Project for students - Testers

There are several Python tasks. You can choose preferred homework and do just one, several or all of them. The sky is the limit.

1) Palindrome

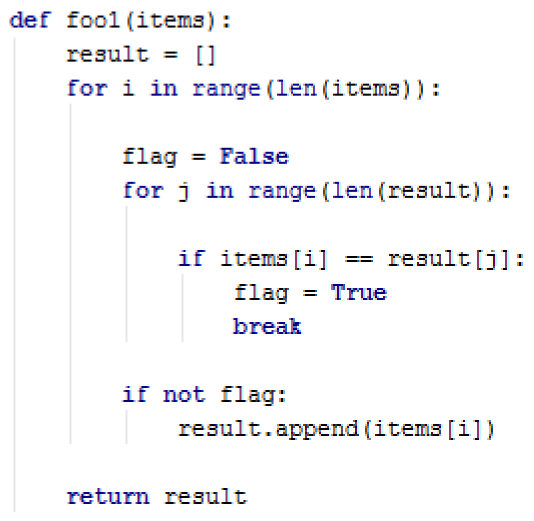
Write a program to check whether the given string is a palindrome:

- String shall be given as command line parameter
- Palindrome example: „abcddcba“, „racecar“, „level“, „mom“

The program shall accept a string to test (for palindrome) using command line parameters.

2) Code review

Simplify the following code:



```python
def foo1(items):
    result = []
    for i in range(len(items)):

        flag = False
        for j in range(len(result)):

            if items[i] == result[j]:
                flag = True
                break

        if not flag:
            result.append(items[i])

    return result
```

Inspect, what is the meaning of above written code and write simplified version.

3) Piškvorky

Write a simple algorithm (or artificial intelligence) for Piškvorky in Python language.

Make it a console application with 2 parameters setting the x and y size of the grid.

Command line run example:

`ai.exe 20 20`

The sequence of moves will be communicated trough standard input and output.

On stdin it should accept commands:

- `start` - gives the AI a first move
- `{x} {y}` - opponent's move, x and y are integers
- `exit` - close the application

On stdout it should print those messages:

- `{x} {y}` - the AI's move
- `wrong` - if the AI thinks the opponent makes invalid moves, or cannot find any possible moves, or anything else went wrong

You can somewhat test your ai with this python project https://github.com/Ormei/pyskvorky

Document your solution.

3) Cesar cipher

Write a simple Caesar cipher implementation in Python language (see Caesar cipher - Wikipedia).

The program shall read plain text from stdin and output encrypted text via stdout.

4) Code generator

Create Python script that generates new file called "output.cpp" based on data stored in JSON file passed as parameter for Python script.

See example of input and output files in "input_example.json" and "output_example.cpp"

output_example.cpp:

- lines 23-25 and 34-36 will be generated by the python script
- all other lines unchanged (see template.cpp)

input_example.json

- structure of file and attribute names: "name", "tag", "type", "size", are fixed. It's name will not be changed in any input json file.
- item values (e.g. "ProductNumber", 14, "string", 90) can by any string/number. Values must be passed as arguments to function initDataItem(..) in output.cpp
- collection names (e.g. "interface", "data") are variable strings (not fixed), you just need to push all items from each collection via function Colection.NAME_OF_COLLECTION->push("NAME_OF_ITEM") in output.cpp
- number of collections, and number of items in collections is variable

Document your solution.

input_example.json

```json
{
    "interface": [
        {
            "name": "ProductNumber",
            "tag": 14,
            "type": "string",
            "size": 90
        },
        {
            "name": "SerialNumber",
            "tag": 18,
            "type": "int",
            "size": 8
        }
    ],
    "data": [
        {
            "name": "DeviceType",
            "tag": 4275,
            "type": "bool",
            "size": 1
        }
    ]
}
```

output_example.cpp

```
#include "modules/TestDevice.hpp"

#include "iolink/iolink.hpp"


#define True true

#define False false


DeviceAB::DeviceAB(uint8_t slot):

    Module(slot, "IODevice")

{

        initItems();

        initCollections();

}

DeviceAB::~DeviceAB()

{

}

void DeviceAB::initItems()

{

        initDataItem("ProductNumber", 14, "string", 90);

        initDataItem("SerialNumber", 18, "int", 8);

        initDataItem("DeviceType", 4275, "bool", 1);

}

void DeviceAB::initCollections()

{

std::shared_ptr<Iolink> Colection = Iolink::getInstance();

        Colection.interface->push("ProductNumber");

        Colection.interface->push("SerialNumber");

        Colection.data->push("DeviceType");

}
```

template.cpp

```
#include "modules/TestDevice.hpp"

#include "iolink/iolink.hpp"


#define True true

#define False false


DeviceAB::DeviceAB(uint8_t slot):

        Module(slot, "IODevice")

{

        initItems();

        initCollections();

}


DeviceAB::~DeviceAB()

{

}


void DeviceAB::initItems()

{

        // generated code here

}

void DeviceAB::initCollections(){

        std::shared_ptr<Iolink> Colection = Iolink::getInstance();

        // generated code here

}
```