計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 6: Basic Python (part 4)

Instructor: 潘國全

kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Warm-up
2. Exercise: Angry bird game
3. Formatted output
4. Read/Write files
5. Exercise: Angry bird game
6. Modules

Today’s plan

1. Python Classes
2. Property, getter, setter
3. Data class
4. Exercise: Angry bird game (again!)
5. Note: Next Monday is a national holiday (no lecture)

The angry bird game (sample code)

1. Check the non-interactive version of the “angry.py” in google classroom
2. The initial velocity and angle are taken from the system arguments in sys.argv

```
try:
    velocity = float(sys.argv[1]) #33.0 # m/s
    angle    = float(sys.argv[2]) #45.0 # degree
except:
    msg = """
    Error: incorrect inputs.
```

Python Classes

Python Class

1. Classes provide a means of bundling data and functionality together.
2. Classes are a fundamental concept in object-oriented programming (OOP)
3. Classes allow you to create your own data types with their own attributes and methods.
4. Classes are the foundation of many Python libraries and frameworks, such as matplotlib, spicy, …etc.

Creating a class

1. To create a class, use the “class” keyword followed by the class name.

2. Class names should start with a capital letter.

ComputationalPhysics > tutorial > tut_08_class.py > ...

```
1    class PhysicalConstant:
2        pass
3
```

Defining Attributes

3. Attributes are variables defined within a class

4. They are also known as instance variables or properties

```python
class PhysicalConstant:
    """
    A class to include physical constants
    """
    def __init__(self, units="cgs"):
        """
        :param units: supprot "cgs" and "mks".
        """
        self.units = units
        self.c_cgs = 2.99792458e10
        self.c_mks = 2.99792458e8
        self.G_cgs = 6.67428e-8
        self.G_mks = 6.67428e-11
        if units=="cgs":
            self.c = self.c_cgs
            self.G = self.G_cgs
        elif units=="mks":
            self.c = self.c_mks
            self.G = self.G_mks
        else:
            print(r"Eror: no such units. units = {units}")
            print(r"Set the units to 'cgs'. ")
            self.c = self.c_cgs
            self.G = self.G_cgs

        return
```

Private variables

5. Private attributes: start with double underscore: __xxx

6. Private attributes cannot be accessed outside the class

```
class ExampleClass:

    def __init__(self):
        self.this_is_public    = "Public"
        self.__this_is_private = "Private"

    def get_private(self):
        print(self.__this_is_private)
```

```
ex = ExampleClass()
#print(ex.__this_is_private)
print(ex.this_is_public)
ex.get_private()
```

Defining methods

7. Methods are functions defined inside a class that can be called on an object of the class. To define a method, you create a function inside the class. The first argument of the method should always be self, which refers to the object calling the method.

```
28      def set_units(self, units):
29          self.units = units
30          if units=="cgs":
31              self.c = self.c_cgs
32              self.G = self.G_cgs
33          elif units=="mks":
34              self.c = self.c_mks
35              self.G = self.G_mks
36          else:
37              print(r"Eror: no such units. units = {units}")
38              print(r"Set the units to 'cgs'. ")
39              self.c = self.c_cgs
40              self.G = self.G_cgs
41          return
```

Creating objects

8. To create an object of a class, you call the class name as if it were a function and pass in any arguments required by the __init__ method.

```
43  if __name__=='__main__':
44
45      const = PhysicalConstant(units="cgs")
46      print(f"[{const.units}] c = {const.c:12.6e}")
47      const.set_units("mks")
48      print(f"[{const.units}] c = {const.c:12.6e}")
49
```

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_08_class.py
[cgs] c = 2.997925e+10
[mks] c = 2.997925e+08
```

property, getter, setter

Property, getter, setter

1. To protect some attributes of a class, we make them private attributes (Encapsulation).
2. Getters: methods to access the private attributes from a class
3. Setters: methods to set the values of private attributes in a class

```
class NormalClass:

    def __init__(self,name):
        self.name = name
```

```
class ExampleClass:

    def __init__(self, name):
        self.__name = name

    def get_name(self):
        return self.__name

    def set_name(self, new_name):
        self.__name = new_name
```

```
if __name__=='__main__':

    a = ExampleClass(name="Albert")
    #print(a.__name)
    print(a.get_name())
    a.set_name("Einstein")
    print(a.get_name())
```

Property, getter, setter

4. The python way to use getter and setter.

```
class ExampleClass2:

    def __init__(self, name):
        self.__name = name

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, new_name):
        self.__name = new_name
```

```
if __name__=='__main__':

    a = ExampleClass(name="Albert")
    #print(a.__name)
    print(a.get_name())
    a.set_name("Einstein")
    print(a.get_name())

    b = ExampleClass2(name="Albert")
    print(b.name)
    b.name = "Einstein"
    print(b.name)
```

Property, getter, setter

5. Property (setter & getter) can help you to avoid inconsistence.

```
class Name:
    def __init__(self,first_name,last_name):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def full_name(self):
        return self.first_name+" "+self.last_name
```

```
c = Name("Albert", "Einstein")
print(c.full_name)
```

Property, getter, setter

6. It is also possible that your setter can be more complicate. Property can be useful for verification.

```
class Celsius:
    def __init__(self, temperature = 0):
        self._temperature = temperature

    def to_fahrenheit(self):
        return (self._temperature * 1.8) + 32

    def to_kevin(self):
        return self._temperature + 273.15

    @property
    def temperature(self):
        return self._temperature

    @temperature.setter
    def temperature(self, value):
        if value < -273.15
            raise ValueError("Temperature cannot below -273.15 C")
        self._temperature = value
```

Data Class

Python data class

1. New feature after Python 3.7
2. With the “@dataclass” decorator.
3. A data class is a class that is designed to only hold data values.
4. They are typically used to store information that will be passed between different parts of a program or a system. (For example, in the angry bird exercise, we use a dictionary “data={}” to store the trajectory information)
5. REF: https://docs.python.org/3/library/dataclasses.html

Python data class

1. The regular approach:

```
class Point_regular:
    def __init__(self,t,x,y):
        self.x = x
        self.y = y
        return
```

2. Use the @dataclass

```
from dataclasses import dataclass

@dataclass
class Point:
    x:float
    y:float
```

Python data class

3. Dataclass supports default values as well:

```
@dataclass
class Person:
    name: str = "Albert Einstein"
    age: int  = 30
```

Exercise: Point and Trajectory class

1. In the angry bird game, we record the trajectory of an angry bird.

2. The trajectory can be used (1) to check if you hit the target, (2) to visualize the motion, (3) can be dumped to a data file.

3. Let's write two dataclass "Point" and "Trajectory" for the angry bird game in a module file named "data.py".

4. The "Point" dataclass contains these variables: t, x, y, vx, vy, ax, ay

5. The "Trajectory" dataclass contains these variables: times: list(float), posx: list(float), posy: list(float), velx: list(float), vely: list(float), accx: list(float), accy: list(float)

6. The "Trajectory" class has the method "append(self, point: Point)" to add new data in a trajectory.

Exercise 1: Point and Trajectory class

7. The “Trajectory” class has the method “dump(self)” to save the trajectory to a txt file.

8. Example usage:

```
if __name__=='__main__':

    p  = Point(t=0,x=0,y=0,vx=10,vy=10,ax=0,ay=0)
    p2 = Point(t=0,x=0,y=0,vx=10,vy=10)

    traj = Trajectory()
    traj.append(p)
    traj.append(p)
    traj.dump()
```

(See demo)

Exercise 2: More class examples

1. Imagine a scenario that there could be more than one target (pig) in the game
2. The targets could move as well.
3. Thus, both the bird and pigs are game characters (but one is the player; others are NPC)
4. Write a “Character” class three properties: name : str, position : Point, traj : Trajectory
5. The “Character” class has two methods: (1) update_do(self, time:float) : to update self.position to a new time, using the analytical solution. (2) distance_to(self, pt: Point): to valuate the distance to another point: Point.

Exercise 2: More class examples

6. Example usage:

```
if __name__=='__main__':

    pt   = Point(0,0,0,10,10)
    bird = Character("Angry Bird", pt)

    times = [0.1*t for t in range(20)]

    for t in times:
        bird.update_to(t)

    print(bird.traj.times)
```

(See demo)

Exercise 3: More class examples

1. Write another class called “Game” to handle the initial conditions and the game controls.

2. Import “Point”, “Trajectory”, and “Character”.

3. The game class has several methods to hand the initial conditions and game controls.

   You could design it by yourself (for example; add_bird(); add_pigs(); shoot(); check(), …)

4. Write a “play()” method to play the game.

Exercise 3: More class examples

5. Example usage:

(See demo)

```
if __name__=='__main__':

    game = Game()

    if len(sys.argv) != 3:
        msg = """
        Incorrect inputs.

        Try: python game.py 30 45

        """
        print(msg)
        quit()

    velocity = float(sys.argv[1]) #33.0 # m/s
    angle    = float(sys.argv[2] )#45.0 # degree

    angle_rad = math.radians(angle)
    vx  = velocity * math.cos(angle_rad)
    vy  = velocity * math.sin(angle_rad)
    ax  = 0.0
    ay  = -9.81

    pt   = Point(0,0,0,vx,vy,ax,ay)
    bird = Character(name="Angry Bird",pt=pt)

    pt1  = Point(0,100,0,0,0,0,0)
    pig1  = Character(name="Pig 1",pt=pt1)

    pt2  = Point(0,100,0,3,0,0,0)
    pig2  = Character(name="Pig 2",pt=pt2)
    game.play(bird,[pig1, pig2])
```

Class inheritance

Inheritance

1. a language feature would not be worthy of the name “class” without supporting inheritance.

2. Example:

```
class AngryBird(Character):

    def __init__(self, name, mass=1):
        self.mass = 1.0
        pt = Point()
        super().__init__(name, pt)
        return

    def set_velocity_and_angle(self, velocity:float, angle:float):
        angle_rad = math.radians(angle)
        vx = velocity * math.cos(angle_rad)
        vy = velocity * math.sin(angle_rad)
        self.point.vx = vx
        self.point.vy = vy
        return

    def apply_force(self, fx, fy):
        m = self.mass
        self.point.ax = fx/m
        self.point.ay = fy/m
        return
```

```
class Pig(Character):

    def __init__(self, name, distance):
        self.distance = distance
        pt = Point()
        pt.x = distance
        super().__init__(name,pt)
        return

    def set_velocity(self, speed):
        self.point.vx = speed
        return
```

Inheritance

3. The game can be simplified to

```
bird = AngryBird(name="Angry Bird")
bird.set_velocity_and_angle(velocity,angle)
bird.apply_force(fx=0,fy=-9.81)

pig1 = Pig(name="Pig 1", distance=100)
pig1.set_velocity(speed= 1.0)
pig2 = Pig(name="Pig 2", distance=100)
pig2.set_velocity(speed= -1.0)

game.play(bird,[pig1, pig2])
```