پایرنج

می‌خواهیم کمک‌کننده‌ای (`helper`) بنویسیم که در موارد مشخص *Exception*ها را لاگ کند و از آن بگذرد.

کریم در حال خواندن سوالات بود که از من به عنوان نویسنده‌ی سوالات بابت توهین‌هایی که به کریم کردم در نداشتن توانایی یاد گرفتن برنامه‌نویسی دلخور شد. برای همین سوالی را طرح کرد که به من بفهماند که من هم توانایی یاد گرفتن برنامه‌نویسی را ندارم. از شما می‌خواهم به جای من سوال زیر را حل کنید.

جزئیات

سوال از این قرار است که باید در پایتون کمک کننده‌ای به نام `PyRanj` پیاده‌سازی کنید به صورتی که این کمک‌کننده **سه قابلیت اصلی** زیر را داشته باشد:

۱. *Wrapper*

هر گاه از `pyranj` به عنوان *wrapper* یک تابع استفاده شود، در تابع مورد نظر نباید هیچ تغییری ایجاد شود و فقط در زمان صدا زدن تابع، در صورتی که *Exception* پرتاب شود، صرفا باید این اتفاق لاگ شود و برنامه ادامه پیدا کند. برای مثال در کد زیر باید خروجی پایینی لاگ شود:

```python
from pyranj import PyRanj as pyranj

@pyranj
def f():
    raise Exception('Ranj')

f()
```

```
[EXCEPTION] :: Ranj
```

۲. *ContextManager*

هرگاه از `pyranj` به عنوان `context` استفاده شود، در عملیات مورد نظر نباید هیچ تغییری ایجاد شود و فقط اگر حین انجام عملیات، *Exception* پرتاب شود، صرفا باید این اتفاق لاگ شود و برنامه ادامه پیدا کند. برای مثال در کد زیر باید خروجی پایینی لاگ شود:

```python
from pyranj import PyRanj as pyranj

with pyranj:
    raise Exception('Ranj')
```

```
[EXCEPTION] :: Ranj
```

۳. *Mixin*

هرگاه از `pyranj` در کلاسی ارث‌بری شود، **در صورتی که آن کلاس دارای متد run باشد**، تغییری که ایجاد می‌شود باید برابر با لاگ‌شدن `Exception` با فرمت مشخص شده باشد. برای مثال در کد زیر باید خروجی پایینی لاگ شود:

```
from pyranj import PyRanj as pyranj

class Runner(pyranj):
    def run(self):
        raise Exception('Ranj')

 Runner().run()
```

```
[EXCEPTION] :: Ranj
```

توجه: برای لاگ کردن باید از متد `log` شی `logger` به شیوه‌ی زیر عمل کنید و به آن یک رشته ورودی دهید (از اینجا می‌توانید نمونه `logger` که در تست‌ها استفاده می‌شود را دانلود کنید):

```
from pyranj import PyRanj as pyranj

from logger import logger

logger.log('PyRanj Log')
```

علاوه بر قابلیت‌های اصلی، کمک‌کننده‌ی شما باید دارای قابلیت‌های زیر نیز باشد.

۱. تغییر دادن `prefix` در متن لاگ

```
@pyranj(prefix='[PREFIX]')
def f():
    raise Exception('Ranj')

f()
```

```
[PREFIX] :: Ranj
```

توجه کنید که این نوع از تغییر prefix در تمام ویژگی‌های اصلی باید وجود داشته باشد.

۲. لغو کردن `prefix` در متن لاگ

```
from pyranj import PyRanj as pyranj

with pyranj(prefix='[PREFIX]')():
    raise Exception('Ranj')
```

```
[EXCEPTION] :: Ranj
```

توجه: متنی که به ازای هر *Exception* لاگ می‌شود باید به فرمت زیر باشد، که در آن مقدار پیشفرض `prefix` ، برابر است با `[EXCEPTION]` .

```
f"{prefix} :: {exception}"
```

برای فهمیدن بهتر سوال می‌توانید مثال زیر و خروجی آن را مشاهده کنید.

```
from pyranj import PyRanj as pyranj

@pyranj
def f1():
    raise Exception('x1')

f1()

pyrannnnnnnj = pyranj()()(prefix='Hey')()

@pyrannnnnnnj(prefix='You')
def f2():
    raise Exception('x2')

f2()

with pyranj(prefix='Yes')()()()()():
    raise Exception('x3')


class A(pyranj):
    def run(self, num):
        raise Exception('x' * num)

A().run(5)


class B(pyranj()()(prefix='Hey there is an error')):
    def run(self):
        raise Exception('run ...')

B().run()
```

```
[EXCEPTION] :: x1
You :: x2
[EXCEPTION] :: x3
[EXCEPTION] :: xxxxx
Hey there is an error :: run ...
```

نکات

- نام برنامه‌ی ارسالی شما باید `pyranj.py` باشد که در آن شی `PyRanj` وجود داشته باشد و ویژگی‌های گفته شده را داشته باشد.
- می‌توانید کد تست نمونه را با استفاده از این لینک دانلود کنید.
- تست‌های اصلی این سوال از ۵ بخش تشکیل شده که هر بخش دارای سه نوع تست است؛ یعنی هنگام ارسال شما نام هر متد تست را ۵ بارمی‌بینید. این بخش‌ها به ترتیب دسته‌های زیر هستند:
  - بخش `BasePyRanjTest`
  - بخش `TestPyRanjInstantiation`
  - بخش `TestPyRanjInstantiationWithPrefix`
  - بخش `TestPyRanjRecursiveInstantiation`
  - بخش `TestPyRanjRecursiveInstantiationWithPrefix`

قسمت آموزشی

در این قسمت راهنمایی‌های سوال به ترتیب در روزهای شنبه، دوشنبه و چهارشنبه ساعت ۱۸ اضافه می‌شود. مشکلات‌تان در راستای حل سوال را می‌توانید از بخش "سوال بپرسید" مطرح کنید.

▼ راهنمایی ۱

توصیه می‌شود ابتدا در مورد مفاهیم wrapper, contextManager, Mixin از این لینک‌ها بخوانید:

- https://wiki.python.org/moin/FunctionWrappers
- https://book.pythontips.com/en/latest/context_managers.html
- https://www.ianlewis.org/en/mixins-and-python

توجه کنید که `helper` مورد نظر لزومی ندارد به صورت عادی تعریف شود و می‌تواند خودش حاصل فراخوانی یک تابع دیگر باشد.

برای این که `helper` مورد نظر به طور همزمان قابل ارث‌بری باشد و همچنین قابل فراخوانی، می‌توان از تابع `__call__` در `metaclass` کمک گرفت. برای اطلاع ازین مورد می‌توانید لینک زیر را بخوانید:

- https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Metaprogramming.html

▼ راهنمایی ۲

برای حل این سوال می‌توانید از کتابخانه `contextlib` پایتون استفاده کنید که کمک زیادی به پیاده‌سازی موارد خواسته شده می‌کند. برای کسب اطلاعات بیشتر درباره این کتابخانه این‌جا را ببینید.

حال برای حل سوال یک کلاس به نام `PyRanj`- تعریف کنید؛ به طوری که `PyRanj` نهایی که `instance` از آن باشد.

▼ راهنمایی ۳

کد نهایی سوال به شکل زیر می‌شود:

```
from logger import logger
from contextlib import ContextDecorator



def decorator(func):
    def wrapper(*args, **kwargs):
        try :
            return func(*args, **kwargs)
        except Exception as e :
            logger.log(f"{_PyRanj.prefix} :: {e.args[0]}")

    return wrapper

class _PyRanj () :

    prefix = "[EXCEPTION]"

    def __init__ ( self ,a=None , b=None, c=None ):
        if a is None and b is None and c is None :
            return
        if "run" in c :
            self._run = c["run"]
        pass

    def run ( self , *args , **kwargs ) :
        try :
            return self._run(self , *args , **kwargs)
        except Exception as e :
            logger.log(f"{_PyRanj.prefix} :: {e.args[0]}")


    def __call__(self , function=None , prefix=None):
        if prefix is not None :
            _PyRanj.prefix = prefix

        if function is None :
            if prefix is None and not hasattr(self, '_run') :
                _PyRanj.prefix = "[EXCEPTION]"
            return self


        if function is None:
            return decorator
        else:
            return decorator(function)

    def __enter__(self):
        return self

```

```
    def __exit__(self, *exc):
        if exc[1] is not None :
            logger.log(f"{_PyRanj.prefix} :: {exc[1].args[0]}")
        return True


PyRanj = _PyRanj()
```