Odoo Python Bootcamp

A journey from zero to +NaN, one loop at a time

From PS-Tech with love

May 5, 2025

Welcome !

Pack your towel, and welcome to the coding bootcamp designed just for you — the functional masterminds who are about to conquer the world of scripting, automation, and backend wizardry. Whether you've dabbled in Excel macros, peeked at a Python file, or just heard the word `"terminal"` in the wild, you're in the right place.

Day 0 — Setup & Survival Guide

Here's what you need to get started:

- Python 3.9 or higher (run `python3 -version` to check)
- A GitHub account
- Terminal
- VS Code or any text editor (Chads go for VIM)
- Internet, curiosity, and some gnack

Folder structure

Organize your work like this:

```
bootcamp/
|__ day01/
    |__ ex00_hello.py
    |__ ex01_loop.py
    |__ ...
|__ day02/
    |__ ...
```

Git Rules of Engagement

- One repo per participant
- One commit per exercise (at least!)
- Clean, readable commit messages (e.g. `feat: ex01 completed`)

How to submit

Create a repository on Github, clone it locally using the ssh link, and after each correct exercise you can push it like this.

```
git add .
git commit -m "done exo 00 - that was easy man"
git push origin main
```

You will share your GitHub link with the reviewer (or with OdooFisherBot 9000 if we ever automate it).

You may be like "Ye sure... what are we supposed to do with this ?" at the moment but in a few hours from now, it'll be clearer !

1. Day 01 — Morning: Shell Games

Learning to use the terminal is like getting access to the command center of your computer. It empowers you to navigate, automate, and manage your environment like a true developer. In this morning session, you'll get comfortable with essential shell commands that you'll rely on daily as a developer.

Each command you run today should be saved into a corresponding file ending in `.sh`. These script files will serve as proof of work for the review happening at the end of the day, and give you a foundation for writing repeatable, automatable tasks.

Exercise 00 — First Contact

Goal: Learn to navigate and manipulate the filesystem.
Command concepts: `mkdir`, `cd`, `touch`, `ls`, `pwd`

Task:

- Create a directory `sandbox` and navigate into it.
- Inside it, create an empty file named `test.txt`.
- List all contents in the directory.
- Print the current working directory.

Example:

```
$> sh first_contact.sh
sandbox/
test.txt
/home/student/sandbox
```

Extra questions:

- What do ‘.‘ and ‘..‘ represent?
- What does ‘/‘ mean at the start of a path?
- What happens if you run ‘touch‘ on a file that already exists?

Exercise 01 — Who are you, really?

Goal: Superficial dive into your deep identity (and some more commands).
Command concepts: `whoami`, `pwd`, `echo`

Task: Print this message in one line:

```
Hello, I am USERNAME and I'm currently in /your/path
```

Example:

```
$> sh whoami_path.sh
Hello, I am chmg and I'm currently in /Users/chmg/bootcamp/day01
```

Extra questions:

- How can you view hidden files or folders in a directory?

Exercise 02 — I'm counting on you

Goal: Find and count files.
Command concepts: `find`, `wc`, `grep`, `| (pipe)`

Task:

- In your current directory, search for all Python files recursively.
- Count how many you find and print the total.

Example:

```
$> sh count_pyfiles.sh
There are 7 Python files in this directory.
```

Extra questions:

- What does the '-name' flag do in 'find'?
- What does 'wc -l' actually count?
- How would you modify the command to search for '.txt' files instead?

Exercise 03 — Alias or die trying

Goal: Create your first useful shell alias. **Command concepts:** `alias`, `.bashrc`, `.zshrc`

Task:

- Create an alias named `gpush` that runs:

```
git add . && git commit -m "autosave" && git push
```

- OPTIONAL: Make the alias available permanently in your shell configuration file.

Example:

```
$> gpush
[main 1234abc] autosave
 1 file changed, 2 insertions(+)
```

Extra questions:

- What does ‘source /.zshrc‘ do?
- What is the difference between ’&&’ and ’;’ ?
- How do you remove an alias?
- Can you override an existing shell command with an alias?

Exercise 04 — Wild Wild Permissions

Goal: Understand and manipulate file permissions. **Command concepts:** `chmod`, `ls -l`

Task:

- Create a script that prints "Permissions test".
- Remove its execute permission, try to run it, and observe the error.
- Restore execute permission and run it again successfully.

Example:

```
$> sh perm_test.sh
sh: perm_test.sh: Permission denied
$> chmod +x perm_test.sh
$> ./perm_test.sh
Permissions test
```

Extra questions:

- What do the permission flags ‘rwx‘ mean?
- What are the differences between ‘chmod +x‘ and ‘chmod 755‘?
- How can you view detailed file permissions for all files in a directory?

Exercise 05 — Script it like it's hot

Goal: Write and run your first shell script. **Command concepts:** `bash`, `#!/bin/bash`, `$1`

Task:

- Create a script called `greet.sh`.
- It should take one argument (a name) and greet the user.

Example:

```
$> ./greet.sh Charlie
Hello, Charlie! Welcome to the bootcamp.
```

Extra questions:

- What happens if you run the script without an argument?
- How can you make a script executable without using ‘bash‘ or ‘sh‘ explicitly?

Exercise 06 — Restore me if you can

Now we're talking ! Using many concepts we've discovered before, let's do a real script that will help us on our daily lives as Odoo devs.

Directory: `day01/ex04/`
Files to submit: `restore.sh`
Forbidden: None
Goal: Combine file manipulation, archive handling, and automation with scripting.

Task: You are given a compressed archive containing a file named `dump.sql`. Your job is to:

- Unzip the archive (name will be provided during the session)
- Locate the `dump.sql` file inside it
- Move it into a folder named `restored/` (create it if necessary)
- Call a provided script `restore_db.sh` that will take `dump.sql` and restore it into a database
- Call a second provided script that will alter the DB to give you an easy access as an admin user

Make your script reproducible and idempotent (i.e., running it multiple times should not break things).

Example:

```
$> sh restore.sh archive.zip
Unzipping archive.zip ...
Moving dump.sql to ~/dumps/. ...
Restoring database from dump.sql ...
Altering it to provide an easy admin access...
Done.
```

Extra questions:

- What does the `unzip` command do?
- How can you check if a folder exists before creating it?
- How would you make your script fail gracefully if the archive or the dump file is missing?

2. Day 01 — Afternoon: Hello, Python!

Now that you're friends with your terminal, it's time to meet Python. This afternoon session will ease you into Python scripting with progressively trickier exercises that play with strings, types, functions, and user interaction.

Each script should be saved in the appropriate folder (e.g., `day01/ex00/hello.py`) and committed to your GitHub repository. Don't forget to test your scripts with real input and edge cases.

Exercise 00 — Rev Alpha

Directory: `day01/ex00/`
Files to submit: `exec.py`
Forbidden: None
Goal: Work with command-line arguments, string reversal, and case swapping.

Task: Make a script that:

- Takes one or more strings as arguments
- Merges them into one string with spaces between them
- Reverses the string and swaps the case of each letter

Example:

```
$> python3 exec.py 'Hello World!'
!DLROw OLLEh
$> python3 exec.py 'Hello' 'my Friend'
DNEIRf YM OLLEh
$> python3 exec.py
```

Extra questions:

- What does ‘sys.argv‘ contain?
- What happens if you run the script with no arguments?
- How could you make the output always end with a newline?

Exercise 01 — The Odd, the Even and the Zero

Directory: `day01/ex01/`
Files to submit: `whois.py`
Forbidden: None
Goal: Practice type conversion, basic conditionals, some errors handling.

Task: Make a script that:

- Takes a single integer argument
- Prints if it's Odd, Even or Zero
- Rejects multiple arguments or non-integers with an error

Example:

```
$> python3 whois.py 12
I'm Even.
$> python3 whois.py 0
I'm Zero.
$> python3 whois.py 3
I'm Odd.
$> python3 whois.py Hello
AssertionError: argument is not an integer
$> python3 whois.py 12 3
AssertionError: more than one argument are provided
```

Extra questions:

- How can you check if a string is an integer?
- What is the difference between '==' and 'is' in Python?
- What happens if you use modulo with a negative number?

Exercise 02 — Functional file

Directory: `day01/ex02/`
Files to submit: `count.py`
Forbidden: None
Goal: Practice writing functions, string analysis, and the use of `__name__ == '__main__'`.

Task:

- Write a function called `text_analyzer` that takes a string and prints:
  - Total number of characters
  - Number of upper- and lowercase letters
  - Number of spaces and punctuation marks
- If the string is not provided or not a string, handle errors appropriately.
- Include a docstring that describes what your function does.
- Use `if __name__ == "__main__"` to make your script executable and testable.

Example:

```
$> python3 count.py "Blazing Fast Odoo!"
The text contains 18 character(s):
- 3 upper letter(s)
- 12 lower letter(s)
- 1 punctuation mark(s)
- 2 space(s)
$> python3 count.py
What is the text to analyze?
>> Ciaooo
The text contains 6 character(s):
- 1 upper letter(s)
- 5 lower letter(s)
```

Extra questions:

- What does `__name__ == '__main__'` allow you to do?
- How can you identify punctuation with built-in Python modules?
- Why is it good practice to write reusable functions instead of printing directly?
- What is the difference between a SCRIPT, a FUNCTION, and a PROGRAM ?

Exercise 03 — Read Me Like a Book

Directory: `day01/ex09/`
Files to submit: `readwrite.py`
Goal: Learn how to read from and write to text files.

Task:

- Create a script that reads from its first argument, capitalize every word in the file and
- Save the result to a new file called as its second argument.

Example:

```
$> cat input.txt
this is a sample file
$> python3 readwrite.py input.txt output.txt
$> cat output.txt
This Is A Sample File
```

Exercise 04 — Greet Everyone

Directory: `day01/ex09/`
Files to submit: `greetings.py`
Goal: Practice using variable arguments (*args) and handling command-line input.

Task: Write a script with a function `greet_everyone(*names)` that:

- Takes multiple names as arguments (via `*args`)
- Prints "Hello *name*!" for each one
- Returns the total number of people greeted
- Also prints the number of people greeted at the end of the script if run directly

Example:

```
$> python3 greetings.py FP WIS CHMG
Hello FP!
Hello WIS!
Hello CHMG!
3

$> python3 greetings.py
0
```

Extra questions:

- What are the advantages of using `*args`?
- How can you detect if no arguments are passed?

Exercise 05 — Count Records in CSV

Directory: `day01/ex06/`
Files to submit: `csv_counter.py`
Goal: Learn to read from CSV files and count rows using Python.

Task: Write a PROGRAM `count_records_in_csv(file_path)` that:

- Takes the path to a CSV file as input (via command-line argument)
- Returns how many data rows the file contains (excluding the header, if any)
- Ignores completely empty lines (even between or after data)

Also write the necessary script logic so the script works from the terminal.

Example:

```
$> python3 csv_counter.py test.csv
3
```

Extra questions:

- What is the purpose of `if __name__ == "__main__"`?
- How would you handle CSVs that use different delimiters (for example semicolons instead of commas)?

Exercise 06 — S.O.S

Directory: `day01/ex07/`
Files to submit: `sos.py`
Goal: Encode a string into Morse code using dictionary mapping.

Task: Create a program that takes one or more string arguments and encodes them into Morse code.

- Accept alphanumeric characters and spaces only
- Merge all arguments into a single string with spaces between them
- Convert each alphanumeric character to its Morse equivalent
- Separate Morse characters with spaces
- Use '/' to represent space characters between words
- If no argument is given, do nothing.
- If unsupported characters (like '/') are passed in input, print an error

Hint: Look up how to use a Python `dictionary` to map letters and digits to Morse code.

Example:

```
$> python3 sos.py "SOS"
... --- ...

$> python3 sos.py
#

$> python3 sos.py "HELLO / WORLD"
ERROR

$> python3 sos.py "SHOW ME THE MONEY" "LEBOWSKY"
... .... --- .-- / -- . / - .... . / -- --- -. . -.-- / .-.. . -... ---
   ↪ .-- ... -.- -.--
```

Extra questions:

- How do you efficiently look up a value for each character in a string?
- What would you change to decode Morse back into normal text?
- Can you ensure lowercase input also works without modifying the dictionary?

Exercise 07 — Draw Me a Diamond

Directory: `day01/ex08/`
Files to submit: `diamond.py`
Goal: Practice nested loops and string formatting.

Task: Write a script that asks for an odd integer and prints a diamond shape using asterisks '*'.

- The widest line of the diamond should have 'n' stars (where 'n' is the number given)
- Each upper and lower row should be centered and symmetrical
- If the number is even or invalid, print an error message

Example:

```
$> python3 diamond.py 5
  *
 ***
*****
 ***
  *
```

Extra questions:

- How can you use string multiplication and padding to create patterns?
- How would you handle invalid inputs (negative numbers, even numbers, etc.)?
- Could you adapt it to print other shapes (pyramids, hourglasses)?

Exercise 08 — Sudoku Solver (Easy Grid)

Directory: `day01/ex09/`
Files to submit: `sudoku.py`
Goal: Implement logic, backtracking, and recursive problem solving. Let's do it in pseudo code first, you'll need your group for it.

Task: Build a basic Sudoku solver for a 4x4 grid. Or go directly for the classic 9x9, you should able to do it ! I trust you :D

- The board is represented as a list of lists (4x4), with 0 indicating empty cells
- Valid numbers are 1–4, each must appear once per row, column, and 2x2 square
- Use recursion and backtracking (google it up) to solve the puzzle

Hint: A Sudoku is a constraint satisfaction problem — only place values that are valid at a given step.

Example:

```
initial_board = [
  [1, 0, 0, 4],
  [0, 0, 0, 0],
  [4, 0, 0, 1],
  [0, 1, 0, 0],
]

# after solving...
# [1, 2, 3, 4]
# [3, 4, 1, 2]
# [4, 3, 2, 1]
# [2, 1, 4, 3]
```

Extra questions:

- Can you implement checks to validate user-input boards?
- How would you measure the complexity or steps used by your solver?