Assignment 2: Instructions

When you're ready to submit your solution, go to the assignments list.

Learn to Program: Crafting Quality Code // Assignment 2

Preface

The due date has been extended: it is now Sunday 28 April. The hard deadline is still 1 May.

Learning activities

Here are the learning activites for this assignment:

- Implement two classes: `Rat` and `Maze`.
- Practice using nested lists, writing loops and `if` statements, and testing.

Required Preparation

This assignment description assumes that you have watched all the week 4 videos and also done the week 4 exercise. If you read this assignment description before you've done all of that, please come back and reread it after you've completed the week 4 exercise.

Rat Race Game

Overview

This assignment has you write a two-player rat race game. Jen and Paul are rats caught in a maze, and they frantically run around and eat Brussels sprouts.

The maze

Here is a sample maze:

```
#######
#.....#
#.###.#
#..@#.#
#@#.@.#
#######
```

`#` is a wall, `.` is a hallway, and `@` is a Brussels sprout.

Here is the same maze with Jen shown (`J`) in the upper-left corner at row 1 and column 1, and Paul (`P`) near the upper-right at row 1 and column 4:

```
#######
#J..P.#
#.###.#
#..@#.#
#@#.@.#
#######
```

Controls

The players will move the rats around using the keyboard.

Here are the controls; both have the classic "inverted T" layout:

```
Jen             Paul
--------        --------
w: up           i: up
s: down         k: down
a: left         j: left
d: right        l: right
```

Game play

The players move Jen and Paul around the maze. They cannot move into walls. If the players try to move the rats into walls, the rats do not move. Both Jen and Paul can occupy the same space, although if this happens only one of them will be shown on the maze. (It doesn't matter which one.)

When they move over a Brussels sprout they eat it and the Brussels sprout character `@` is replaced by the hallway character `.` . Each rat keeps track of how many

Brussels sprouts it has eaten. When there are no Brussels sprouts left, the game ends. The players can still move the rats around the maze, but there will not be any more Brussels sprouts to eat and so the scores will not change.

What to do

Step 1: Download and read the `a2.py` starter code.

In this assignment, we are providing starter code. Here is a file containing headers for the two classes that you will write:

- a2.py

Constants

As you read the starter code, you will see several *constants*. Constants are variables whose values should never change: once a constant is assigned a value, it should not be assigned a different value elsewhere in the program. Following the standard Python style, constants are named with uppercase letters to distinguish them from other variables.

`a2.py` has the following constants: `WALL`, `HALL`, `SPROUT`, `LEFT`, `RIGHT`, `NO_CHANGE`, `UP`, `DOWN`, `RAT_1_CHAR`, and `RAT_2_CHAR`. Rather than use a string like `'@'` in your code, use the constant, `SPROUT`, that refers to that value.

Step 2: Write class `Rat`'s methods.

In class `Rat` in `a2.py`, write the following methods. We **strongly** recommend that you follow the Function Design Recipe and start by writing example calls on these methods!

Method name: (parameter types) -> return type	Description
`__init__`: `(Rat, str, int, int) -> NoneType`	The first parameter represents the rat to be initialized, the second parameter represents the 1-character symbol for the rat, the third parameter represents the row where the rat is located, and the fourth parameter represents the column where the rat is

Method name: (parameter types) -> return type	Description
	located. Initialize the rat's four instance variables: • `symbol`: the 1-character symbol for the rat • `row`: the row where the rat is located • `col`: the column where the rate is located • `num_sprouts_eaten`: the number of sprouts that this rat has eaten, which is initially `0`. Here is an example: `Rat('P', 1, 4)`
`set_location`: `(Rat, int, int) -> NoneType`	The first parameter represents a rat, the second represents a row, and the third represents a column. Set the rat's `row` and `col` instance variables to the given row and column.
`eat_sprout`: `(Rat) -> NoneType`	The first parameter represents a rat. Add one to the rat's instance variable `num_sprouts_eaten`. Yuck.
`__str__`: `(Rat) -> str`	The first parameter represents a rat. Return a string representation of the rat, in this format: `symbol at (row, col) ate num_sprouts_eaten sprouts.` For example: `'J at (4, 3) ate 2 sprouts.'` Do **not** put a newline character (`'\n'`) at the end of the string.

Step 3: Write class `Maze`'s methods.

In class `Maze` in `a2.py`, write the following methods. We **strongly** recommend that you follow the Function Design Recipe and start by writing example calls on these methods!

Method name: (parameter types) -> return type	Description

Method name: (parameter types) -> return type	Description
`__init__`: `(Maze, list of list of str, Rat, Rat) -> NoneType`	The first paramter represents the maze to be initialized, the second parameter represents the contents of the maze, the third parameter represents the first rat in the maze, and the fourth parameter represents the second rat in the maze. Initialize this maze's four instance variables: • `maze`: a maze with contents specified by the second parameter. • `rat_1`: the first rat in the maze. • `rat_2`: the second rat in the maze. • `num_sprouts_left`: the number of uneaten sprouts in this maze. Here is an example call: `Maze([['#', '#', '#', '#', '#', '#', '#'], ['#', '.', '.', '.', '.', '.', '#'], ['#', '.', '#', '#', '#', '.', '#'], ['#', '.', '.', '@', '#', '.', '#'], ['#', '@', '#', '.', '@', '.', '#'], ['#', '#', '#', '#', '#', '#', '#']], Rat('J', 1, 1), Rat('P', 1, 4))` Notice that the rat symbols do not appear in the `list of list of str`.
`is_wall`: `(Maze, int, int) -> bool`	The first parameter represents a maze, the second represents a row, and the third represents a column. Return `True` if and only if there is a wall at the given row and column of the maze.
`get_character`: `(Maze, int, int) -> str`	The first parameter represents a maze, the second represents a row, and the third represents a column. Return the character in the maze at the given row and column. If there is a rat at that location, then its character should be returned rather than `HALL`.
`move`: `(Maze, Rat, int, int) -> bool`	The first parameter represents a maze, the second represents a rat, the third represents a vertical direction change (`UP`, `NO_CHANGE` or `DOWN`), and the fourth

Method name: (parameter types) -> return type	Description
	represents a horizontal direction change (`LEFT`, `NO_CHANGE` or `RIGHT`). Move the rat in the given direction, unless there is a wall in the way. Also, check for a Brussels sprout at that location and, if present: • have the rat eat the Brussels sprout, • make that location a `HALL`, and • decrease the value that `num_sprouts_left` refers to by one. Return `True` if and only if there wasn't a wall in the way.

`__str__`: `(Maze) -> str`	The first parameter represents a maze. Return a string representation of the maze, using the format shown in this example:
	`#######`
	`#J..P.#`
	`#.###.#`
	`#..@#.#`
	`#@#.@.#`
	`#######`
	`J at (1, 1) ate 0 sprouts.`
	`P at (1, 4) ate 0 sprouts.`
	
	Do **not** put a newline character (`'\n'`) at the end of the string.

Step 4: Download `rat_race.py`.

Here is the code that you will use to display the graphical user interface (GUI) for your game and to handle the keyboard input:

- rat_race.py

You will not submit this file and you do not need to modify it. We are providing it as an example of a `tkinter` GUI.

The version of `a2.py` that you submit for grading must work with this code. However, once you have finished with the assignment, feel free to modify and extend this code.

Step 5: Download `maze.txt`.

Here is a sample maze that you can use to test your game:

- maze.txt

Step 6: Test your code.

There are several ways to test your code, including `doctest` and `unittest`. (Your tests will not be marked.)

You can also test your code by running `rat_race.py` and playing the game. To do so, put your `a2.py` and the sample `maze.txt` in the same folder as `rat_race.py`. In IDLE, open `rat_race.py` and choose Run.

Step 7: Submit your work.

Go to the Assignments page and click the appropriate Submit button. Choose your completed **a2.py** file. It should be marked within a few minutes. You can see your results by clicking on your Score.

You can submit **a2.py** once every 10 minutes. Notice that this means that you can submit a lot of times before the due date if you start early. You can even submit before you've finished all of the methods. We will take the highest score out of all of your submissions.

Fun stuff: post your mazes!

We provided one sample maze, but the possibilities are endless! A maze must:

- be rectangular,
- the outside edges must be made up entirely of `WALL`s, and
- the rest of the maze must be made up of `WALL`s, `HALL`s, `SPROUT`s, `RAT_1_CHAR`, and `RAT_2_CHAR`.

Visit the forum board Assignment 2: post your mazes! to post your mazes and to download mazes to test your code with.