(/)

Curriculum
SE Foundations
Average: **108.76%**

0x0B. Python - Input/Output

Python

By: Guillaume

Weight: 1

Project over - took place from Jun 13, 2023 6:00 AM to Jun 14, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell...

- **Auto QA review:** 126.0/140 mandatory & 5.0/21 optional
- **Altogether: 111.43%**
  - Mandatory: 90.0%
  - Optional: 23.81%
  - Calculation: 90.0% + (90.0% * 23.81%) == **111.43%**

Resources

Read or watch:

- 7.2. Reading and Writing Files (/rltoken/hFlrZ9E1XROVWcjwwyF52A)
- 8.7. Predefined Clean-up Actions (/rltoken/0OZ9fzPRjmKWZsID9IRJSg)
- Dive Into Python 3: Chapter 11. Files (/rltoken/0osPfNU5d3Shh9PFWgYm9A) (*until "11.4 Binary Files" (included)*)
- JSON encoder and decoder (/rltoken/l0B9_pFn1tgBvE7FrT14Zw)
- Learn to Program 8 : Reading / Writing Files (/rltoken/ZvtAdnUzjnEVu1sjg3m_tQ)
- Automate the Boring Stuff with Python (/rltoken/Ej8YjhxLXpzHW7_rNMd9XQ) (*ch. 8 p 180-183 and ch. 14 p 326-333*)
- All about py-file I/O (/rltoken/TUatlpPV27S4zPogmQIPnQ)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/x2TxSf8LF65dpNOPSGtXgQ), **without the help of Google**:

General

- Why Python programming is awesome
- How to open a file
- How to write text in a file
- How to read the full content of a file
- How to read a file line by line
- How to move the cursor in a file
- How to make sure a file is closed after using it
- What is and how to use the `with` statement
- What is `JSON`
- What is serialization
- What is deserialization
- How to convert a Python data structure to a JSON string
- How to convert a JSON string to a Python data structure

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Help

Requirements

Python Scripts

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be interpreted/compiled on Ubuntu 20.04 LTS using python3 (version 3.8.5)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/usr/bin/python3`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the pycodestyle (version `2.8.*`)
- All your files must be executable
- The length of your files will be tested using `wc`

Python Test Cases

- Allowed editors: `vi`, `vim`, `emacs`
- All your files should end with a new line
- All your test files should be inside a folder `tests`
- All your test files should be text files (extension: `.txt`)
- All your tests should be executed by using this command: `python3 -m doctest ./tests/*`
- All your modules should have a documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)
- All your classes should have a documentation (`python3 -c 'print(__import__("my_module").MyClass.__doc__)'`)
- All your functions (inside and outside a class) should have a documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'` and `python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)'`)
- A documentation is not a simple word, it's a real sentence explaining what's the purpose of the module, class or method (the length of it will be verified)
- We strongly encourage you to work together on test cases, so that you don't miss any edge case

Tasks

0. Read file

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that reads a text file (`UTF8`) and prints it to stdout:

- Prototype: `def read_file(filename=""):`
- You must use the `with` statement
- You don't need to manage `file permission` or `file doesn't exist` exceptions.
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0B$ cat 0-main.py
#!/usr/bin/python3
read_file = __import__('0-read_file').read_file

read_file("my_file_0.txt")

guillaume@ubuntu:~/0x0B$ cat my_file_0.txt
We offer a truly innovative approach to education:
focus on building reliable applications and scalable systems, take on real-world challenges, collaborate with your peers.

A school every software engineer would have dreamt of!
guillaume@ubuntu:~/0x0B$ ./0-main.py
We offer a truly innovative approach to education:
focus on building reliable applications and scalable systems, take on real-world challenges, collaborate with your peers.

A school every software engineer would have dreamt of!
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `0-read_file.py`

Done! | Help | Check your code | Get a sandbox | QA Review

1. Write to a file (/)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that writes a string to a text file (`UTF8`) and returns the number of characters written:

- Prototype: `def write_file(filename="", text=""):`
- You must use the `with` statement
- You don't need to manage file permission exceptions.
- Your function should create the file if doesn't exist.
- Your function should overwrite the content of the file if it already exists.
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0B$ cat 1-main.py
#!/usr/bin/python3
write_file = __import__('1-write_file').write_file

nb_characters = write_file("my_first_file.txt", "This School is so cool!\n")
print(nb_characters)

guillaume@ubuntu:~/0x0B$ ./1-main.py
29
guillaume@ubuntu:~/0x0B$ cat my_first_file.txt
This School is so cool!
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `1-write_file.py`

Done! | Help | Check your code | Get a sandbox | QA Review

2. Append to a file

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that appends a string at the end of a text file (`UTF8`) and returns the number of characters added:

- Prototype: `def append_write(filename="", text=""):`
- If the file doesn't exist, it should be created
- You must use the `with` statement
- You don't need to manage `file permission` or `file doesn't exist` exceptions.
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0B$ cat 2-main.py
#!/usr/bin/python3
append_write = __import__('2-append_write').append_write

nb_characters_added = append_write("file_append.txt", "This School is so cool!\n")
print(nb_characters_added)

guillaume@ubuntu:~/0x0B$ cat file_append.txt
cat: file_append.txt: No such file or directory
guillaume@ubuntu:~/0x0B$ ./2-main.py
29
guillaume@ubuntu:~/0x0B$ cat file_append.txt
This School is so cool!
guillaume@ubuntu:~/0x0B$ ./2-main.py
29
guillaume@ubuntu:~/0x0B$ cat file_append.txt
This School is so cool!
This School is so cool!
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `2-append_write.py`

Done! | Help | Check your code | Get a sandbox | QA Review

3. To JSON string

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that returns the JSON representation of an object (string):

- Prototype: `def to_json_string(my_obj):`
- You don't need to manage exceptions if the object can't be serialized.

```
guillaume@ubuntu:~/0x0B$ cat 3-main.py
#!/usr/bin/python3
to_json_string = __import__('3-to_json_string').to_json_string

my_list = [1, 2, 3]
s_my_list = to_json_string(my_list)
print(s_my_list)
print(type(s_my_list))

my_dict = {
    'id': 12,
    'name': "John",
    'places': [ "San Francisco", "Tokyo" ],
    'is_active': True,
    'info': {
        'age': 36,
        'average': 3.14
    }
}
s_my_dict = to_json_string(my_dict)
print(s_my_dict)
print(type(s_my_dict))

try:
    my_set = { 132, 3 }
    s_my_set = to_json_string(my_set)
    print(s_my_set)
    print(type(s_my_set))
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/0x0B$ ./3-main.py
[1, 2, 3]
<class 'str'>
{"id": 12, "is_active": true, "name": "John", "info": {"average": 3.14, "age": 36}, "places": ["San Francisco", "Tokyo"]}
<class 'str'>
[TypeError] {3, 132} is not JSON serializable
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `3-to_json_string.py`

Done! | Help | Check your code | Get a sandbox | QA Review

4. From JSON string to Object

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that returns an object (Python data structure) represented by a JSON string:

- Prototype: `def from_json_string(my_str):`
- You don't need to manage exceptions if the JSON string doesn't represent an object.

```
guillaume@ubuntu:~/0x0B$ cat 4-main.py
#!/usr/bin/python3
from json_string = __import__('4-from_json_string').from_json_string

s_my_list = "[1, 2, 3]"
my_list = from_json_string(s_my_list)
print(my_list)
print(type(my_list))

s_my_dict = """
{"is_active": true, "info": {"age": 36, "average": 3.14},
"id": 12, "name": "John", "places": ["San Francisco", "Tokyo"]}
"""
my_dict = from_json_string(s_my_dict)
print(my_dict)
print(type(my_dict))

try:
    s_my_dict = """
    {"is_active": true, 12 }
    """
    my_dict = from_json_string(s_my_dict)
    print(my_dict)
    print(type(my_dict))
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/0x0B$ ./4-main.py
[1, 2, 3]
<class 'list'>
{'id': 12, 'is_active': True, 'name': 'John', 'info': {'age': 36, 'average': 3.14}, 'places': ['San Francisco', 'Tokyo']}
<class 'dict'>
[ValueError] Expecting property name enclosed in double quotes: line 2 column 25 (char 25)
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `4-from_json_string.py`

Done! | Help | Check your code | Get a sandbox | QA Review

5. Save Object to a file

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that writes an Object to a text file, using a JSON representation:

- Prototype: `def save_to_json_file(my_obj, filename):`
- You must use the `with` statement
- You don't need to manage exceptions if the object can't be serialized.
- You don't need to manage file permission exceptions.

```
guillaume@ubuntu:~/0x0B$ cat 5-main.py
#!/usr/bin/python3
save_to_json_file = __import__('5-save_to_json_file').save_to_json_file

filename = "my_list.json"
my_list = [1, 2, 3]
save_to_json_file(my_list, filename)

filename = "my_dict.json"
my_dict = {
    'id': 12,
    'name': "John",
    'places': [ "San Francisco", "Tokyo" ],
    'is_active': True,
    'info': {
        'age': 36,
        'average': 3.14
    }
}
save_to_json_file(my_dict, filename)

try:
    filename = "my_set.json"
    my_set = { 132, 3 }
    save_to_json_file(my_set, filename)
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/0x0B$ ./5-main.py
[TypeError] {3, 132} is not JSON serializable
guillaume@ubuntu:~/0x0B$ cat my_list.json ; echo ""
[1, 2, 3]
guillaume@ubuntu:~/0x0B$ cat my_dict.json ; echo ""
{"name": "John", "places": ["San Francisco", "Tokyo"], "id": 12, "info": {"average": 3.14, "age": 36}, "is_active": true}
guillaume@ubuntu:~/0x0B$ cat my_set.json ; echo ""

guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `5-save_to_json_file.py`

☑ Done! | Help | Check your code | >_ Get a sandbox | QA Review

6. Create object from a JSON file

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that creates an Object from a “JSON file”:

- Prototype: `def load_from_json_file(filename):`
- You must use the `with` statement
- You don’t need to manage exceptions if the JSON string doesn’t represent an object.
- You don’t need to manage file permissions / exceptions.

```
guillaume@ubuntu:~/0x0B$ cat my_fake.json
{"is_active": true, 12 }
guillaume@ubuntu:~/0x0B$ cat 6-main.py
#!/usr/bin/python3
load_from_json_file = __import__('6-load_from_json_file').load_from_json_file

filename = "my_list.json"
my_list = load_from_json_file(filename)
print(my_list)
print(type(my_list))

filename = "my_dict.json"
my_dict = load_from_json_file(filename)
print(my_dict)
print(type(my_dict))

try:
    filename = "my_set_doesnt_exist.json"
    my_set = load_from_json_file(filename)
    print(my_set)
    print(type(my_set))
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

try:
    filename = "my_fake.json"
    my_fake = load_from_json_file(filename)
    print(my_fake)
    print(type(my_fake))
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/0x0B$ cat my_list.json ; echo ""
[1, 2, 3]
guillaume@ubuntu:~/0x0B$ cat my_dict.json ; echo ""
{"name": "John", "places": ["San Francisco", "Tokyo"], "id": 12, "info": {"average": 3.14, "age": 36}, "is_active": true}
guillaume@ubuntu:~/0x0B$ cat my_fake.json ; echo ""
{"is_active": true, 12 }
guillaume@ubuntu:~/0x0B$ ./6-main.py
[1, 2, 3]
<class 'list'>
{'name': 'John', 'info': {'age': 36, 'average': 3.14}, 'id': 12, 'places': ['San Francisco', 'Tokyo'], 'is_active': True}
<class 'dict'>
[FileNotFoundError] [Errno 2] No such file or directory: 'my_set_doesnt_exist.json'
[ValueError] Expecting property name enclosed in double quotes: line 1 column 21 (char 20)
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `6-load_from_json_file.py`

Done! | Help | Check your code | Get a sandbox | QA Review

7. Load, add, save

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Write a script that adds all arguments to a Python list, and then save them to a file:

- You must use your function `save_to_json_file` from `5-save_to_json_file.py`
- You must use your function `load_from_json_file` from `6-load_from_json_file.py`
- The list must be saved as a JSON representation in a file named `add_item.json`
- If the file doesn't exist, it should be created
- You don't need to manage file permissions / exceptions.

```
guillaume@ubuntu:~/0x0B$ cat add_item.json
cat: add_item.json: No such file or directory
guillaume@ubuntu:~/0x0B$ ./7-add_item.py
guillaume@ubuntu:~/0x0B$ cat add_item.json ; echo ""
[]
guillaume@ubuntu:~/0x0B$ ./7-add_item.py Best School
guillaume@ubuntu:~/0x0B$ cat add_item.json ; echo ""
["Best", "School"]
guillaume@ubuntu:~/0x0B$ ./7-add_item.py 89 Python C
guillaume@ubuntu:~/0x0B$ cat add_item.json ; echo ""
["Best", "School", "89", "Python", "C"]
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `7-add_item.py`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

8. Class to JSON

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that returns the dictionary description with simple data structure (list, dictionary, string, integer and boolean) for JSON serialization of an object:

- Prototype: `def class_to_json(obj):`
- `obj` is an instance of a Class
- All attributes of the `obj` Class are serializable: list, dictionary, string, integer and boolean
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0B$ cat 8-my_class.py
#!/usr/bin/python3
""" My class module
"""

class MyClass:
    """ My class
    """

    def __init__(self, name):
        self.name = name
        self.number = 0

    def __str__(self):
        return "[MyClass] {} - {:d}".format(self.name, self.number)

guillaume@ubuntu:~/0x0B$ cat 8-main.py
#!/usr/bin/python3
MyClass = __import__('8-my_class').MyClass
class_to_json = __import__('8-class_to_json').class_to_json

m = MyClass("John")
m.number = 89
print(type(m))
print(m)

mj = class_to_json(m)
print(type(mj))
print(mj)

guillaume@ubuntu:~/0x0B$ ./8-main.py
<class '8-my_class.MyClass'>
[MyClass] John - 89
<class 'dict'>
{'name': 'John', 'number': 89}
guillaume@ubuntu:~/0x0B$
guillaume@ubuntu:~/0x0B$ cat 8-my_class_2.py
#!/usr/bin/python3
""" My class module
"""

class MyClass:
    """ My class
    """

    score = 0

    def __init__(self, name, number = 4):
        self.__name = name
        self.number = number
        self.is_team_red = (self.number % 2) == 0

    def win(self):
        self.score += 1

    def lose(self):
        self.score -= 1

    def __str__(self):
        return "[MyClass] {} - {:d} => {:d}".format(self.__name, self.number, self.score)

guillaume@ubuntu:~/0x0B$ cat 8-main_2.py
#!/usr/bin/python3
MyClass = __import__('8-my_class_2').MyClass
class_to_json = __import__('8-class_to_json').class_to_json

m = MyClass("John")
m.win()
print(type(m))
print(m)

mj = class_to_json(m)
print(type(mj))
print(mj)

guillaume@ubuntu:~/0x0B$ ./8-main_2.py
<class '8-my_class_2.MyClass'>
[MyClass] John - 4 => 1
<class 'dict'>
```

```
{'number': 4, '_MyClass__name': 'John', 'is_team_red': True, 'score': 1}
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `8-class_to_json.py`

Done! | Help | Check your code | Get a sandbox | QA Review

9. Student to JSON

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a class `Student` that defines a student by:

- Public instance attributes:
  - `first_name`
  - `last_name`
  - `age`
- Instantiation with `first_name`, `last_name` and `age`: `def __init__(self, first_name, last_name, age):`
- Public method `def to_json(self):` that retrieves a dictionary representation of a `Student` instance (same as `8-class_to_json.py`)
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0B$ cat 9-main.py
#!/usr/bin/python3
Student = __import__('9-student').Student

students = [Student("John", "Doe", 23), Student("Bob", "Dylan", 27)]

for student in students:
    j_student = student.to_json()
    print(type(j_student))
    print(j_student['first_name'])
    print(type(j_student['first_name']))
    print(j_student['age'])
    print(type(j_student['age']))

guillaume@ubuntu:~/0x0B$ ./9-main.py
<class 'dict'>
John
<class 'str'>
23
<class 'int'>
<class 'dict'>
Bob
<class 'str'>
27
<class 'int'>
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `9-student.py`

Done! | Help | Check your code | Get a sandbox | QA Review

10. Student to JSON with filter

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a class `Student` that defines a student by: (based on `9-student.py`)

- Public instance attributes:

(/)

  - `first_name`
  - `last_name`
  - `age`
- Instantiation with `first_name`, `last_name` and `age`: `def __init__(self, first_name, last_name, age):`
- Public method `def to_json(self, attrs=None):` that retrieves a dictionary representation of a `Student` instance (same as `8-class_to_json.py`):
  - If `attrs` is a list of strings, only attribute names contained in this list must be retrieved.
  - Otherwise, all attributes must be retrieved
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0B$ cat 10-main.py
#!/usr/bin/python3
Student = __import__('10-student').Student

student_1 = Student("John", "Doe", 23)
student_2 = Student("Bob", "Dylan", 27)

j_student_1 = student_1.to_json()
j_student_2 = student_2.to_json(['first_name', 'age'])
j_student_3 = student_2.to_json(['middle_name', 'age'])

print(j_student_1)
print(j_student_2)
print(j_student_3)

guillaume@ubuntu:~/0x0B$ ./10-main.py
{'age': 23, 'last_name': 'Doe', 'first_name': 'John'}
{'age': 27, 'first_name': 'Bob'}
{'age': 27}
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `10-student.py`

Done! | Help | Check your code | Get a sandbox | QA Review

11. Student to disk and reload

mandatory

Score: 50.0% (*Checks completed: 100.0%*)

Write a class `Student` that defines a student by: (based on `10-student.py`)

- Public instance attributes:
  - `first_name`
  - `last_name`
  - `age`
- Instantiation with `first_name`, `last_name` and `age`: `def __init__(self, first_name, last_name, age):`
- Public method `def to_json(self, attrs=None):` that retrieves a dictionary representation of a `Student` instance (same as `8-class_to_json.py`):
  - If `attrs` is a list of strings, only attributes name contain in this list must be retrieved.
  - Otherwise, all attributes must be retrieved
- Public method `def reload_from_json(self, json):` that replaces all attributes of the `Student` instance:
  - You can assume `json` will always be a dictionary
  - A dictionary key will be the public attribute name
  - A dictionary value will be the value of the public attribute
- You are not allowed to import any module

Now, you have a simple implementation of a serialization and deserialization mechanism (concept of representation of an object to another format, without losing any information and allow us to rebuild an object based on this representation)

```
guillaume@ubuntu:~/0x0B$ cat 11-main.py
#!/usr/bin/python3
import os
import sys

Student = __import__('11-student').Student
read_file = __import__('0-read_file').read_file
save_to_json_file = __import__('5-save_to_json_file').save_to_json_file
load_from_json_file = __import__('6-load_from_json_file').load_from_json_file

path = sys.argv[1]

if os.path.exists(path):
    os.remove(path)

student_1 = Student("John", "Doe", 23)
j_student_1 = student_1.to_json()
print("Initial student:")
print(student_1)
print(type(student_1))
print(type(j_student_1))
print("{} {} {}".format(student_1.first_name, student_1.last_name, student_1.age))


save_to_json_file(j_student_1, path)
read_file(path)
print("\nSaved to disk")


print("Fake student:")
new_student_1 = Student("Fake", "Fake", 89)
print(new_student_1)
print(type(new_student_1))
print("{} {} {}".format(new_student_1.first_name, new_student_1.last_name, new_student_1.age))


print("Load dictionary from file:")
new_j_student_1 = load_from_json_file(path)

new_student_1.reload_from_json(j_student_1)
print(new_student_1)
print(type(new_student_1))
print("{} {} {}".format(new_student_1.first_name, new_student_1.last_name, new_student_1.age))

guillaume@ubuntu:~/0x0B$ ./11-main.py student.json
Initial student:
<11-student.Student object at 0x7f832826eda0>
<class '11-student.Student'>
<class 'dict'>
John Doe 23
{"last_name": "Doe", "first_name": "John", "age": 23}
Saved to disk
Fake student:
<11-student.Student object at 0x7f832826edd8>
<class '11-student.Student'>
Fake Fake 89
Load dictionary from file:
<11-student.Student object at 0x7f832826edd8>
<class '11-student.Student'>
John Doe 23
guillaume@ubuntu:~/0x0B$ cat student.json ; echo ""
{"last_name": "Doe", "first_name": "John", "age": 23}
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `11-student.py`

Done! | Help | Check your code | Get a sandbox | QA Review

12. Pascal's Triangle

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Technical interview preparation:

- (/) You are not allowed to google anything
- Whiteboard first

Create a function `def pascal_triangle(n):` that returns a list of lists of integers representing the Pascal's triangle of `n`:

- Returns an empty list if `n <= 0`
- You can assume `n` will be always an integer
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0B$ cat 12-main.py
#!/usr/bin/python3
"""
12-main
"""
pascal_triangle = __import__('12-pascal_triangle').pascal_triangle

def print_triangle(triangle):
    """
    Print the triangle
    """
    for row in triangle:
        print("[{}]".format(",".join([str(x) for x in row])))


if __name__ == "__main__":
    print_triangle(pascal_triangle(5))

guillaume@ubuntu:~/0x0B$
guillaume@ubuntu:~/0x0B$ ./12-main.py
[1]
[1,1]
[1,2,1]
[1,3,3,1]
[1,4,6,4,1]
guillaume@ubuntu:~/0x0B$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `12-pascal_triangle.py`

☑ Done! | Help | Check your code | >_ Get a sandbox | QA Review

13. Search and update

#advanced

Score: 50.0% (*Checks completed: 100.0%*)

Write a function that inserts a line of text to a file, after each line containing a specific string (see example):

- Prototype: `def append_after(filename="", search_string="", new_string=""):`
- You must use the `with` statement
- You don't need to manage `file permission` or `file doesn't exist` exceptions.
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0B$ cat 100-main.py
#!/usr/bin/python3
append_after = __import__('100-append_after').append_after

append_after("append_after_100.txt", "Python", "\"C is fun!\"\n")

guillaume@ubuntu:~/0x0B$ cat append_after_100.txt
At Holberton School,
Python is really important,
But it can be very hard if:
- You don't get all Pythonic tricks
- You don't have strong C knowledge.
guillaume@ubuntu:~/0x0B$ ./100-main.py
guillaume@ubuntu:~/0x0B$ cat append_after_100.txt
At School,
Python is really important,
"C is fun!"
But it can be very hard if:
- You don't get all Pythonic tricks
"C is fun!"
- You don't have strong C knowledge.
guillaume@ubuntu:~/0x0B$ ./100-main.py
guillaume@ubuntu:~/0x0B$ cat append_after_100.txt
At School,
Python is really important,
"C is fun!"
But it can be very hard if:
- You don't get all Pythonic tricks
"C is fun!"
"C is fun!"
- You don't have strong C knowledge.
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `100-append_after.py`

Done! | Help | Check your code | Get a sandbox | QA Review

14. Log parsing

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a script that reads `stdin` line by line and computes metrics:

- Input format: `<IP Address> - [<date>] "GET /projects/260 HTTP/1.1" <status code> <file size>`
- Each 10 lines and after a keyboard interruption (`CTRL + C`), prints those statistics since the beginning:
  - Total file size: `File size: <total size>`
  - where is the sum of all previous (see input format above)
  - Number of lines by status code:
    - possible status code: `200` , `301` , `400` , `401` , `403` , `404` , `405` and `500`
    - if a status code doesn't appear, don't print anything for this status code
    - format: `<status code>: <number>`
    - status codes should be printed in ascending order

```
guillaume@ubuntu:~/0x0B$ cat 101-generator.py
#!/usr/bin/python3
import random
import sys
from time import sleep
import datetime

for i in range(10000):
    sleep(random.random())
    sys.stdout.write("{:d}.{:d}.{:d}.{:d} - [{}] \"GET /projects/260 HTTP/1.1\" {} {}\n".format(
        random.randint(1, 255), random.randint(1, 255), random.randint(1, 255), random.randint(1, 255),
        datetime.datetime.now(),
        random.choice([200, 301, 400, 401, 403, 404, 405, 500]),
        random.randint(1, 1024)
    ))
    sys.stdout.flush()

guillaume@ubuntu:~/0x0B$ ./101-generator.py | ./101-stats.py
File size: 5213
200: 2
401: 1
403: 2
404: 1
405: 1
500: 3
File size: 11320
200: 3
301: 2
400: 1
401: 2
403: 3
404: 4
405: 2
500: 3
File size: 16305
200: 3
301: 3
400: 4
401: 2
403: 5
404: 5
405: 4
500: 4
^CFile size: 17146
200: 4
301: 3
400: 4
401: 2
403: 6
404: 6
405: 4
500: 4
Traceback (most recent call last):
  File "./101-stats.py", line 15, in <module>
Traceback (most recent call last):
  File "./101-generator.py", line 8, in <module>
    for line in sys.stdin:
KeyboardInterrupt
    sleep(random.random())
KeyboardInterrupt
guillaume@ubuntu:~/0x0B$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0B-python-input_output`
- File: `101-stats.py`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review