Advanced Python Programming

Assignment 3

'Reflecting upon itself'

Welcome to the new assignment! This one is going to be easier than any other before. We promise! (suspicious face) This time we will undertake so called "Reflection" techniques. Does it splash a tide of fear to you? Don't worry dear, each task will gradually make you bolder as we move along. We keep our plank as usual – **5 tasks** and **2 points** per each. So 10 points is what you get when all is done. Our methodology of giving the points stays the same: 1 point – "well..that's ok", 2 – "well done!", 0 – "where is the task n?".

As you know, whatever happens – ask for any appropriate help. When you're done, upload your assignment as an archive to the"Assignments" tab of the course team in Microsoft Teams.

1. Create a decorator that dumps original source code of the function. Here is how:

```
$ cat reflect.py
# Python 3.7.4 (no shebang line)
...here is your @reflect implementation...

@reflect
def foo():
    print("bar")

if __name__ == "__main__":
    foo()

$ python reflect.py
def foo():
    print("bar")
```

   Hm...is that **quine** or what? Give us a brief comment right in the *docstring* of your decorator. (*brief* from now on means no more than 1-3 sentences!)

2. First, extend your decorator so that it can work with any arbitrary function (ie with any *signature*). Second, let it dump a bit more information:

```
$ cat reflect.py
...your decorator's code...

@reflect
def foo(bar1, bar2=""):
    """
    This function does nothing useful
    :param bar1: description
    :param bar2: description
    """
    print("some\nmultiline\noutput")
```

```
if __name__ == "__main__":
    foo(None, bar2="")

$ python reflect.py
Name:    foo
Type:    <class 'function'>
Sign:    (bar1, bar2='')

Args:    positional (None,)
         key=worded {'bar2': ''}

Doc:     This function does nothing useful
         :param bar1: description
         :param bar2: description

Source:  @reflect
         def foo(bar1, bar2=""):
             """
             This function does nothing useful
             :param bar1: description
             :param bar2: description
             """
             print("some\nmultiline\noutput")

Output:  some
         multiline
         output
```

Notice how neat the **alignment** of the output is! Attribute's name goes first, then the line(s) of an output indented into a column. We want you to do exactly the same! But what we do *not* want is to implement you a general-purpose decorator that can recognize classes, modules, etc. Only functions! Do not overcomplicate, just keep it simple!

3. Now, apply your `@reflect` decorator against its own definition! Does your head start spinning? Is that possible? If no why? Put a brief comment in the "main" section and give us *workaround* instead:

```
$ cat reflect.py
@reflect # hm?
...your decorator definition goes here...

@reflect
def foo(bar1, bar2=""):
    """
    This function does nothing useful
    :param bar1: description
```

```
        :param bar2: description
        """
        print("some\nmultiline\noutput")

    if __name__ == "__main__":
        foo(None, bar2="")
        # explanation
        ...workaround...

    $ python reflect.py
    Name:    foo
    Type:    <class 'function'>
    Sign:    (bar1, bar2='')
    ...

    Name:    reflect
    Type:    <class 'function'>
    ...
```

4. Now, we're going to add one more attribute. This time it will be a Complexity! We promised to be easy this turn. So we just want you to prepare for the next assignment. Modify your decorator so that it shows `Complx:` which counts *one* particular word from within the `Source:`. The word might be `for`, `print` or `if`. Example:

```
$ python reflect.py
Name:    foo
Type:    <class 'function'>
Sign:    (bar1, bar2='')
Args:    ...
Doc:     ...
Complx:  {'print': 1}
Source:  ...
Output:  ...

Name:    reflect
Type:    <class 'function'>
Sign:    ...
Args:    ...
Doc:     ...
Complx:  {'print': 14}
Source:  ...
Output:  ...
```

   Important! Regex is forbidden.

5. And finally, make out of your decorator's code a complete python module (singe-filed though)! We want you to be able to import it as a good library for your own

needs. The `foo` function you can surely throw away. What we've got:

```
$ ls ./
reflect.py
myprogram.py
$ cat myprogram.py
from reflect import reflect

@reflect
def myfunction():
    print("nothing\nuseful")

myfunction()
# move your workaround with an explanation here
# so we keep the module clean

$ python myprogram.py | tee output.txt
Name:    myfunction
Type:    <class 'function'>
...

Name:    reflect
Type:    <class 'function'>
...

(It works! Thank you dear!)

$ ls ./
reflect.py
myprogram.py
output.txt
```

Resources

1. https://docs.python.org/3/library/inspect.html
2. https://www.python.org/dev/peps/pep-0257/
3. https://docs.python.org/3/library/contextlib.html#contextlib.redirect_stdout