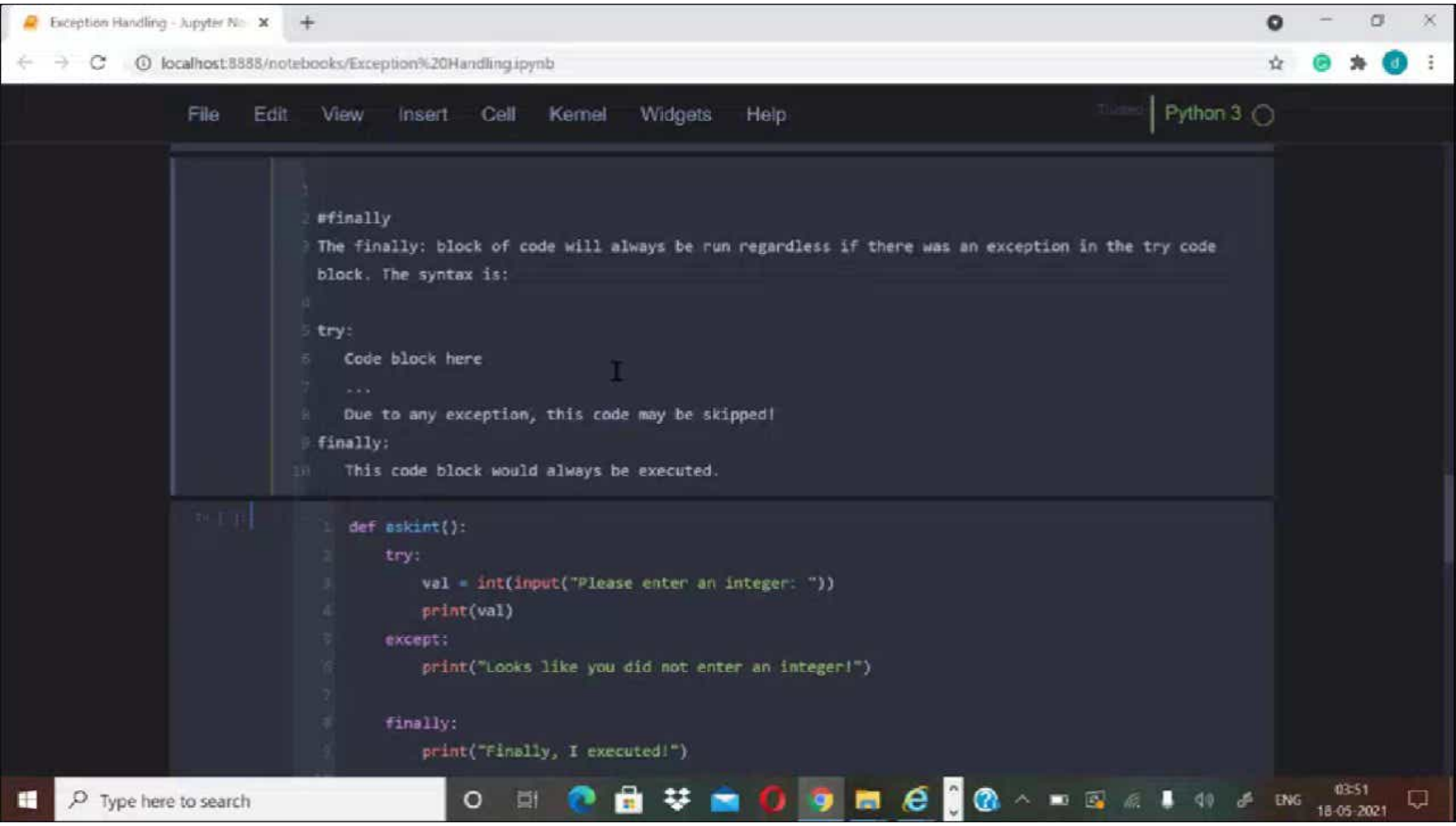
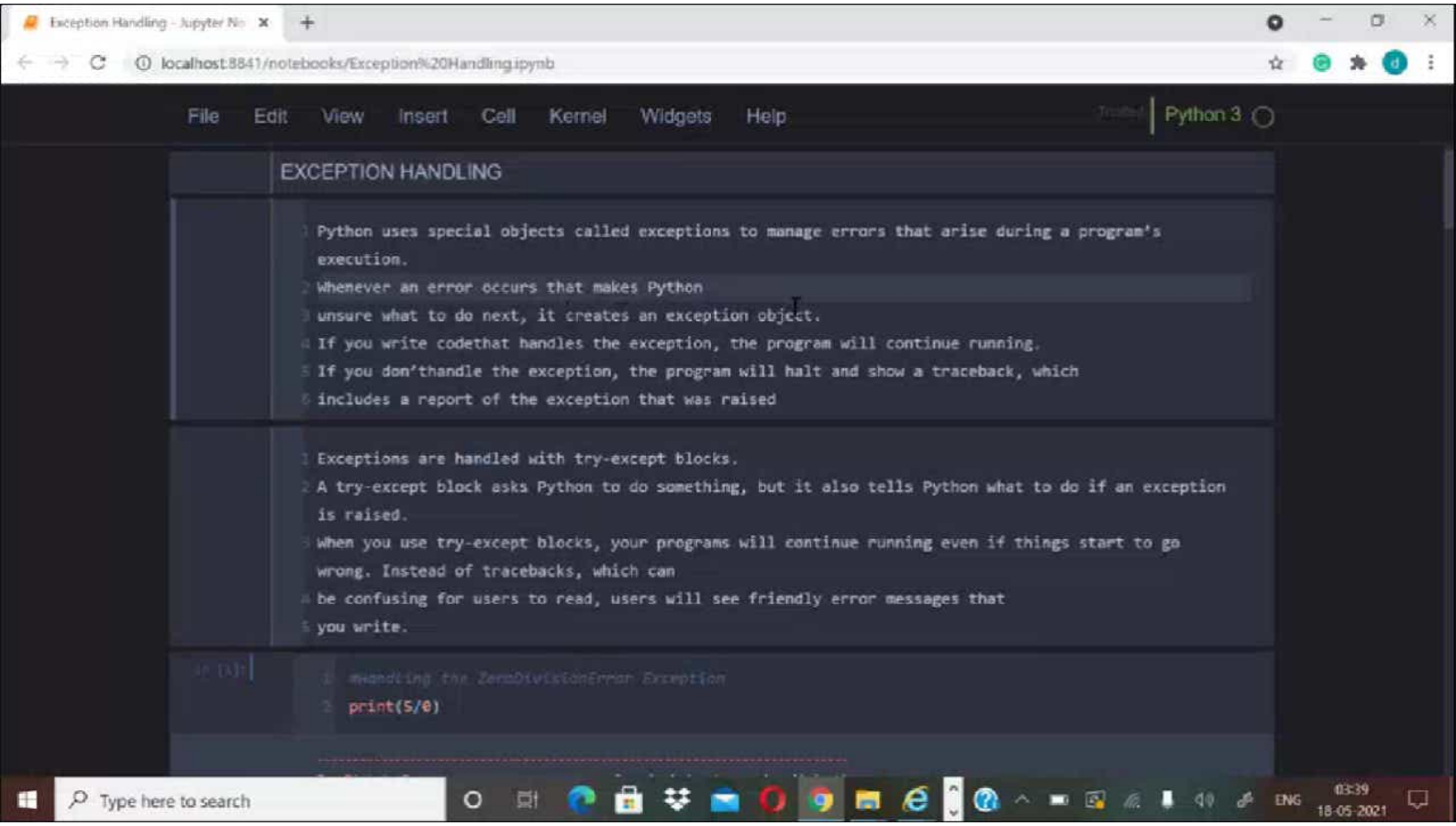
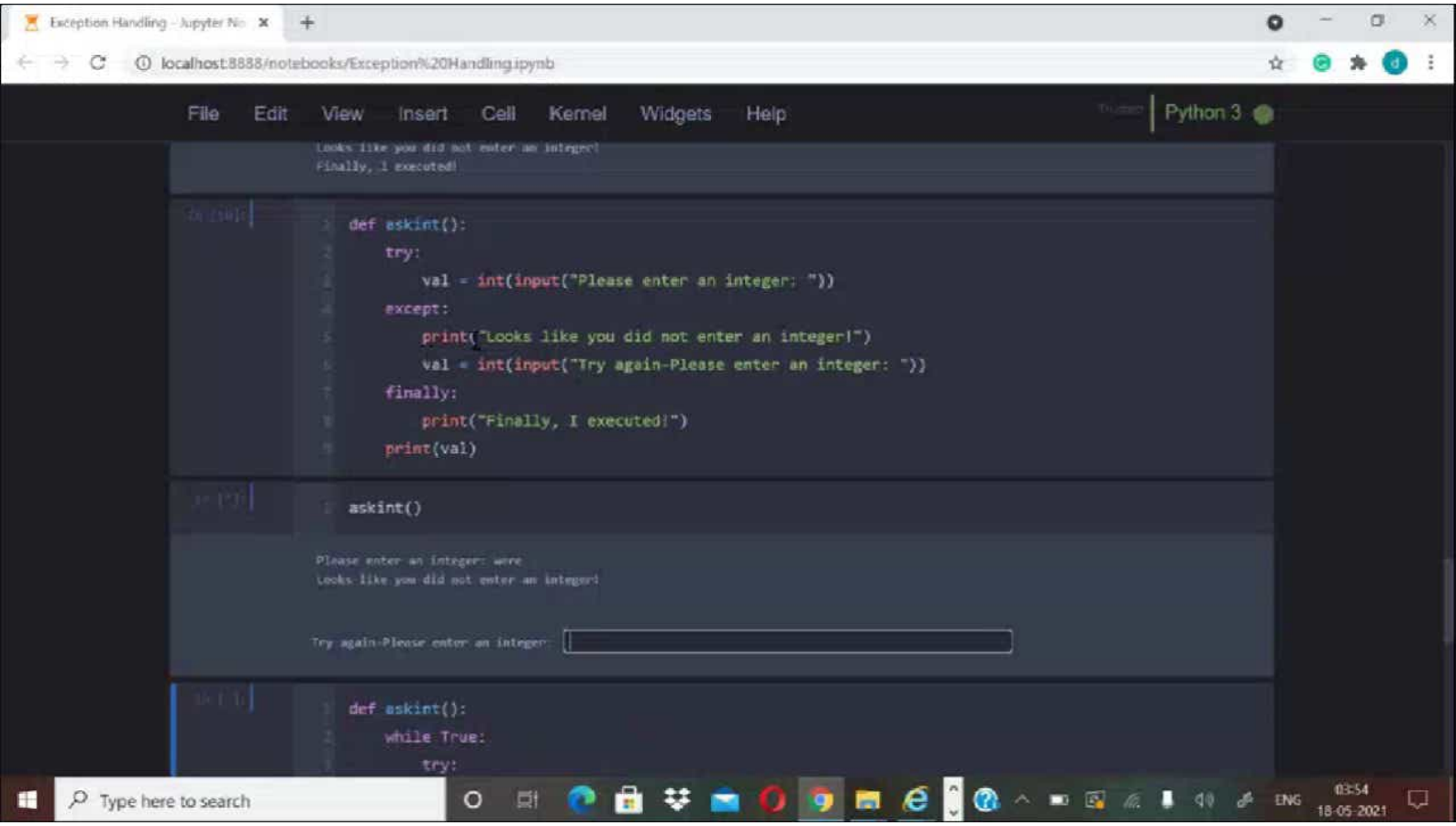
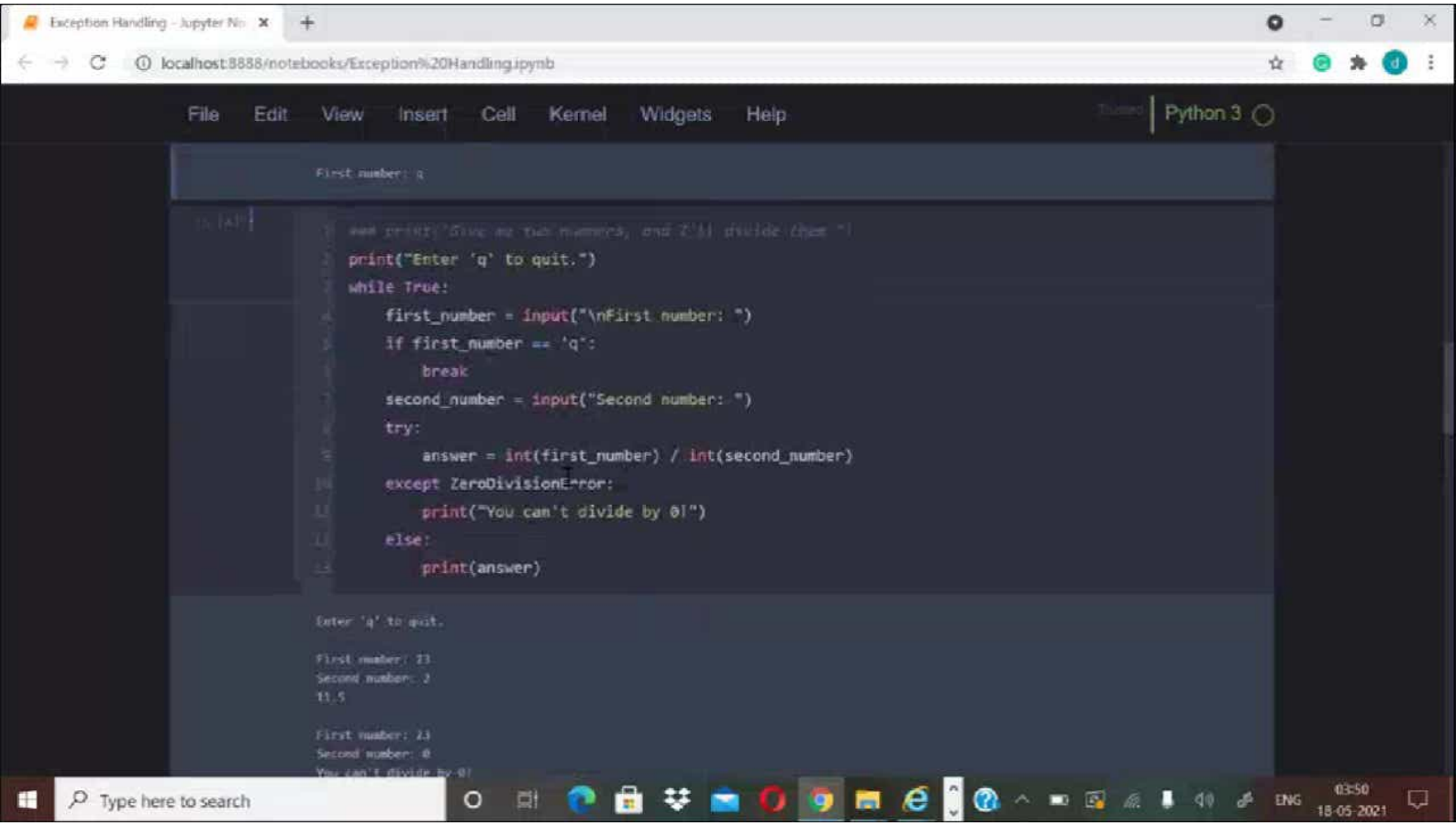
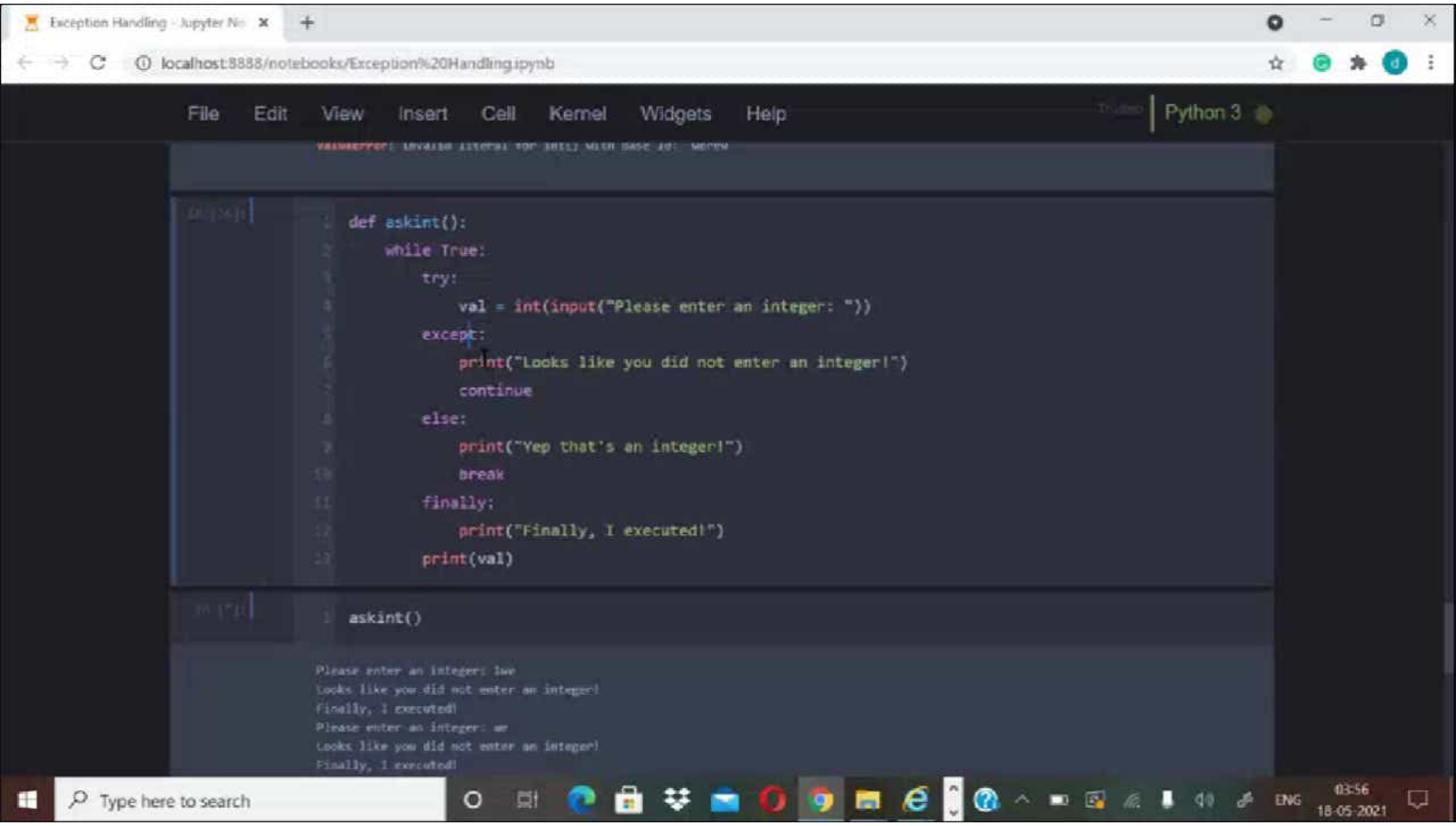
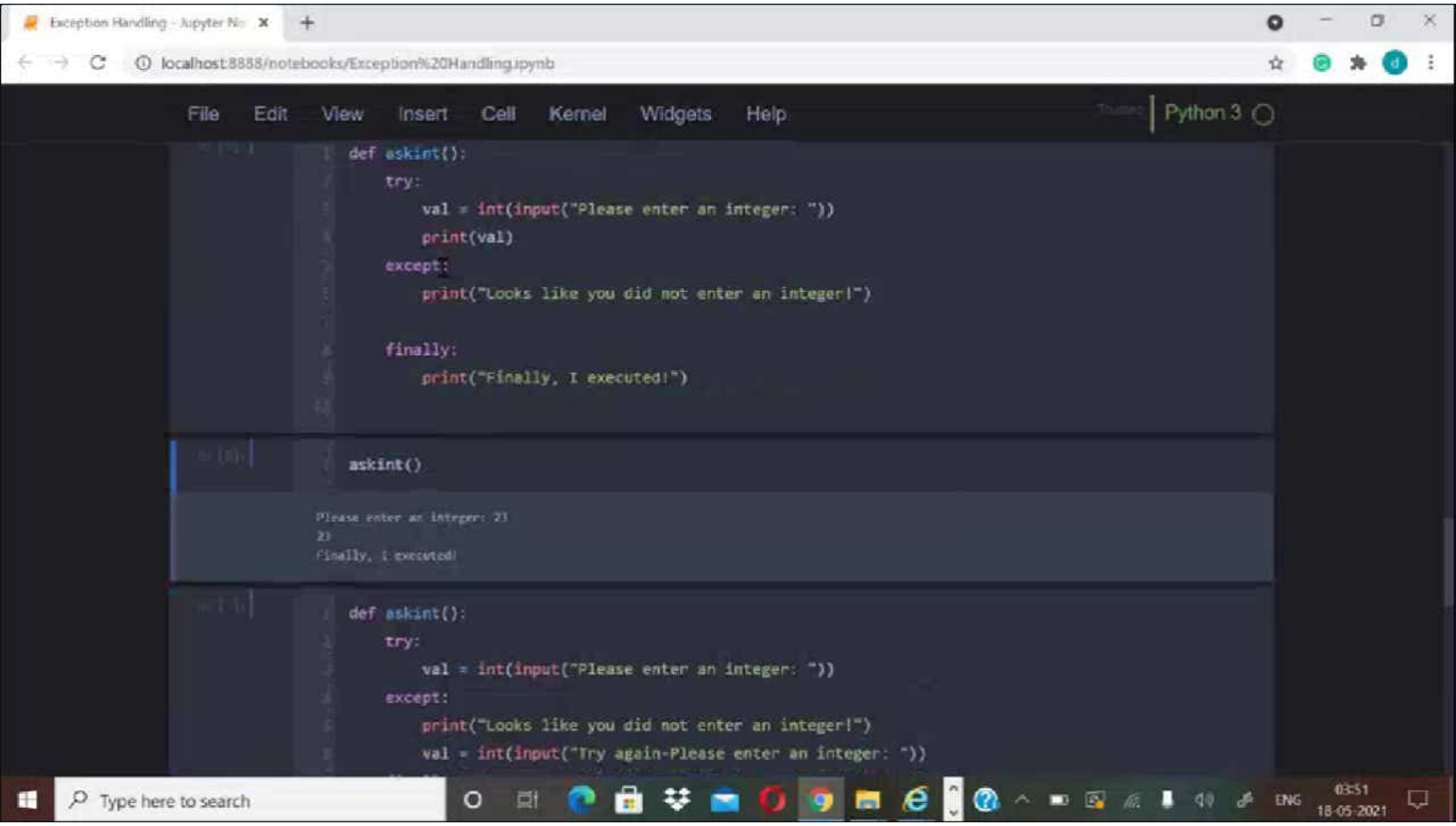

Exception Handling - Jupyter No
localhost:8888/notebooks/Exception%20Handling.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
#finally
The finally: block of code will always be run regardless if there was an exception in the try code block. The syntax is:
try:
Code block here
...
Due to any exception, this code may be skipped!
finally:
This code block would always be executed.
def askint():
try:
val = int(input("Please enter an integer: "))
print(val)
except:
print("Looks like you did not enter an integer!")
finally:
print("Finally, I executed!")
Type here to search
ENG
03:51
18-05-2021

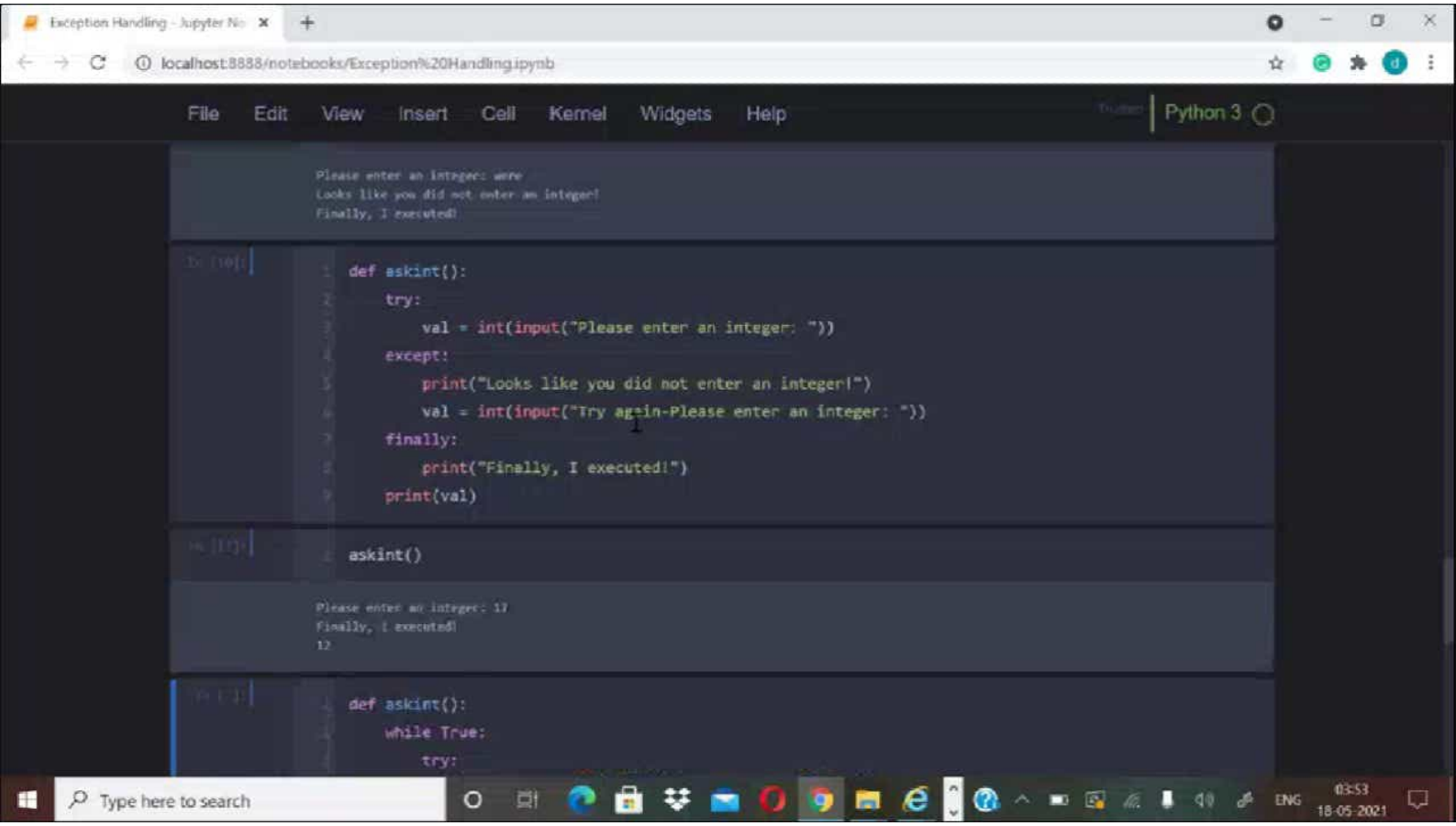

Exception Handling - Jupyter No
localhost:8841/notebooks/Exception%20Handling.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
EXCEPTION HANDLING
Python uses special objects called exceptions to manage errors that arise during a program's execution.
Whenever an error occurs that makes Python
unsure what to do next, it creates an exception object.
If you write codethat handles the exception, the program will continue running.
If you don'thandle the exception, the program will halt and show a traceback, which
includes a report of the exception that was raised
Exceptions are handled with try-except blocks.
A try-except block asks Python to do something, but it also tells Python what to do if an exception is raised.
When you use try-except blocks, your programs will continue running even if things start to go wrong. Instead of tracebacks, which can
be confusing for users to read, users will see friendly error messages that
you write.
In [1]:
#handling the ZeroDivisionError Exception
print(5/0)
Type here to search
ENG
03:39
18-05-2021

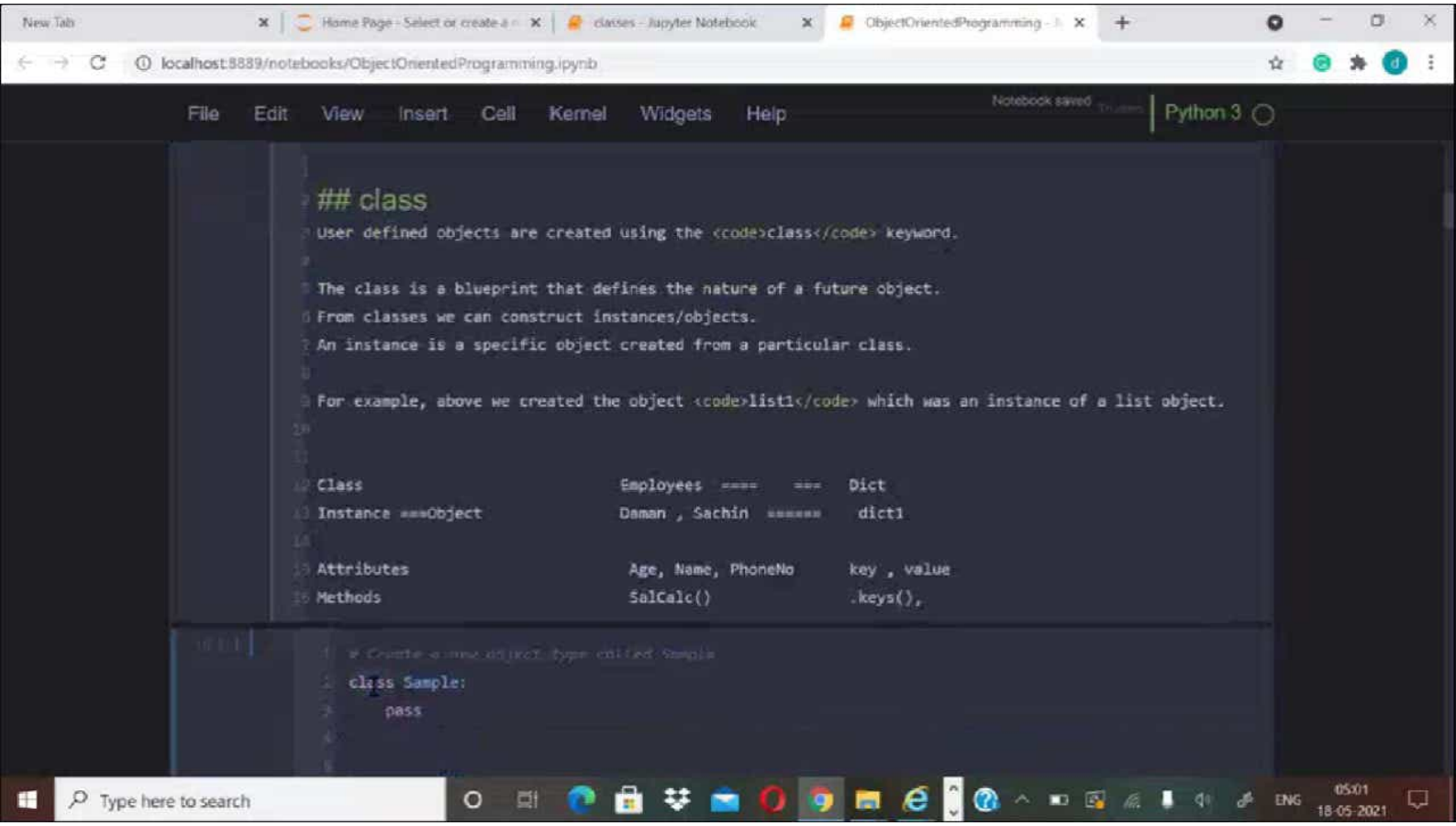

Exception Handling - Jupyter No
localhost:8888/notebooks/Exception%20Handling.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
Looks like you did not enter an integer!
Finally, I executed!
def askint():
try:
val = int(input("Please enter an integer: "))
except:
print("Looks like you did not enter an integer!")
val = int(input("Try again-Please enter an integer: "))
finally:
print("Finally, I executed!")
print(val)
askint()
Please enter an integer: were
Looks like you did not enter an integer!
Try again-Please enter an integer:
def askint():
while True:
try:
Type here to search
ENG
03:54
18-05-2021

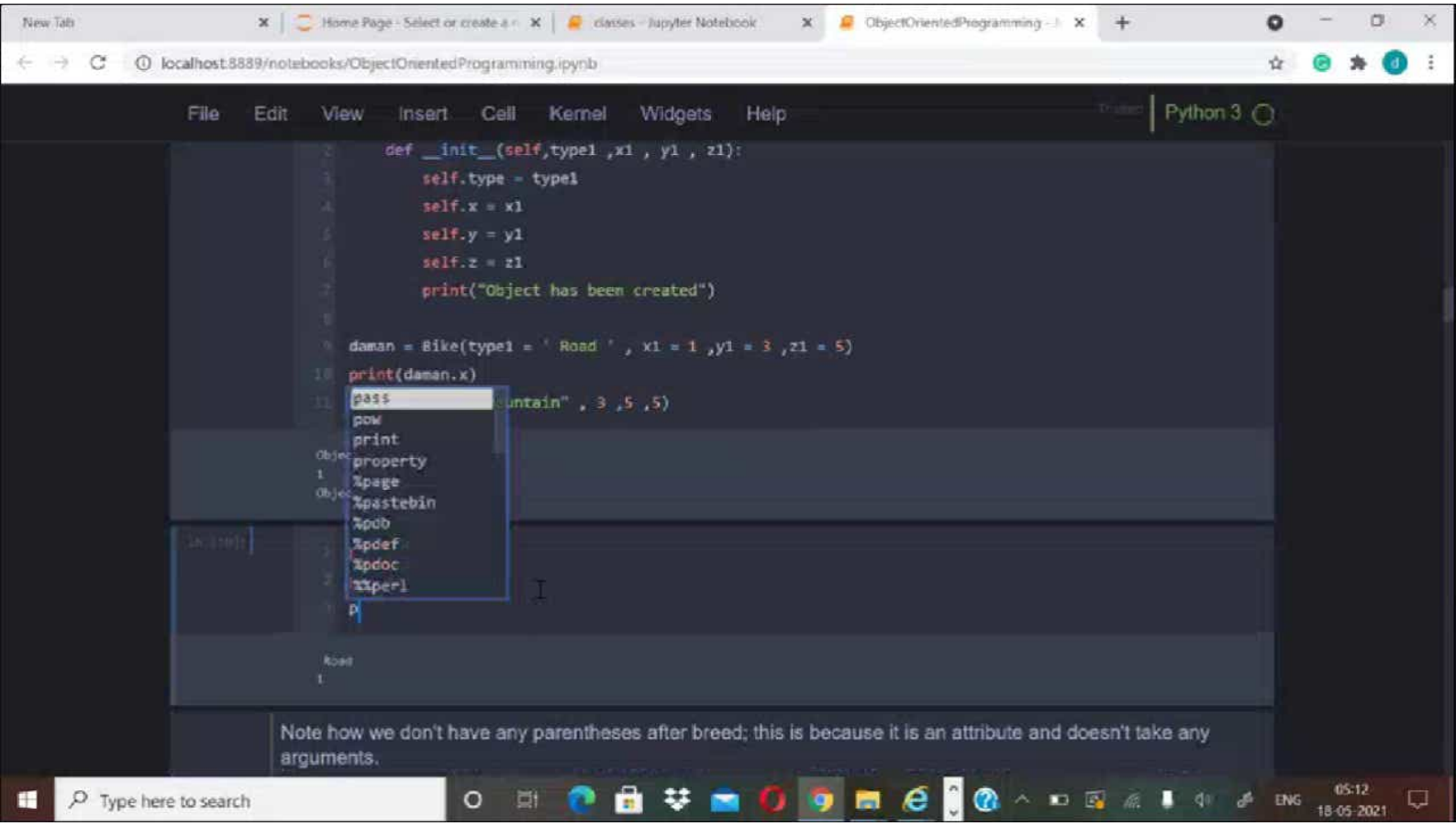

Exception Handling - Jupyter No
localhost:8888/notebooks/Exception%20Handling.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
First number: q
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")
while True:
first_number = input("\nFirst number: ")
if first_number == 'q':
break
second_number = input("Second number: ")
try:
answer = int(first_number) / int(second_number)
except ZeroDivisionError:
print("You can't divide by 0!")
else:
print(answer)
Enter 'q' to quit.
First number: 23
Second number: 2
11.5
First number: 23
Second number: 0
You can't divide by 0!
Type here to search
ENG
03:50
18-05-2021

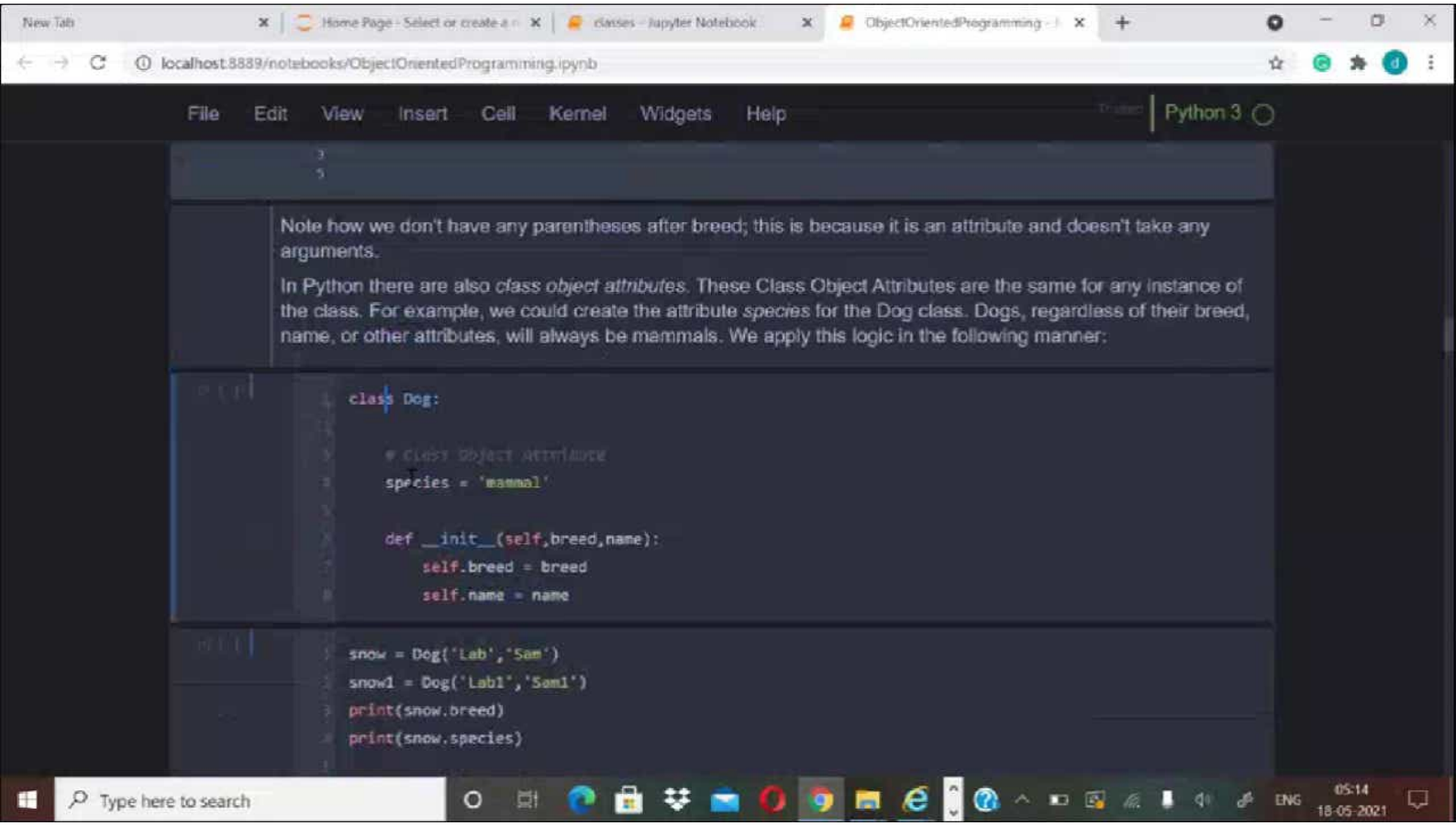

Exception Handling - Jupyter No
localhost:8888/notebooks/Exception%20Handling.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
def askint():
while True:
try:
val = int(input("Please enter an integer: "))
except:
print("Looks like you did not enter an integer!")
continue
else:
print("Yep that's an integer!")
break
finally:
print("Finally, I executed!")
print(val)
askint()
Please enter an integer: lwe
Looks like you did not enter an integer!
Finally, I executed!
Please enter an integer: we
Looks like you did not enter an integer!
Finally, I executed!
Type here to search
ENG
03:56
18-05-2021

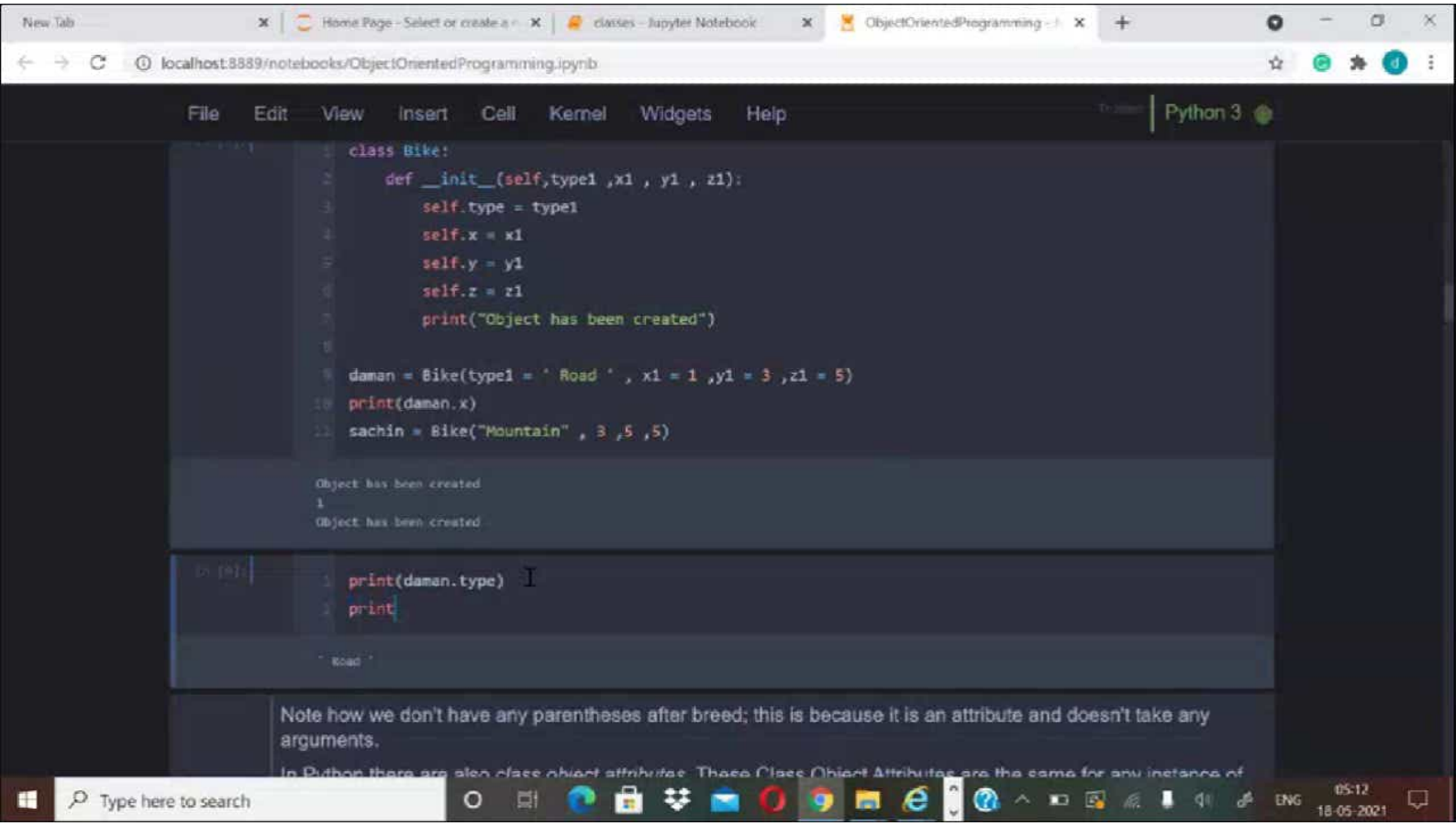

Exception Handling - Jupyter No
localhost:8888/notebooks/Exception%20Handling.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
def askint():
try:
val = int(input("Please enter an integer: "))
print(val)
except:
print("Looks like you did not enter an integer!")
finally:
print("Finally, I executed!")
askint()
Please enter an integer: 23
23
Finally, I executed!
def askint():
try:
val = int(input("Please enter an integer: "))
except:
print("Looks like you did not enter an integer!")
val = int(input("Try again-Please enter an integer: "))
Type here to search
ENG
03:51
18-05-2021

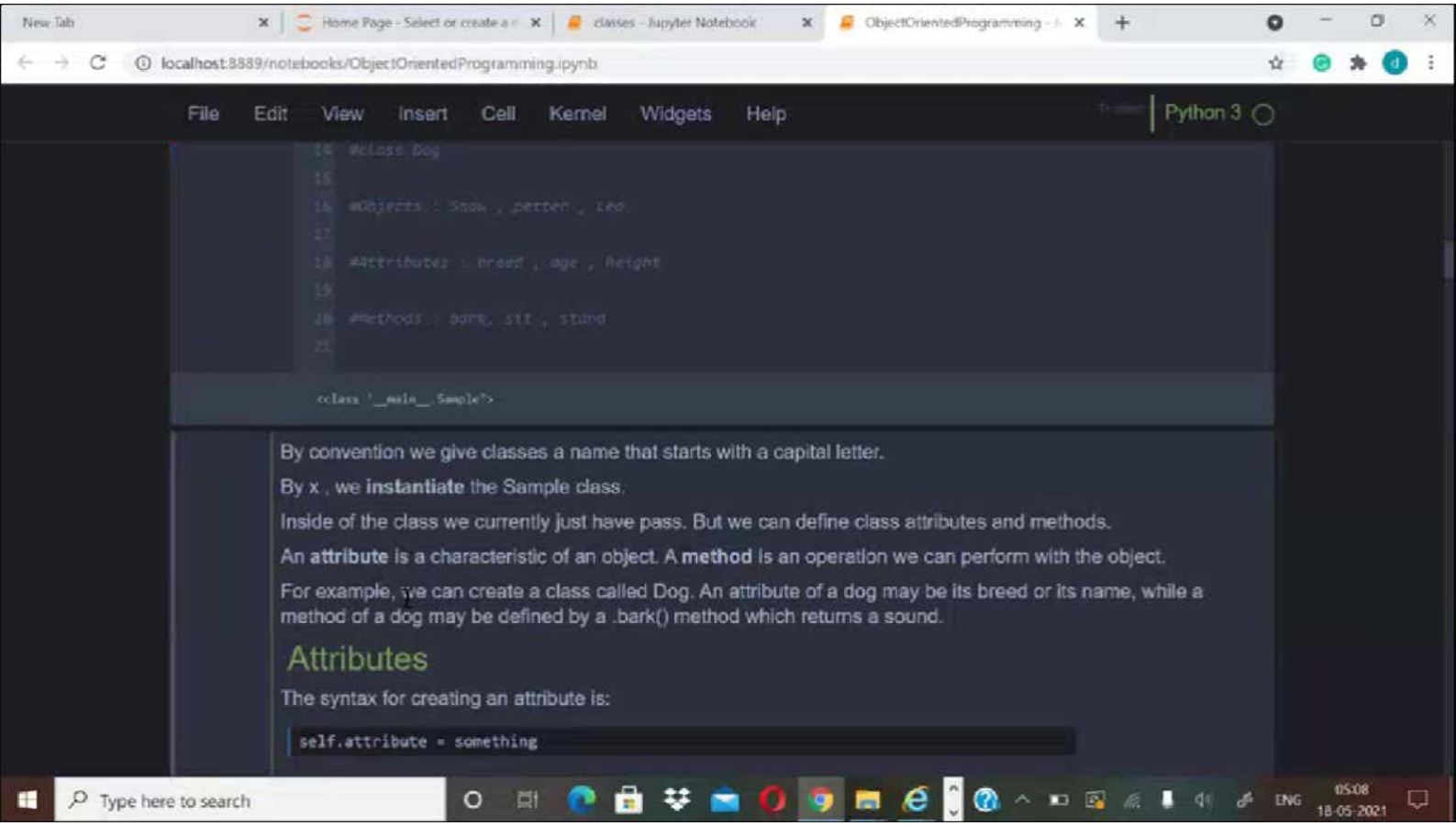

Exception Handling - Jupyter No
localhost:8888/notebooks/Exception%20Handling.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
Please enter an integer: were
Looks like you did not enter an integer!
Finally, I executed!
def askint():
try:
val = int(input("Please enter an integer: "))
except:
print("Looks like you did not enter an integer!")
val = int(input("Try again-Please enter an integer: "))
finally:
print("Finally, I executed!")
print(val)
askint()
Please enter an integer: 12
Finally, I executed!
12
def askint():
while True:
try:
Type here to search
ENG
03:53
18-05-2021

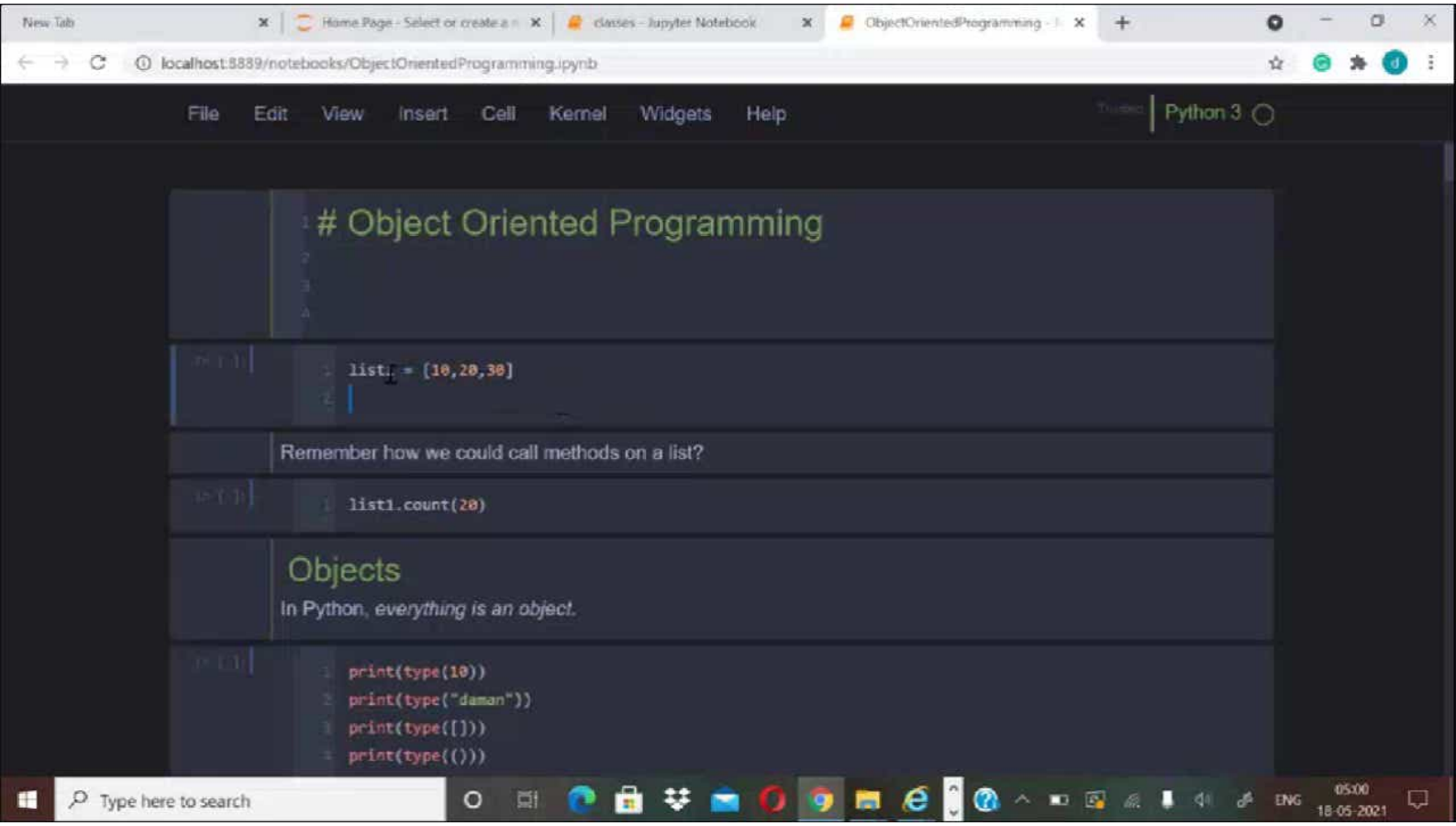

New Tab
Home Page - Select or create a n
classes - Jupyter Notebook
ObjectOrientedProgramming - J
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Notebook saved
Python 3
class
User defined objects are created using the <code>class</code> keyword.
The class is a blueprint that defines the nature of a future object.
From classes we can construct instances/objects.
An instance is a specific object created from a particular class.
For example, above we created the object <code>list1</code> which was an instance of a list object.
Class Employees ==== === Dict
Instance ===Object Daman , Sachin ====== dict1
Attributes Age, Name, PhoneNo key , value
Methods SalCalc() .keys(),
class Sample:
pass
Type here to search
ENG
05:01
18-05-2021

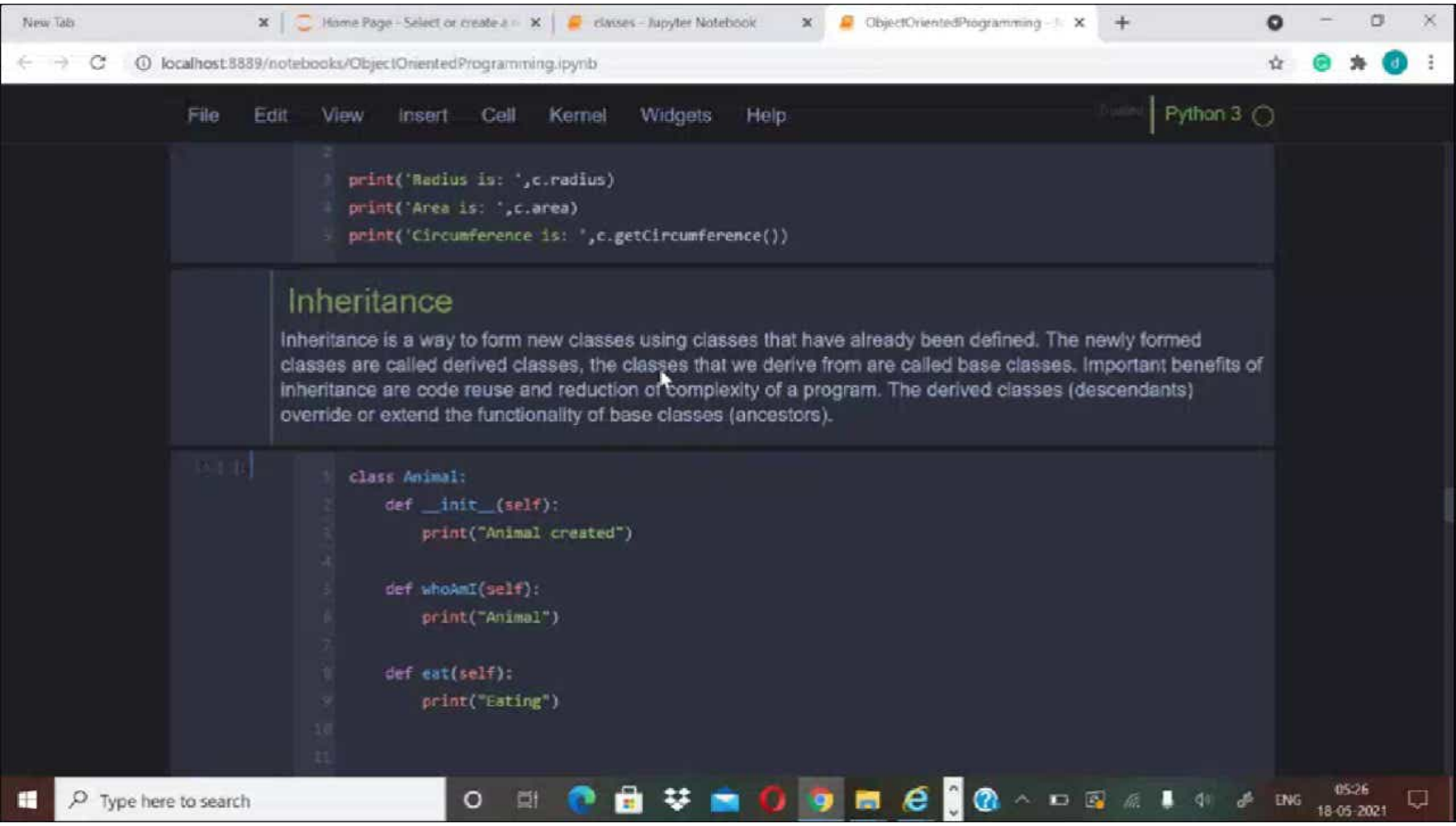

New Tab
Home Page - Select or create a
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
def __init__(self,type1 ,x1 , y1 , z1):
self.type = type1
self.x = x1
self.y = y1
self.z = z1
print("Object has been created")
daman = Bike(type1 = ' Road ' , x1 = 1 ,y1 = 3 ,z1 = 5)
print(daman.x)
pass
pow
print
property
%page
%pastebin
%pdb
%pdef
%pdoc
%%perl
Road
Note how we don't have any parentheses after breed; this is because it is an attribute and doesn't take any arguments.
Type here to search
ENG
05:12
18-05-2021

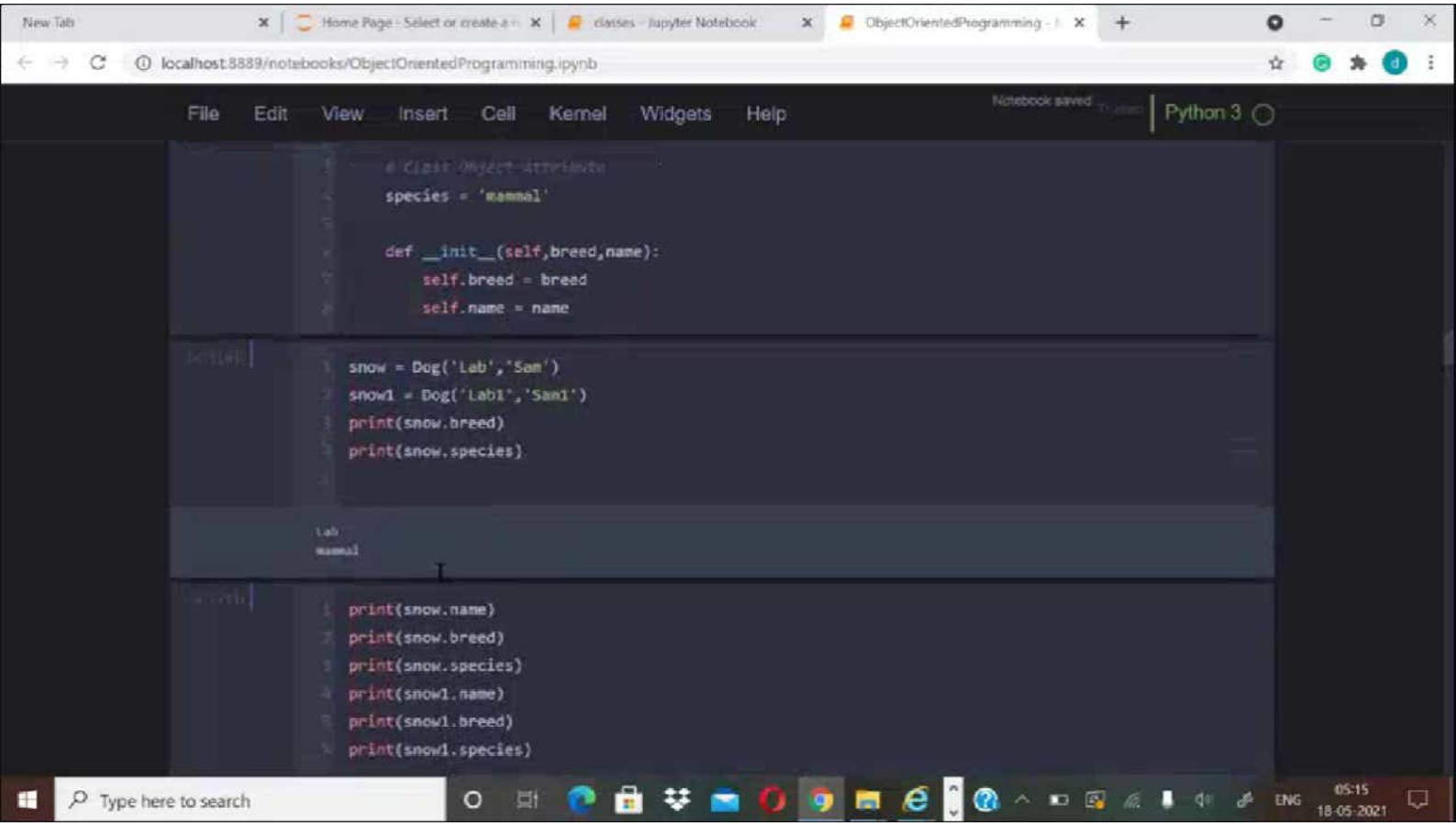

New Tab
Home Page - Select or create a n
classes - Jupyter Notebook
ObjectOrientedProgramming - J
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
Note how we don't have any parentheses after breed; this is because it is an attribute and doesn't take any arguments.
In Python there are also *class object attributes*. These Class Object Attributes are the same for any instance of the class. For example, we could create the attribute *species* for the Dog class. Dogs, regardless of their breed, name, or other attributes, will always be mammals. We apply this logic in the following manner:
class Dog:
# Class Object Attribute
species = 'mammal'
def __init__(self,breed,name):
self.breed = breed
self.name = name
snow = Dog('Lab','Sam')
snow1 = Dog('Lab1','Sam1')
print(snow.breed)
print(snow.species)
Type here to search
ENG
05:14
18-05-2021

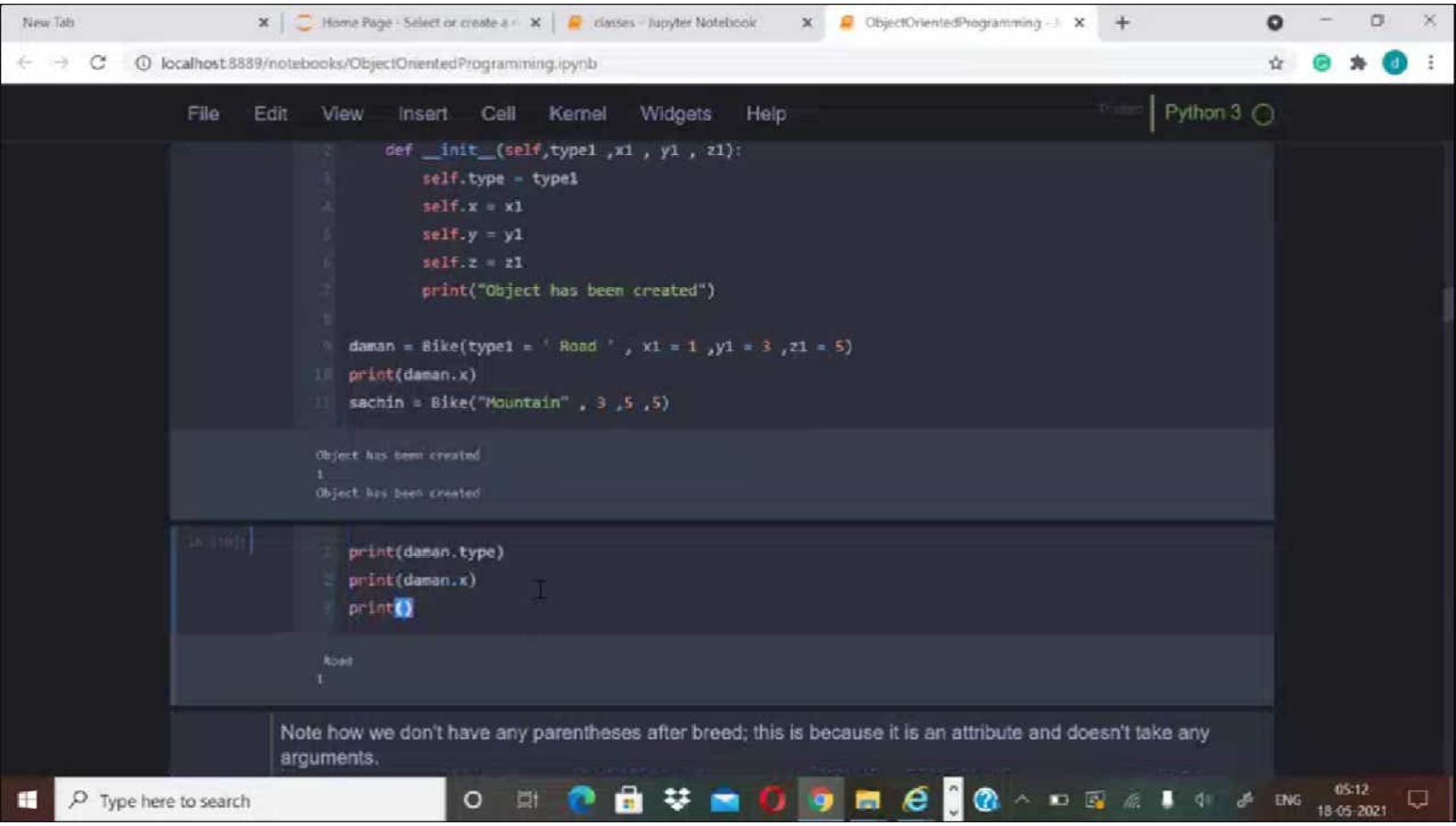

New Tab
Home Page - Select or create a
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
class Bike:
def __init__(self,type1 ,x1 , y1 , z1):
self.type = type1
self.x = x1
self.y = y1
self.z = z1
print("Object has been created")
daman = Bike(type1 = ' Road ' , x1 = 1 ,y1 = 3 ,z1 = 5)
print(daman.x)
sachin = Bike("Mountain" , 3 ,5 ,5)
Object has been created
1
Object has been created
print(daman.type)
print
' Road '
Note how we don't have any parentheses after breed; this is because it is an attribute and doesn't take any arguments.
Type here to search
ENG
05:12
18-05-2021

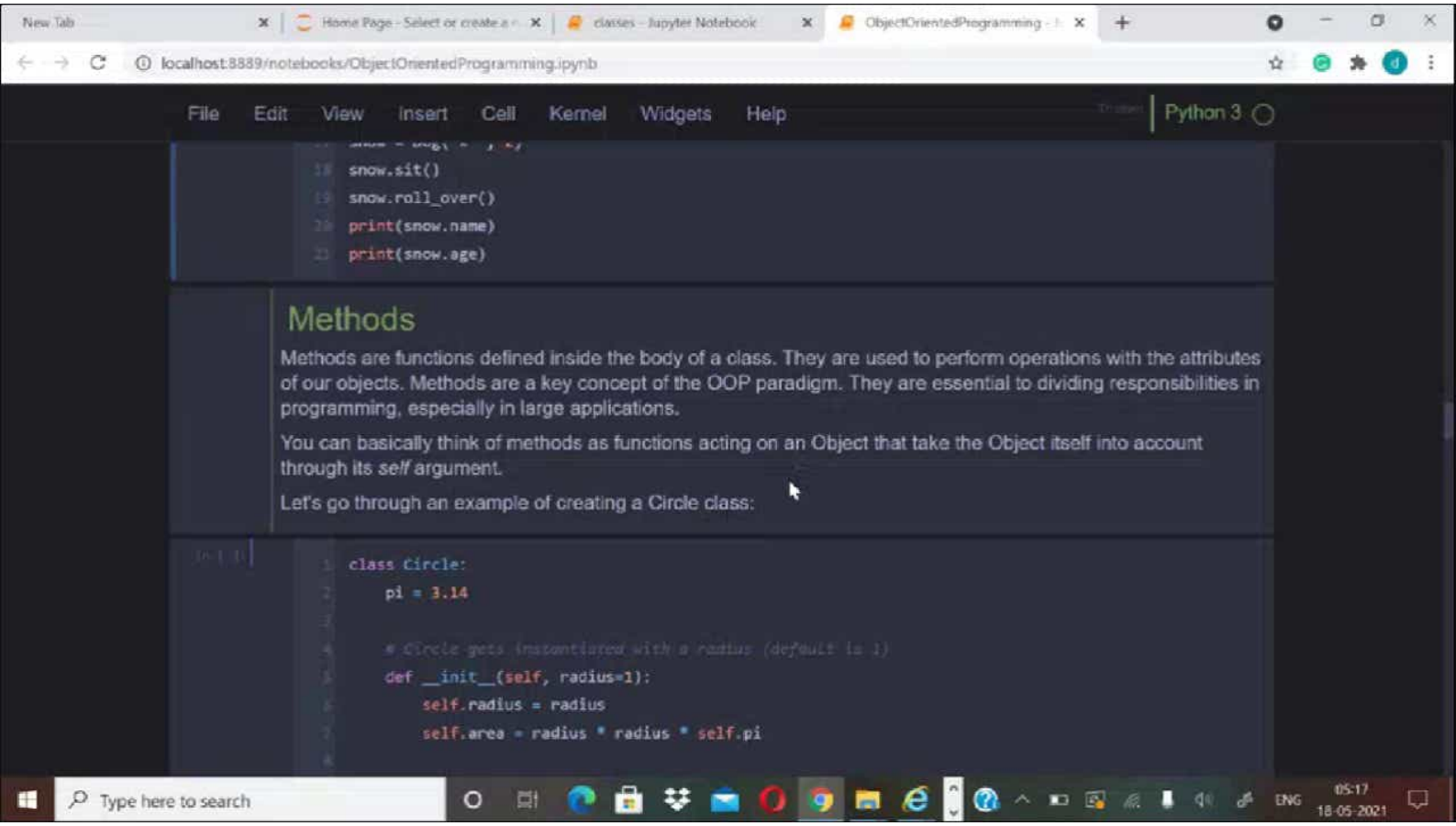

New Tab
Home Page - Select or create a
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
#Class Dog
#Objects : Snow , petter , leo
#Attributes : breed , age , height
#Methods : bark, sit , stand
<class '__main__.Sample'>
By convention we give classes a name that starts with a capital letter.
By x , we instantiate the Sample class.
Inside of the class we currently just have pass. But we can define class attributes and methods.
An attribute is a characteristic of an object. A method is an operation we can perform with the object.
For example, we can create a class called Dog. An attribute of a dog may be its breed or its name, while a method of a dog may be defined by a .bark() method which returns a sound.
Attributes
The syntax for creating an attribute is:
self.attribute = something
Type here to search
ENG
05:08
18-05-2021

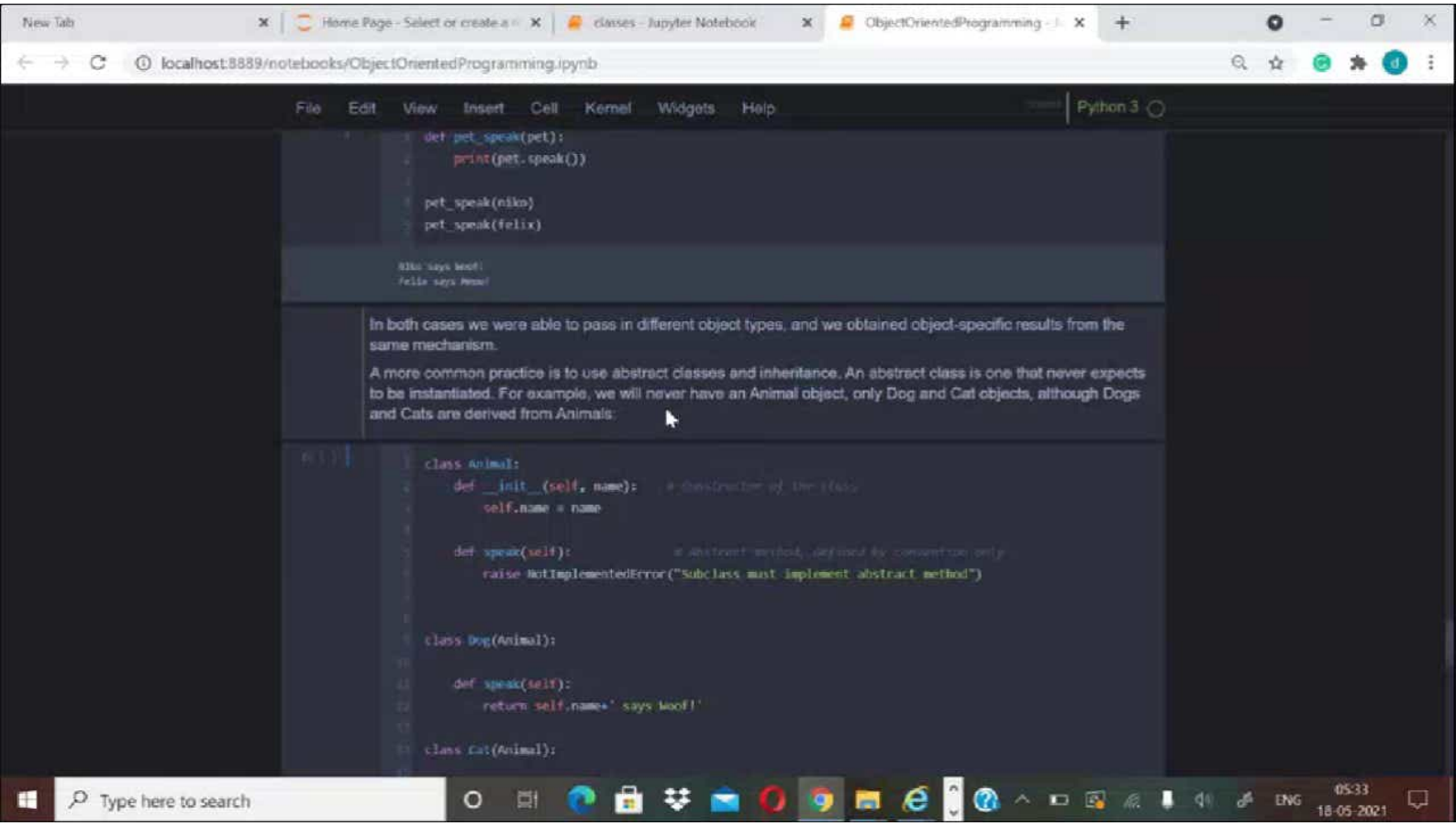

New Tab
Home Page - Select or create a n
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
# Object Oriented Programming
list1 = [10,20,30]
Remember how we could call methods on a list?
list1.count(20)
Objects
In Python, everything is an object.
print(type(10))
print(type("daman"))
print(type([]))
print(type(()))
Type here to search
ENG
05:00
18-05-2021

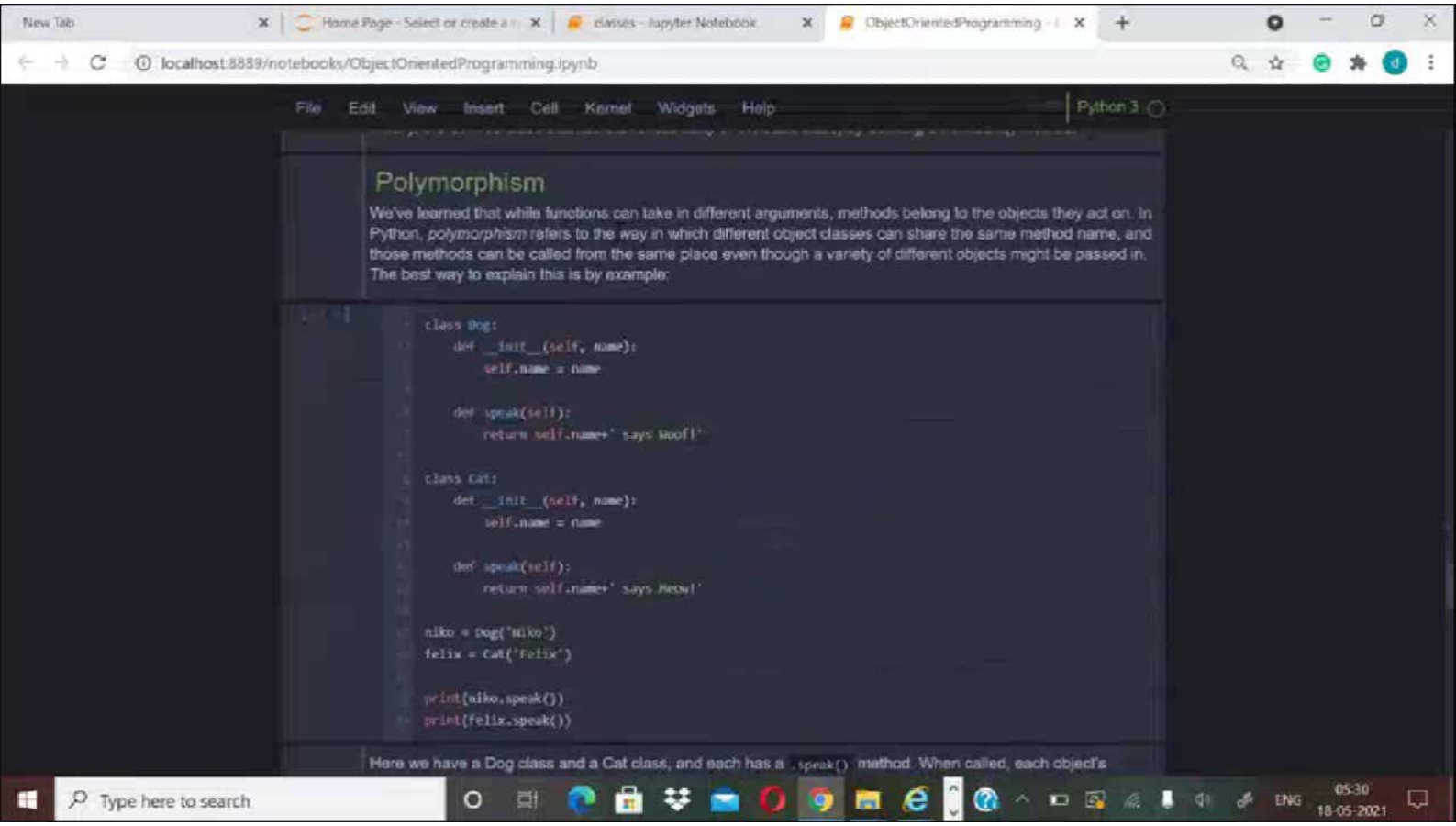

New Tab
Home Page - Select or create a n
classes - Jupyter Notebook
ObjectOrientedProgramming - J
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
print('Radius is: ',c.radius)
print('Area is: ',c.area)
print('Circumference is: ',c.getCircumference())
Inheritance
Inheritance is a way to form new classes using classes that have already been defined. The newly formed classes are called derived classes, the classes that we derive from are called base classes. Important benefits of inheritance are code reuse and reduction of complexity of a program. The derived classes (descendants) override or extend the functionality of base classes (ancestors).
class Animal:
def __init__(self):
print("Animal created")
def whoAmI(self):
print("Animal")
def eat(self):
print("Eating")
Type here to search
ENG
05:26
18-05-2021

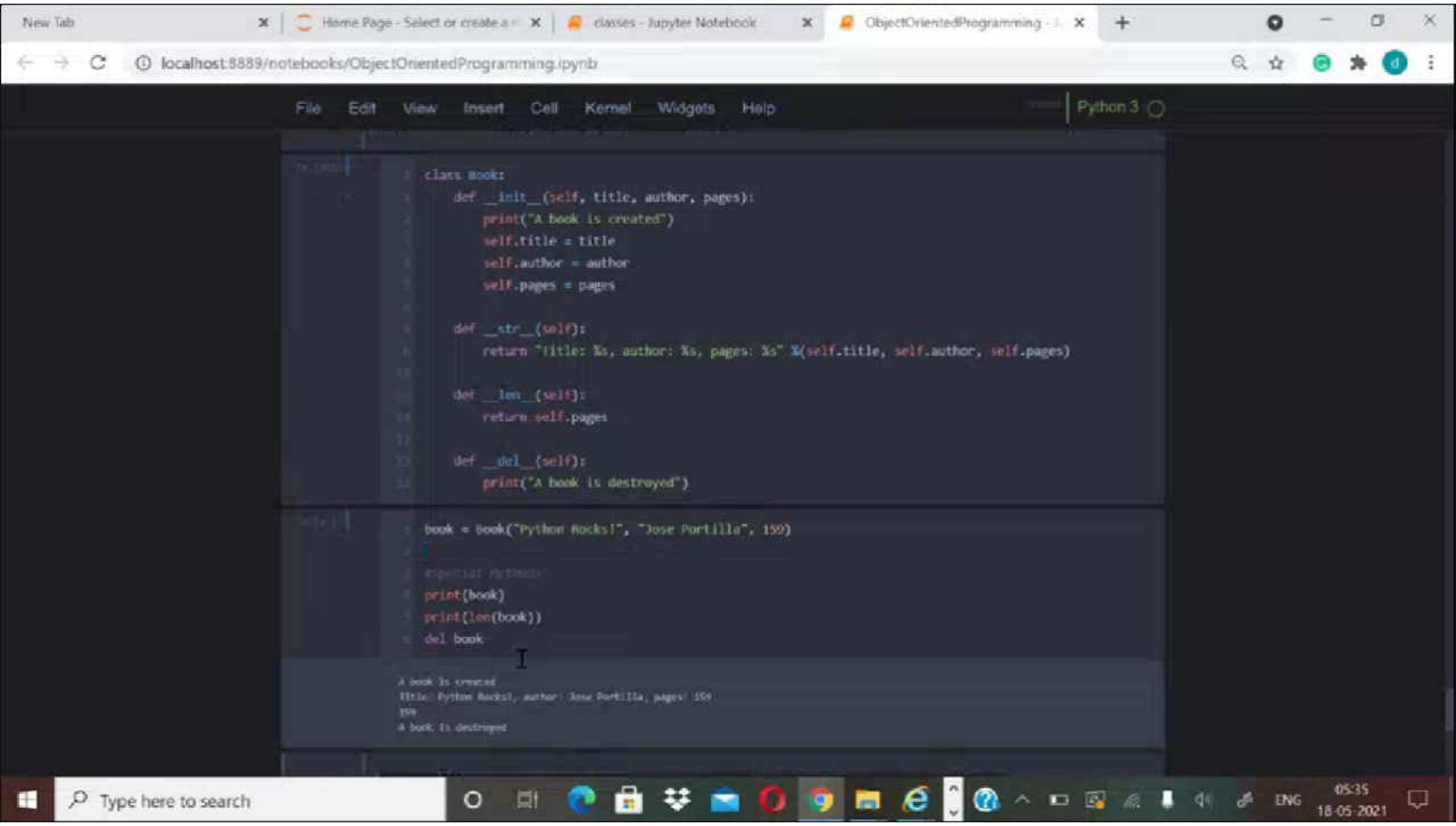

New Tab
Home Page - Select or create a n
classes - Jupyter Notebook
ObjectOrientedProgramming - J
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Notebook saved
Python 3
species = 'mammal'
def __init__(self,breed,name):
self.breed = breed
self.name = name
snow = Dog('Lab','Sam')
snow1 = Dog('Lab1','Sam1')
print(snow.breed)
print(snow.species)
Lab
mammal
print(snow.name)
print(snow.breed)
print(snow.species)
print(snow1.name)
print(snow1.breed)
print(snow1.species)
Type here to search
ENG
05:15
18-05-2021

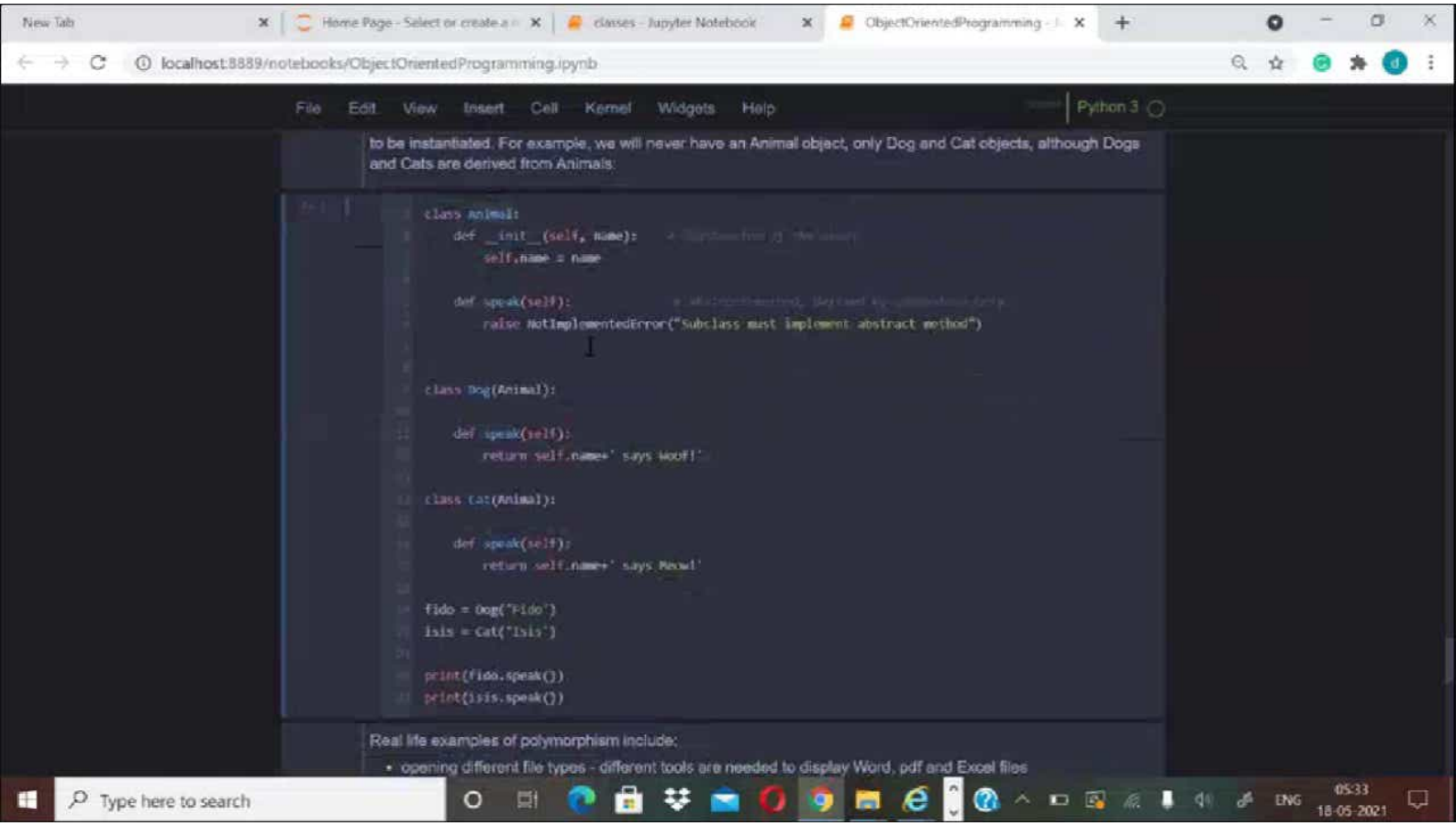

New Tab
Home Page - Select or create a
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
def __init__(self,type1 ,x1 , y1 , z1):
self.type = type1
self.x = x1
self.y = y1
self.z = z1
print("Object has been created")
daman = Bike(type1 = ' Road ' , x1 = 1 ,y1 = 3 ,z1 = 5)
print(daman.x)
sachin = Bike("Mountain" , 3 ,5 ,5)
Object has been created
1
Object has been created
print(daman.type)
print(daman.x)
print()
Road
1
Note how we don't have any parentheses after breed; this is because it is an attribute and doesn't take any arguments.
Type here to search
ENG
05:12
18-05-2021


New Tab
Home Page - Select or create a
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
snow.sit()
snow.roll_over()
print(snow.name)
print(snow.age)
Methods
Methods are functions defined inside the body of a class. They are used to perform operations with the attributes of our objects. Methods are a key concept of the OOP paradigm. They are essential to dividing responsibilities in programming, especially in large applications.
You can basically think of methods as functions acting on an Object that take the Object itself into account through its self argument.
Let's go through an example of creating a Circle class:
class Circle:
pi = 3.14
# Circle gets instantiated with a radius (default is 1)
def __init__(self, radius=1):
self.radius = radius
self.area = radius * radius * self.pi
Type here to search
ENG
05:17
18-05-2021


New Tab
Home Page - Select or create a
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
def pet_speak(pet):
print(pet.speak())
pet_speak(niko)
pet_speak(felix)
Niko says Woof!
Felix says Meow!
In both cases we were able to pass in different object types, and we obtained object-specific results from the same mechanism.
A more common practice is to use abstract classes and inheritance. An abstract class is one that never expects to be instantiated. For example, we will never have an Animal object, only Dog and Cat objects, although Dogs and Cats are derived from Animals.
class Animal:
def __init__(self, name):
self.name = name
def speak(self):
raise NotImplementedError("Subclass must implement abstract method")
class Dog(Animal):
def speak(self):
return self.name+' says Woof!'
class Cat(Animal):
Type here to search
ENG
05:33
18-05-2021


New Tab
Home Page - Select or create a n
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
Polymorphism
We've learned that while functions can take in different arguments, methods belong to the objects they act on. In Python, polymorphism refers to the way in which different object classes can share the same method name, and those methods can be called from the same place even though a variety of different objects might be passed in. The best way to explain this is by example:
class Dog:
def __init__(self, name):
self.name = name
def speak(self):
return self.name+' says Woof!'
class Cat:
def __init__(self, name):
self.name = name
def speak(self):
return self.name+' says Meow!'
niko = Dog('Niko')
felix = Cat('Felix')
print(niko.speak())
print(felix.speak())
Here we have a Dog class and a Cat class, and each has a .speak() method. When called, each object's
Type here to search
ENG
05:30
18-05-2021


New Tab
Home Page - Select or create a n
classes - Jupyter Notebook
ObjectOrientedProgramming - J
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
class Book:
def __init__(self, title, author, pages):
print("A book is created")
self.title = title
self.author = author
self.pages = pages
def __str__(self):
return "title: %s, author: %s, pages: %s" %(self.title, self.author, self.pages)
def __len__(self):
return self.pages
def __del__(self):
print("A book is destroyed")
book = Book("Python Rocks!", "Jose Portilla", 159)
print(book)
print(len(book))
del book
A book is created
title: Python Rocks!, author: Jose Portilla, pages: 159
159
A book is destroyed
Type here to search
ENG
05:35
18-05-2021


New Tab
Home Page - Select or create a
classes - Jupyter Notebook
ObjectOrientedProgramming
localhost:8889/notebooks/ObjectOrientedProgramming.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
to be instantiated. For example, we will never have an Animal object, only Dog and Cat objects, although Dogs and Cats are derived from Animals:
class Animal:
def __init__(self, name):
self.name = name
def speak(self):
raise NotImplementedError("Subclass must implement abstract method")
class Dog(Animal):
def speak(self):
return self.name+' says Woof!'
class Cat(Animal):
def speak(self):
return self.name+' says Meow!'
fido = Dog('Fido')
isis = Cat('Isis')
print(fido.speak())
print(isis.speak())
Real life examples of polymorphism include:
opening different file types - different tools are needed to display Word, pdf and Excel files
Type here to search
ENG
05:33
18-05-2021