.gitignore

# Byte-compiled / optimized / DLL files

\_\_pycache\_\_/

\*.py[cod]

\*$py.class

# C extensions

\*.so

# Distribution / packaging

.Python

build/

develop-eggs/

dist/

downloads/

eggs/

.eggs/

lib/

lib64/

parts/

sdist/

var/

wheels/

pip-wheel-metadata/

share/python-wheels/

\*.egg-info/

.installed.cfg

\*.egg

MANIFEST

# PyInstaller

# Usually these files are written by a python script from a template

# before PyInstaller builds the exe, so as to inject date/other infos into it.

\*.manifest

\*.spec

# Installer logs

pip-log.txt

pip-delete-this-directory.txt

# Unit test / coverage reports

htmlcov/

.tox/

.nox/

.coverage

.coverage.\*

.cache

nosetests.xml

coverage.xml

\*.cover

\*.py,cover

.hypothesis/

.pytest\_cache/

# Translations

\*.mo

\*.pot

# Django stuff:

\*.log

local\_settings.py

db.sqlite3

db.sqlite3-journal

# Flask stuff:

instance/

.webassets-cache

# Scrapy stuff:

.scrapy

# Sphinx documentation

docs/\_build/

# PyBuilder

target/

# Jupyter Notebook

.ipynb\_checkpoints

# IPython

profile\_default/

ipython\_config.py

# pyenv

.python-version

# pipenv

# According to pypa/pipenv#598, it is recommended to include Pipfile.lock in version control.

# However, in case of collaboration, if having platform-specific dependencies or dependencies

# having no cross-platform support, pipenv may install dependencies that don't work, or not

# install all needed dependencies.

#Pipfile.lock

# PEP 582; used by e.g. github.com/David-OConnor/pyflow

\_\_pypackages\_\_/

# Celery stuff

celerybeat-schedule

celerybeat.pid

# SageMath parsed files

\*.sage.py

# Environments

.env

.venv

env/

venv/

ENV/

env.bak/

venv.bak/

# Spyder project settings

.spyderproject

.spyproject

# Rope project settings

.ropeproject

# mkdocs documentation

/site

# mypy

.mypy\_cache/

.dmypy.json

dmypy.json

# Pyre type checker

.pyre/

**Restaurant.py**

from Table import Table

"""

Restaurant Class

----------

This class represents the entire restaurant which manages the tables and incoming customers.

The Restaurant has no properties attached to it - you are free to add any properties you need.

Remember that the data structure backing the restaurant must be able to satisfy the

time requirements of the functions given below. Think about which data structure is best.

The class also supports the following functions:

- \_\_init\_\_(self): Initializes the restaurant

- start\_new\_table(self) -> Table: Starts a new table at the restaurant

- add\_hobbit(self): Adds a hobbit to the restaurant, following the requirements for seating

- add\_elf(self): Adds an elf to the restaurant, following the requirements for seating

- add\_dwarf(self): Adds a dwarf to the restaurant, following the requirements for seating

- add\_human(self): Adds a human to the restaurant, following the requirements for seating

- get\_least\_crowded\_table(self) -> Table: Returns the least crowded table in the restaurant. Should run in O(1) time.

- get\_number\_tables(self) -> int: Returns the number of tables in the restaurant. Should run in O(1) time.

- get\_number\_diners(self) -> int: Returns the number of diners in the restaurant. Should run in O(1) time.

- get\_number\_hobbits(self) -> int: Returns the number of hobbits in the restaurant. Should run in O(1) time.

- get\_number\_elves(self) -> int: Returns the number of elves in the restaurant. Should run in O(1) time.

- get\_number\_dwarves(self) -> int: Returns the number of dwarves in the restaurant. Should run in O(1) time.

- get\_number\_humans(self) -> int: Returns the number of humans in the restaurant. Should run in O(1) time.

Your task is to complete the following functions which are marked by the TODO comment.

You are free to add properties and functions to the class as long as the given signatures remain identical.

Good Luck!

"""

class Restaurant:

# TODO: Add your own properties here for your data structure

def \_\_init\_\_(self):

"""

The constructor for the Restaurant class.

"""

# TODO: FILL THIS IN

def start\_new\_table(self) -> Table:

"""

Starts a new table at the restaurant.

:return: The new table created.

"""

# TODO: FILL THIS IN

def add\_hobbit(self) -> None:

"""

Adds a hobbit to the least crowded table in the restaurant.

If there are multiple tables with the same number of diners,

add the hobbit to the table closest to the door. If all the

tables are full, start a new table.

"""

# TODO: FILL THIS IN

def add\_elf(self) -> None:

"""

Adds an elf to the first table closest to the door that consists

solely of elves. If such a table does not exist, start a new table

"""

# TODO: FILL THIS IN

def add\_dwarf(self) -> None:

"""

Adds a dwarf to the table with the fewest elves. If multiple tables have

an equal amount of elves, the one closest to the door is chosen. If no such

table can be found or all the tables are full, start a new table.

"""

# TODO: FILL THIS IN

def add\_human(self) -> None:

"""

Adds a human to the closest table that is not dwarf-only,

elf-only or hobbit-only. If no such table exists, start a new table.

"""

# TODO: FILL THIS IN

def get\_least\_crowded\_table(self) -> Table:

"""

Returns the least crowded table in the restaurant.

If there are multiple tables with the same number of diners,

return the table closest to the door.

This should run in O(1) time.

:return: The least crowded table in the restaurant. If there are no tables, return None.

"""

# TODO: FILL THIS IN

def get\_number\_tables(self) -> int:

"""

Returns the number of tables in the restaurant.

This should run in O(1) time.

:return: The number of tables in the restaurant.

"""

# TODO: FILL THIS IN

def get\_number\_diners(self) -> int:

"""

Returns the total number of diners in the restaurant.

This should run in O(1) time.

:return: The total number of diners in the restaurant.

"""

# TODO: FILL THIS IN

def get\_number\_hobbits(self) -> int:

"""

Returns the total number of hobbits in the restaurant.

This should run in O(1) time.

:return: The total number of hobbits in the restaurant.

"""

# TODO: FILL THIS IN

def get\_number\_elves(self) -> int:

"""

Returns the total number of elves in the restaurant.

This should run in O(1) time.

:return: The total number of elves in the restaurant.

"""

# TODO: FILL THIS IN

def get\_number\_dwarves(self) -> int:

"""

Returns the total number of dwarves in the restaurant.

This should run in O(1) time.

:return: The total number of dwarves in the restaurant.

"""

# TODO: FILL THIS IN

def get\_number\_humans(self) -> int:

"""

Returns the total number of humans in the restaurant.

This should run in O(1) time.

:return: The total number of humans in the restaurant.

"""

# TODO: FILL THIS IN

# TODO :Add any other functions that you need

**Table.py**

"""

Table Class

----------

This class represents a table inside the restaurant.

Each Table has the following static class properties:

- capacity: The maximum number of people that can sit at the table

Each Table also has the following instance properties:

- distance: The distance from the table to the door. The closest door is a distance of 1, the next closest is a distance of 2, etc.

- hobbits: The number of hobbits sitting at the table

- elves: The number of elves sitting at the table

- dwarves: The number of dwarves sitting at the table

- humans: The number of humans sitting at the table

The class also supports the following functions:

- \_\_init\_\_(self, distance, \*\*kwargs): Initializes the table with the given distance from the door

- is\_table\_full(self): Returns True if the table is full, False otherwise

- is\_elves\_only(self): Returns True if the table consists only of elves

- is\_dhe\_only(self): Returns True if the table consists of only dwarves, only hobbits or only elves

- get\_total\_diners(self): Returns the total number of diners at the table

- get\_elves(self): Returns the number of elves at the table

- get\_distance(self): Returns the distance from the table to the door

- add\_hobbit(self): Adds a hobbit to the table

- add\_elf(self): Adds an elf to the table

- add\_dwarf(self): Adds a dwarf to the table

- add\_human(self): Adds a human to the table

Your task is to complete the following functions which are marked by the TODO comment.

You are free to add properties and functions to the class as long as the given signatures remain identical.

Good Luck!

"""

class Table:

# This is the defined class property as above. This should not be changed.

capacity: int = 7

# These are the defined instance properties as above. Feel free to add any extra properties you need.

distance: int

hobbits: int

elves: int

dwarves: int

humans: int

def \_\_init\_\_(self, distance: int, \*\*kwargs) -> None:

"""

The constructor for the Table class.

:param distance: The distance of the table from the door.

:param kwargs: Any extra arguments to be passed to the constructor.

"""

# TODO: FILL THIS IN

def is\_table\_full(self) -> bool:

"""

Returns whether the table is full or not.

:return: True if the table is full, False otherwise.

"""

# TODO: FILL THIS IN

def is\_elves\_only(self) -> bool:

"""

Returns whether the table is only elves or not.

There must be at least one elf at the table to be considered elves-only.

:return: True if the table is only elves, False otherwise.

"""

# TODO: FILL THIS IN

def is\_dhe\_only(self) -> bool:

"""

Returns whether the table is made up of only dwarves, only hobbits, or only elves.

Note that at least one of the creatures must be at the table to be considered

dwarf-only, hobbit-only, or elf-only.

:return: True if the table is made up of only dwarves, only hobbits, or only elves.

"""

# TODO: FILL THIS IN

def get\_total\_diners(self) -> int:

"""

Returns the total number of diners at the table.

:return: The total number of diners at the table.

"""

# TODO: FILL THIS IN

def get\_elves(self) -> int:

"""

Returns the number of elves at the table.

:return: The number of elves at the table.

"""

# TODO: FILL THIS IN

def get\_distance(self) -> int:

"""

Returns the distance of the table from the door.

:return: The distance of the table from the door.

"""

# TODO: FILL THIS IN

def add\_hobbit(self) -> None:

"""

Adds a hobbit to the table.

If the table is already full, this function should do nothing.

"""

# TODO: FILL THIS IN

def add\_elf(self) -> None:

"""

Adds an elf to the table.

If the table is already full, this function should do nothing.

"""

# TODO: FILL THIS IN

def add\_dwarf(self) -> None:

"""

Adds a dwarf to the table.

If the table is already full, this function should do nothing.

"""

# TODO: FILL THIS IN

def add\_human(self) -> None:

"""

Adds a human to the table.

If the table is already full, this function should do nothing.

"""

# TODO: FILL THIS IN