Estimated time

20-45 minutes

Level of difficulty

Easy/Medium

Objectives

- improve the student's skills in defining classes;
- using existing classes to create new classes equipped with new functionalities.

Scenario

We've showed you recently how to extend *Stack* possibilities by defining a new class (i.e., a subclass) which retains all inherited traits and adds some new ones.

Your task is to extend the `Stack` class behavior in such a way so that the class is able to count all the elements that are pushed and popped (we assume that counting pops is enough). Use the `Stack` class we've provided in the editor.

Follow the hints:

- introduce a property designed to count pop operations and name it in a way which guarantees hiding it;
- initialize it to zero inside the constructor;
- provide a method which returns the value currently assigned to the counter (name it `get_counter()`).

Complete the code in the editor. Run it to check whether your code outputs 100.

Code:

```
class Stack:
    def __init__(self):
        self.__stk = []

    def push(self, val):
        self.__stk.append(val)

    def pop(self):
        val = self.__stk[-1]
        del self.__stk[-1]
        return val


class CountingStack(Stack):
    def __init__(self):
    #
    # Fill the constructor with appropriate actions.
    #

    def get_counter(self):
    #
    # Present the counter's current value to the world.
    #

    def pop(self):
    #
    # Do pop and update the counter.
    #


stk = CountingStack()
for i in range(100):
    stk.push(i)
    stk.pop()
print(stk.get_counter())
```

Estimated time

20-45 minutes

Level of difficulty

Easy/Medium

Objectives

- improving the student's skills in defining classes from scratch;
- implementing standard data structures as classes.

Scenario

As you already know, a *stack* is a data structure realizing the so-called LIFO (Last In - First Out) model. It's easy and you've already grown perfectly accustomed to it.

Let's taste something new now. A *queue* is a data model characterized by the term **FIFO: First In - Fist Out**. Note: a regular queue (line) you know from shops or post offices works exactly in the same way - a customer who came first is served first too.

Your task is to implement the `Queue` class with two basic operations:

- `put(element)`, which puts an element at end of the queue;
- `get()`, which takes an element from the front of the queue and returns it as the result (the queue cannot be empty to successfully perform it.)

Follow the hints:

- use a list as your storage (just like we did in stack)
- `put()` should append elements to the beginning of the list, while `get()` should remove the elements from the list's end;
- define a new exception named `QueueError` (choose an exception to derive it from) and raise it when `get()` tries to operate on an empty list.

Complete the code we've provided in the editor. Run it to check whether its output is similar to ours.

Expected output

```
1
```

```
dog
```

```
False
```

```
Queue error
```

Code:

```
class QueueError(???):  # Choose base class for the new exception.
   #
   #  Write code here
   #


class Queue:
   def __init__(self):
      #
      # Write code here
      #

   def put(self, elem):
      #
      # Write code here
      #

   def get(self):
      #
      # Write code here
      #


que = Queue()
que.put(1)
que.put("dog")
que.put(False)
try:
   for i in range(4):
      print(que.get())
except:
   print("Queue error")
```

Estimated time

15-30 minutes

Level of difficulty

Easy/Medium

Objectives

- improving the student's skills in defining subclasses;
- adding a new functionality to an existing class.

Scenario

Your task is to slightly extend the `Queue` class' capabilities. We want it to have a parameterless method that returns `True` if the queue is empty and `False` otherwise.

Complete the code we've provided in the editor. Run it to check whether it outputs a similar result to ours.

Expected output

```
1

dog

False

Queue empty
```

Code:

```
class QueueError(???):
    pass


class Queue:
    #
    # Code from the previous lab.
    #
```

```
class SuperQueue(Queue):
   #
   # Write new code here.
   #


que = SuperQueue()
que.put(1)
que.put("dog")
que.put(False)
for i in range(4):
   if not que.isempty():
      print(que.get())
   else:
      print("Queue empty")
```