Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкция языков прграммирования»

Отчет по лабораторной работе №4

«Объектно-ориентированные возможности языка Python»

Выполнил:
студент группы ИУ5-36Б
Каверина С.Г
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Нардид А. Н.

Подпись и дата:

Цель лабораторной работы:

Необходимо создать виртуальное окружение и установить в него хотя бы один внешний пакет с использованием pip.

Необходимо разработать программу, реализующую работу с классами. Программа должна быть разработана в виде консольного приложения на языке Python 3.

Все файлы проекта (кроме основного файла main.py) должны располагаться в пакете lab_python_oop.

Каждый из нижеперечисленных классов должен располагаться в отдельном файле пакета lab_python_oop.

Абстрактный класс «Геометрическая фигура» содержит абстрактный метод для вычисления площади фигуры. Подробнее про абстрактные классы и методы Вы можете прочитать здесь.

Класс «Цвет фигуры» содержит свойство для описания цвета геометрической фигуры. Подробнее про описание свойств Вы можете прочитать здесь.

Класс «Прямоугольник» наследуется от класса «Геометрическая фигура». Класс должен содержать конструктор по параметрам «ширина», «высота» и «цвет». В конструкторе создается объект класса «Цвет фигуры» для хранения цвета. Класс должен переопределять метод, вычисляющий площадь фигуры.

Класс «Круг» создается аналогично классу «Прямоугольник», задается параметр «радиус». Для вычисления площади используется константа math.pi из модуля math.

Класс «Квадрат» наследуется от класса «Прямоугольник». Класс должен содержать конструктор по длине стороны. Для классов «Прямоугольник», «Квадрат», «Круг»:

1. Определите метод "repr", который возвращает в виде строки основные параметры фигуры, ее цвет и площадь. Используйте метод format - https://pyformat.info/
2. Название фигуры («Прямоугольник», «Квадрат», «Круг») должно задаваться в виде поля данных класса и возвращаться методом класса.

В корневом каталоге проекта создайте файл main.py для тестирования Ваших классов (используйте следующую конструкцию - https://docs.python.org/3/library/__main__.html). Создайте следующие объекты и выведите о них информацию в консоль (N - номер Вашего варианта по списку группы):

1. Прямоугольник синего цвета шириной N и высотой N.
2. Круг зеленого цвета радиусом N.
3. Квадрат красного цвета со стороной N.
4. Также вызовите один из методов внешнего пакета, установленного с использованием pip.

Текст программы:

Field.py

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]

def field(items, *args):
    assert len(args) > 0
```

```
    for item in items:
        if len(args) == 1:
            key = args[0]
            if key in item and item[key] is not None:
                yield item[key]
        else:
            filtered_item = {key: item[key] for key in args if key in item and item[key] is
not None}
            if filtered_item:
                yield filtered_item
```

```
# Проверка для одного аргумента
for title in field(goods, 'title'):
    print(title)
```

```
# Проверка для двух аргументов
for item in field(goods, 'title', 'price'):
    print(item)
```

Gen_random.py

```
import random

def gen_random(count, begin, end):
    for i in range(count):
        yield random.randint(begin, end)
```

```
for num in gen_random(5, 1, 3):
    print(num)
```

Print_result.py

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(result, list):
            for item in result:
                print(item)
        elif isinstance(result, dict):
            for key, value in result.items():
                print(f"{key} = {value}")
        else:
            print(result)
        return result
    return wrapper
```

```python
# Примеры использования декоратора
@print_result
def test_1():
    return 1
```

```python
@print_result
def test_2():
    return 'iu5'
```

```python
@print_result
def test_3():
    return {'a': 1, 'b': 2}
```

```python
@print_result
def test_4():
    return [1, 2]
```

```python
if __name__ == '__main__':
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

Process_data.py

```python
import json
import sys
import time
from contextlib import contextmanager


path = "C:/Users/ACITO/OneDrive/Рабочий стол/data_light.json"
```

```python
with open(path, encoding="utf8") as f:
    data = json.load(f)
```

```python
@contextmanager
def cm_timer_1():
    start_time = time.time()
    yield
    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"time: {elapsed_time}")
```

```python
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(result)
        return result
    return wrapper
```

```python
@print_result
def f1(arg):
    return sorted(list(set([job['job-name'].lower() for job in arg])))
```

```python
@print_result
def f2(arg):
    return list(filter(lambda job: job.startswith('программист'), arg))
```

```python
@print_result
def f3(arg):
    return list(map(lambda job: job + ', с опытом Python', arg))
```

```python
@print_result
def f4(arg):
    salary = [str(i) for i in range(100000, 200001)]
    return list(zip(arg, salary))
```

```python
if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Sort.py

```python
# with Lamda
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
print(data)
result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
print(result_with_lambda)

# without Lamda
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
print(data)
result_without_lambda = sorted(data, key=abs, reverse=True)
print(result_without_lambda)
```

Uniqe/py

```python
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = iter(items)  # Преобразуем входные данные в итератор
        self.ignore_case = kwargs.get('ignore_case', False)  # Получаем ignore_case из kwargs,
по умолчанию False
        self.seen = set()

    def __next__(self):
        while True:
            try:
                item = next(self.items)
                if self.ignore_case and isinstance(item, str):
                    item = item.lower()  # При ignore_case=True приводим строки к нижнему
регистру
                if item not in self.seen:
                    self.seen.add(item)
                    return item
            except StopIteration:
                raise StopIteration
```

```python
    def __iter__(self):
        return self
```

```python
# Примеры использования:
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
for item in Unique(data1):
    print(item)
```

```python
print("\n")
```

```
data2 = (i for i in range(10) if i % 3 == 0) # Генератор
for item in Unique(data2):
    print(item)
```

```
print("\n")
```

```
data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
for item in Unique(data3):
    print(item)
```

```
print("\n")
```

```
for item in Unique(data3, ignore_case=True):
    print(item)
```

Вывод: Я изучила объектно-ориентированные возможности языка python.