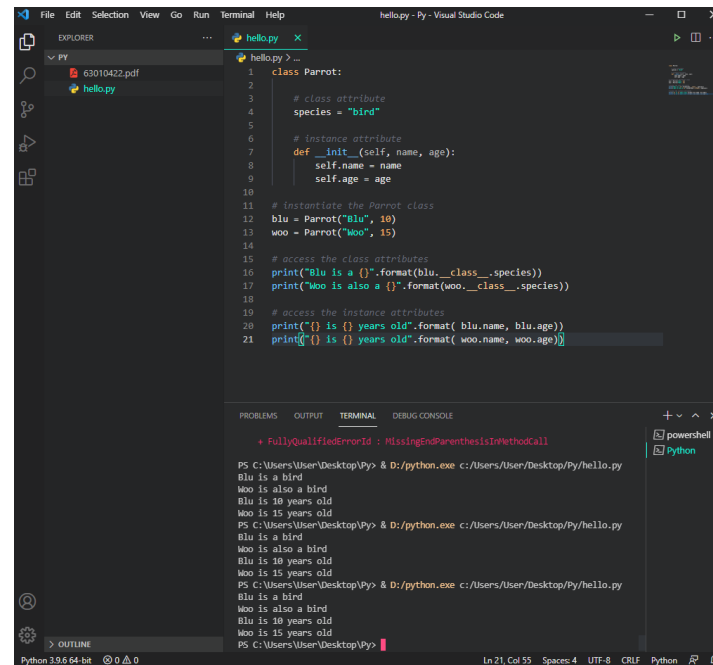


: Creating Class and Object in Python

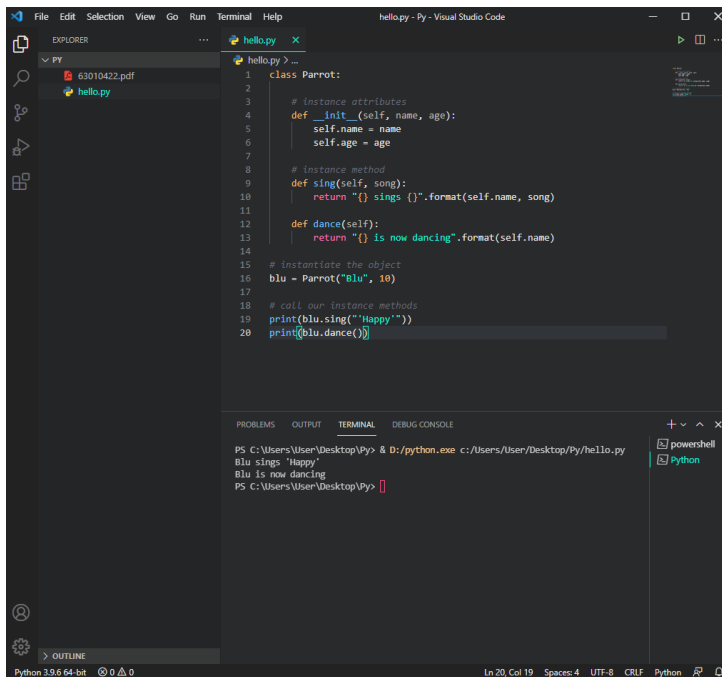


The screenshot shows a Visual Studio Code editor with a file named `hello.py`. The code defines a `Parrot` class with a class attribute `species = "bird"` and an `__init__` method that takes `name` and `age` as arguments. Two objects are instantiated: `blu = Parrot("blu", 10)` and `woo = Parrot("woo", 15)`. The code then prints the class and instance attributes for both objects. A syntax error is highlighted in the code: `print("{} is {} years old".format(woo.name, woo.age))` (line 21). The error message in the terminal is: `PS C:\Users\User\Desktop\Py> & D:/python.exe c:/Users/User/Desktop/Py/hello.py` followed by `Blu is a bird`, `Woo is also a bird`, `Blu is 10 years old`, `Woo is 15 years old`, `Blu is a bird`, `Woo is also a bird`, `Blu is 10 years old`, `Woo is 15 years old`, `Blu is a bird`, `Woo is also a bird`, `Blu is 10 years old`, `Woo is 15 years old`, `Blu is a bird`, `Woo is also a bird`, `Blu is 10 years old`, `Woo is 15 years old`.

```
1 class Parrot:
2
3     # class attribute
4     species = "bird"
5
6     # instance attribute
7     def __init__(self, name, age):
8         self.name = name
9         self.age = age
10
11 # instantiate the Parrot class
12 blu = Parrot("blu", 10)
13 woo = Parrot("woo", 15)
14
15 # access the class attributes
16 print("Blu is a {}".format(blu.__class__.species))
17 print("Woo is also a {}".format(woo.__class__.species))
18
19 # access the instance attributes
20 print("{} is {} years old".format( blu.name, blu.age))
21 print("{} is {} years old".format( woo.name, woo.age))
```

PS C:\Users\User\Desktop\Py> & D:/python.exe c:/Users/User/Desktop/Py/hello.py
Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old
PS C:\Users\User\Desktop\Py> & D:/python.exe c:/Users/User/Desktop/Py/hello.py
Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old
PS C:\Users\User\Desktop\Py> & D:/python.exe c:/Users/User/Desktop/Py/hello.py
Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old
PS C:\Users\User\Desktop\Py> & D:/python.exe c:/Users/User/Desktop/Py/hello.py
Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old
PS C:\Users\User\Desktop\Py>

: Creating Methods in Python

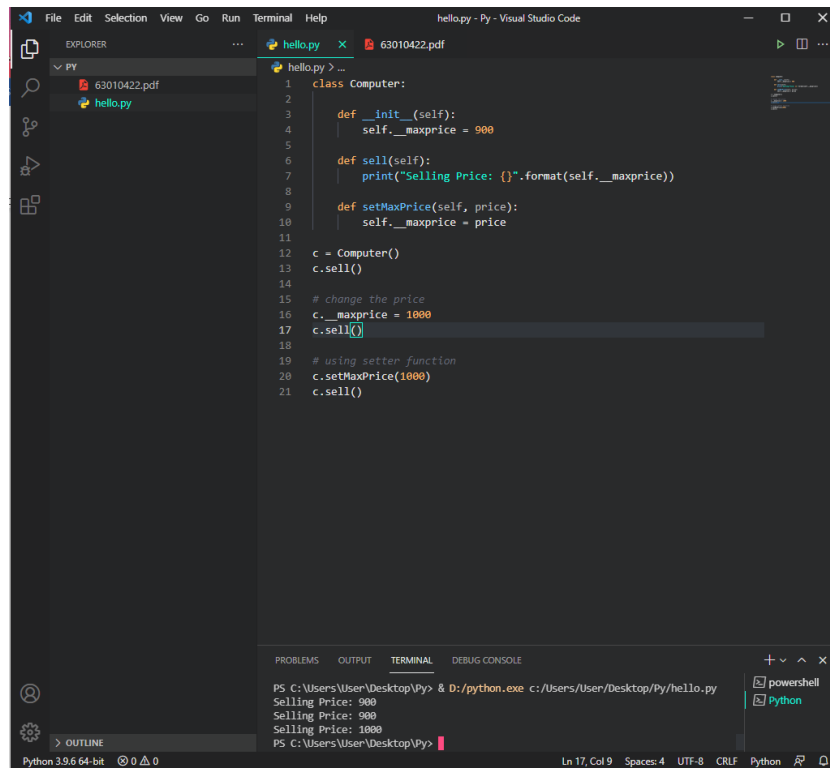


The screenshot shows a Visual Studio Code editor with a file named `hello.py`. The code defines a `Parrot` class with an `__init__` method and two instance methods: `sing` and `dance`. The `sing` method takes a `song` argument and returns a string. The `dance` method returns a string. Two objects are instantiated: `blu = Parrot("blu", 10)`. The code then calls the `sing` and `dance` methods on the `blu` object. The terminal output shows the results of the method calls.

```
1 class Parrot:
2
3     # instance attributes
4     def __init__(self, name, age):
5         self.name = name
6         self.age = age
7
8     # instance method
9     def sing(self, song):
10         return "{} sings {}".format(self.name, song)
11
12     def dance(self):
13         return "{} is now dancing".format(self.name)
14
15 # instantiate the object
16 blu = Parrot("blu", 10)
17
18 # call our instance methods
19 print(blu.sing("Happy"))
20 print(blu.dance())
```

PS C:\Users\User\Desktop\Py> & D:/python.exe c:/Users/User/Desktop/Py/hello.py
Blu sings "Happy"
Blu is now dancing
PS C:\Users\User\Desktop\Py>

: Data Encapsulation in Python



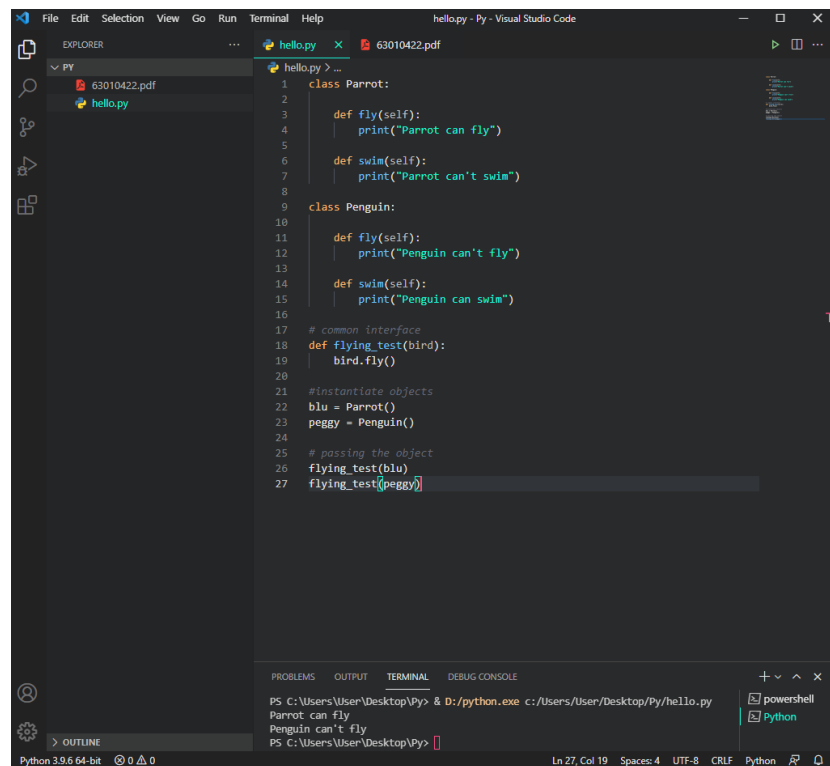
The screenshot shows a Visual Studio Code editor with a Python file named `hello.py`. The code defines a `Computer` class with an `__init__` method that sets `__maxprice` to 900, a `sell` method that prints the selling price, and a `setMaxPrice` method that updates the price. The script creates an instance `c` of the `Computer` class, calls `c.sell()`, changes the price using `c.__maxprice = 1000` and `c.sell()`, and then uses the `setMaxPrice` method to set the price to 1000 and calls `c.sell()` again. The terminal output shows the execution results.

```
1 class Computer:
2
3     def __init__(self):
4         self.__maxprice = 900
5
6     def sell(self):
7         print("Selling Price: {}".format(self.__maxprice))
8
9     def setMaxPrice(self, price):
10        self.__maxprice = price
11
12 c = Computer()
13 c.sell()
14
15 # change the price
16 c.__maxprice = 1000
17 c.sell()
18
19 # using setter function
20 c.setMaxPrice(1000)
21 c.sell()
```

Terminal Output:

```
PS C:\Users\User\Desktop\Py> & D:/python.exe c:/Users/User/Desktop/Py/hello.py
Selling Price: 900
Selling Price: 900
Selling Price: 1000
PS C:\Users\User\Desktop\Py>
```

: Using Polymorphism in Python



The screenshot shows a Visual Studio Code editor with a Python file named `hello.py`. The code defines two classes, `Parrot` and `Penguin`, each with `fly` and `swim` methods. A common interface `flying_test` is defined, which calls the `fly` method of the passed object. The script instantiates objects `blu` (Parrot) and `peggy` (Penguin) and calls `flying_test` on both. The terminal output shows the execution results.

```
1 class Parrot:
2
3     def fly(self):
4         print("Parrot can fly")
5
6     def swim(self):
7         print("Parrot can't swim")
8
9 class Penguin:
10
11     def fly(self):
12         print("Penguin can't fly")
13
14     def swim(self):
15         print("Penguin can swim")
16
17 # common interface
18 def flying_test(bird):
19     bird.fly()
20
21 # instantiate objects
22 blu = Parrot()
23 peggy = Penguin()
24
25 # passing the object
26 flying_test(blu)
27 flying_test(peggy)
```

Terminal Output:

```
PS C:\Users\User\Desktop\Py> & D:/python.exe c:/Users/User/Desktop/Py/hello.py
Parrot can fly
Penguin can't fly
PS C:\Users\User\Desktop\Py>
```

