

exercise six

This set of exercises will guide you towards improving your adventure game. We will implement a more advanced parser for the player's input and also implement the concept of items and inventory. If you encounter difficulties, do not hesitate to seek assistance from the teaching associates — they are here to support you.

- 1 Download **game2_template.zip** from Learning Central and unzip it into a folder. This template improves upon the previous version of the game. I have made some minor structural changes (see Appendix for the summary), but the core logic of the game remains largely unchanged.

The file **items.py** now contains the definitions of items one may encounter in the game (either carried by the player, or lying around in the rooms). Similar to rooms, items are represented as dictionaries, containing fields for **"id"**, **"name"**, and **"description"**.

Rooms now have an additional field, **items**, which is a list of items found in the room. Similarly, **inventory** (a global variable in module **player.py**) is a list of items carried by the player.

- 2 Modify the **normalise_input** function (now located in **parser.py**) to return a *list* of words from the player's input, rather than a string. *Hint*: you may find the **.split()** method useful. The **normalise_input** function should, as before, strip punctuation and whitespace. Additionally, it should now remove all *stop words*.¹ These are common words that do not contribute significantly to the meaning of a command, such as articles and prepositions. Refer to the comments and examples provided for the **normalise_input** function for more detailed information.

To achieve this, first complete a helper function **filter_words** which takes a list of words **words** and returns a copy of the list from which all words provided in the list **skip_words** have been removed.

¹See https://en.wikipedia.org/wiki/Stop_word



Remember to test your `normalise_input` function with various inputs to ensure it handles different cases correctly.

- 3 Implement the `list_of_items` function in `game.py`. This function should take a list of items as input and return a string containing a comma-separated list of item names.
- 4 Now, using the function `list_of_items` from the previous question, complete the functions `print_room_items` and `print_inventory_items` according to their respective documentation.
- 5 Enhance the `print_room` function (previously named `display_room`) to include a list of items found in the room in its output.
- 6 Modify the `print_menu` function to display not only the available exits but also the possible **take** and **drop** actions for items in the room and inventory. Note that exits are now presented as **go** actions. See the documentation in the comments and examples for more information.
- 7 The `menu` and `main` functions have been updated. The revised main game loop now operates as follows:
 - The current situation is displayed using `print_room` (which now additionally prints the items in the room) and `print_inventory_items` which shows what the player is carrying.
 - The list of actions is shown to the player using the updated `menu` function, which also takes the player's input and normalises it (but does not check for correctness).
 - Finally, the `execute_command` function tries to make sense of and execute the player's command.
- 8 The essential `execute_command` function has been provided in the template. It takes the player's input (as a list of words) and, depending on the first word, which

is treated as the name of the action, executes either `execute_go`, `execute_take`, or `execute_drop`. The second word of the command is supplied as an argument to these functions. The first of these, `execute_go`, attempts to move the player in the specified direction (if the direction is a valid exit), the other two update the inventory and the list of items in a room accordingly. Complete these functions.

The game should be basically playable now. At this point, you have developed a reasonably intelligent *verb-object* parser capable of interpreting sentences such as “*I would like to drop my laptop here.*”

Moreover, you can now manipulate the game world (carry items around, drop, and pick them up). Next, try making your game more interesting, as suggested in the following optional exercises.

- 9 Implement a **"mass"** property for each item and modify your game to impose a weight limit (*e.g.* three kilograms) on the player's inventory. 
- 10 Define some victory conditions for the game, and check (in the main game loop, after each move) whether the player has won. For example, the objective may be to find all items in the game and drop them off at the reception. 

Appendix: changes to the game template

I have made some changes to the game template since the previous version to enhance its logical structure. This is a summary of changes:

- The text processing functions like `normalise_input` have been moved into a separate module `gameparser.py`.
- The function `display_room` was renamed `print_room` for consistency.
- The function `print_menu_line` was renamed `print_exit` since the menu now offers not only a list of exits, but also a list of actions on items.
- The function `menu` no longer checks for correctness of the command (it is checked elsewhere).
- `current_room` is now a global variable in module `player.py`.