API for `cspbase.py`

Table of Contents:

`Variable` class:

```
cspbase.Variable(
    name, domain=[]
)
```

- Arguments:
  - **name**: `string` type. The name of this variable.
    - Ex: Variable cell (1,1) should have name `Cell(1,1)`.
    - Ex: A cage contains cell (1,1) and cell (1,2), with operation '?' and expected number 12. The operand variable should have name:
      - `Cage_op(12:?:[Var-Cell(1,1), Var-Cell(1,2)])`
  - **domain**: `[int]` or `[string]` type. A list of `int` or `string` representing the PERMANENT domain of this variable.
    - PERMANENT domain: never changes during filtering.
    - CURRENT domain(will see later): can be changed by pruning/unpruning values during filtering.
    - Hint: `int` for cell variables and `string` for operand variables.

- Methods:
  - `domain()`:
    - Return the variable's PERMANENT domain.
  - `domain_size()`:
    - Return the size of the PERMANENT domain.

- `add_domain_values(values)`:
  - Add additional domain values to the PERMANENT domain.
  - `values`: a collection of `int` or `string` to add.
- `prune_value(value)`:
  - Remove `value` from CURRENT domain.
- `unprune_value(value)`:
  - Restore `value` to CURRENT domain.
- `restore_curdom()`:
  - Restore all values back into CURRENT domain.
  - Now CURRENT domain is the same as PERMANENT domain.
- `cur_domain()`:
  - Return list of values in CURRENT domain.
  - If assigned, only assigned value is viewed as being in current domain.
- `in_cur_domain(value)`:
  - Check if `value` is in CURRENT domain (without constructing lists).
  - If assigned, only the assigned value is viewed as being in current domain.
  - *Implemented by searching and indexing in domain, so this method is cheap.*
- `cur_domain_size()`:
  - Return the size of the variables in CURRENT domain, (without constructing lists).
  - *Implemented by traversing once in domain, so this method is cheap.*
- `is_assigned()`:
  - Return `True` if this variable is assigned with a value.
- `get_assigned_value()`:
  - Return the assigned value to this variable.
  - If this variable is not assigned, `None` is returned.
- `assign()`:
  - Assign a value to the variable and remove all other values from the CURRENT DOMAIN.
- `unassign()`:
  - Unassign the variable and restore the previous CURRENT DOMAIN.

`Constraint` class:

```
cspbase.Constraint(
    name, scope
)
```

- Arguments:
  - **name**: `string` type. The name of this constraint.
    - Can be any descriptive and unique name among other constraints.
  - **scope**: `[Variable]` type. The list of all variables involved in this constraint.

- Methods:
    - `add_satisfying_tuples(tuples)`:
        - Specify the constraint by adding its complete list of satisfying tuples.
        - `tuples`: a list of tuples of satisfying values.
    - `get_scope()`:
        - Return a list of variables that are involved in this constraint.
    - `check_tuple(tuple)`:
        - Return `True` if the given tuple is a satisfying tuple for this constraint. `False` otherwise.
    - `get_n_unasgn()`:
        - Return the number of unassigned variables in the constraint's scope.
    - `get_unasgn_vars()`:
        - Return list of unassigned variables in constraint's scope.
        - Caution: this method is computationally expensive. See if `get_n_unasgn()` is enough to do the job.
    - `check_var_val(var, val)`:
        - Return `True` if:
            - Suppose we want to assign variable `var` with value `val`, there are still satisfying tuples in this constraint (in the CURRENT domain of all variables in the scope).
        - Return `False` otherwise.

`CSP` class:

```
cspbase.CSP(
    name, vars=[]
)
```

- Arguments:
    - **name**: `string` type. The name of this CSP object.
        - Can be any descriptive and unique name among other CSP objects.
    - **vars**: `[Variables]` type. The list of all variables in this CSP.

- Methods:
    - `add_var(var)`:
        - Add variable `var` to CSP.
    - `add_constraint(con)`:
        - Add constraint `con` to CSP.
        - All variables in the constraint's scope must already have been added to the CSP.
    - `get_all_vars()`:
        - Return a list of all variables in the CSP
    - `get_all_unasgn_vars()`:

        - Return a list of unassigned variables in the CSP
    - `get_all_cons()`:
        - Return a list of all constraints in the CSP.
    - `get_cons_with_var(var)`:
        - Return a list of constraints that include variable `var` in their scope.
    - `get_all_nary_cons(n)`:
        - Return a list of all constraints that have exactly `n` variables in its scope.
    - `print_all()`:
        - Debugging method. Prints all the variables and constraints in the CSP.
    - `print_soln()`:
        - Debugging method. Prints all the variables and their assigned values in the CSP.

Misc:

- If you have any concerns about this file, feel free to post on the forum, linked on onQ.