# Question 1: The Circle Class

Create a class called `cirlce` which contains the following:

## Attributes:

1. private attribute: radius (of the type float and greater than 0) (use assert in the init method to force this behaviour)
2. private attribute: color (of the type string), with default value of 1.0 and "red", respectively.

## Properties

Use the `@property` function decorator to create the following:

1. radius (both getter and setter): Raise a TypeError if setter used with a non numeric value. Raise a ValueError if setter used with radius <= 0.
2. colour (both getter and setter):

## Methods:

1. get_area(), which return the area this instance
2. get_circumference(), which return the circumference this instance
3. `__str__` : return a string containing the radius of the circle.

# Question 2: The Point Class

Create a class called **Point** which has the following data and behaviour:

## Data Attributes:

1. x (float, private) represents x coordinate of the point
2. y (float, private) represents y coordinate of the point

## Properties:

1. x (only getter i.e. read-only)
2. y (only getter i.e. read-only)

## Methods:

1. `__init__` takes x and y from the user as floats.
2. `__str__` returns point in the format: (x, y). Example, if p = Point(2,3) print(p) should output (2, 3)

## Question 3: The Line Class

The line class will represent a line segment with a start point and an end point.

### `__init__` takes:

1. start_point: Point, private: complain if type != Point
2. end_point: Point, private

### properties

1. start_point: (getter only)
2. end_point(getter and setter)

### derived properties:

1. length: returns the length of the line. Calculate using the distance formula: (https://www.wikihow.com/Use-Distance-Formula-to-Find-the-Length-of-a-Line)

## Question 4: The Rectangle Class

### `__init__` takes:

1. bottom_left_corner: Point, private
2. top_right_corner: Point, private

### Properties:

1. bottom_left_corner
2. top_right_corner

### Derived Properties:

1. bottom_right_corner: using the bottom_left_corner.y and top_right_corner.x create a new Point object and return that.
2. top_left_corner: similar process as bottom_right_corner. Return a Point object.
3. area: returns a float representing the area of the rectangle. (Hint: using two of the four corners, you can create a line and then get the length of the line using the length method in the Line class. Create two such lines, one for each side, and multiply their lengths to find the area of the rectangle)
4. perimeter: returns float representing the perimeter around the rectangle.

# Question 5: The Date Class

Create a class "Date".

- This exercise must be performed without using the datetime module at all.

## Data Attributes:

1. day: int, private, getter property, setter property
2. month: int, private, getter property, setter property
3. year: int, private, getter property, setter property

## Derived Properties:

1. is_leap_year: bool. Returns T/F based on if the year is a leap year or not.
2. is_valid_date: bool. Returns T/F based on if its a valid date or not.

## Methods:

1. **init** takes 3 arguments: year, month, day. raise TypeErrors if they are not ints. raise ValueErrors if values are outside the expected bounds.
2. **str** Returns date in dd/mm/yyyy format

hint for **str** method:

```
d = 5
print(d.zfill(2)) # prints 05
```

## Optional:

implement next_day and a previous_day methods that returns an object of the Date class but represents the next day or previous day. Must be a valid date. Hint: add one to the day, and check if its is valid, if not, add one to the month and check if valid and so on. repeat with minus for previous date.

# Question 6: Class Student

## Private Data Attributes

1. id: int
2. name:string
3. address:string
4. program:string
5. current_courses:[] # Contains a list of course IDs
6. completed_courses:{} # Contains a mapping from CourseID to marks obtained.

### Properties:

1. name (getter only)
2. id (getter only)
3. address (getter and setter)

### Derived Properties

1. average_score: Returns average score of all the completed courses. Return -1 if no courses have been completed.

### Methods:

1. add_course. Takes a course id and adds to the list of current courses.
2. drop_course. Removes course id from list of courses if present, otherwise raise ValueError
3. course_completed: Takes course_id and marks_obtained and stores in the completed_courses dict if course_id in current_courses list.

## Question 7: The Employee Class

Create a class called Employee that has the following data and behaviours:

### Data Attributes:

1. first_name (string, private)
2. last_name (string, private)
3. employee_id (string of the format E0001 - E9999)
4. salary (float, private)

### Properties:

1. employee_id (getter only)

### Derived Properties:

1. annual_salary (returns 12 * salary)
2. name (returns first_name-space-last_name)

### Methods

1. raise_salary(raise_percent): Increases the salary by the specified percent.

## Question 8: Bank Account

Create a class called BankAccount that has the following data and behaviour:

### Data Attributes:

1. account_id: string, private, getter
2. balance: float, private, getter
3. transaction_list: list of Transaction objects

### Methods:

1. deposit(amount): returns True if successful, else false
2. withdraw(amount): returns True if successful, else false. Only possibe if there is enough money

## Question 9: Class Transaction

### Data Attributes:

1. from_account: BankAccount, private
2. to_account: BankAccount, private
3. amount: float

in the init methods, you should also call the withdraw method of the from account and if successful, call the deposit method of the to_account. If the withdraw is unsuccessful, raise a ValueError saying Insufficient funds.

## Question 10: Make Transaction and Bank Account objects and perform transactions

1. Create multiple bank account objects
2. Create several transactions
3. Print the balance of the bank accounts involved in the transactions after the transaction is created.
4. If the transaction fails, the bank accounts should not be affected.
5. Test and make sure it works perfectly.