consistent, generate_and_test`
 	
`new_csp = arc_consistent(cryptic_puzzle)`
`print(sorted(new_csp.var_domains['w']))`
	`[0, 2, 3, 4, 5, 6, 7, 8, 9]`

Test

```
from csp import *
from collections import OrderedDict
from student_answer import cryptic_puzzle, arc_consistent,
generate_and_test

new_csp = arc_consistent(cryptic_puzzle)
solutions = []
for solution in generate_and_test(new_csp):
    solutions.append(sorted((x, v) for x, v in solution.items()
                            if x in "twofur"))
print(len(solutions))
solutions.sort()
print(solutions[0])
print(solutions[5])
```

Result

```
7
[('f', 1), ('o', 4), ('r', 8), ('t', 7), ('u', 6),
('w', 3)]
[('f', 1), ('o', 8), ('r', 6), ('t', 9), ('u', 5),
('w', 2)]
```

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```
import itertools, copy
from csp import scope, satisfies, CSP

def arc_consistent(csp):
    csp = copy.deepcopy(csp)
    to_do = {(x, c) for c in csp.constraints for x in csp.var_domains.keys()} # C
    while to_do:
        x, c = to_do.pop()
        ys = scope(c) - {x}
        new_domain = set()
        for xval in csp.var_domains[x]: # COMPLETE
            assignment = {x: xval}
            for yvals in itertools.product(*[csp.var_domains[y] for y in ys]):
                assignment.update({y: yval for y, yval in zip(ys, yvals)})
                if satisfies(assignment, c): # COMPLETE
                    new_domain.add(xval) # COMPLETE
                    break
        if csp.var_domains[x] != new_domain:
            for cprime in set(csp.constraints) - {c}:
                if x in scope(cprime):
                    for z in scope(cprime): # COMPLETE
                        if x != z: # COMPLETE
                            to_do.add((z, cprime))
            csp.var_domains[x] = new_domain     #COMPLETE
    return csp

def generate_and_test(csp):
    names, domains = zip(*csp.var_domains.items())
    for values in itertools.product(*domains):
        assignment = {names[i]: values[i] for i in range(len(names))}
        if all([satisfies(assignment, constraint) for constraint in csp.constrain
            yield assignment

domains = {x: set(range(10)) for x in "twofur"}
domains.update({'c1':{0, 1}, 'c2': {0, 1}}) # domains of the carry overs

cryptic_puzzle = CSP(
    var_domains=domains,
    constraints={
        lambda o, r, c1    :      o + o == r + 10 * c1, # one of the constraints
        lambda w, u, c1, c2:      w + w + c1 == u + 10 * c2,
        lambda t, o, c2, f:       t + t + c2 == o + 10 * f,
        lambda f: f != 0,
        lambda t: t != 0,
        lambda t, w, o, f, u, r: len({t, w, o, f, u, r}) == 6
        })

```

	Test	Expected	Got	
✓	`from csp import *` `from student_answer import cryptic_puzzle, arc_consistent, generate_and_test` `print(set("twofur") <= set(cryptic_puzzle.var_domains.keys()))` `print(all(len(cryptic_puzzle.var_domains[var]) == 10 for var in "twofur"))`	`True` `True`	`True` `True`	✓
✓	`from csp import *` `from student_answer import cryptic_puzzle, arc_consistent, generate_and_test` `new_csp = arc_consistent(cryptic_puzzle)` `print(sorted(new_csp.var_domains['r']))`	`[0, 2, 4, 6, 8]`	`[0, 2, 4, 6, 8]`	✓
✓	`from csp import *` `from student_answer import cryptic_puzzle, arc_consistent, generate_and_test` `new_csp = arc_consistent(cryptic_puzzle)` `print(sorted(new_csp.var_domains['w']))`	`[0, 2, 3, 4, 5, 6, 7, 8, 9]`	`[0, 2, 3, 4, 5, 6, 7, 8, 9]`	✓
✓	`from csp import *` `from collections import OrderedDict` `from student_answer import cryptic_puzzle, arc_consistent, generate_and_test` `new_csp = arc_consistent(cryptic_puzzle)` `solutions = []` `for solution in generate_and_test(new_csp):` `    solutions.append(sorted((x, v) for x, v in solution.items()` `                            if x in "twofur"))` `print(len(solutions))` `solutions.sort()` `print(solutions[0])` `print(solutions[5])`	`7` `[('f', 1), ('o', 4), ('r', 8), ('t', 7), ('u', 6), ('w', 3)]` `[('f', 1), ('o', 8), ('r', 6), ('t', 9), ('u', 5), ('w', 2)]`	`7` `[('f', 1), ('o', 4), ('r', 8), ('t', 7), ('u', 6), ('w', 3)]` `[('f', 1), ('o', 8), ('r', 6), ('t', 9), ('u', 5), ('w', 2)]`	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.