Day 02 – Piscine Python for Data Science

Intro to Python: OOP skills

Summary: This day will help you to get the basic knowledge about OOP approach in Python.

Contents

Chapter I

Preamble

A common complaint to data scientists is that they write shitcode (by the way, only for educational purposes you may find a lot of examples of Python shitcode here). Why? Because an average data scientist uses a lot of inefficient techniques, hard coded variables, and neglects object-oriented programming. Do not be like them.

Just the top examples from the website mentioned above:

- How to get the absolute value on just 6 lines of python

```
def absolute_value(value):
    if str(value)[0]=='-':
        value = -1 * value
        return value
    else:
        return value
```

- How to evaluate factorial of 40000 in approximately 1 second:

```
for module in next_possible_modules:
    import math; math.factorial(40000) # approx. a 1 second operation
    end_time = start_time + timedelta(minutes=module.duration)
```

- Gotta check that date

```
if (SelectionAndTimeData[1] < 2000 or \
        SelectionAndTimeData[2] < 1 or SelectionAndTimeData[2] > 12 or \
        SelectionAndTimeData[3] < 1 or SelectionAndTimeData[3] > 31 or \
        SelectionAndTimeData[4] < 0 or SelectionAndTimeData[4] > 24 or \
        SelectionAndTimeData[5] < 0 or SelectionAndTimeData[5] > 60 or \
        SelectionAndTimeData[2] < 0 or SelectionAndTimeData[2] >60):

    print('************************************************************')
    print(' Entered date is not valid')
    print('************************************************************')
```

Chapter II

Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Here and further we use `Python 3` as the only correct version of Python.
- The python files for python exercises (d01, d02, d03) must have a block in the end: `if __name__ == '__main__'`.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google.
- Remember to discuss on the Intra Piscine forum.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!

Chapter III

Specific instructions of the day

- No code in the global scope. Use functions!
- Each file must be ended by a function call in a condition similar to:

```
if __name__ == '__main__':
    # your tests and your error handling
```

- Any exception not caught will invalidate the work, even in the event of an error that was asked you to test.
- No imports allowed, except those explicitly mentioned in the section “Authorized functions” of the title block of each exercise.
- The interpreter to use is `Python 3`.

Chapter IV

Exercise 00

Exercise 00 : Reporting
Directory to store your solution : `ex00/`
Files to be in the directory : `make_report.py, report.template, numbers.py`
Authorized functions : `import sys, os, re`

Data scientists sometimes produce results that are used by other people, managers, for example. Managers need data to make better decisions. In this exercise, you need to create a report for such a case. You decided not to do it every time manually since the structure is the same and only some numbers change, but to automize your work and to save your time for something more interesting.

Write a program make_report.py that:

- takes a file with .template extension as a parameter,
- reads the content of that file,
- replaces certain patterns by values defined in the numbers.py,
- writes the result to a file with .txt extension.

The following example should be able to be exactly reproduced by your program:

```
$ cat numbers.py
revenue_change = “+21%”
profit_change = “+5%”
average_check = “$321”

$ cat file.template
In the previous quartal the revenue changed by {revenue_change}. The profit change was {profit_change} comparing to the same quartal of the previous year. And the current level of average check is {average_check}.

$ python3 make_report.py file.template

$ cat file.txt
In the previous quartal the revenue changed by +21%. The profit change was +5% comparing to the same quartal of the previous year. And the current level of average check is $321.
```

The file `numbers.py` does not have to have the block `if __name__ == '__main__:`.

Errors, including the wrong file extension, a file does not exist, or the wrong number of arguments will have to be handled.

Chapter V

Exercise 01

Exercise 01 : Communicating with managers
Directory to store your solution : `ex01/`
Files to be in the directory : `manager.py`
Authorized functions : `n/a`

You continue thinking about how to automate your communications with managers.

Create the `Manager` class with the following functionalities:

- A constructor taking a string as a parameter and assigning its value to the `Name` attribute. The default value, "Yet another manager" should be implemented.
- An `__str()__` method that returns the `Name` attribute of the instance.
- An Analytics class (nested) with a simple `__str()__` method that returns the string "`Here is my report, please!`"
- A `working()` method (in the scope of Analytics class) that throws an exception (use the type `Exception`) with the text "`I'm working on your report. Need a little more time.`"
- A `make_analysis()` method that returns an instance of `Analytics` class that you implemented in the `Manager` class.

In your tests, you must instantiate the Manager class twice: once without a name, and the second time with the name "John". Display the name of each instance. Create an analysis and display the result. Answer to the manager that you are working on the report right now. You have to handle the exceptions in your test.

Chapter VI

Exercise 02

Exercise 02 : Employees
Directory to store your solution : `ex02/`
Files to be in the directory : `employees.py`
Authorized functions : `n/a`

You are responsible for creating a beta-version of the employees database.

Create an `Employee` class with the following features:

- A `salary` attribute with a value of 3 000.
- A `name` attribute with the value "data scientist".
- A `description` attribute returning a description of the instance. The value of the description will be "A professional who can analyze data with code."
- An `__str()__` method returning a description of the instance in this form:

```
name : <name attribute>
salary : <salary attribute in EUR per month>
description : <instance's description>
```

- for example, displaying an Employee instance would give:

```
name : "data scientist"
salary : 3000
description : "A professional who can analyze data with code."
```

Then perform the following classes derived from `Employee`.

Janitor :

```
name : "Janitor"
salary : 1000
description : "A profession who keeps the facilities clean."
```

Receptionist :

```
name : "Receptionist"
salary : 1300
description : "A profession that helps visitors to connect with the staff."
```

Cook :

```
name : "Cook"
salary : 3000
description : "A profession who cooks a meal for the staff."
```

Chapter VII

Exercise 03

Exercise 03 : Work and leisure
Directory to store your solution : `ex03/`
Files to be in the directory : `hr.py, employees.py`
Authorized functions : `import random`

Ok, great. Now you have a database where the information about the employees stores. What do we need to do next? We can ask them to work on a task. Let us design such a program!

Create an `HR` class containing:

- A constructor.
- A `Retire` class inheriting from Employee with the name "Retiring employee", the salary 2100, and the description "I am retiring, give me a bonus!" Copy the `employees.py` to your directory for this exercise to be able to use the classes it contains.
- An `ExhaustionException` class inheriting from Exception with the text "This employee is exhausted and cannot work anymore". This text must be defined in the constructor of the exception.
- A `vacation()` method that sends an employee to vacation and gives the ability to be able to work again.
- A `work()` method that will have the following characteristics:
    - **Parameters**: a single parameter (other than self) that will be derived from the class `Employee`.
    - **Return**: once out of two (randomly), the method returns an instance of the class passed as a parameter, and once out of two – an instance of `Retire`.
    - **Exhaustion**: the employees got tired after 10 months of work and stop working.
    - **In the event of a failure**: calling the `work()` method must cause the exception of type `HR.ExhaustionException` until the `vacation()` method is called.
    - **Vacation**: After a call of the `vacation()` method, the `work()` method can operate again without an exception during a cycle of 10 months before the employee gets exhausted again.

In your tests, instantiate `HR class`. Ask various employees to work from the `employees.py` file and display the employee until they get exhausted (you must manage the exception). Send the employee to vacation and try again until they get exhausted again (manage the exception).

Chapter VIII

Exercise 04

Exercise 04 : Autoscaler requests
Directory to store your solution : `ex04/`
Files to be in the directory : `autoscaler.py`
Authorized functions : `n/a`

In the next three exercises, you will work on the same project. You will need to create an autoscaler. What is autoscaler? It is a program that watches the utilization of a cluster. If it is high, it adds several nodes to the cluster. If it is low, it takes several nodes from it and saves money. You are going to work on a prototype of this program.

The first part of the project is to create class `ClusterManagement`. You cannot just suddenly take resources from a cluster. You need to do it gracefully:

- turn all the services off on a node,
- turn the maintenance mode on,
- decommission the node from the cluster.

To add a node into a cluster you need to:

- recommission the node to the cluster,
- turn the maintenance mode off,
- turn all the services on.

The class `ClusterManagement` has the following characteristics:

- A constructor that takes as parameters the cluster URL, the cluster name, the headers needed for API requests (contains a dict `{'X-Requested-By': 'ambari'}`), and the auth credentials (contains a tuple with a login and a password `('admin', 'password')`).

- A method `switcher()` that turns all the services off or on depending on the parameter given. Besides, it takes the list of services names that should be taken into the account as well as the name of the node we are operating on. The function sends an API request (PUT) to the cluster URL with the headers and auth given in the constructor. For the data of the request leave a placeholder. The function must return the status code of the request.

- A method `maintenance()` that turns the maintenance mode off or on depending on the parameter given. Besides, it takes the name of the node to do the job. It sends an API request (PUT) to the cluster URL with the headers and auth given in the constructor. For the data of the request leave a placeholder. The function must return the status code of the request.

- A method `commission()` that decommissions or recommissions a node depending on the parameter given. Besides, it takes the name of the node to do the job. It sends an API request (PUT) to the cluster URL with the headers and auth given in the constructor. For the data of the request leave a placeholder. The function must return the status code of the request.

The cases that need to be handled:

- if a parameter in any method or constructor is not given, raise an exception with the name of the parameter(s).

Chapter IX

Exercise 05

Exercise 05 : Autoscaler logic
Directory to store your solution : `ex05/`
Files to be in the directory : `autoscaler.py`
Authorized functions : `n/a`

By now we have a class that allows us to make some operations with the cluster nodes, but we need to implement some logic: when to scale-up, when to scale-down. To make these decisions we need to have the data about cluster utilization.

You need to create a class `Scaler` which has the following characteristics:

- A constructor that takes as parameters:
  - list of the nodes available for scaling procedure,
  - an instance of `ClsuterManagement`,
  - number of nodes in the cluster,
  - RAM per node,
  - CPU per node.
- In the constructor, you need to instantiate queues that will store the metrics about the current available CPU and RAM in the cluster.
- A method `get_params()` that makes API requests to get the current available CPU and RAM in the cluster, calculates the available percentage of both resources and returns them.
- A method `make_queue()` that takes as a parameter the evaluation period during which we monitor the changes in the availability of the resources. The method checks if we collected data for more than the evaluation period. If it is true, then we need to start deleting values from the queues from the left. If it is false, we need to collect more data into our queues using `get_params()`. The method returns the queues.
- A method `assert_down()` that checks if there are the conditions to scale down a node in the cluster. It takes as parameters `scaled_down_thresholds`: the percentage of available resources. It checks if the queues are stable in the values (only one unique value during the evaluation period). It checks if the percentage of available resources is higher than the thresholds. It returns `True` if both conditions are met, `False` otherwise.

- An identical method `assert_up()` that does the same thing, except it checks if the percentage of available resources is lower than the thresholds. It returns `True` if both conditions are met, `False` otherwise.
- A method `scale()` that takes as parameters the name of the node and the mode (up or down). It launches the sequence of the commands from the previous exercise depending on the mode given.

Chapter X

Exercise 06

Exercise 06 : Finishing touches
Directory to store your solution : `ex06/`
Files to be in the directory : `autoscaler.py, config.py, monitor.py`
Authorized functions : `n/a`

By now you wrote some logic of the autoscaler. But what if during the production there will be some problems that we will need to debug? How are you going to do it? That is right! You need to log it. So the first task of the exercise: each and every method in all the classes should log useful information for debugging.

The second task is you need to store all the configuration values in a file `config.py` (cluster URL, cluster name, CPU, and RAM per node, etc.).

The third task is you need to write a program `monitor.py` where you can import `autoscaler.py` and `config.py`. The program should constantly monitor if the right conditions are met. If they are met, then call the corresponding methods.

The fourth task. Write a function that sends a message to a Slack channel using webhooks. The message should contain: "`A node has been successfully scaled-down`" or "`A node has been successfully scaled-up`". Yeah, we know that you do not have admin rights to create a custom integration in the School workspace, but be creative, create your own Slack workspace! It will be helpful in that branch.

The checklist for the day.

The survey for the day.