Week-10, Activity Questions

Note

- The first 15 problems are based on some subtle theoretical ideas in object oriented programming in Python.
- We will solve them during the live sessions.

Problem-1

Observe the output of the following snippet of code. Why do you think this happens?

```
obj = Foo()

class Foo:
    x = None
```

Answer

We get a `NameError`. The class definition should come before object creation. Python must first know that there is a class named `Foo` before it can create an object of type `Foo`.

Problem-2

Observe the output of the following snippet of code. Why do you think this happens?

```
class Foo:
    def __init__(self):
        print('Foo object created')
obj = Foo()
obj.x = 10
print(obj.x)
```

Answer

This is the output:

```
Foo object created
10
```

`x` is an attribute of object `obj` of class `Foo`. This code snippet is meant to demonstrate the fact that object attributes can be created on the fly. They need not be explicitly defined inside the constructor every time.

Problem-3

Observe the output of the following snippet of code. Why do you think this happens?

```
1  class Foo:
2      x = 0
3
4  obj = Foo()
5  print(obj.x)
6  obj.x = 10
7  print(obj.x, Foo.x)
```

Answer

The output is as follows:

```
1  0
2  10 0
```

From the definition of the class `Foo`, we see it has a class attribute, namely `x`. Recall that we can always access and also update the class attribute using the class' name — `Foo.x` In line-5, `obj.x` points to the class attribute. At this stage, we are able to access the class attribute using the object as well.

However, line-6 introduces an object attribute with the same name. In such a case, while trying to access the attribute with name `x` using the object, Python gives precedence to the object attribute. This is why, the output of line-7 is `10 0`. `obj.x` is going to refer to the object attribute, while `Foo.x` is will continue to refer to the class attribute.

Consider a slight variation of this code:

```
1  class Foo:
2      x = 0
3
4  obj = Foo()
5  print(obj.x)
6  obj.x = obj.x + 10
7  print(obj.x, Foo.x)
```

Note that in line-6, we are creating an object attribute with the same name as the class attribute, but we are doing this by using the current value of the class attribute.

Some important lessons from this problem:

- A class attribute's state can never be explicitly altered by an object. However, it can be changed by using methods.
- A class attribute can be referenced by an object, provided there is no object attribute with the same name.
- If the same variable name is shared by a class attribute and an object attribute, while trying to access attributes using an object, Python gives precedence to the object attribute.

Problem-4

Observe the output of the following snippet of code. Why do you think this happens?

```
class Foo:
    x = 0
    def __init__(self):
        self.x = 100

obj = Foo()
print(obj.x, Foo.x)
```

Answer

This is similar to the previous problem. Inside the constructor, we have the line `self.x = 100`. This creates an object attribute with the same name as the class attribute, namely `x`.

Problem-5

Observe the output of the following snippet of code. Why do you think this happens?

```
class Foo:
    def __init__(self):
        x = 1

obj = Foo()
print(obj.x)
```

Answer

We get an `AttributeError`: `Foo` object has no attribute `x`. This is because, the `x` defined inside the constructor is a local variable for the function `__init__`. It is not an attribute of the object.

Problem-6

Observe the output of the following snippet of code. Why do you think this happens?

```
1 class Foo:
2     self.x = 1
3
4 obj = Foo()
```

Answer

This time, we get a `NameError`: name 'self' is not defined. This is because, the `self` keyword in line-2 is not within a method. It is just dangling there in mid-air. The class doesn't know what `self` is and therefore throws a `NameError`. The important lesson here is that using the `self` keyword makes sense only inside methods of the class.

Problem-7

`how` accepts one argument, namely `self`. But we are not passing any arguments to it in line-6, or are we?

```
1  class Foo:
2      def how(self):
3          print('No arguments, but how?')
4
5  obj = Foo()
6  obj.how()
```

Answer

Though it is not explicit, the object `obj` is passed as the first argument to the method `how`. In some sense, this is how Python interprets the function call: `how(obj)`.

Problem-8

Observe the output of the following snippet of code. Why do you think this happens?

```
class Foo:
    foo = 'foo'

class Bar(Foo):
    bar = 'bar'

obj = Bar()
print(obj.foo)
print(obj.bar)
```

Answer

The output is as follows:

```
foo
bar
```

This is an example of inheritance. Class `Bar` inherits attribute `foo` from `Foo` class. Therefore, both attributes `foo` and `bar` are present in `obj` which is an instance of class `Bar`.

Problem-9

What statement should be inserted between lines 6 and 7 so that the code runs without any error?

```
1  class Foo:
2      def __init__(self):
3          self.foo = 'foo'
4
5  class Bar(Foo):
6      def __init__(self):
7          self.bar = 'bar'
8
9  obj = Bar()
10 print(obj.bar)
11 print(obj.foo)
```

Answer

This will show `AttributeError: 'Bar' object has no attribute 'foo'`. Class `Bar` inherits class `Foo`. All the class attributes of class `Foo` becomes available to class `Bar`. However, the `foo` attribute is an instance attribute in the class `Foo`. It is defined inside the `__init__()` method of class `Foo`. In order to make available in the object of the class `Bar`, we should call the `__init__()` method of the parent class. This can be done by adding `super().__init__()` at line-7. The correct code will be:

```
1  class Foo:
2      def __init__(self):
3          self.foo = 'foo'
4
5  class Bar(Foo):
6      def __init__(self):
7          super().__init__()
8          self.bar = 'bar'
9
10 obj = Bar()
11 print(obj.bar)
12 print(obj.foo)
```

Problem-10

What statement should be inserted between lines 10 and 11 so that the code runs without any error?

```
1  class Foo:
2      def foo_fun(self):
3          self.foo = 'foo'
4
5  class Bar(Foo):
6      def __init__(self):
7          self.bar = 'bar'
8
9  obj = Bar()
10 print(obj.bar)
11 print(obj.foo)
```

Answer

Similar to the previous case, it will show `AttributeError: 'Bar' object has no attribute 'foo'`. The `foo` attribute is an instance attribute in the class `Foo` and is defined inside the instance method `foo_fun()`. To make available in the object of the class `Bar`, we should call the `foo_fun()` method on the object `obj`. Alternatively, this can also be done by adding `super().foo_fun()` at line-7. The correct code will be:

```
1  class Foo:
2      def foo_fun(self):
3          self.foo = 'foo'
4
5  class Bar(Foo):
6      def __init__(self):
7          #super().foo_fun()
8          self.bar = 'bar'
9
10 obj = Bar()
11 print(obj.bar)
12 obj.foo_fun()
13 print(obj.foo)
```

Problem-11

Observe the output of the following snippet of code. Why do you think this happens?

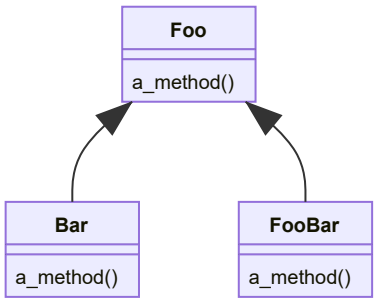
```
class Foo:
    def a_method(self):
        print('calling from Foo')

class Bar(Foo):
    def a_method(self):
        print('calling from Bar')

class FooBar(Foo):
    def a_method(self):
        super().a_method()
        print('calling from FooBar as well')

obj_1 = Foo()
obj_2 = Bar()
obj_3 = FooBar()

obj_1.a_method()
obj_2.a_method()
obj_3.a_method()
```

Answer

The output will be:

```
calling from Foo
calling from Bar
calling from Foo
calling from FooBar as well
```

In the code snippet, class `Bar` and `FooBar` inherits class `Foo`. This is an example of `hierarchical` inheritance. Here, `obj_1`, `obj_2` and `obj_3` are the instance of class `Foo`, `Bar` and `FooBar` respectively.



The method `a_method()` in class `Bar` does not call the parent class version of `a_method()`. In class `FooBar`, the method `a_method()` first calls the parent class version of `a_method()` followed by print statement.

Therefore, `obj_3.a_method()` prints below output:

```
calling from Foo
calling from FooBar as well
```

Problem-12

Observe the output of the following snippet of code. Why do you think this happens?

```
1  class Foo:
2      pass
3
4  class Bar(Foo):
5      def __init__(self):
6          self.x = 0
7
8  obj_1 = Foo()
9  obj_2 = Bar()
10 print(obj_1.x)
```

Answer

This will result in `AttributeError: 'Foo' object has no attribute 'x'`. At line-10, we are trying to access variable `x` from the object of class `Foo`, which was not defined inside the class. The object of the parent class `Foo` will have no idea of what attributes are part of the object of child class `Bar`.

Problem-13

Observe the output of the following snippet of code. Why do you think this happens?

```
1  class Foo:
2      pass
3
4  class Bar(Foo):
5      pass
6
7  obj_1 = Foo()
8  obj_2 = Bar()
9  obj_1.x = 100
10 print(obj_2.x)
```

Answer

This will result in `AttributeError: 'Bar' object has no attribute 'x'`. At line-10, we are trying to access variable `x` from the object of class `Bar`, which was not defined inside the class. The object of the class `Bar` will have no idea of what attributes are part of the object of class `Foo`.

Problem-14

A slight variation from the previous question. Observe the output of the following snippet of code. Why do you think this happens?

```
class Foo:
    pass

class Bar(Foo):
    pass

obj_1 = Foo()
obj_1.x = 100
obj_2 = Bar()
print(obj_2.x)
```

Answer

Same as Problem-13

Problem-15

Observe the output of the following snippet of code. Why do you think this happens?

```
class Foo:
    def __init__(self, x):
        self.x = x
        print(f'new object with x = {self.x}')

objects = [ ]
for i in range(5):
    obj = Foo(i)
    objects.append(obj)
```

Answer

The program output is:

```
new object with x = 0
new object with x = 1
new object with x = 2
new object with x = 3
new object with x = 4
```

In the program, an object of class `Foo` is created with each iteration and the object is appended to the list called `objects`. When an object of class `Foo` is created, the constructor method `__init__()` is called which assigns instance variable `x` with the value of the loop variable `i` and also executes the print statement inside it.

Problem-16

Write a class named `Country` that has the following attributes and methods:

Attributes

- `name`: name of the country
- `capital`: capital of the country

Methods

- `__init__(self)`: this method should set the values of all attributes to `None`
- `set_name(self, name)`: this method should update the `name` of the country
- `set_capital(self, capital)`: this method should update the `capital` of the country
- `display`: this method should print the following string

`<capital> is the capital of <name>`

Create at least three different objects of class `Country`.

Answer

```
class Country:
    def __init__(self):
        self.name = None
        self.capital = None
    def set_name(self, name):
        self.name = name
    def set_capital(self, capital):
        self.capital = capital
    def display(self):
        print(f"{self.capital} is the capital of {self.name}")

c1 = Country()
c2 = Country()
c3 = Country()
c1.set_name("India")
c1.set_capital("New Delhi")
c2.set_name("Australia")
c2.set_capital("Canberra")
c3.set_name("France")
c3.set_capital("Paris")
c1.display()
c2.display()
c3.display()
```

Problem-17

Consider the file country-list.csv. It is a csv file that contains a list of countries and their capitals.

(1) First open this file and try to understand how the data is represented.

(2) Read this file in Python.

(3) Create an object of class `Country` that was introduced in the previous problem for each country that you find in this file.

(4) Store all these objects in a list named `countries`.

(5) Loop through the list of countries. Break whenever you find `South Africa`. Change its capital from "Pretoria" to "Cape Town". You must use the object's method to do this.

Answer

```
class Country:
    def __init__(self):
        self.name = None
        self.capital = None
    def set_name(self, name):
        self.name = name
    def set_capital(self, capital):
        self.capital = capital
    def display(self):
        print(f"{self.capital} is the capital of {self.name}")

countries = []
f = open("country-list.csv", "r")
lines = f.readlines()[1:]
for line in lines:
    name, capital = line.strip().split(",")
    c = Country()
    c.set_name(name)
    c.set_capital(capital)
    countries.append(c)

for country in countries:
    if country.name == 'South Africa':
        country.set_capital("Cape Town")
        break
```

Problem-18

Create a class named `Book`. It should have the following attributes and methods:

Attributes

- `name`
- `author`
- `pages`
- `genre`

Methods

- `__init__(self, name, author pages, genre)`: initialize the attributes with the argument values
- `is_fiction(self)`: returns `True` if the book is belongs to the fiction category, and `False` otherwise
- `is_nonFiction(self)`: returns `True` if the book belongs to the non-fiction category and `False` otherwise
- `time_to_read(self)`:
  - returns the string `5 day` if the number of pages is less than 100
  - returns the string `20 days` if the number of pages lies between 100 and 500
  - returns the string `infinite` if the number of pages exceeds 500
- `same_author(self, book)`: returns `True` if `book` is written by the same author. `book` is an argument of type `Book`.

Create at least three different objects of class `Book`.

Answer

```
class Book:
    def __init__(self, name, author, pages, genre):
        self.name = name
        self.author = author
        self.pages = pages
        self.genre = genre
    def is_fiction(self):
        if self.genre == "Fiction":
            return True
        else:
            return False
    def is_nonFiction(self):
        if self.genre == "Nonfiction":
            return True
        else:
            return False
    def time_to_read(self):
        if self.pages < 100:
            return "5 day"
        if 100 <= self.pages <= 500:
            return "20 days"
        if self.pages > 500:
            return "infinite"
    def same_author(self, book):
```

```
        if self.author == book.author:
            return True
        else:
            return False

book1 = Book("Igniting Minds", "Kalam", 178, "Nonfiction")
book2 = Book("Wings of Fire", "Kalam", 180, "Nonfiction")
book3 = Book("Harry Potter-1", "Rowling", 309, "Fiction")
book4 = Book("Harry Potter-2", "Rowling", 341, "Fiction")
```

Problem-19

Consider the following csv file named `library.csv`:

```
Name,Author,Genre,Pages
Igniting Minds,Kalam,Nonfiction,English,178
Wings of Fire,Kalam,Nonfiction,English,180
Harry Potter-1,Rowling,Fiction,English,309
Harry Potter-2,Rowling,Fiction,English,341
Harry Potter-3,Rowing,Fiction,English,435
Harry Potter-4,Rowling,Fiction,English,636
The Casual Vacancy,Rowling,Fiction,English,503
1984,Orwell,Fiction,English,328
Animal farm,Orwell,Fiction,English,112
Burmese days,Orwell,Fiction,English,300
Algorithms,Dasgupta,Nonfiction,English,320
A Brief History of Time,Hawking,Nonfiction,English,256
The Universe in a Nutshell,Hawking,Nonfiction,English,224
The Code Book,Simon,Nonfiction,English,416
Big Bang,Simon,Nonfiction,English,560
```

(1) Read this file in Python.

(2) Create an object of class `Books` that was introduced in the previous problem for each book that you find in this file.

(3) Store all these objects in a list named `library`.

(4) Loop through the list of books in `library` and create the following sub-lists:

- all books that belong to the fiction genre
- all books that take less than 20 days to read
- all books that are written by `Rowling`

For each of these questions, make maximum use of the methods in the class `Books`.

Answer

```
books = []
f = open("library.csv", "r")
lines = f.readlines()[1:]
for line in lines:
    name, author, genre, subject, pages = line.strip().split(",")
    b =  Book(name, author, int(pages), genre)
    books.append(b)

fiction_books = []
for book in books:
    if book.is_fiction():
        fiction_books.append(book.name)
books_less_than_20_days = []
for book in books:
    if book.time_to_read() == "5 day":
        books_less_than_20_days.append(book.name)
rowling_books = []
for book in books:
```

```
    if book.author == 'Rowling':
        rowling_books.append(book.name)
print(fiction_books)
print(books_less_than_20_days)
print(rowling_books)
```

Problem-20

Consider the following class:

```
class Shape:
    def __init__(self):
        self.name = None
        self.area = None

    def display(self):
        print(f'{self.name} has an area of {self.area}')

    def get_area(self):
        return self.area

    def is_larger(self, shape):
        return self.area > shape.area
```

Create the following sub-classes that are derived from `Shape`. Each sub-class should override the method `get_area` that returns the area of the shape. Add appropriate attributes for each of these three classes. Assume that all polygons are regular (sides of equal length).

(1) `Triangle`:

(2) `Square`

(3) `Circle`

Answer

```
class Triangle:
    def __init__(self, side):
        super().__init__()
        self.name = "Triangle"
        self.side = side
    def get_area(self):
        self.area = round((3 ** 0.5) * (self.side ** 2) / 4, 2)
        return self.area
class Square:
    def __init__(self, side):
        super().__init__()
        self.name = "Square"
        self.side = side
    def get_area(self):
        self.area = round(self.side ** 2, 2)
        return self.area
class Circle:
    def __init__(self, side):
        super().__init__()
        self.name = "Circle"
        self.side = side
    def get_area(self):
        self.area = round(3.14 * (self.side ** 2), 2)
        return self.area
```

Problem-21

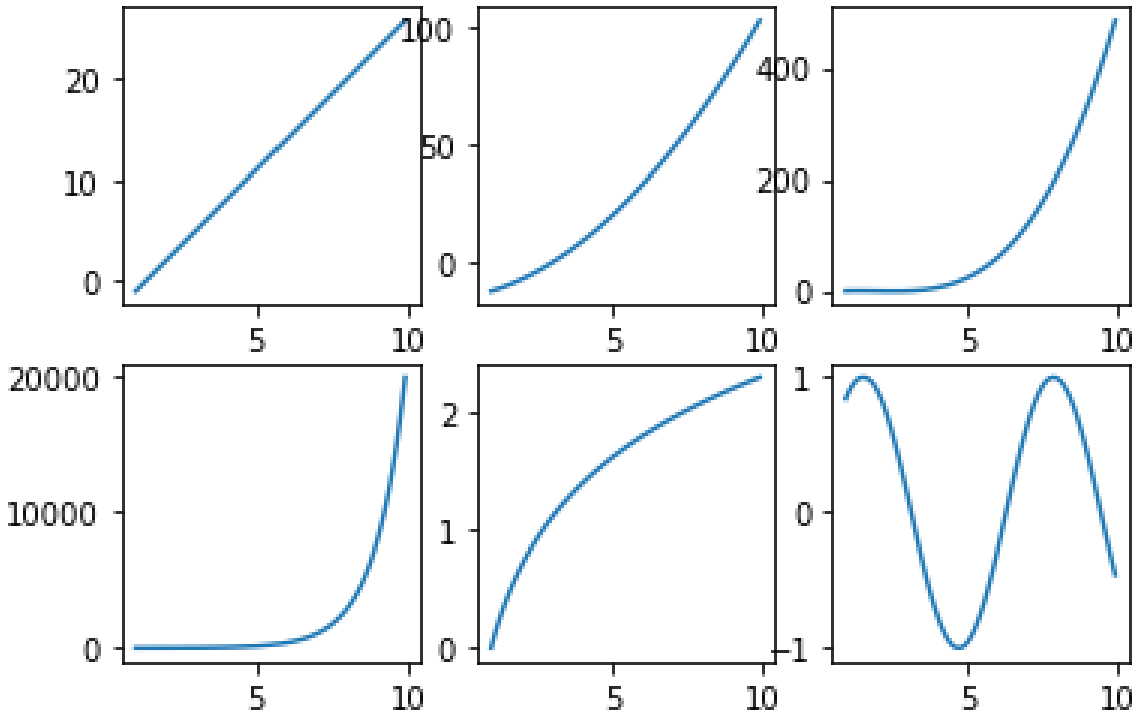
Plot the following functions using Matplotlib:

- $f(x) = 3x - 4$
- $f(x) = x^2 + 2x - 15$
- $f(x) = 5(x - 1)(x - 2)(x - 3)$
- $f(x) = e^x$
- $f(x) = \log x$
- $f(x) = \sin x$

Answer

```
import matplotlib.pyplot as plt
import math
import numpy as np
x = np.arange(1, 10, 0.1)

p = np.poly1d([3, -4])
y = p(x)
plt.subplot(2, 3, 1)
plt.plot(x, y)

p = np.poly1d([1, 2, -15])
y = p(x)
plt.subplot(2, 3, 2)
plt.plot(x, y)

p = np.poly1d([1, 2, 3], True)
y = p(x)
plt.subplot(2, 3, 3)
plt.plot(x, y)

y =  np.power(math.e, x)
plt.subplot(2, 3, 4)
plt.plot(x, y)

y =  np.log(x)
plt.subplot(2, 3, 5)
plt.plot(x, y)

y = np.sin(x)
plt.subplot(2, 3, 6)
plt.plot(x, y)

plt.show()
```

Graph:


20
10
0
5
10
100
50
0
5
10
400
200
0
5
10
20000
10000
0
5
10
2
1
0
5
10
1
0
−1
5
10

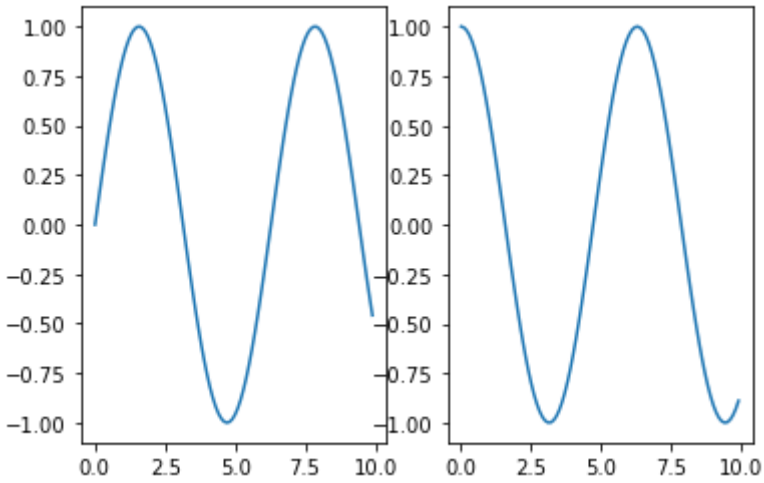
Problem-22

Plot the functions $f(x) = \sin x$ and $g(x) = \cos x$ on the same graph.

Answer

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0,10, 0.1)

y = np.sin(x)
plt.subplot(1, 2, 1)
plt.plot(x, y)

y = np.cos(x)
plt.subplot(1, 2, 2)
plt.plot(x, y)

plt.show()
```

Graph:



Problem-23

Consider the following file, named `scores.csv` that contains the scores of 25 students in Mathematics and Physics.

```
Maths,Physics
20,80
10,30
40,30
20,30
10,5
80,90
99,100
76,84
29,100
100,30
95,92
100,100
70,74
65,88
90,93
89,91
20,40
10,30
20,25
15,34
35,25
50,70
45,55
34,43
60,67
```

Construct a scatter plot with Mathematics scores on the X-Axis and Physics scores on the Y-Axis. Do you observe anything?

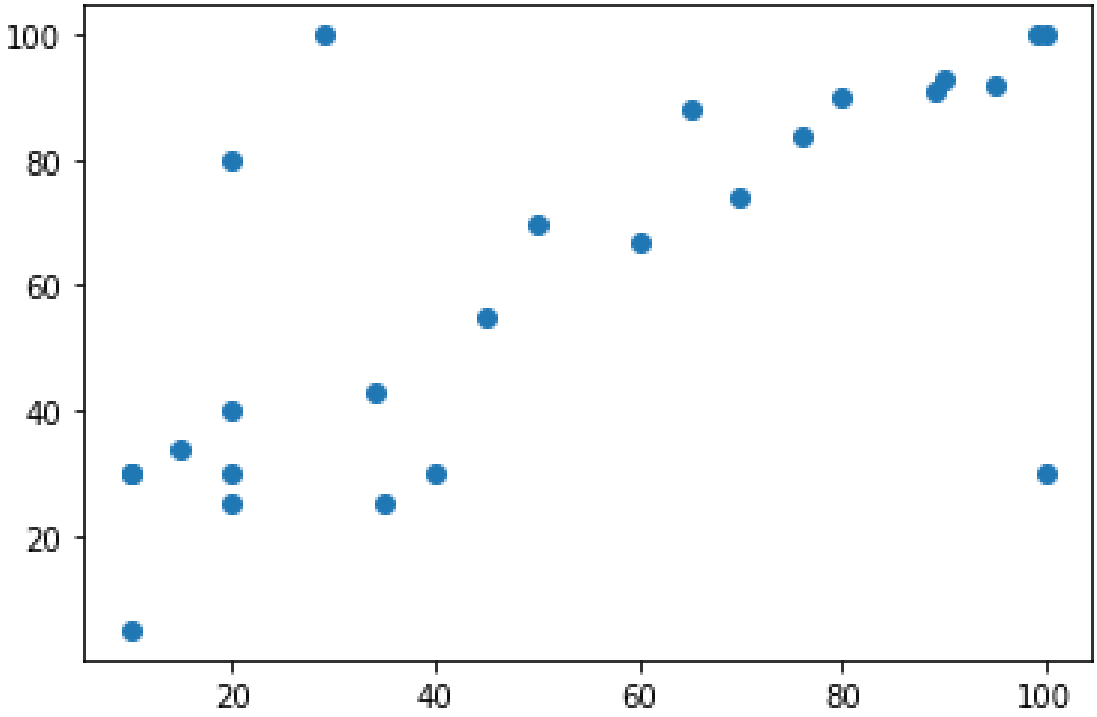
Answer

```
f = open('scores.csv', 'r')
lines =  f.readlines()[1:]
maths = []
physics = []
for line in lines:
    m, p = line.strip().split(",")
    m, p = int(m), int(p)
    maths.append(m)
    physics.append(p)

plt.scatter(maths, physics)
plt.show()
```

Alternatively, you can use `Pandas` library.

```
import matplotlib.pyplot as plt
import pandas as pd
data = pd.read_csv('scores.csv')
plt.scatter(data['Maths'], data['Physics'])
plt.show()
```

Graph:



Problem-24

Perform the following experiment:

(1) Throw a standard, unbiased, six-sided dice n times.

(2) Get the frequency of occurrence of the numbers from 1 to 6.

(3) Convert the frequencies to probabilities.

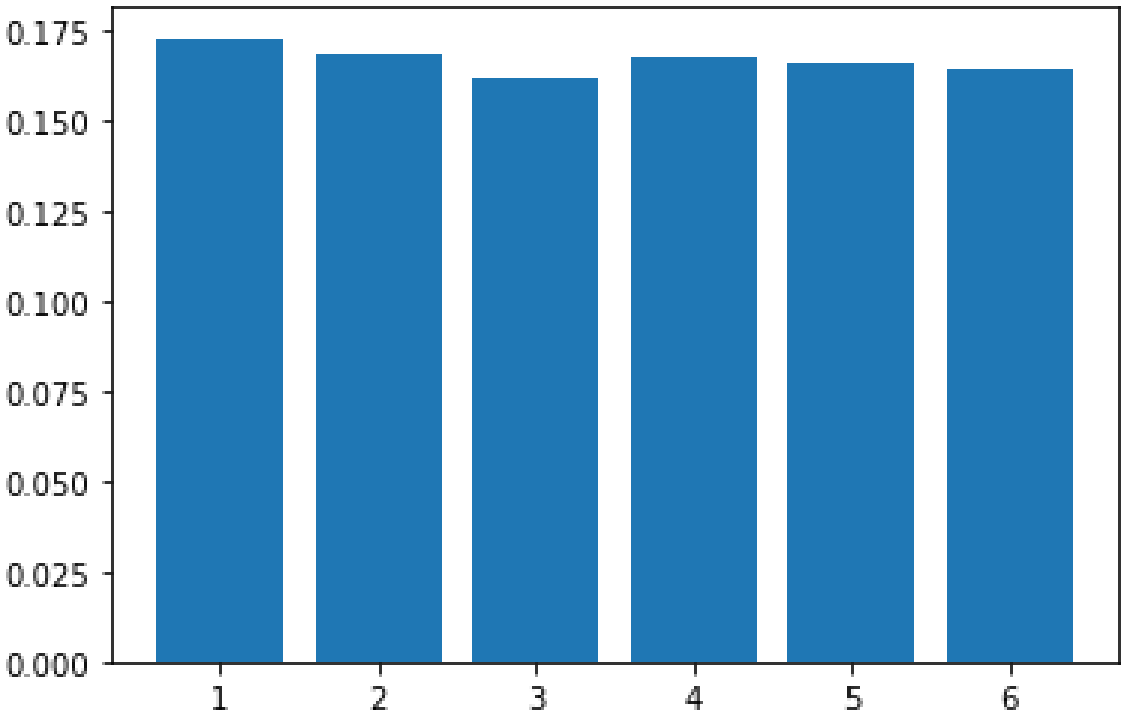
(4) Represent the probabilities of obtaining each of the six numbers in the form of a bar plot.

Run this experiment for the following values of n: 10, 100, 1000, 10000, 100000, 1000000 and note down your observations.

Answer

```
import matplotlib.pyplot as plt
import numpy as np
import random

n = int(input())
trials_list = []
for i in range(n):
    trials_list.append(random.randrange(1, 7))
dice_num_list = range(1, 7)
prob_list = []
for i in dice_num_list:
    count = trials_list.count(i)
    prob_list.append(count/len(trials_list))
x = np.array(dice_num_list)
y = np.array(prob_list)
plt.bar(x, y)
plt.show()
```

Graph: All the probabilities are nearly equal as `n` becomes larger and larger



Problem-25

The equation of a circle centered at the origin of radius r is given by the following equation:

$$x^2 + y^2 = r^2$$

Draw a circle of radius 5 using Matplotlib using a scatter plot.

Answer

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-10, 10, 0.001)
y = np.sqrt(25 - (x ** 2))
plt.scatter(x, y, color = 'k')
plt.scatter(x, -y, color = 'k')
plt.axis('equal')
plt.show()
```

Graph: