**Exercise 1- Inheritance**

**1. What are the parent and child classes here?**

**A.** spell is the parent class in the program and Accio and Confundo both are the child classes of spell.

**2. What does the code print out? (Try figuring it out without running it in Python)**

**A. :**

|  |  |  |
| --- | --- | --- |
| **Code** |  | **Output** |
| spell = Accio()  spell.execute() | - | Accio |
| study\_spell(spell) | - | Summoning Charm Accio |
| study\_spell(Confundo()) | - | Confundus Charm Confundo  Causes the Victim to become confused and befuddled |

**3. Which get description method is called when ‘study spell(Confundo())’ is executed? Why?**

**A.** The method inside the class Confundo will be called, because similar methods are present in both classes so the child class will override the parent class method

**4. What do we need to do so that ‘print Accio()’ will print the appropriate description (‘This charm summons an object to the caster, potentially over a significant distance’)? Write own the code that we need to add and/or change.**

**A.** We can make a new get\_description method in Accio class and print it with the print(Accio()) command.

Code:

def get\_description(self):

return 'This charm summons an object to the caster, potentially over a significant distance.'

**Exercise 2- Overriding**

**Program:**

class Address:

def \_\_init\_\_ (self, street, num):

self.street\_name = street

self.number = num

class CampusAddress(Address):

def \_\_init\_\_ (self, office\_number, num = 77, street = "Massachusetts Ave"):

#Address.\_\_init\_\_(self, street,num)

self.street\_name = street

self.number = num

self.office\_number = office\_number

Sarina\_addr = CampusAddress("32-G904")

print(Sarina\_addr.office\_number)

print(Sarina\_addr.street\_name)

print(Sarina\_addr.number)

**Output:**

32-G904

Massachusetts Ave

77

**Screen Shot:**

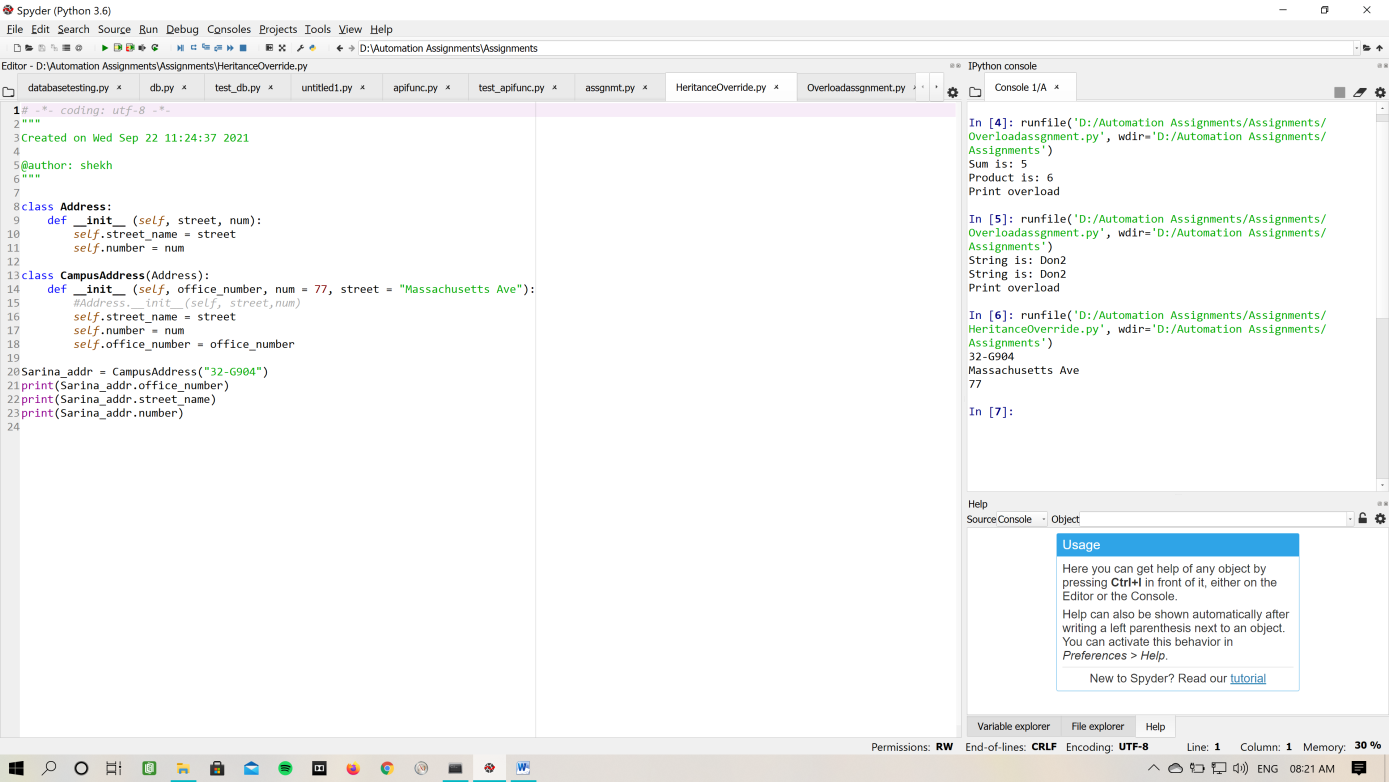


Figure 1 Overriding

**Exercise 3: Method Overload.**

**Program:**

class Param:

def \_\_init\_\_(self, n, v):

self.n = n

self.v = v

def add(self):

if isinstance(self.n,int):

print("Sum is:", self.n + self.v)

else:

print("String is:", self.n + str(self.v))

def mul(self):

if isinstance(self.n,int):

print("Product is:", self.n \* self.v)

else:

print("String is:", self.n + str(self.v))

def \_\_str\_\_(self):

return "Print overload"

ob = Param("Don",2)

ob.add()

ob.mul()

print(ob

**Output:**

String is: Don2

String is: Don2

Print overload

**Screen Shot:**

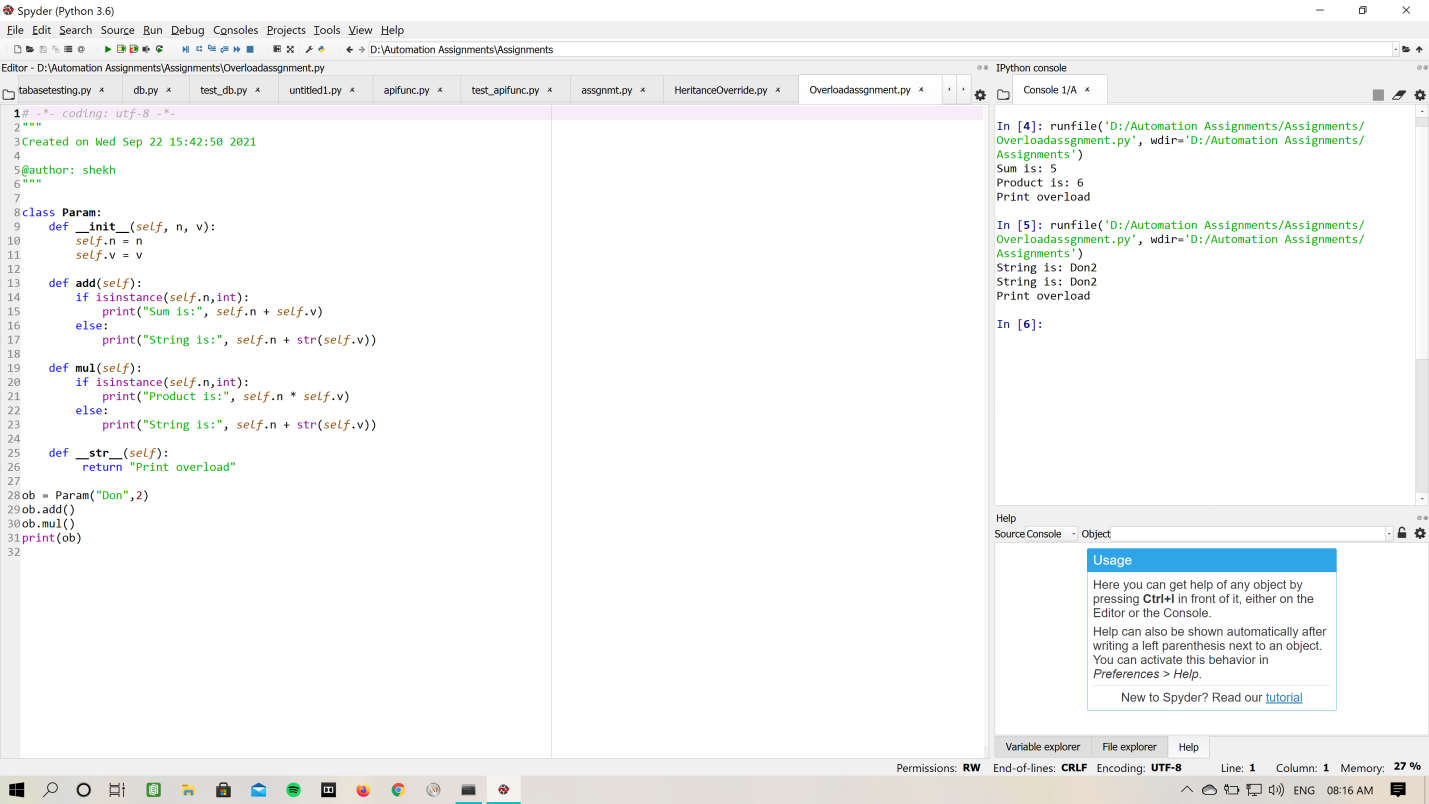


Figure Method Overload

**Exercise 4: Stopwatch**

**Program:**

import time

class Stopwatch:

def \_\_init\_\_(self):

self.startTime = time.clock()

self.endTime = None

def getStartTime(self):

return self.startTime

def getEndTime(self):

return self.endTime

def start(self):

self.startTime = time.time()

def stop(self):

self.endTime = time.time()

def getElapsedTime(self):

elapsedTime = None;

if (self.endTime != None):

elapsedTime = (self.endTime - self.startTime) \* 1000

return elapsedTime

def measure\_time():

sum = 0

timer = Stopwatch()

timer.start()

for i in range(1, 1000000):

sum = sum + i

timer.stop()

print("Execution time :", timer.getElapsedTime());

measure\_time()

**Output:**

Execution time : 64.62454795837402

**Screen Shot:**

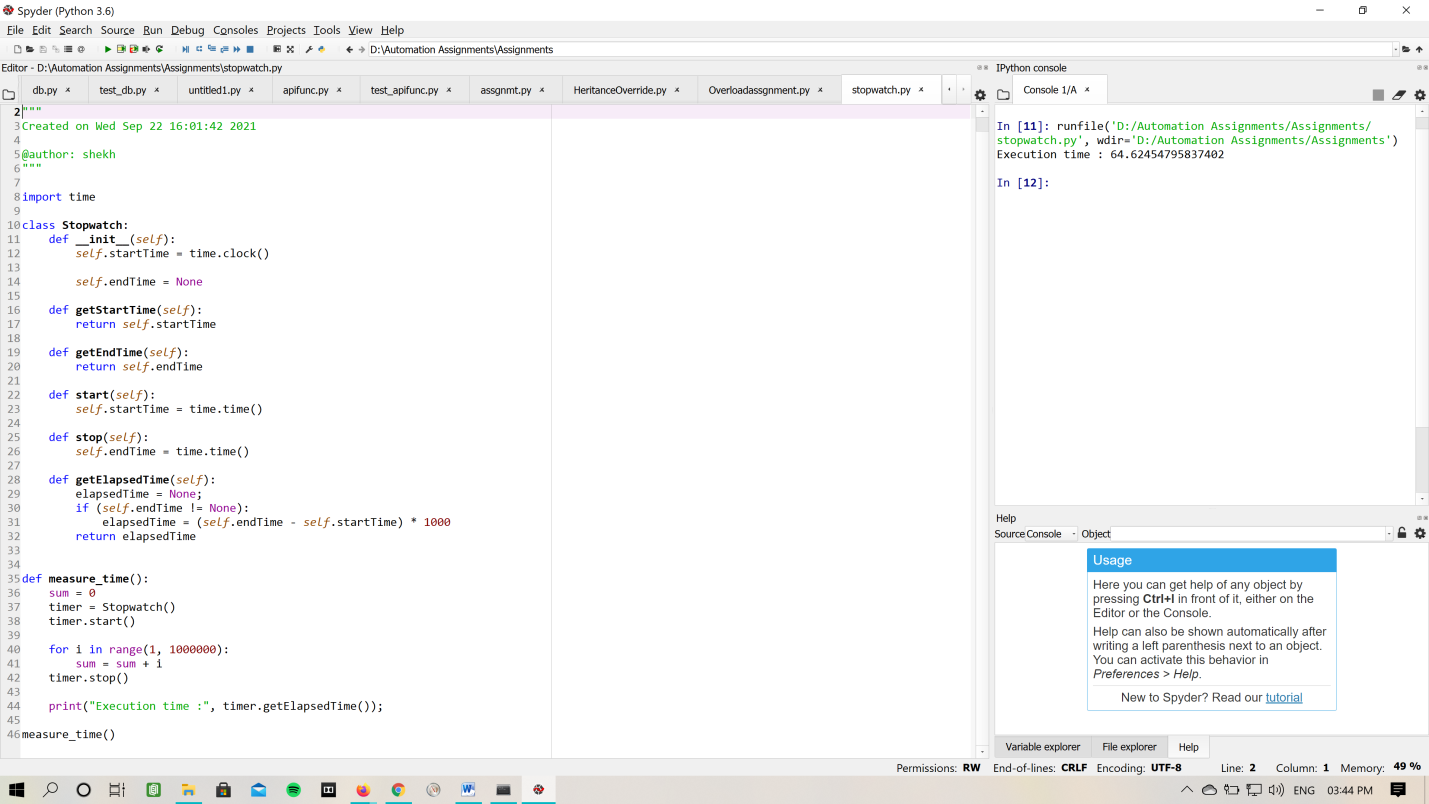
****

Figure Stop Watch