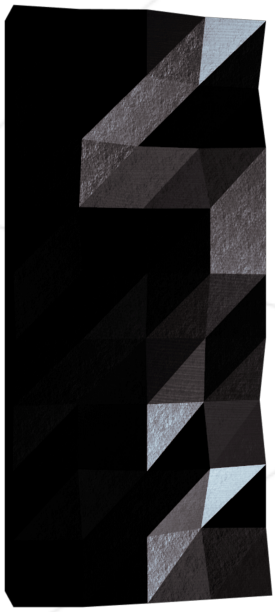

Python-Django Training - Rush 00

Moviemon

Summary: Here is the subject for the rush00 of the piscine Python-Django.

Contents

Chapter I

Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the `OCaml` syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercices must be done in order. The graduation will stop at the first failed exercice. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlopt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerfull ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the `OCaml` piscine. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.
- You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the

exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.

- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter II

Day-specific rules

- You will use the `Python3` interpreter.
- The paths relative to an application must be located in a `urls.py` file in the folder of this application.
- Each displayed page must be properly formatted (including a doctype, `html` tag couples, `body`, `head`), proper management of special characters. No awkward display).
- For this rush, your will use the default Django development server provided with the `manage.py` utility.
- Only the required URLs will return a page without an error any error message. If `/ex00` is requested, `/ex00foo` must return a 404 error.
- You must turn-in a requirement.txt file (via pip freeze) containing all the libraries your project requires to work.

Chapter III

Preamble

Some quotes from the movie Be Kind Rewind with Jack Black:

Figure III.1: The video store.

Jerry: *I am Robocop. Anything you say can and will be held against you in a court of Robocop.*

Jerry: *I will shoot you, and I know robot karate.*

Jerry: *I'm not saying your uncle is illerate, maybe he just needs to go to nightschool.*

Jerry: *I was going to make you into the next Marylin Monroe! But no! Now you're just going to be.... Laundry girl!*

Jerry: *Hey, hey, don't put your shoes in the refrigerator... 'cause they'll get cold!*

Mike: (holding the Ghost Busters tape) *I'll be Bill Murray and you'll be everyone else.*

Jerry: *[sung, poorly, to the tune of the Ghostbusters theme song] When you're walkin' down the street...*

Jerry: *[singing] ... and you see a little ghost...*

Jerry: *[singing] ... whatcha gonna do about -*

Jerry: *Ghostbusters?*

Mike: *What? What is that?*

Jerry: *That's the Ghostbusters theme song.*

Mike: *No.*

Jerry: *I'm pretty sure it is.*

Alma: *Are you in love with me?*

Mike: *Huh?*

Alma: *Mm-hm.*

Mike: *Well, how do I know that?*

Alma: *You know you're in love with a person when you talk to them for a minimum of 20 minutes a day in your head.*

Mike: *What if I talk to a guy in my head for 20 minutes? What would that mean?*

Alma: *You're in love with Jerry.*

Your name it, we shoot it

Sometimes the best movies are the ones we make up

source

Chapter IV

Mandatory part

IV.1 Introduction - FAQ

This rush will make your code a little solo game with a web interface.

`What is this game's goal?`
This game is called `MovieMon`. Its goal is to capture all the Moviemons hiding in a game grid, using Movieballs.

`Hey, that reminds of... Isn't it exactly like...?`
I don't know what you're talking about, man.

`What's a Moviemon?`
A `Moviemon` is a movie you can find on IMDb. Preferably, a monster movie.

`How can you catch a Moviemon?`
Throwing a `movieball` at them, of course!. A `Moviemon`'s strenght is equal to their IMDb rating. A high rating makes the `Moviemon` harder to catch than one with a low rating. The player's strength - that increases with the number of `Moviemons` they own- will increase their chance to catch one.

`How does a typical game run?`
When a player starts a new game, the game requests all the necessary films on IMDb and sends them on the 'Worldmap', the main page of the game.
The player will move freely over this page, square after square on a fixed grid. As they progress, they will harvest `movieballs`, or tumble over a `Moviemon`.
When they do find a `Moviemon`, if they believe they can catch it, the player tries to capture them.
They throw a movieball at them, misses, tries again, misses, tries again, and catches them! The player can then proudly consult their `Moviedex` that lists all the `Moviemons` they caught before returning to hunt, trying to catch them all! (Moviemons, that is, of course...)

IV.2 Instructions

IV.2.1 Settings

You must add the games settings to your own settings, that is:

- The grid size. It must be 10x10 squares minimum.
- The starting point of the player on this grid.
- The list of names or IDs of at least 10 films you can request on IMDb. Remember, movie monsters are better.

You must use these settings in your code.

You can add other settings if you like, but they must be relevant and working!.

The rating of a film will be the Moviemon's strenght. Try to make a balanced dispatch of monsters with enough strong and weak monsters. Try to use at least 3 films with a lower score than 4 and 3 films with a higher score thant 7.

IMDb's API being quite obscure, you can use OMDB, which is an unofficial API.

IV.2.2 Game data

During a game, you will need to save the data from one page to another. The website would naturally use cookies, or a session system on the server side. But this is not your average website.

You must stock the data summing up the game's state in a file that must be created in your project.

This file is not saved for the player, it's used to stock the ongoing game state. It must not include any logic and its content must be stocked in binary thanks to the `pickle` library (included with `Python`).

In your project, you aslo must create the logic necessary to the update and use of this file, that will contain the following informations:

- Player position on the map.
- The number of owned `Movieballs`.
- The names (or IDs) of all the `Moviemons` in the `Moviedex`.

- The complete informations of all the game's `Moviemons` as obtained on `IMDb`.

IV.2.3 Data management

You also must create a `Python` class that will have to manage the game data. This class must at least contain the following methods:

- 'load': Load the game data passed in parameters in the class instance. Returns the current instance.
- 'dump': Returns the game's data.
- 'get_random_movie': Returns a random Moviemon among the `Moviemons` not yet captured.
- 'load_default_settings': Loads the game data in the class instance from the settings. Requests and stocks all the `Moviemons` details on `IMDB`. Returns the current intance.
- 'get_strength': Returns the player's strength.
- 'get_movie': Returns a `Python` dictionary containing all the details of the `Moviemon` name passed in parameter and necessary to the `Detail` page.

You can add any method and attributes you'll see fit to this class.

`You will never be asked to modify this file "by hand" or delete this file or any ongoing saving file during the evaluation.`

IV.2.4 Game's esthetics

The game will naturally be displayed in your browser via `HTML` and `CSS`. You cannot use `Javascript`.

The game display will be cut in two parts the player must be able to perfectly identify:

- `The screen`: Displays what's going on in the game. There are no possible interaction in this place. It must contain neither link nore form.
- `Controls`: Located below or on both sides od the screen, they allow to interact with the game and are contextual. This means they change behavior depending on the game state. This also means they don't have to be always active. Still, even when inactive, they must remain `visible and in place`.

There must be 8 'buttons':

- 4 `directions`, like the ones you would find on a gamepad:
  Top, right, bottom, left
- A `select` button.
- A `start` button.
- An `A` button.
- A `B` button.

You cannot add or delete 'buttons' in this space. You cannot display any information other than the 'buttons' name in this space either.

Beyond these specification, the game's esthetic won't count in the mandatory part. But of course, making it look good will get you some bonus points.

Button behavior for each view is described in the following part.

`A 'button' doesn't have to be an HTML <button> tag. An inactive 'button' can be a dead link, an image, a text or a special character.`

IV.2.5 Pages

You must create the following pages/views/behaviors. A 'button' not mentioned in a page is an inactive 'button'. Besides, if the destination is not specified for a control, this means it sends back to the same page, potentially modified.

`TitleScreen`

- Description : Title screen.
- Screen: Must display the game's title as well as 'A - New Game' and 'B - Load'.
- Url: basic one, domain name and port.
- Controls:
  - `A`: A link to the `Worldmap` page.
    Before it's being displayed, the file containing the current game informations must be reinitialized with the `Settings` parameters and the `Moviemons` must be requested once again.
  - `B`: link to the `Load` page.

`Worldmap`

- Description: a world map, where the character moves, grabs `movieballs` and hunts `Moviemons`.
- Screen: A grid the size defined in the settings. A representation of the player (image, character...) must stand on the square matching their current position. It must be clearly visible.

  The screen must also display:

  - The number of `movieballs`.
  - A message when a `movieball` is found.
  - A message when a `Moviemon` flushed out, as well as an indication of the button to push to start the catching process.]
- Url : `'/worldmap'`
- Controls:
  - `Directions`: Each direction must move the character one case in the same direction. The player must not be able to exit the map.

    Each move gives a chance to flush out a `Moviemon` or grab a `movieball`.

    If a `Moviemon` is flushed out, it is randomly picked among the `Moviemons` that have not yet been caught.
  - `A`: Only if a `Moviemon` is flushed out: Link to `Battle` page of the battle against this `Moviemon`.
  - `start`: Link to the `Option` page.
  - `select`: Link to the `Moviedex` page.

`Refeshing this page must not change the player position on the map.`

`Battle`

- Description: Try to catch the Moviemon you've just flushed out!

- URL: `'/battle/<moviemon_id>'`. <moviemon_id> will be replaced by the `Moviemon`'s ID you're about to battle.
- Screen: Displays the poster and the strength of the `Moviemon`, the number of `movieballs` in stock, the stength of the player and the winning rate (see below).

  If caught, you must also display a catch phrase (no pun intended) like "You caught it" to mark the event.

  If you failed, you must also display a phrase like "You missed !".

  As long as the `Moviemon` is not caught, the screen must also display 'A - Launch movieball'.

  Anyway, you must always show that the 'button' `B` takes the player back to the `Worldmap`.

- Controls :
  - `A` : Throw a `movieball`

    If the player has none, the action is effectless (you can display a little taunt fromm the ennemy on the screen).

    Otherwise, the number of movieballs is decreases by 1 and a luck roll calculates whether or not the `Moviemon` is caught.

    Luck rate `C` is calculated as follows:
    `C` = 50 - (monster strength * 10) + (player strength * 5)
    Et 1 <= `C` <= 90

    For instance:
    For a monster with a 8.2 strength and a strength 2 player:
    C = 50 - 82 + 10 = -22
    This monster has 1% chance to get caught.

    For a monster with a 5 strength, and a strength 8 player:
    C = 50 - 50 + 40 = 40
    This monster has 40% chance to get caught.

    For a monster with a 2 strength, and a strength 14 player:
    C = 50 - 20 + 70 = 100
    This monster has 90% chance to get caught.

    If successful, the `Moviemon` is caught and stocked in the `MovieDex`. The A button becomes inactive.

In case of a failure, the player can throw as many movie balls they have in their possession.

- `B`: Returns to `Worldmap`

`Moviedex`

- Description: List of newly caught `Moviemons`. You must be able to select a film to access its informations.
- URL: `'/moviedex'`
- Screen: All the caught `Moviemons` posters must show on screen. The selected poster must be clearly set appart by a graphic element like a blue frame.

  You will also add 'A - More information' and 'select - Back'.
- Controls:
  - `Directions`: Directions allow you to select a different film. You must use at least 2 directions: left and right or top and bottom.
  - `select`: Link to the `Worldmap` page.
  - `A`: Link to the `Detail` page of the selected `Moviemon`.

`As a default setting, when reaching the page, the film selected will be the first in the list.`

`Detail`

- Description: Detail of a moviemon.
- URL: `'/moviedex/<moviemon>'` <moviemon> must be replaced with the `Moviemon` ID.
- Screen: Must display the name, poster, director, year, rating, synopsis and actors of the Moviemon as well as 'B-Back'.
- Controls:
  - B Button: Link to the Moviedex page

`Option`

- Description: Game's options.
- URL: `'/options'`
- Screen: Displays the game's options: 'A - Save', 'B - Quit' as well as 'start - cancel'
- Controls:
  - Start button: Link to the `Worldmap` page.
  - A button: Link to the `Save` page.
  - B button: Link to the `TitleScreen` page.

`Save`

- Description: Allows to save an ongoing game in either one of the three available slots.
- URL: `/options/save_game`
- Screen: must display 3 slots: 'Slot A', 'Slot B' and 'Slot C'.

  A selected slot must be marked with a specific graphic element, like an arrow preceding its name.

  If a slot is empty, it must be followed by 'Free'. Otherwise, it must be followed by the numbers of `Moviemons` caught.

  For instance: 'Slot A: 2/15' means this save file contains a game in which 2 `Moviemons` out of 15 have been caught.

  The screen must also display 'A - Save' and 'B - Cancel'.
- Controls:
  - `Directions`: Directions 'top' and 'bottom' are used to select a slot.
  - `A`: Copies the file containing the current state of the game in another file that must be stocked in the folder 'saved_game' of your project.

    The file name must be 'slot<n>_<score>.mmg'. <n> will be replaced by the slot's name 'a', 'b' or 'c' and <score> will be replaced by the game's score. For instance: slotb_1_15.mmg. (Please, don't try to put a slash in a file's name).
  - `B`: Link to the `Option` page

`Load`

- Description: Allows to load a saved game in either one of the three slots.
- URL: `/options/load_game`
- Screen: must display three slots: 'Slot A', 'Slot B' and 'Slot C'.

  A selected slot must be marked with a specific graphic element, like an arrow preceding its name.

  If a slot is empty, it must be followed by 'Free'. Otherwise, it must be followed by the numbers of `Moviemons` caught.

  For instance: 'Slot A: 2/15' means this save file contains a game in which 2 `Moviemons` out of 15 have been caught.

  The screen must also display 'A - Save' and 'B - Cancel'.
  Once the game is loaded,'A - Load' must be replaced by 'A - start game'.
- `Directions`: Directions 'top' and 'bottom' are used to select a slot.
- `A`:
  If no game is loaded: Copy the content of the file matching the selected slot in the file used to stock the current state of the game.

  If a game was just loaded, the 'button' is used as a link to the `Worldmap` page.

  Obviously, trying to load a 'Free' slot mustn't have any effect.
- `B`: Link to the `TitleScreen` page.

Chapter V

Bonus part

Once the mandatory part is thoroughly completed, you can implement additional functionalities that will earn you bonus points.

For your assessor to consider these functionalities, you will have to prove they're useful and functional.
An error or a flaw will invalidate the proposed functionality.

This bonus part is the only moment in this project where Javascript will be tolerated as long as it doesn't alter the operations of the mandatory part (which must run without any Javascript). AJAX or Websockets are not authorized.

Here are a few bonus ideas you can implement:

- Your website could look like a game console (see examples):
    - The screen will look like a real screen.
    - The controls must be mapped with pics that make them look like a real console controller.
- As the game is, it's very easy to guess a combat address and catch the Moviemons without looking to hard for them.
  Create a token system that is hard to see and impossible to guess, associated to some `Moviemons`. You will use it in the URL to prevent cheating.
- Associate the controls to keyboard keys so the player doesn't have to use a mouse.
- To create a variety in your game, each new game should load a different set of monsters.
- In order to make the `Worldmap` more dynamic, add a radar that will display the `Moviemons/moviballs` available on the squares adjacent to the player's position. Spice up this bonus making the radar a object you can grab like a Movieball, that will activate only once you've grabbed it.

Chapter VI

Turn-in and peer-evaluation

You will turn-in a perfectly configured `Django` project.

Except for all the mandatory elements in the subject, you're free to organize the project as you see fit.

No server error will be tolerated. Thoroughly test the behaviors of your views.

You must provide a requirement.txt file containing all the libraries necessary to your project operation.

`Django automatically returns a page 404 if the requested URL cannot be found. You could come across occurrences where Django finds a matching URL which is an error in your logic. In this case, use "raise Http404('my message')" in your views.`

Chapter VII

Turn-in example

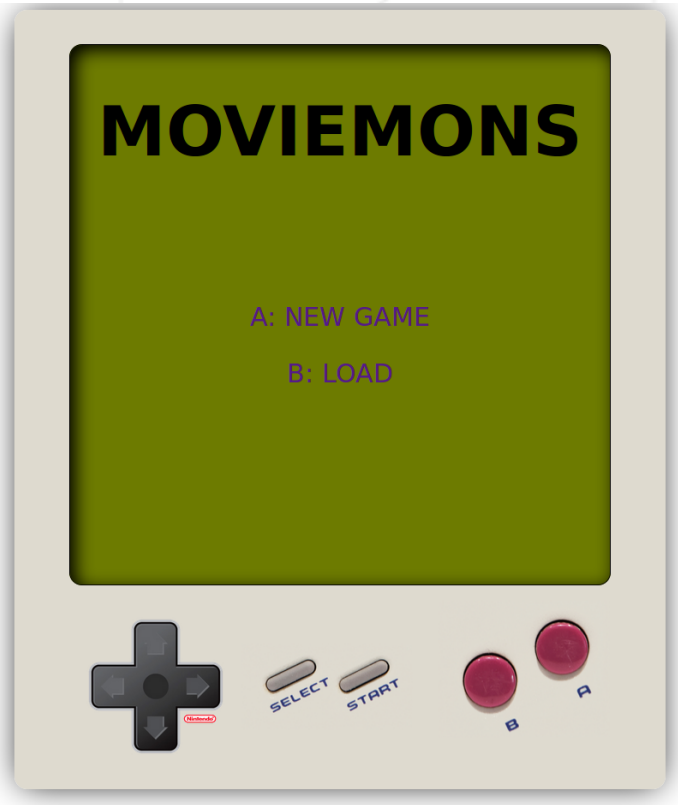

Figure VII.1: Your titlescreen could look like that

Figure VII.2: This Moviemon game appears to happen in Belgium