```
(base) PS C:\Users\Abi Rahman> python
Python 3.10.9 | packaged by conda-forge | (main, Jan 11 2023, 15:15:40) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> #9. Classes
>>> #9.2.1. Scopes and Namespaces Example
>>> def scope_test():
...     def do_local():
...         spam = "local spam"
...     def do_nonlocal():
...         nonlocal spam
...         spam = "nonlocal spam"
...     def do_global():
...         global spam
...         spam = "global spam"
...     spam = "test spam"
...     do_local()
...     print("After local assignment:", spam)
...     do_nonlocal()
...     print("After nonlocal assignment:", spam)
...     do_global()
...     print("After global assignment:", spam)
...
>>> scope_test()
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
>>> print("In global scope:", spam)
In global scope: global spam
>>>
>>>
```

```
>>>
>>> #9.3.2. Class Objects
>>> class MyClass:
...     """A simple example class"""
...     i = 12345
...     def f(self):
...         return 'hello world'
...
>>> x = MyClass()
>>> def __init__(self):
...     self.data = []
...
>>> x = MyClass()
>>>
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
>>>
>>>
>>> #9.3.3. Instance Objects
>>> x.counter = 1
>>> while x.counter < 10:
...     x.counter = x.counter * 2
...
>>> print(x.counter)
```

```
>>> del x.counter
>>>
>>>
>>> #9.3.5. Class and Instance Variables
>>> class Dog:
...     kind = 'canine'         # class variable shared by all instances
...     def __init__(self, name):
...         self.name = name    # instance variable unique to each instance
...
>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind
'canine'
>>> e.kind
'canine'
>>> d.name
'Fido'
>>> e.name
'Buddy'
>>>
>>>
>>> class Dog:
...     tricks = []             # mistaken use of a class variable
...     def __init__(self, name):
...         self.name = name
...     def add_trick(self, trick):
...         self.tricks.append(trick)
...
>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
```

```
>>> e.add_trick('play dead')
>>> d.tricks
['roll over', 'play dead']
>>>
>>>
>>> class Dog:
...     def __init__(self, name):
...         self.name = name
...         self.tricks = []    # creates a new empty list for each dog
...     def add_trick(self, trick):
...         self.tricks.append(trick)
...
>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play dead']
>>>
>>>
>>> #9.4. Random Remarks
>>> class Warehouse:
...     purpose = 'storage'
...     region = 'west'
...
>>> w1 = Warehouse()
>>> print(w1.purpose, w1.region)
storage west
>>> w2 = Warehouse()
```

```
>>> w2.region = 'east'
>>> print(w2.purpose, w2.region)
storage east
>>>
>>>
>>> # Function defined outside the class
>>> def f1(self, x, y):
...     return min(x, x+y)
...
>>> class C:
...     f = f1
...     def g(self):
...         return 'hello world'
...     h = g
...
>>>
>>> class Bag:
...     def __init__(self):
...         self.data = []
...     def add(self, x):
...         self.data.append(x)
...     def addtwice(self, x):
...         self.add(x)
...         self.add(x)
...
>>>
>>>
>>>#9.6. Private Variables
>>> class Mapping:
...     def __init__(self, iterable):
...          self.items_list = []
```

```
...         self.__update(iterable)
...     def update(self, iterable):
...         for item in iterable:
...             self.items_list.append(item)
...     __update = update   # private copy of original update() method
...
>>> class MappingSubclass(Mapping):
...     def update(self, keys, values):
...         # provides new signature for update()
...         # but does not break __init__()
...         for item in zip(keys, values):
...             self.items_list.append(item)
...
>>>
>>>
>>> #9.7. Odds and Ends
>>> from dataclasses import dataclass
>>> @dataclass
... class Employee:
...     name: str
...     dept: str
...     salary: int
...
>>> john = Employee('john', 'computer lab', 1000)
>>> john.dept
'computer lab'
>>> john.salary
1000
>>>
>>>
>>> #9.8. Iterators
```

```
>>> for element in [1, 2, 3]:
...     print(element)
...
1
2
3
>>> for element in (1, 2, 3):
...     print(element)
...
1
2
3
>>> for key in {'one':1, 'two':2}:
...     print(key)
...
one
two
>>> for char in "123":
...     print(char)
...
1
2
3
>>> for line in open("myfile.txt"):
...     print(line, end='')
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'myfile.txt'
>>>
>>> s = 'abc'
```

```
>>> it = iter(s)
>>> it
<str_iterator object at 0x00000229B1B387C0>
>>> next(it)
'a'
>>> next(it)
'b'
>>> next(it)
'c'
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
>>>
>>> class Reverse:
...     """Iterator for looping over a sequence backwards."""
...     def __init__(self, data):
...         self.data = data
...         self.index = len(data)
...     def __iter__(self):
...         return self
...     def __next__(self):
...         if self.index == 0:
...             raise StopIteration
...         self.index = self.index - 1
...         return self.data[self.index]
...
>>> rev = Reverse('spam')
>>> iter(rev)
<__main__.Reverse object at 0x00000229B1AB8D00>
```

```
>>> for char in rev:
...     print(char)
...
m
a
p
s
>>>
>>>
>>> #9.9. Generators
>>> def reverse(data):
...     for index in range(len(data)-1, -1, -1):
...         yield data[index]
...
>>> for char in reverse('golf'):
...     print(char)
...
f
l
o
g
>>>
>>> #9.10. Generator Expressions
>>> sum(i*i for i in range(10))                 # sum of squares
285
>>> xvec = [10, 20, 30]
>>> yvec = [7, 5, 3]
>>> sum(x*y for x,y in zip(xvec, yvec))         # dot product
260
>>> unique_words = set(word for line in page  for word in line.split())
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
NameError: name 'page' is not defined. Did you mean: 'range'?
>>> valedictorian = max((student.gpa, student.name) for student in graduates)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'graduates' is not defined
>>> data = 'golf'
>>> list(data[i] for i in range(len(data)-1, -1, -1))
['f', 'l', 'o', 'g']
>>>
```