

CLASSES

Numbers
Strings
Booleans

Lists
Dictionaries|

```
point.get_distance|
```



```
class Point:  
    def move(self):  
        print("move")  
  
    def draw(self):  
        print("draw")  
  
point1 = Point()
```

```
point1 = Point()
point1.|  
    ⚪ draw(self)           Point  
    ⚪ move(self)          Point  
    ⚫ __annotations__     object  
    ⚫ __class__            object  
    ⚪ __delattr__(self... object
```

```
class Point:  
    def move(self):  
        print("move")  
  
    def draw(self):  
        print("draw")
```

```
point1 = Point()  
point1.draw()
```

```
point1 = Point()  
point1.x = 10  
point1.y = 20  
print(point1.x)  
point1.draw()
```

```
point2 = Point()  
print(point2.x)
```

```
    print(point2.x)  
AttributeError: 'Point' object has no attribute 'x'
```

```
Process finished with exit code 1
```

CONSTRUCTORS

```
point2 = Point()  
print(point2.x)
```

```
pp x  
    print(point2.x)  
AttributeError: 'Point' object has no attribute 'x'
```

Process finished with exit code 1

```
class Point:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
  
    def move(self):  
        print("move")  
  
    def draw(self):  
        print("draw")
```



```
point = Point(10, 20) I
print(point.x)
```

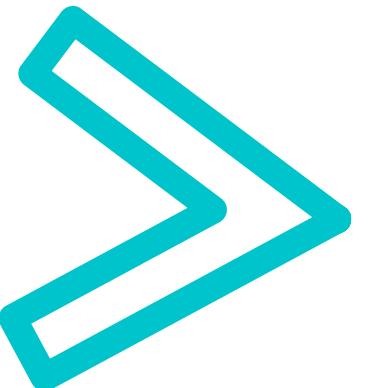
Person

- name
- talk()

Solution

```
class Person:  
    def __init__(self, name):  
        self.name = name  
  
    def talk(self):  
        print("talk")  
  
john = Person("John Smith")  
print(john.name)  
john.talk()
```

```
bob = Person("Bob Smith")  
bob.talk()
```



```
Hi, I am Bob Smith
```

```
I
```

```
Process finished with exit code 0
```

INHERITANCE

```
class Dog:  
    def walk(self):  
        print("walk")
```

```
class Dog:  
    def walk(self):  
        print("walk")
```

```
class Cat:  
    def walk(self):  
        print("walk")
```



```
class Mammal:  
    def walk(self):  
        print("walk")  
  
class Dog(Mammal):  
    pass  
  
class Cat:  
    def walk(self):  
        print("walk")
```

```
class Dog(Mammal):  
    pass
```

```
class Cat(Mammal):  
    pass
```

```
dog1 = Dog()  
dog1.|
```



A screenshot of a Python code editor showing a code completion dropdown. The code above has created a Dog object named dog1. A cursor is at the end of 'dog1.' and a dropdown menu is open. It contains two items: 'walk(self)' and '__annotations__'. To the right of the items, the type 'Mammal' is shown above the word 'object'.

- walk(self) Mammal
- __annotations__ object

```
class Dog(Mammal):  
    def bark(self):  
        print("bark")
```

```
class Cat(Mammal):  
    pass
```

```
dog1 = Dog()
```

```
dog1.walk()
```

• walk(self)

Mammal

• bark(self)

Dog

```
class Cat(Mammal):
    def be_annoying(self):
        print("annoying")
```

```
cat1 = Cat()
```

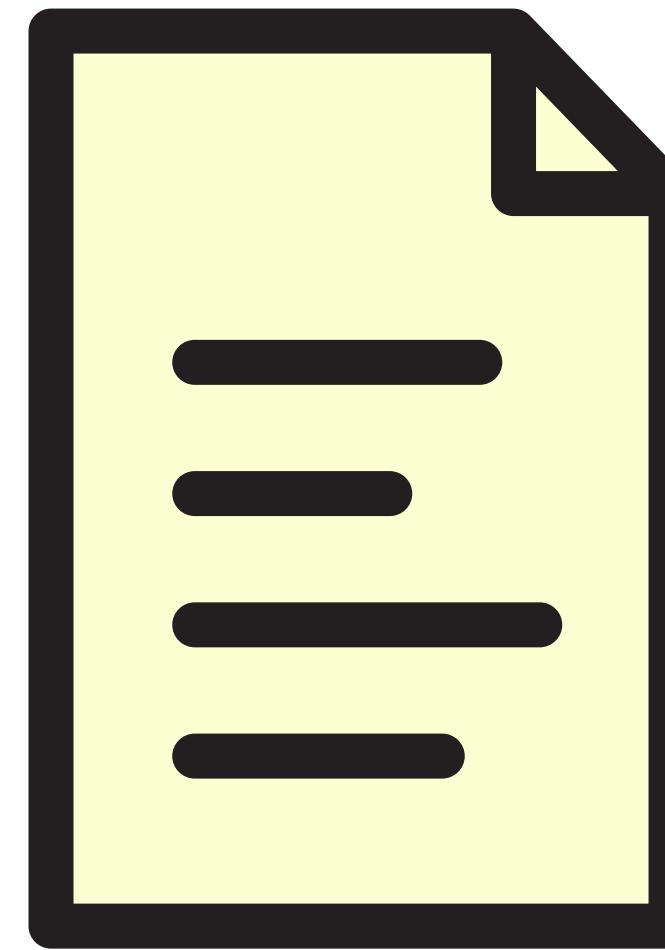
```
cat1.|
```

• walk(self)	Mammal
• be_annoying(self)	Cat
• __annotations__	object
• class	object

MODULES



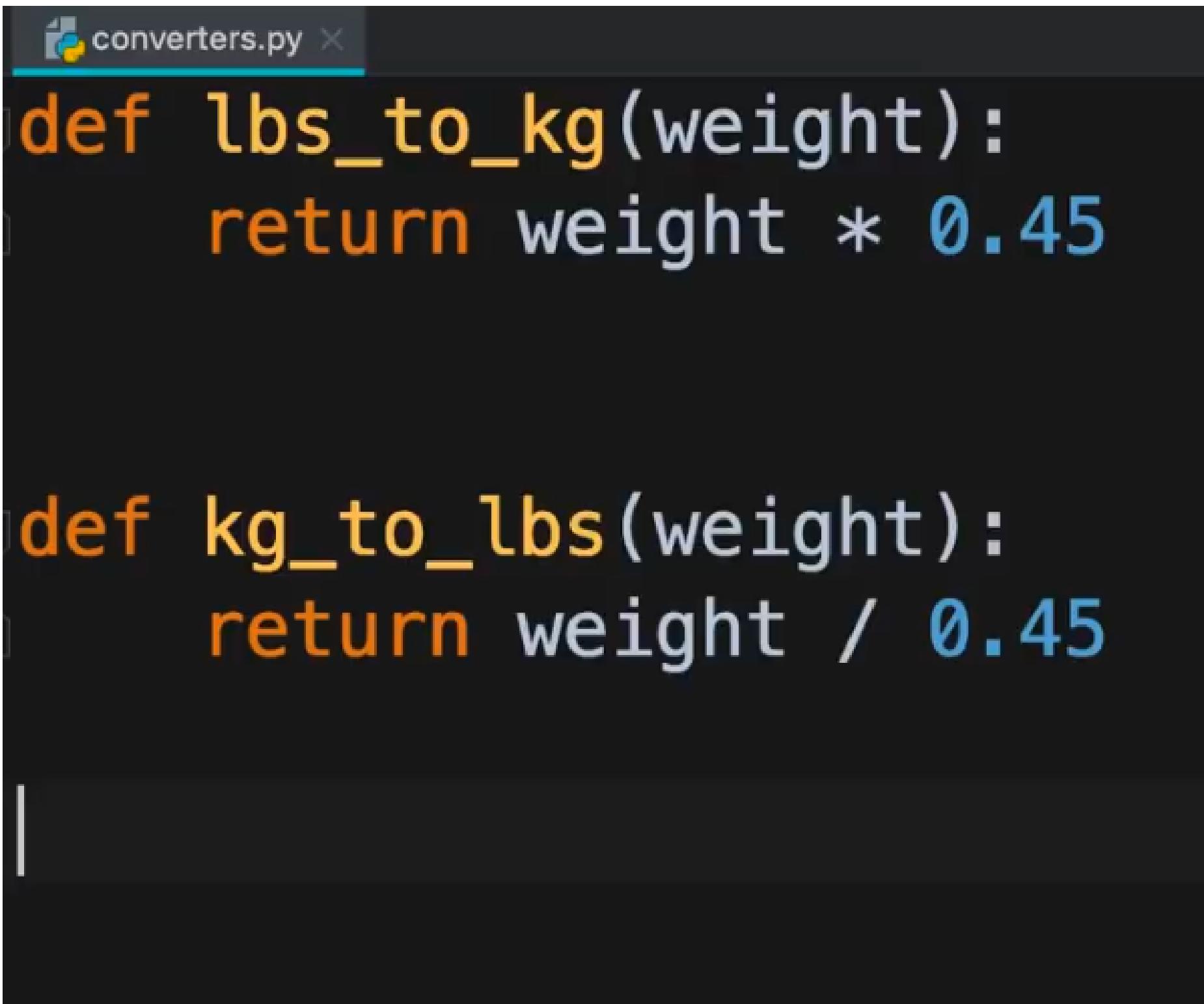
app.py



converter.py



Both app.py and converter.py are separate modules



```
converters.py x
def lbs_to_kg(weight):
    return weight * 0.45

def kg_to_lbs(weight):
    return weight / 0.45
```

```
import converters  
print(converters.kg_to_lbs(70))
```

A screenshot of a code editor window titled "converters.py". The code shown is:

```
import converters
from converters import_
```

The word "from" is highlighted in orange. A completion dropdown menu is open at the end of the line, showing two suggestions: "kg_to_lbs" and "lbs_to_kg". The "kg_to_lbs" option is highlighted with a blue background. A tooltip at the bottom of the dropdown says "Did you know that Quick Definition View (��Sp...)".

A screenshot of a code editor window titled "converters.py". The code shown is:

```
import converters
from converters import kg_to_lbs
```

The line "from converters import kg_to_lbs" is partially completed. The cursor is positioned after "kg_to_lbs", and the text "kg_to_lbs(100)" is displayed in a green box, indicating it is a suggested completion. Below this, another line of code is partially visible: "print(converters.kg_to_lbs(70))".

Modules

A module is a file with some Python code. We use modules to break up our program into multiple files. This way, our code will be better organized. We won't have one gigantic file with a million lines of code in it!

There are 2 ways to import modules: we can import the entire module, or specific objects in a module.

Modules

A module is a file with some Python code. We use modules to break up our program into multiple files. This way, our code will be better organized. We won't have one gigantic file with a million lines of code in it!

There are 2 ways to import modules: we can import the entire module, or specific objects in a module.

EXERCISE

```
numbers = [10, 3, 6, 2]
max = numbers[0]
for number in numbers:
    if number > max:
        max = number
print(max)
```

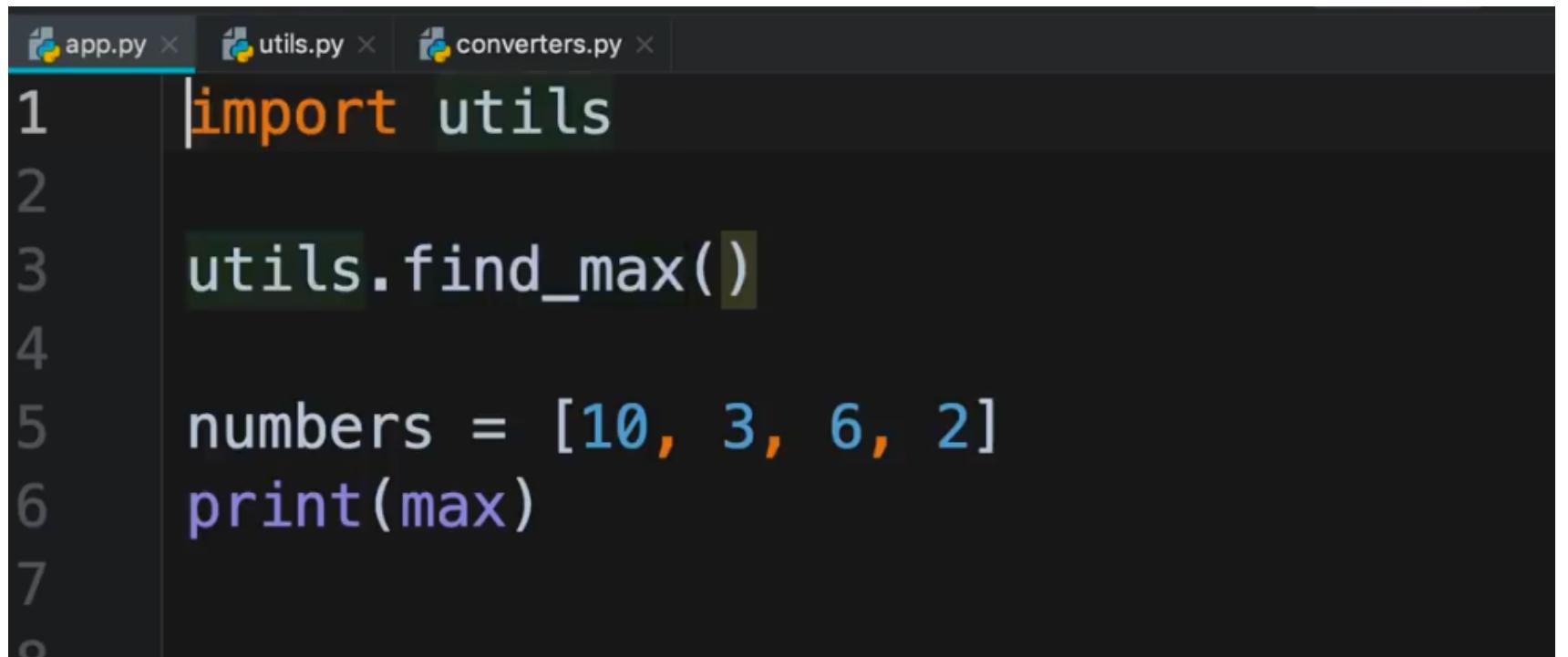


```
find_max()
utils
```

Solution

The screenshot shows a code editor interface with two tabs at the top: "utils.py" and "converters.py". The "utils.py" tab is active, displaying the following Python code:

```
def find_max(numbers):
    max = numbers[0]
    for number in numbers:
        if number > max:
            max = number
    return max
```



```
app.py × utils.py × converters.py ×
1 import utils
2
3     utils.find_max()
4
5 numbers = [10, 3, 6, 2]
6 print(max)
7
8
```

```
from utils import find_max

numbers = [10, 3, 6, 2]
max = find_max(numbers)
|
print(max)
```

The screenshot shows a Python code editor with three tabs at the top: `app.py`, `utils.py`, and `converters.py`. The `app.py` tab is active. The code in the editor is:

```
1 from utils import find_max
2
3 numbers = [10, 3, 6, 2]
4 max = find_max(numbers)
```

A yellow squiggly underline is underlined under the variable `max` in line 4. A tooltip window is displayed below the line, containing the text: "Shadows built-in name 'max' more... (⌘F1)". Lines 6 and 7 are partially visible at the bottom of the editor.

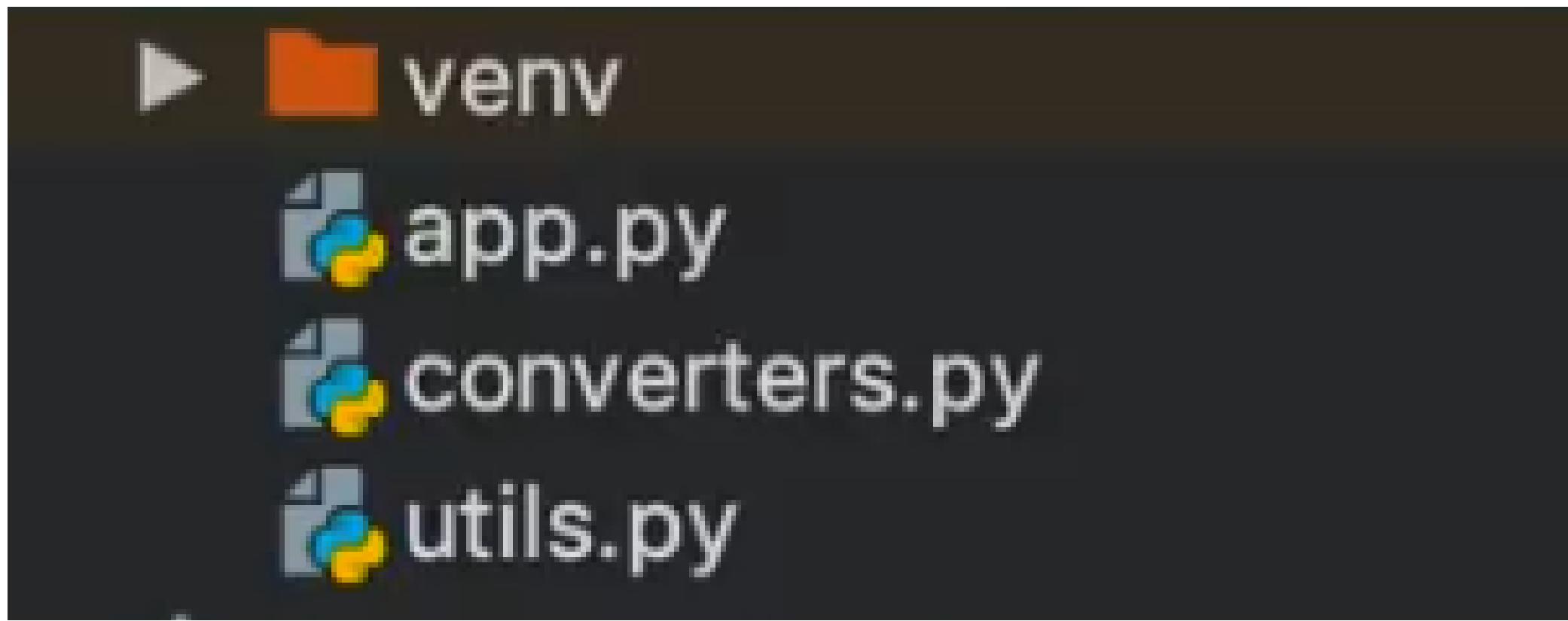
```
from utils import find_max

numbers = [10, 3, 6, 2]
# max = find_max(numbers)
print(max)
```

```
from utils import find_max

numbers = [10, 3, 6, 2]
# max = find_max(numbers)
print(max(numbers))
```

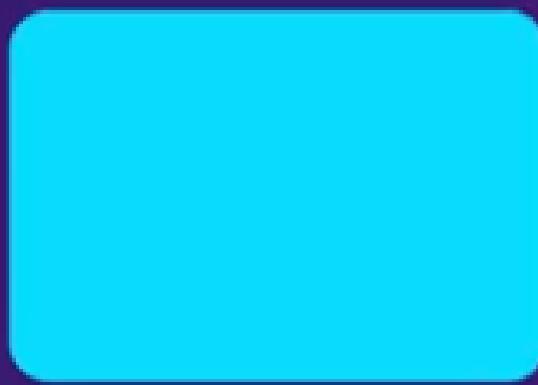
PACKAGES



Men's



Women's



Kids'



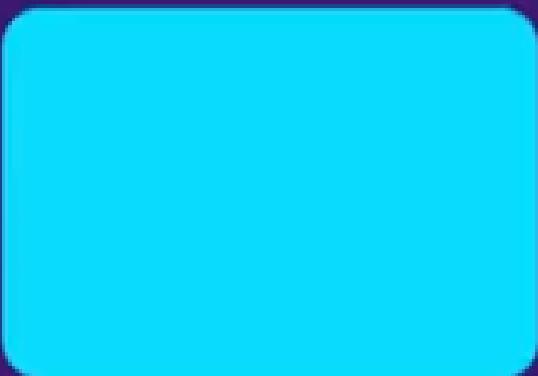
PACKAGE

Men's

Women's

Kids'

PACKAGE

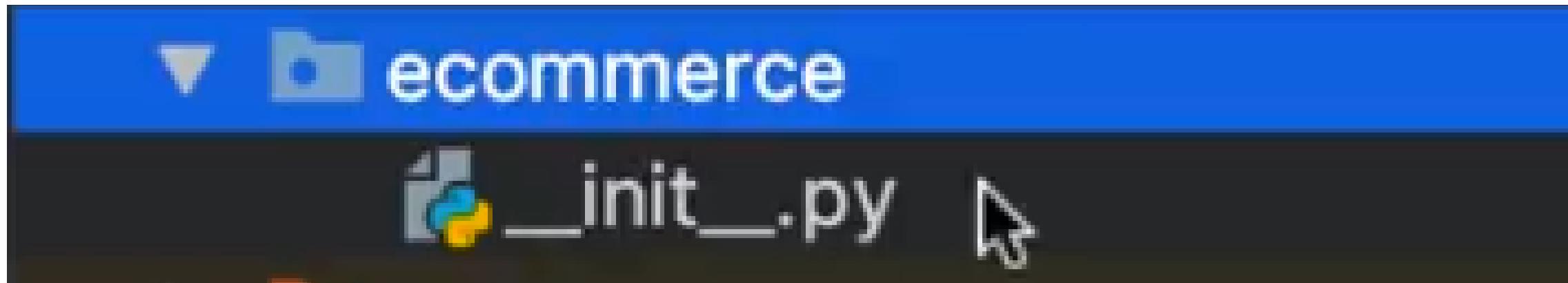
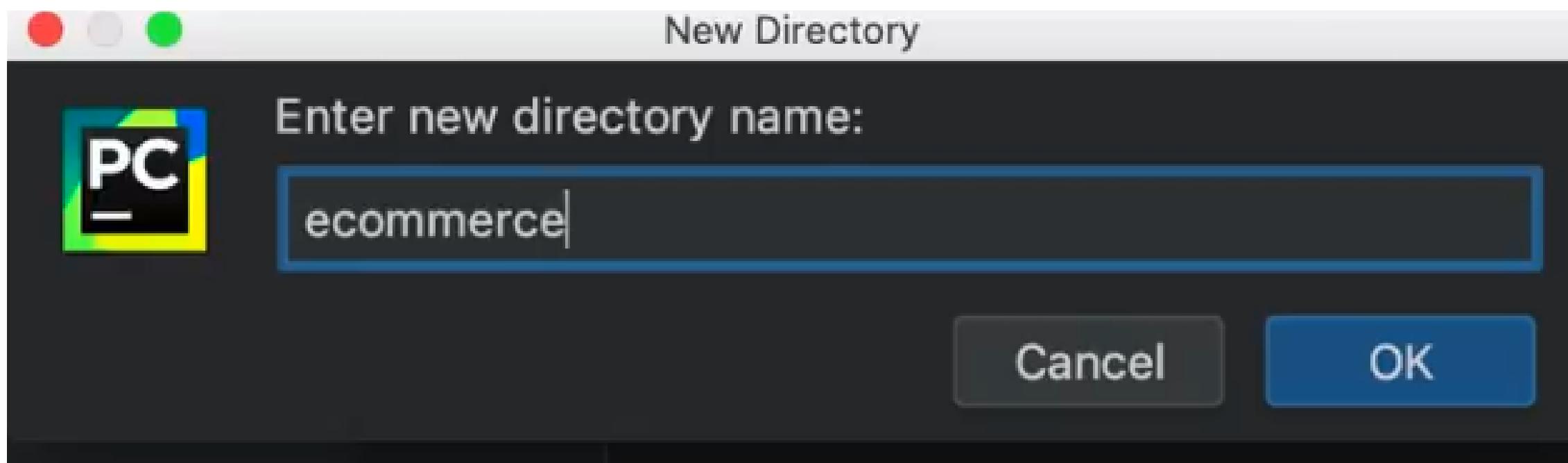


Shoes

T-shirts

Jackets

MODULE



> ecommerce > shipping.py

shipping.py ×

```
def calc_shipping():
    print("calc_shipping")
```

The screenshot shows the PyCharm IDE interface. The top navigation bar has tabs for 'app.py' and 'shipping.py'. The main editor window contains the following Python code:

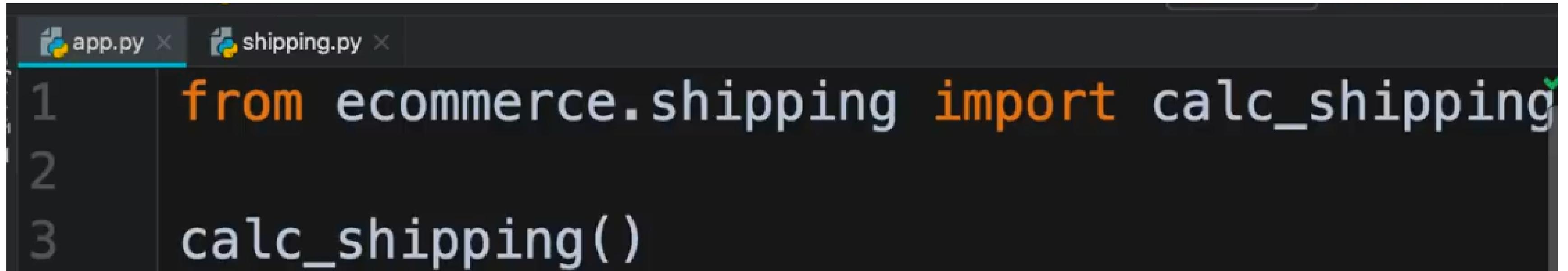
```
1 import ecommerce.shipping
2 ecommerce.shipping.calc_shipping()
3 |
```

The code is highlighted with syntax coloring. A yellow dot is positioned under the word 'calc_shipping()' in the second line. The bottom panel is the 'Run' tool window, which displays the output of the run:

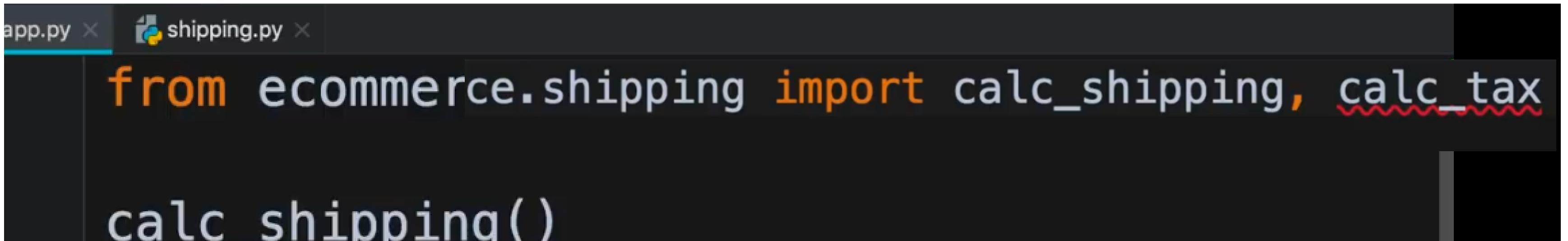
Run: app

```
/Users/moshteghamedani/PycharmProjects/Hell
calc_shipping
```

Process finished with exit code 0



```
app.py x shipping.py x
1 from ecommerce.shipping import calc_shipping
2
3 calc_shipping()
```



```
app.py x shipping.py x
from ecommerce.shipping import calc_shipping, calc_tax
calc_shipping()
```

```
app.py × shipping.py ×
1 from ecommerce import shipping
2
3 shipping.calc_shipping()
```

Packages

A package is a directory with `__init__.py` in it. It can contain one or more modules.

```
# importing the entire sales module
from ecommerce import sales
sales.calc_shipping()

# importing one function in the sales module
from ecommerce.sales import calc_shipping
calc_shipping()
```

GENERATING RANDOM VARIABLES

Python Standard Library

Python comes with a huge library of modules for performing common tasks such as sending emails, working with date/time, generating random values, etc.

Random Module

```
import random

random.random()          # returns a float between 0 to 1
random.randint(1, 6)    # returns an int between 1 to 6

members = ['John', 'Bob', 'Mary']
leader = random.choice(members) # randomly picks an item
```

Challenge

(3, 1)

(3, 6)

(5, 6)

Dice
- roll()

Solution

```
import random

class Dice:
    def roll(self):
        first = random.randint(1, 6)
        second = random.randint(1, 6)
        return (first, second)
```

```
import random

class Dice:
    def roll(self):
        first = random.randint(1, 6)
        second = random.randint(1, 6)
        return first, second

dice = Dice()
print(dice.roll())
|
```

WORKING WITH DIRECTORIES

PYPI AND PIP

THANK YOU!