**Object-Oriented Programming**

* Introduction

10.1 Q1: Which of the following statements is false?

a. Everything in Python is an object.

b. Just as houses are built from blueprints, classes are built from objects—one of the core technologies of object-oriented programming.

c. Building a new object from even a large class is simple—you typically write one statement.

d. All of the above statements are true.

Answer: b. Just as houses are built from blueprints, classes are built from objects—one of the core technologies of object-oriented programming. Actually, objects are built from classes.

10.1 Q2: Which of the following statements is false?

a. You’ll use lots of classes created by other people.

b. You can create your own custom classes.

c. The core technologies of object-oriented programming are classes, objects, inheritance and polymorphism.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.1 Q3: Which of the following statements is false?

a. The vast majority of object-oriented programming you’ll do in Python is object-based programming in which you primarily use objects of new custom classes you create.

b. To take maximum advantage of Python you must familiarize yourself with lots of preexisting classes.

c. Over the years, the Python open-source community has crafted an enormous number of valuable classes and packaged them into class libraries, available on the Internet at sites like GitHub, BitBucket, SourceForge and more. This makes it easy for you to reuse existing classes rather than “reinventing the wheel.”

d. Widely used open-source library classes are more likely to be thoroughly tested, bug free, performance tuned and portable across a wide range of devices, operating systems and Python versions.

Answer: a. The vast majority of object-oriented programming you’ll do in Python is object-based programming in which you primarily use objects of new custom classes you create. Actually, the vast majority of object-oriented programming you’ll do in Python is object-based programming in which you primarily create and use objects of *existing* classes.

10.1 Q4: Which of the following statements is false?

a. Classes are new function types.

b. Most applications you’ll build for your own use will commonly use either no custom classes or just a few.

c. You can contribute your custom classes to the Python open-source community, but you are not obligated to do so. Organizations often have policies and procedures related to open-sourcing code.

d. All of the above statements are true.

Answer: a. Classes are new function types. Actually, classes are new *data* types.

10.1 Q5: Which of the following statements is false?

a. New classes can be formed quickly through inheritance and composition from classes in abundant class libraries.

b. Eventually, software will be constructed predominantly from standardized, reusable components, just as hardware is constructed from interchangeable parts today. This will help meet the challenges of developing ever more powerful software.

c. When creating a new class, instead of writing all new code, you can designate that the new class is to be formed initially by inheriting the attributes and methods of a previously defined base class (also called a subclass) and the new class is called a derived class (or superclass).

d. After inheriting, you then customize the derived class to meet the specific needs of your application. To minimize the customization effort, you should always try to inherit from the base class that’s closest to your needs.

Answer: c. When creating a new class, instead of writing all new code, you can designate that the new class is to be formed initially by inheriting the attributes and methods of a previously defined base class (also called a subclass) and the new class is called a derived class (or superclass). Actually, with inheritance, the previously defined base class is also called a *superclass* and the new class is called a derived class or *subclass*.

10.1 Q6: Which of the following statements is false?

a. Polymorphism enables you to conveniently program “in the general” rather than “in the specific.”

b. With polymorphism, you simply send the same method call to objects possibly of many different types. Each object responds by “doing the right thing.” So the same method call takes on many forms, hence the term “poly-morphism.”

c. In Python, as in other major object-oriented programming languages, you can implement polymorphism only via inheritance.

d. All of the above statements are true.

Answer: c. In Python, as in other major object-oriented programming languages, you can implement polymorphism only via inheritance. Actually, in Python, you can implement polymorphism via inheritance *and through duck typing*.

* Custom Class **Account**

No questions.

* Test-Driving Class Account

10.2 Q1: Each new class you create becomes a new data type that can be used to create objects. This is one reason why Python is said to be a(n) \_\_\_\_\_\_\_\_ language.

a. dynamic typing

b. comprehensive

c. extensible

d. None of the above

Answer: c. extensible

10.2 Q2: To create a Decimal object, we can write:

value = Decimal('3.14159')

This is known as a(n) \_\_\_\_\_\_\_\_ expression because it builds and initializes an object of the class.

a. builder

b. maker

c. assembler

d. constructor

Answer: d. constructor

10.2 Q3 Which of the following statements about code snippets that use class Account is false?

a. The following code uses a constructor expression to create an Account object and initialize it with an account holder’s name (a string) and balance (a Decimal):

account1 = Account('John Green', Decimal('50.00'))

b. The following expressions accesse an Account object’s name and balance attributes:

account1.name

account1.balance

c. The following snippets deposit an amount into an Account and access the new balance:

account1.deposit(Decimal('25.53'))

account1.balance

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

* Account Class Definition

10.2 Q4: Which of the following statements is false?

a. A class definition begins with the keyword class followed by the class’s name and a colon (:). This line is called the class header.

b. The Style Guide for Python Code recommends that you begin each word in a multi-word class name with an uppercase letter (for example, CommissionEmployee).

c. Every statement in a class’s suite is indented.

d. Each class must provide a descriptive docstring in the line or lines immediately following the class header. To view any class’s docstring in IPython, type the class name and a question mark, then press Enter.

Answer: d. Each class must provide a descriptive docstring in the line or lines immediately following the class header. To view any class’s docstring in IPython, type the class name and a question mark, then press *Enter*. Actually, each class typically provides a descriptive docstring, but it is **optional**.

10.2 Q5: Which of the following statements is false?

a. The following constructor expression creates a new object, then initializes its data by calling the class’s \_\_init\_\_ method:

account1 = Account('John Green', Decimal('50.00'))

b. Each new class you create can provide an \_\_init\_\_ method that specifies how to initialize an object’s data attributes.

c. Returning a value other than Null from \_\_init\_\_ results in a TypeError. Null is returned by any function or method that does not contain a return statement.

d. Class Account’s \_\_init\_\_ method below initializes an Account object’s name and balance attributes if the balance is valid:

def \_\_init\_\_(self, name, balance):

"""Initialize an Account object."""

# if balance is less than 0.00, raise an exception

if balance < Decimal('0.00'):

raise ValueError('Initial balance must be >= to 0.00.')

self.name = name

self.balance = balance

Answer: c. Returning a value other than Null from \_\_init\_\_ results in a TypeError. Null is returned by any function or method that does not contain a return statement.

Actually, returning a value other than *None* from \_\_init\_\_ results in a TypeError. *None* is returned by any function or method that does not contain a return statement.

10.2 Q6 Which of the following statements is false?

a. When you call a method for a specific object, Python implicitly passes a reference to that object as the method’s first argument, so all methods of a class must specify at least one parameter.

b. All methods must have a first parameter self—a class’s methods must use that reference to access the object’s attributes and other methods.

c. When an object of a class is created, it does not yet have any attributes. They’re added dynamically via assignments, typically of the form self.attribute\_name = value.

d. All of the above statements are true.

Answer: b. All methods must have a first parameter self—a class’s methods must use that reference to access the object’s attributes and other methods. Actually, *by convention* most Python programmers call a method’s first parameter self, but this is not required.

10.2 Q7: Python class \_\_\_\_\_\_\_\_ defines the special methods that are available for all Python objects.

a. object

b. special

c. class

d. root

Answer: a. object

* Composition: Object References as Members of Classes

10.2 Q8: An object’s attributes are references to objects of other classes. Embedding references to objects of other classes is a form of software reusability known as \_\_\_\_\_\_\_\_ and is sometimes referred to as the \_\_\_\_\_\_\_\_ relationship.

a. composition, "is a"

b. inheritance, "has a"

c. composition, "has a"

d. inheritance, "is a"

Answer: c. composition, "has a"

* Controlling Access to Attributes

10.3 Q1: Which of the following statements is false?

a. Methods commonly use a class’s attributes to get the values of those attributes.

b. We also can use a class’s attributes to modify their values.

c. Consider the Account object in the following IPython session:

In [1]: from account import Account

In [2]: from decimal import Decimal

In [3]: account1 = Account('John Green', Decimal('50.00'))

In [4]: account1.balance

Out[4]: Decimal('50.00')

Initially, account1 contains a valid balance. The following code shows that we can set the balance attribute to an invalid negative value, then display the balance:

In [5]: account1.balance = Decimal('-1000.00')

In [6]: account1.balance

Out[6]: Decimal('-1000.00')

d. Like methods, data attributes can validate the values you assign to them.

Answer: d. Like methods, data attributes can validate the values you assign to them. Actually, unlike methods, data attributes **cannot** validate the values you assign to them.

10.3 Q2: Most object-oriented programming languages enable you to encapsulate (or hide) an object’s data from the client code. Such data in these languages is said to be \_\_\_\_\_\_\_\_ data.

a. public

b. protected

c. private

d. None of the above

Answer: c. private

10.3 Q3: Which of the following statements is false?

a. Python does not have private data. Instead, you use naming conventions to design classes that encourage correct use.

b. An attribute name beginning with an underscore (\_) is never accessible by clients of a class.

c. Client code should use the class’s methods and properties to interact with each object’s internal-use data attributes.

d. All of the above statements are true.

Answer: b. An attribute name beginning with an underscore (\_) is never accessible by clients of a class. Actually, by convention, Python programmers know that any attribute name beginning with an underscore (\_) is for a class’s internal use only. Even when we use this convention, attributes are always accessible.

* Properties for Data Access

10.4 Q1: Properties look like \_\_\_\_\_\_\_\_ to client-code programmers, but control the manner in which they get and modify an object’s data.

a. function calls

b. method calls

c. data attributes

d. None of the above

Answer: c. data attributes

* Test-Driving Class Time

10.4 Q2: Assume the following sets hour, minute and second to zero.

In [2]: wake\_up = Time(hour=6, minute=30)

This means that:

a. If at least two arguments have default arguments, the third automatically defaults to zero.

b. Any omitted argument is automatically set to zero.

c. second has a default argument of zero.

d. None of the above.

Answer: c. second is a default argument that defaults to zero.

10.4 Q3: When you evaluate a variable in IPython it calls the object’s \_\_\_\_\_\_\_\_ special method to produce a string representation of the object.

a. \_\_init\_\_

b. \_\_str\_\_

c. \_\_string\_\_

d. \_\_repr\_\_

Answer: d. \_\_repr\_\_

10.4 Q4: b. Properties are implemented as \_\_\_\_\_\_\_\_, so they may contain additional logic, such as specifying the format in which to return a data attribute’s value or validating a new value before using it to modify a data attribute.

a. functions

b. suites

c. classes

d. methods

Answer: d. methods

10.4 Q5: Which of the following statements is false?

a. A class can also support setting its attribute values individually via its properties. The following statement changes an hour property value to 6:

wake\_up.hour = 6

b. Though the statement snippet in Part (a) appears to simply assign a value to a data attribute, it’s actually a call to an hour method that takes 6 as an argument.

c. The hour method mentioned in Part (b) can validate the value, then assign it to a corresponding data attribute (which we might name \_hour, for example).

d. Each of the above statements is true.

Answer: d. Each of the above statements is *true*.

10.4 Q6: The following code and traceback shows that class Time’s hour property validates the values you assign to it:

In [10]: wake\_up.hour = 100

---------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-10-1fce0716ef14> in <module>()

----> 1 wake\_up.hour = 100

~/Documents/examples/ch10/timewithproperties.py in hour(self, hour)

20 """Set the hour."""

21 if not (0 <= hour < 24):

---> 22 raise ValueError(f'Hour ({hour}) must be 0-23')

23

24 self.\_hour = hour

ValueError: Hour (100) must be 0-23

Which of the following statements is false?

a. The code attempts to set the hour property to an invalid value.

b. The code checks that the hour property is in the range 0, 1, 2, ... 22, 23, 24.

c. The value 100 is out of range so the code raises a ValueError.

d. All of the above statements are true.

Answer: b. The code checks that the hour property is in the range 0, 1, 2, ... 22, 23, 24. Actually, the code checks that the hour property is in the range 0, 1, 2, ... 21, 22, 23.

* Class Time Definition

10.4 Q7: Consider the following class Time \_\_init\_\_ method:

def \_\_init\_\_(self, hour=0, minute=0, second=0):

"""Initialize each attribute."""

self.hour = hour # 0-23

self.minute = minute # 0-59

self.second = second # 0-59

Which of the following statements is false?

a. Class Time’s \_\_init\_\_ method specifies hour, minute and second parameters, each with a default argument of 0.

b. The self parameter is a reference to the Time object being initialized.

c. The statements containing self.hour, self.minute and self.second appear to create hour, minute and second attributes for the new Time object (self). However, these statements may actually call methods that implement the class’s hour, minute and second properties.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.4 Q8: Consider code from class Time below:

@property

def hour(self):

"""Return the hour."""

return self.\_hour

@hour.setter

def hour(self, hour):

"""Set the hour."""

if not (0 <= hour < 24):

raise ValueError(f'Hour ({hour}) must be 0-23')

self.\_hour = hour

10.4 Q8: Consider code from class Time below:

a. This code defines a read-write property named hour that manipulates a data attribute named \_hour.

b. The single-leading-underscore (\_) naming convention indicates that client code can safely access \_hour directly.

c. Properties look like data attributes to programmers working with objects. Properties are implemented as methods.

d. All of the above statements are true.

Answer: b. The single-leading-underscore (\_) naming convention indicates that client code can safely access \_hour directly. Actually, the single-leading-underscore (\_) naming convention indicates that client code should *not* access \_hour directly.

10.4 Q9: Which of the following statements is false?

a. Each property defines a getter method which gets (that is, returns) a data attribute’s value and can optionally define a setter method which sets a data attribute’s value.

b. The @property decorator precedes a property’s getter method, which receives only a self parameter.

c. Behind the scenes, a decorator adds code to the decorated function—in this case to make the hour function work with attribute syntax.

d. All of the above statements are true.

Answer: d. All of the above statements are **true**.

10.4 Q10: A read-only property has \_\_\_\_\_\_\_\_.

a. only a setter

b. only a getter

c. a setter and a getter

d. neither a setter nor a getter

Answer: b. only a getter

10.4 Q11: Which of the following statements is false?

a. When you pass an object to built-in function repr—which happens implicitly when you evaluate a variable in an IPython session—the corresponding class’s \_\_repr\_\_ special method is called to get a string representation of the object. The Python documentation indicates that \_\_repr\_\_ returns the “official” string representation of the object.

b. Typically the string returned by \_\_repr\_\_ looks like a constructor expression that creates and initializes the object, as in:

'Time(hour=6, minute=30, second=0)'

c. Python has a built-in function eval that could receive the preceding string as an argument and use it to create and initialize a Time object containing values specified in the string.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.4 Q12: The \_\_\_\_\_\_\_\_ special method is called implicitly when you convert an object to a string with the built-in function str, such as when you print an object or call str explicitly.

a. \_\_repr\_\_

b. \_\_string\_\_

c. \_\_str\_\_

d. None of the above

Answer: c. \_\_str\_\_

* Class Time Definition Design Notes

10.4 Q13: Which of the following statements is false?

a. Class Time’s properties and methods define the class’s \_\_\_\_\_\_\_\_ interface, that is, the set of properties and methods programmers should use to interact with objects of the class.

a. public

b. private

c. protected

d. None of the above

Answer: a. public

10.4 Q14: Which of the following statements is false?

a. When you design a class, carefully consider the class’s interface before making that class available to other programmers.

b. Unfortunately, existing code will break if you update the class’s implementation details—that is, the internal data representation or how its method bodies are implemented.

c. If Python programmers follow convention and do not access attributes that begin with leading underscores, then class designers can evolve class implementation details without breaking client code.

d. All of the above statements are true.

Answer: b. Unfortunately, existing code will break if you update the class’s implementation details—that is, the internal data representation or how its method bodies are implemented. Actually, ideally, you’ll design a class’s interface such that existing code *will not break* if you update the class’s implementation details—that is, the internal data representation or how its method bodies are implemented.

10.4 Q15: Which of the following statements is false?

a. It may seem that providing properties with both setters and getters has no benefit over accessing the data attributes directly, but there are subtle differences.

b. A getter seems to allow clients to read the data at will, but the getter can control the formatting of the data.

c. A setter can scrutinize attempts to modify the value of a data attribute to prevent the data from being set to an invalid value.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.4 Q16: Which of the following statements about a class’s utility methods is false?

a. They serve as part of a class’s interface.

b. They are used inside the class.

c. They should be named with a single leading underscore.

d. In other object-oriented languages like C++, Java and C#, utility methods typically are implemented as private methods.

Answer: a. They serve as part of a class’s interface. Actually, they do not serve as part of a class’s interface—they’re meant for internal use only.

10.4 Q17: In professional Python development, rather than building your own classes to represent times and dates, you’ll typically use the Python Standard Library’s \_\_\_\_\_\_\_\_ module capabilities.

a. date

b. calendar

c. time

d. datetime

Answer: d. datetime

* Simulating “Private” Attributes

which would set the hour to an invalid value. Which of the following statements is false?

wake\_up.\_hour = 100

which would set the hour to an invalid value. Which of the following statements is false?

a. Rather than \_hour, we can name the attribute \_\_hour with two leading underscores. This convention indicates that \_\_hour is “private” and should not be accessible to the class’s clients.

b. To help prevent clients from accessing “private” attributes, Python renames them by preceding the attribute name with \_ClassName, as in \_Time\_\_hour. This is called name mangling.

c. If you try to assign to \_\_hour, as in

wake\_up.\_\_hour = 100

Python raises a ValueError, indicating that the class does not have an \_\_hour attribute.

d. IPython does not show attributes with one or two leading underscores when you try to auto-complete an expression like

wake\_up.

by pressing Tab. Only attributes that are part of the wake\_up object’s “public” interface are displayed in the IPython auto-completion list.

Answer: c. If you try to assign to \_\_hour, as in

wake\_up.\_\_hour = 100

Python raises a *ValueError*, indicating that the class does not have an \_\_hour attribute. Actually, if you try to assign to \_\_hour, as in

wake\_up.\_\_hour = 100

Python raises an AttributeError, indicating that the class does not have an \_\_hour attribute.

* Case Study: Card Shuffling and Dealing Simulation

No questions.

* Test-Driving Classes Card and DeckOfCards

No questions.

* Class Card—Introducing Class Attributes

10.6 Q1: Which of the following statements is false?

a. Sometimes, an attribute should be shared by all objects of a class. A class attribute (also called a class variable) represents class-wide information—it belongs to the class, not to a specific object of that class.

b. You define a class attribute by assigning a value to it inside the class’s definition, but not inside any of the class’s methods or properties (in which case, they’d be local variables).

c. Class attributes are typically accessed through any object of the class.

d. Class attributes exist as soon as you import their class’s definition.

Answer: c. Class attributes are typically accessed through any object of the class. Actually, class attributes **can** be accessed through any object of the class, but are **typically** accessed through the **class’s name**.

* Class DeckOfCards

No questions.

* Displaying Card Images with Matplotlib

No questions.

* Inheritance: Base Classes and Subclasses

10.7 Q1: Which of the following statements is false?

a. Often, an object of one class is an object of another class as well.

b. A CarLoan is a Loan as are HomeImprovementLoans and MortgageLoans. Class CarLoan can be said to inherit from class Loan.

c. In the context of Part (b), class CarLoan is a base class, and class Loan is a subclass.

d. In the context of Part (b) a CarLoan is a specific type of Loan, but it’s incorrect to claim that every Loan is a CarLoan—the Loan could be any type of loan.

Answer: c. In the context of Part (b), class CarLoan is a base class, and class Loan is a subclass. Actually, In the context of Part (b), class Loan is a base class, and class CarLoan is a subclass.

10.7 Q2: The following table lists several simple examples of base classes and subclasses:

|  |  |
| --- | --- |
| **Base class** | **Subclasses** |
| Student | GraduateStudent, UndergraduateStudent |
| Shape | Circle, Triangle, Rectangle, Sphere, Cube |
| Loan | CarLoan, HomeImprovementLoan, MortgageLoan |
| Employee | Faculty, Staff |
| BankAccount | CheckingAccount, SavingsAccount |

Assume there are no base class objects. Without actually knowing the number of objects of each subclass—and assuming that no object of one subclass is also an object of another subclass, which of the following statements about this table is guaranteed to be true?

a. There are more Students than Shapes.

b. There are fewer Faculty than Staff.

c. The number of BankAccounts is equal to the number of CheckingAccounts plus the number of SavingsAccounts.

d. All of the above statements are false.

Answer: c. The number of BankAccounts is equal to the number of CheckingAccounts plus the number of SavingsAccounts.

10.7 Q3: Which of the following statements is false?

a. Base classes tend to be “more specific” and subclasses “more general.”

b. Because every subclass object is an object of its base class, and one base class can have many subclasses, the set of objects represented by a base class is often larger than the set of objects represented by any of its subclasses.

c. The base class Vehicle represents all vehicles, including cars, trucks, boats, bicycles and so on. Subclass Car represents a smaller, more specific subset of vehicles.

d. All of the above statements are true.

Answer: a. Base classes tend to be “more specific” and subclasses “more general.” Actually, base classes tend to be “more general” and subclasses “more specific”

10.7 Q4: Which of the following statements is false?

a. A base class exists in a hierarchical relationship with its subclasses.

b. With single inheritance, a class is derived from one base class. With multiple inheritance, a subclass inherits from two or more base classes.

c. One reason to avoid multiple inheritance is the “diamond problem in Python multiple inheritance.”

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.7 Q5: Consider a Shape inheritance hierarchy that begins with base class Shape, which has subclasses TwoDimensionalShape and ThreeDim-ensional-Shape. Class TwoDimensionalShape has subslasses Circle, Square and Triangle. Class ThreeDimensionalShape has subclasses Sphere, Cube and Tetrahedron. Which of the following statements is false?

a. Each Shape is either a TwoDimensionalShape or a Three-Dimensional-Shape.

b. The third level of this hierarchy contains specific types of Two-Dimensional-Shapes and ThreeDimensionalShapes.

c. In this hierarchy, a Triangle is a TwoDimensionalShape and is a Shape, while a Sphere is a ThreeDimensionalShape and is a Shape.

d. A TwoDimensionalShape is a Square.

Answer: d. A TwoDimensionalShape is a Square. Actually, the "is-a" relationship of inheritance works only from a subclass to its base class—so a Square is a TwoDimensionalShape**, but not vice versa.**

* Building an Inheritance Hierarchy; Introducing Polymorphism

No questions.

* Base Class CommissionEmployee

10.8 Q1: Which of the following statements is false?

a. You use inheritance to create new classes from existing ones.

b. When you do not explicitly specify the base class for a new class, Python assumes that the class inherits directly from class object.

c. The Python class hierarchy begins with class object, the direct or indirect base class of every class. So, class CommissionEmployee’s header could have been written as

class CommissionEmployee(object):

d. The parentheses after CommissionEmployee indicate inheritance and must contain a single class for inheritance.

Answer: d. The parentheses after CommissionEmployee indicate inheritance and must contain a single class for inheritance. Actually, the parentheses after CommissionEmployee indicate inheritance and may contain a single class for single inheritance or a *comma-separated list of base classes for multiple inheritance*.

* Subclass SalariedCommissionEmployee

10.8 Q2: Which of the following statements is false?

a. Python provides two built-in functions—issubclass and isinstance—for testing “is a” relationships.

b. Function issubclass determines whether one class is derived from another.

c. Function isinstance determines whether an object has an “is a” relationship with a specific type.

d. If SalariedCommissionEmployee inherits from CommissionEmployee, and if object s is a SalariedCommissionEmployee, then the following snippet results are correct:

In [19]: isinstance(s, CommissionEmployee)

Out[19]: False

In [20]: isinstance(s, SalariedCommissionEmployee)

Out[20]: True

Answer: d. If SalariedCommissionEmployee inherits from CommissionEmployee, and if object s is a SalariedCommissionEmployee, then the following snippet results are correct:

In [19]: isinstance(s, CommissionEmployee)

Out[19]: False

In [20]: isinstance(s, SalariedCommissionEmployee)

Out[20]: True

Actually, isinstance(s, CommissionEmployee) should return True.

* Processing CommissionEmployees and SalariedCommissionEmployees Polymorphically

10.8 Q3: Which of the following statements is false?

a. With inheritance, every object of a subclass also may be treated as an object of that subclass’s base class.

b. We can take advantage of this “subclass-object-is-a-base-class-object” relationship to place objects related through inheritance into a list, then iterate through the list and treat each element as a base-class object.

c. The following code places CommissionEmployee and SalariedCommissionEmployee objects (c and s) in a list, then for each element displays its string representation and earnings—this is an example of polymorphism:

In [21]: employees = [c, s]

In [22]: for employee in employees:

...: print(employee)

...: print(f'{employee.earnings():,.2f}\n')

...:

CommissionEmployee: Sue Jones

social security number: 333-33-3333

gross sales: 20000.00

commission rate: 0.10

2,000.00

SalariedCommissionEmployee: Bob Lewis

social security number: 444-44-4444

gross sales: 10000.00

commission rate: 0.05

base salary: 1000.00

1,500.00

d. All of the above statements are true?

Answer: d. All of the above statements are *true*?

* A Note About Object-Based and Object-Oriented Programming

10.8 Q4: Which of the following statements is false?

a. Inheritance with method overriding is a powerful way to build software components that are like existing components but need to be customized to your application’s unique needs.

b. In the Python open-source world, there are a huge number of well-developed class libraries for which your programming style is know what libraries are available, know what classes are available, make objects of existing classes, and call their methods. This style of programming called object-based programming (OBP).

c. When you do composition with objects of known classes, you’re still doing object-based programming. Adding inheritance with overriding to customize methods to the unique needs of your applications and possibly process objects polymorphically is called object-oriented programming (OOP). If you do composition with objects of inherited classes, that’s also object-oriented programming.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

* Duck Typing and Polymorphism

10.9 Q1: Which of the following statements is false?

b. All classes inherit from object directly or indirectly, so they all inherit the default methods for obtaining string representations that print can display.

b. Python also has a more flexible concept called duck typing, which the Python documentation describes as:

A programming style which does not look at an object’s type to determine if it has the right interface; instead, the method or attribute is simply called or used (“If it looks like a duck and quacks like a duck, it must be a duck.”).

c. When Python processes an object at execution time, its type does not matter. As long as the object has the data attribute, property or method (with the appropriate parameters) you wish to access, the code will work.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.9 Q2: Consider the following loop, which processes a list of employees:

for employee in employees:   
 print(employee)   
 print(f'{employee.earnings():,.2f}\n')

Which of the following statements is false?

a. In Python, this loop works properly as long as employees contains only objects that:

* can be displayed with print (that is, they have a string representation)

b. All classes inherit from object directly or indirectly, so they all inherit the default methods for obtaining string representations that print can display.

c. If a class has an earnings method that can be called with no arguments, we can include objects of that class in the list employees, even if the object’s class does not have an “is a” relationship with class CommissionEmployee.

d. All of the above statements are true.

Answer: a) In Python, this loop works properly as long as employees contains only objects that:

* can be displayed with print (that is, they have a string representation)

Actually, in Python, this loop works properly as long as employees contains only objects that:

* can be displayed with print (that is, they have a string representation) and
* have an earnings method which can be called with no arguments.
* Operator Overloading

10.10 Q1: Which of the following statements is false?

a. Method-call notation can be cumbersome for certain kinds of operations, such as arithmetic. In these cases, it would be more convenient to use Python’s rich set of built-in operators. Use operator overloading to define how Python’s operators should handle objects of your own custom types.

b. The overloaded + operator is used for adding numeric values, concatenating lists, concatenating strings and adding a value to every element in a NumPy array.

c. The overloaded [] operator is used for accessing elements in lists, tuples, strings and arrays and for accessing the value for a specific key in a dictionary. The overloaded \* operator is used for multiplying numeric values, repeating a sequence and multiplying every element in a NumPy array by a specific value.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.10 Q2: Which of the following statements is false?

a. You can overload most operators.

b. For every overloadable operator, class object defines a special method, such as \_add\_ for the addition (+) operator or \_mul\_ for the multiplication (\*) operator.

c. Overriding the special methods mentioned in part b) enables you to define how a given operator works for objects of your custom classes.

d. All of the above statements are true.

Answer: b. For every overloadable operator, class object defines a special method, such as \_add\_ for the addition (+) operator or \_mul\_ for the multiplication (\*) operator. Actually, for every overloadable operator, class object defines a special method, such as \_\_add\_\_ for the addition (+) operator or \_\_mul\_\_ for the multiplication (\*) operator—each of these special methods begins and ends with **double underscores**.

10.10 Q3: Which of the following statements is false?

a. The precedence of an operator cannot be changed by overloading. As in algebra, parentheses can be used to force evaluation order of operators in an expression.

b. The left-to-right or right-to-left grouping of an operator cannot be changed by overloading. The “arity” of an operator—that is, whether it’s a unary or binary operator—cannot be changed. You cannot create new operators—only existing operators can be overloaded.

c. The meaning of how an operator works on objects of built-in types cannot be changed—you cannot, for example, change + so that it subtracts two integers.

d. Operator overloading works only with objects of custom classes.   
Answer: d. Operator overloading works only with objects of custom classes. Actually, operator overloading works only with objects of custom classes *or with a mixture of an object of a custom class and an object of a built-in type*.

* Test-Driving Class Complex

No questions.

* Class Complex Definition

No questions.

* Exception Class Hierarchy and Custom Exceptions

10.11 Q1: Exception classes inherit directly or indirectly from base class BaseException and are defined in module exceptions. Python defines four primary BaseException subclasses—SystemExit, KeyboardInterrupt, GeneratorExit and Exception:

a. SystemExit terminates program execution (or terminates an interactive session) and when uncaught does not produce a traceback like other exception types. KeyboardInterrupt exceptions occur when the user types the interrupt command—Ctrl + C (or control + C) on most systems.

c. GeneratorExit exceptions occur when a generator closes—normally when a generator finishes producing values or when its close method is called explicitly. Exception is the base class for most common exceptions you’ll encounter.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.11 Q2: Which of the following statements is false?

a. One of the benefits of the exception class hierarchy is that an except handler can catch exceptions of a particular type or can use a base-class type to catch those base-class exceptions and all related subclass exceptions.

b. An except handler that specifies the base class Exception can catch objects of any subclass of Exception.

c. An except handler that catches type Exception should be placed before other except handlers to ensure that all exceptions are properly caught.

d. All of the above statements are true.

Answer: c. An except handler that catches type Exception should be placed before other except handlers to ensure that all exceptions are properly caught. Actually, Placing an except handler that catches type Exception before other except handlers is a logic error, because all exceptions would be caught before other exception handlers could be reached. Thus, subsequent exception handlers are unreachable.

* Named Tuples

10.12 Q1: Which of the following statements is false?

a. The Python Standard Library’s collections module also provides named tuples that enable you to reference a tuple’s members by name rather than by index number.

b. Function namedtuple creates a base class of the built-in tuple type.

c. The function’s first argument is your new type’s name and the second is a list of strings representing the identifiers you’ll use to reference the new type’s members—for example.

In [2]: Card = namedtuple('Card', ['face', 'suit'])

d. All of the above statements are true.

Answer: b. Function namedtuple creates a base class of the built-in tuple type. Actually, function namedtuple creates a *subclass* of the built-in tuple type.

10.12 Q2: Which of the following statements is false?

a. Each named tuple type has additional methods.

b. The type’s \_make class method (that is, a method called on the class) receives an iterable of values and returns an object of the named tuple type. For example:

In [7]: values = ['Queen', 'Hearts']

In [8]: card = Card.\_make(values)

In [9]: card

Out[9]: Card(face='Queen', suit='Hearts')

c. For a given object of a named tuple type, you can get an OrderedDict dictionary representation of the object’s member names and values—an OrderedDict remembers the order in which its key–value pairs were inserted in the dictionary.

d. All of the above statements are true.

Answer: Actually, d. All of the above statements are *true*.

* A Brief Intro to Python 3.7’s New Data Classes

10.13 Q1: Which of the following statements is false?

a. Though named tuples allow you to reference their members by name, they’re still just tuples, not classes.

b. For some of the benefits of named tuples, plus the capabilities that traditional Python classes provide, you can use Python 3.7’s data classes from the Python Standard Library’s dataclasses module.

c. One problem with data classes is that they typically require more development time than traditional classes.

d. Data classes could become the preferred way to define many Python classes.

Answer: c. One problem with data classes is that they typically require more development time than traditional classes. Actually, data classes help you build classes *faster* by using more concise notation and by autogenerating boilerplate code that’s common in most classes.

10.13 Q2: Which of the following statements is false?

a. Most classes you’ll define provide an \_\_init\_\_ method to create and initialize an object’s attributes and a \_\_repr\_\_ method to specify an object’s custom string representation.

b. Data classes also autogenerate method \_\_eq\_\_, which overloads the = operator.

c. Any class that has an \_\_eq\_\_ method also implicitly supports !=.

d. All classes inherit class object’s default \_\_ne\_\_ (not equals) method implementation, which returns the opposite of \_\_eq\_\_ (or NotImplemented if the class does not define \_\_eq\_\_).

Answer: b. Data classes also autogenerate method \_\_eq\_\_, which overloads the = operator. Actually, data classes also autogenerate method \_\_eq\_\_, which overloads the == operator.

* Creating a Card Data Class

10.13 Q3: Which of the following statements is false?

a. The Python Standard Library’s dataclasses module defines decorators and functions for implementing data classes.

b. Use the @dataclass decorator to specify that a new class is a data class and causes various code to be written for you.

c. ClassVar from the Python Standard Library’s typing module can be used to indicate that a variable is a class variable.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.13 Q4: Which of the following statements is false?

a. To specify that a class is a data class, precede its definition with the @dataclass decorator, as in:

@dataclass

class Card:

b. Optionally, the @dataclass decorator may specify parentheses containing arguments that help the data class determine what autogenerated methods to include.

c. The decorator @dataclass(order=True) would cause the data class to autogenerate overloaded comparison operator methods for <, <=, > and >=. This might be useful, for example, if you need to sort your data-class objects.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.13 Q5: Which of the following statements is false?

a. Like regular classes, data classes declare both class attributes and data attributes inside the class, but outside the class’s methods.

b. Data classes require additional information, or hints, to distinguish class attributes from data attributes, which also affects the autogenerated methods’ implementation details.

c. The following code defines and initializes the class attributes FACES and SUITS:

FACES: ClassVar[List[str]] = ['Ace', '2', '3', '4', '5', '6', '7',

'8', '9', '10', 'Jack', 'Queen', 'King']

SUITS: ClassVar[List[str]] = ['Hearts', 'Diamonds', 'Clubs', 'Spades']

d. The notation in Part (c)

: ClassVar[List[str]]

is a variable annotation (sometimes called a type hint) specifying that FACES and SUITS are class attributes (ClassVar) which refers to a list of strings (List[str]). Class variables are initialized in their definitions and are specific to the class, not individual objects of the class.

Answer: a. Like regular classes, data classes declare both class attributes and data attributes inside the class, but outside the class’s methods. Actually, unlike regular classes, data classes declare both class attributes and data attributes inside the class, but outside the class’s methods—in a regular class, only class attributes are declared this way, and data attributes typically are created in \_\_init\_\_.

10.13 Q6: Which of the following statements is false?

a. You can specify variable annotations using built-in type names (like str, int and float), class types or types defined by the typing module (such as ClassVar and List).

b. Even with type annotations, Python is still a dynamically typed language.

c. Type annotations are not enforced at execution time.

d. All of the above statements are true.

Answer: d. All of the above statements are true

* Using the Card Data Class

No questions.

* Data Class Advantages over Named Tuples

10.13 Q7: Data classes offer several advantages over named tuples. Which of the following statements is false?

a. Although each named tuple technically represents a different type, a named tuple is a tuple and all tuples can be compared to one another. So, objects of different named tuple types could compare as equal if they have the same number of members and the same values for those members. Comparing objects of different data classes always returns False, as does comparing a data class object to a tuple object.

b. If you have code that unpacks a tuple, adding more members to that tuple breaks the unpacking code. Data class objects cannot be unpacked. So you can add more data attributes to a data class without breaking existing code.

c. A data class can be a base class or a subclass in an inheritance hierarchy.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

* Data Class Advantages over Traditional Classes

10.13 Q8: Data classes also offer various advantages over the traditional Python classes you saw earlier in this chapter. Which of the following statements is false?:

a. A data class autogenerates \_\_init\_\_, \_\_repr\_\_ and \_\_eq\_\_, saving you time.

b. A data class cannot autogenerate the special methods that overload the <, <=, > and >= comparison operators.

c. The required variable annotations for class attributes and data attributes enable you to take advantage of static code analysis tools. So, you might be able to eliminate additional errors before they can occur at execution time.

d. Some static code analy-sis tools and IDEs can inspect variable annotations and issue warnings if your code uses the wrong type. This can help you locate logic errors in your code before you execute it.

Answer: b. A data class cannot autogenerate the special methods that overload the <, <=, > and >= comparison operators. Actually, a data class optionally *can* autogenerate the special methods that overload the <, <=, > and >= comparison operators.

* Unit Testing with Docstrings and **doctest**

10.14 Q1: Which of the following statements is false?

a. Even with extensive testing, however, your code may still contain bugs. According to the famous Dutch computer scientist Edsger Dijkstra, “Testing shows the \_\_\_\_\_\_\_\_, not the \_\_\_\_\_\_\_\_ of bugs.”

a. possibility, absence

b. presence, possibility

c. presence, absence

d. None of the above

Answer: c. presence, absence

10.14 Q2: Which of the following statements is false?

a. The Python Standard Library provides the doctest module to help you test your code and conveniently retest it after you make modifications.

b. When you execute the doctest module’s testmod function, it inspects your functions’, methods’ and classes' docstrings looking for sample Python statements preceded by >>>, each followed on the next line by the given statement’s expected output (if any). The testmod function then executes those statements and confirms that they produce the expected output. If they do not, testmod reports errors indicating which tests failed so you can locate and fix the problems in your code.

c. Each test you define in a docstring typically tests a specific unit of code, such as a function, a method or a class. Such tests are called integration tests.

d. All of the above statements are true.

Answer: c. Each test you define in a docstring typically tests a specific unit of code, such as a function, a method or a class. Such tests are called integration tests. Actually, such tests are called unit tests.

10.14 Q3: Which of the following statements is false?

a. When you load any module, Python assigns a string containing the module’s name to a global attribute of the module called \_\_name\_\_.

b. When you execute a Python source file (such as accountdoctest.py) as a script, Python uses the string '\_\_main\_\_' as the module’s name.

c. You can use \_\_name\_\_ in an if statement to specify code that should execute only if the source file is executed as a script.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.14 Q4: Which of the following statements is false?

a. A convenient way to create doctests for existing code is to use an IPython interactive session to test your code, then copy and paste that session into a docstring.

b. IPython’s In [] and Out[] prompts are compatible with doctest.

c. The %doctest\_mode magic toggles between the two prompt styles. The first time you execute the magic, IPython switches to >>> prompts for input and no output prompts. The second time you execute %doctest\_mode, IPython switches back to In [] and Out[] prompts.

d. All of the above statements are true.

Answer: b. IPython’s In [] and Out[] prompts are compatible with doctest. Actually, IPython’s In [] and Out[] prompts are *not* compatible with doctest, so IPython provides the magic %doctest\_mode to display prompts in the correct doctest format.

* Namespaces and Scopes

10.15 Q1: Which of the following statements is false?

a. Each identifier has a scope that determines where you can use it in your program, and we introduced the local and global scopes.

b. Scopes are determined by namespaces, which associate identifiers with objects and are implemented “under the hood” as dictionaries.

c. All namespaces are dependent on one another. So, the same identifier may appear in only one namespace.

d. There are three primary namespaces—local, global and built-in.

Answer: c. All namespaces are dependent on one another—so, the same identifier may appear in only one namespace. Actually, all namespaces are independent of one another—so, the same identifier may appear in multiple namespaces.

10.15 Q2: Which of the following statements is false?

a. Each function and method has a local namespace that associates local identifiers (such as, parameters and local variables) with objects.

b. The local namespace exists from the moment the function or method is called until it terminates and is accessible only to that function or method.

c. In a function’s or method’s suite, assigning to a variable that does not exist creates a local variable and adds it to the local namespace.

d. Identifiers in the local namespace are in scope from the point at which you define them until the program terminates.

Answer: d. Identifiers in the local namespace are in scope from the point at which you define them until the program terminates. Actually, identifiers in the local namespace are in scope from the point at which you define them until the *function or method* terminates.

10.15 Q3: Which of the following statements is false?

a. Each module has a global namespace that associates a module’s global identifiers (such as global variables, function names and class names) with objects. An IPython session has its own global namespace for all the identifiers you create in that session.

b. Python creates a module’s global namespace when it loads the module. A module’s global namespace exists and its identifiers are in scope to the code within that module until the program (or interactive session) terminates.

c. Each module’s global namespace also has an identifier called \_\_name\_\_ containing the module’s name, such as 'math' for the math module.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.15 Q4: When you use an identifier, Python searches for that identifier in the currently accessible namespaces, proceeding from \_\_\_\_\_\_\_\_ to \_\_\_\_\_\_\_\_ to \_\_\_\_\_\_\_\_.

a. global, built-in, local

b. built-in, local, global

c. local, built-in, global

d. local, global, built-in

Answer: d. local, global, built-in

10.15 Q5: Which of the following statements is false?

a. Python allows you to define nested functions inside other functions or methods.

b. If a function or method performs the same task several times, you might define a nested function to avoid repeating code in the enclosing function.

c. When you access an identifier inside a nested function, Python searches the nested function’s local namespace first, then the global namespace, then the built-in namespace and finally the enclosing function’s namespace —this is sometimes referred to as the LGBE (local, global, built-in, enclosing) rule.

d. All of the above statements are true.

Answer: c. When you access an identifier inside a nested function, Python searches the nested function’s local namespace first, then the global namespace, then the built-in namespace and finally the enclosing function’s namespace —this is sometimes referred to as the LGBE (local, global, built-in, enclosing) rule. Actually, when you access an identifier inside a nested function, Python searches the nested function’s local namespace first, then the enclosing function’s namespace, then the global namespace and finally the built-in namespace—this is sometimes referred to as the LEGB (local, enclosing, global, built-in) rule.

10.15 Q6: Which of the following statements is false?

a. Each object has its own namespace containing the object’s methods and data attributes.

b. The class’s \_\_init\_\_ method starts with an empty object (self) and adds each attribute to the object’s namespace.

c. Once you define an attribute in an object’s namespace, clients using the object may access the attribute’s value.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

* Intro to Data Science: Time Series and Simple Linear Regression

10.16 Q1: Which of the following statements is false?

a. Time series are sequences of values called observations associated with points in time.

b. Some examples of time series are daily closing stock prices, hourly temperature readings, the changing positions of a plane in flight, annual crop yields, quarterly company profits, and the stream of time-stamped tweets coming from Twitter users worldwide.

c. You can use simple linear regression to make predictions from time series data.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.16 Q2: \_\_\_\_\_\_\_\_ time series have one observation per time, such as the average of the January high temperatures in New York City for a particular year; \_\_\_\_\_\_\_\_ time series have two or more observations per time, such as temperature, humidity and barometric pressure readings in a weather application.

a. Univariate, bivariate

b. Single, multivariate

c. Univariate, multivariate

d. Single, mixed

Answer: c. Univariate, multivariate

10.16 Q3: Given a Fahrenheit temperature, we can calculate the corresponding Celsius temperature using the following formula:

c = 5 / 9 \* (f - 32)

In this formula, f (the Fahrenheit temperature) is the \_\_\_\_\_\_\_\_ variable, and c (the Celsius temperature) is the \_\_\_\_\_\_\_\_ variable.

a. dependent, independent

b. separated, connected

c. independent, dependent

d. All of the above statements are true.

Answer: c. independent, dependent

10.16 Q4: Simple linear regression determines the slope (m) and intercept (b) of a straight line that best fits your data. Consider the following diagram, which shows some time-series data points and a corresponding regression line—we added vertical lines to indicate each data point’s distance from the regression line.

A close up of a map

Description automatically generated

Which of the following statements is false?

a. The simple linear regression algorithm iteratively adjusts the slope and intercept and, for each adjustment, calculates the square of each point’s distance from the line. The “best fit” occurs when the slope and intercept values minimize the sum of those squared distances. This is known as an ordinary least squares calculation.

c. The SciPy (Scientific Python) library is widely used for engineering, science and math in Python. The linregress function (from the scipy.stats module) performs simple linear regression for you. After calling linregress, you plug the resulting slope and intercept into the y = mx + b equation to make predictions.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

10.16 Q5: Which of the following statements is false?

a. Seaborn function regplot’s x and y keyword arguments are two-dimensional arrays of the same length representing the x-y coordinate pairs to plot.

b. Pandas automatically creates attributes for each column name if the name can be a valid Python identifier.

c. Seaborn and Matplotlib auto-scale the axes, based on the data’s range of values.

d. All of the above statements are true.

Answer: a. Seaborn function regplot’s x and y keyword arguments are two-dimensional arrays of the same length representing the *x-y* coordinate pairs to plot. Actually, function regplot’s x and y keyword arguments are *one*-dimensional arrays of the same length representing the *x-y* coordinate pairs to plot.