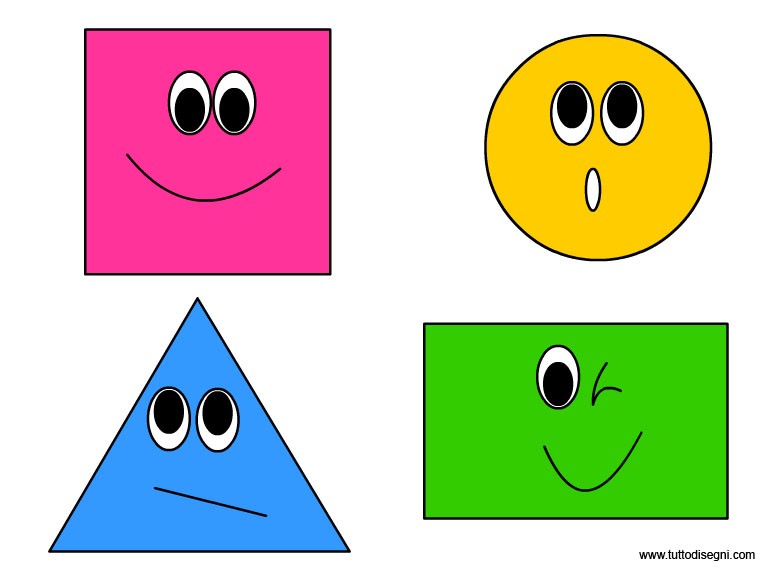
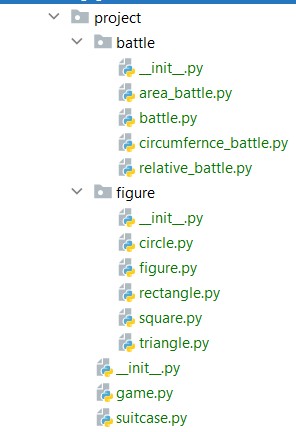
**THE COMBAT OF THE FIGURES**



***The figures are happy, but they can fight strongly!***

**This is a skeleton, which includes all the folders and files that you will need.**

**Note: You are not allowed to change the folder and file structure and change their names!**



**Judge Upload**

**Still not supported to the judge system…**

**Structure and Functionality - [Solution](https://github.com/borislavstoychev/Soft_Uni/tree/master/soft_uni_OOP/EXAM_10_04_2020/aqua_shop_project/project)**

Our first task is to implement the structure and functionality of all the classes (properties, methods, inheritance, etc.)

**1. Figure**

In the base\_figure.py file the class Figure should be implemented. It is a base class of any type of figures, and it **should not be able to be instantiated**.

**For all heirs of the base class Figure validate it arguments(side, radius,...) in init method to be a positive number, if get negative - raise ValueError with message "Negative number!", only the name can be string**

**Structure**

**The class should have the following attributes:**

**name: str - passed upon initialization.**

**Methods**

* **\_\_init\_\_(name: str)**

The **init** method should have a name .

* **calculate\_area()**

Method will be implemented in the subclasses.

* **calculate\_circumference()**

Method will be implemented in the subclasses.

* **relativity**

A property, returns calculated relative of area to circumference for the figure: (area / circumference)

* **str()**

**Overwrite str() returns formatted string:**

**Figure name: {name}  
Parameters: {side}, ...{}  
Area: {area}  
Circumference: {circumference}  
Relativity: {relativity}**

**2. Circle**

In the file circle.py the class Circle should be implemented.

**Structure**

The class should inherit from the Figure class.

**Methods**

* **\_\_init\_\_(name: str, radius: float)**

An instance of the Circle class will have name and radius as atributes.

* **calculate\_area()**

Returns the area of circle.

* **calculate\_circumference()**

Returns the circumference of circle

**3. Rectangle**

In the file rectangle.py the class Rectangle should be implemented.

**Structure**

The class should inherit from the Figure class.

**Methods**

* **\_\_init\_\_(name, side\_a, side\_b)**

An instance of the Circle class will have name, side\_a and side\_b as atributes.

* **calculate\_area()**

Returns the area of rectangle.

* **calculate\_circumference()**

Returns the circumference of rectangle

**4. Square**

In the file square.py the class Square should be implemented.

**Structure**

The class should inherit from the Figure class.

**Methods**

* **\_\_init\_\_(name, side)**

An instance of the Square class will have name and side as atributes.

* **calculate\_area()**

Returns the area of square.

* **calculate\_circumference()**

Returns the circumference of square

**5. Triangle**

In the base\_triangle.py file the class Triangle should be implemented.

**Structure**

The class should inherit from the Figure class.

**Methods**

* **\_\_init\_\_(name, side\_a, height\_a, side\_b, side\_c)**

An instance of the Triangle class will have name , side\_a, height\_a, side\_b and side\_c as atributes.

* **calculate\_area()**

Returns the area of triangle.

* **calculate\_circumference(self)**

Returns the circumference of triangle.

**6. Battle**

In the file battle.py the class Buttle should be implemented. It is a base class of any type of battles, and it **should not be able to be instantiated.**

**Structure**

The class does not accept initialization attributes

**Methods**

* **battle(figure\_1, figure\_2)**

Method will be implemented in the subclasses.

**7. AreaBattle**

In the file area\_battle.py the class AreaBattle should be implemented.

**Structure**

The class should inherit from the Battle class.

**Methods**

* **battle(figure\_1, figure\_2)**

Returns the figure with bigger area, if they are equal returns None.

**7. CircumferenceBattle**

In the file circumference\_battle.py the class CircumferenceBattle should be implemented

**Structure**

The class should inherit from the Battle class.

**Methods**

* **battle(figure\_1, figure\_2)**

Returns the figure with bigger circumference, if they are equal returns None.

**7. RelativeBattle**

In the file relative\_battle.py the class RelativeBattle should be implemented

**Structure**

The class should inherit from the Battle class.

**Methods**

* **battle(figure\_1, figure\_2)**

Returns the figure with bigger relativity, if they are equal returns None.

**10. Suitcase**

In the file suitcase.py the class Suitcase should be implemented.

**Structur**

**The class should have the following attributes:**

**repository: list – empty list upon initialization.**

**Methods**

* **\_\_init\_\_()**
* **add(figure)**

Add a figure object to figures repository, if figure type is in valid types:(Circle, Rectangle, Triangle and Square) and figure name not exist in figure repository.

If figure type is invalid, raise TypeError with message: "The type of Figure is incorrect!"

If figure name exist in figure repository, raise KeyError with message: "Figure name already exist!"

If figure successful added, returns message: "Figure: {name} added."

* **remove(figure\_name)**

Remove a figure with that name from figure repository if exist.

If figure not in repository, raise KeyError with message: "Figure: {figure\_name} not exist!"

If figure successful removed, returns message: "Figure: {figure\_name} removed."

* **get\_figure(figure\_name)**

Returns a figure with same name in repository.

* **\_\_repr\_\_()**

Returns the names of the figures in repository separated by comma and space – “, “.

**11. Game**

In the file game.py the class Game should be implemented.

**Structure**

**The class should have the following attributes:**

* **figures: an instance of Suitcase class**

**Methods**

* **\_\_init\_\_()**

An instance of the Suitcase class

* **area\_battle(figure\_1, figure\_2)**

Gets the result of battle() method in AreaBattle class,

and if have winning figure returns it name, else returns None.

* **circumference\_battle(figure\_1, figure\_2)**

Gets the result of battle() method in CircumferenceBattle class,

and if have winning figure returns it name, else returns None.

* **relative\_battle(figure\_1, figure\_2)**

Gets the result of battle() method in RelativeBattle class,

and if have winning figure returns it name, else returns None.

* **total\_battle()**

Take first two figures in self figures and combat them in area\_battle() and circumference\_battle(), if results of the duels are equal(not winner in battles) combat them in relative\_battle – after that add the winner to self figures, and repeat the procedure until one figure remains.

Returns the winner

* **\_\_repr\_\_()**

Returns formatted string (use the str() method of Figure):

**The winner is:**

**Figure name: {name}  
Parameters: {side}, ...{}  
Area: {area}  
Circumference: {circumference}  
Relativity: {relativity}**

|  |  |
| --- | --- |
| Test code | Expected output |
| tri1 = Triangle("tri1", 9, 4, 6.5, 4.5) tri2 = Triangle("tri2", 5, 2.4, 3, 4) cir1 = Circle("cir1", 3) rec1 = Rectangle("rec1", 1, 7) squ1 = Square("squ1", 6) g = Game() print(g.figures.add(tri1)) print(g.figures.add(tri2)) print(g.figures.add(cir1)) print(g.figures.add(rec1)) print(g.figures.add(squ1)) print(g.area\_battle(cir1, tri1)) print(g.circumference\_battle(cir1, tri1)) print(g.relative\_battle(cir1, tri1)) print(g.figures.remove("squ1")) print(g.figures) print("-------------") print(g) | Figure: tri1 added.  Figure: tri2 added.  Figure: cir1 added.  Figure: rec1 added.  Figure: squ1 added.  cir1  tri1  cir1  Figure: squ1 removed.  tri1, tri2, cir1, rec1  -------------  The winner is:  Figure name: cir1  Parameters: 3  Area: 28.274333882308138  Circumference: 18.84955592153876  Relativity: 1.5 |