# *Test Bank for Fluent Python 2nd Edition by Ramalho*

ISBN: 9781492056355

# Fluent Python

ISBN 9781492056355 | Chapter 1: The Python Data Model

*Total Questions: 100*

## Multiple Choice

**Q1.**
In the chapter, the Python Data Model is primarily described as what?

**A) A framework-level description of Python's core object interfaces ✓ Correct**
**B)** A debugging tool for tracing interpreter execution
**C)** A package for scientific computing with arrays
**D)** A style guide for naming variables and functions

*Rationale: The chapter describes the data model as a description of Python as a framework that formalizes core language interfaces.*

**Q2.**
What naming pattern identifies Python special methods?

**A)** A single leading underscore
**B)** A trailing underscore only
**C) Leading and trailing double underscores ✓ Correct**
**D)** Uppercase letters separated by underscores

*Rationale: Special methods are written with double underscores before and after the method name.*

**Q3.**
Which special method supports the syntax obj[key]?

**A)** __contains__
**B) __getitem__ ✓ Correct**
**C)** __iter__
**D)** __setattr__

*Rationale: The interpreter uses __getitem__ to evaluate item access with square brackets.*

**Q4.**
The slang term "dunder" refers to what aspect of a special method name?

**A)** Its use in debugging output
**B)** Its role in operator overloading
**C)** Its inheritance from object
**D) Its double underscores before and after the name ✓ Correct**

*Rationale: "Dunder" is shorthand for "double underscore before and after."*

**Q5.**
Which of the following was highlighted as a significant update in this chapter's second edition coverage?

**A)** Removal of iterator support from the data model
**B) Addition of asynchronous special methods to the overview tables ✓ Correct**
**C)** Replacement of namedtuple with dataclass in all examples
**D)** Deprecation of __getitem__ for sequence classes

*Rationale: The chapter notes the addition of asynchronous special methods and other newer features to the tables.*

**Q6.**
Which string formatting style does the author say was adopted throughout the second edition for readability and convenience?

**A)** Template strings
**B)** Percent-formatting
**C) f-strings ✓ Correct**
**D)** Concatenation with plus operators

*Rationale: The chapter explicitly states that f-string syntax was adopted throughout the second edition.*

**Q7.**
In the card deck example, Card is created using which standard library factory?

**A)** dataclasses.make_dataclass
**B) collections.namedtuple ✓ Correct**
**C)** typing.NamedTupleMeta
**D)** types.SimpleNamespace

*Rationale: The example defines Card with collections.namedtuple('Card', ['rank', 'suit']).*

**Q8.**
In the FrenchDeck example, implementing which two special methods is sufficient to make the deck behave like a sequence?

**A)** __iter__ and __contains__
**B)** __repr__ and __str__
**C) __len__ and __getitem__ ✓ Correct**
**D)** __setitem__ and __delitem__

*Rationale: The chapter emphasizes that __len__ and __getitem__ make FrenchDeck behave like a standard sequence.*

**Q9.**
What does len(deck) return for a newly created FrenchDeck instance?

**A)** 13
**B)** 26
**C) 52 ✓ Correct**
**D)** 54

*Rationale: The example shows len(deck) returning 52.*

**Q10.**

Which built-in function is used in the example to select a random card from the deck?

**A)** random.sample
**B)** random.shuffle
**C)** random.randint
**D) random.choice ✓ Correct**

*Rationale: The example imports and uses random.choice on the deck instance.*

**Q11.**

Why does the FrenchDeck example automatically support slicing?

**A)** Because it inherits from collections.abc.Sequence
**B) Because its __getitem__ delegates to the underlying list ✓ Correct**
**C)** Because slicing is built into namedtuple
**D)** Because __len__ alone enables slicing

*Rationale: Delegating __getitem__ to self._cards means list-style indexing and slicing work automatically.*

**Q12.**

In the chapter, a doctest ellipsis (...) in displayed output indicates what?

**A)** A syntax error in the example
**B) A placeholder for omitted output ✓ Correct**
**C)** A required operator in the expression
**D)** An asynchronous pause point

*Rationale: The ellipsis marks elided output when the full result is too long to display.*

**Q13.**

If a collection has no __contains__ method, how does the in operator typically behave according to the chapter?

**A)** It always raises TypeError
**B)** It compares object identities only
**C) It performs a sequential scan if the object is iterable ✓ Correct**
**D)** It calls __len__ and returns True if length is nonzero

*Rationale: The chapter explains that in falls back to a sequential scan when __contains__ is absent and the object is iterable.*

**Q14.**

In the card-ranking example, which suit is assigned the highest value?

**A)** Clubs
**B)** Diamonds
**C)** Hearts
**D) Spades ✓ Correct**

*Rationale: The suit_values mapping assigns spades a value of 3, the highest among the suits.*

**Q15.**
According to the chapter, the current FrenchDeck implementation cannot be shuffled mainly because it is what?

**A)** Ordered alphabetically
**B) Immutable ✓ Correct**
**C)** Asynchronous
**D)** Hash-based

*Rationale: The chapter states that the deck cannot be shuffled because it is immutable without __setitem__.*

**Q16.**
Who is identified in the chapter as the only frequent caller of most special methods?

**A)** The standard library only
**B)** The debugger only
**C) The Python interpreter ✓ Correct**
**D)** The metaclass system only

*Rationale: The text states that the Python interpreter is the only frequent caller of most special methods.*

**Q17.**
For built-in variable-sized collections implemented in C, len(x) is especially fast in CPython because it can read which field?

**A)** tp_name
**B) ob_size ✓ Correct**
**C)** ma_used
**D)** nb_index

*Rationale: The chapter explains that CPython reads the ob_size field from PyVarObject for built-in variable-sized collections.*

**Q18.**
What function call does the statement for i in x: effectively trigger first?

**A)** next(x)
**B)** len(x)
**C) iter(x) ✓ Correct**
**D)** repr(x)

*Rationale: The chapter states that a for loop actually causes the invocation of iter(x).*

**Q19.**
Which special method is mentioned as the one user code most often calls directly, typically to invoke a superclass initializer?

**A)** __call__
**B)** __new__
**C)** __repr__
**D) __init__ ✓ Correct**

*Rationale: The chapter notes that __init__ is the special method most frequently called directly by user code.*

**Q20.**
In the Vector example, which special method provides the string used for inspection in the console and debugger?

**A)** __str__
**B) __repr__ ✓ Correct**
**C)** __format__
**D)** __bytes__

*Rationale: The repr built-in, console, and debugger use __repr__ for object inspection.*

**Q21.**
Which built-in function is used in the Vector example to compute vector magnitude?

**A)** sum
**B)** round
**C) abs ✓ Correct**
**D)** pow

*Rationale: The example uses abs(v) to calculate the magnitude of a vector.*

**Q22.**
Which special method implements scalar multiplication in the Vector class example?

**A)** __imul__
**B)** __matmul__
**C) __mul__ ✓ Correct**
**D)** __rmul__

*Rationale: The Vector example defines __mul__ to multiply a vector by a scalar.*

**Q23.**
In the Vector example as presented, what is the expected behavior of infix operators such as + and *?

**A)** They modify the left operand in place
**B) They create and return new objects ✓ Correct**
**C)** They always return primitive numeric types
**D)** They are valid only for built-in classes

*Rationale: The chapter states that infix operators are expected to create new objects without modifying their operands.*

**Q24.**
Without a custom __repr__, how would a Vector instance typically appear in the console?

**A)** As a JSON object
**B)** As a tuple of coordinates
**C) As a memory-address style default object representation ✓ Correct**
**D)** As a formatted mathematical expression

*Rationale: The chapter gives the default style as something like <Vector object at 0x...>.*

**Q25.**

In an f-string, what does the !r conversion request?

**A)** A rounded numeric value
**B)** A recursive expansion of attributes
**C) The standard repr-style representation ✓ Correct**
**D)** A raw byte sequence

*Rationale: The !r conversion tells Python to use repr on the value.*

**Q26.**

According to the chapter, what should a good __repr__ string ideally do?

**A)** Hide implementation details from developers
**B) Match the source code needed to recreate the object when possible ✓ Correct**
**C)** Return localized text for end users
**D)** Always be shorter than __str__

*Rationale: The chapter says __repr__ should be unambiguous and, if possible, resemble code to recreate the object.*

**Q27.**

Which special method is implicitly used by the print function?

**A)** __repr__
**B)** __bytes__
**C) __str__ ✓ Correct**
**D)** __format__

*Rationale: The chapter states that __str__ is called by str() and implicitly used by print.*

**Q28.**

If a class implements only one of __str__ or __repr__, which one does the chapter recommend implementing?

**A)** __str__
**B) __repr__ ✓ Correct**
**C)** __format__
**D)** __bytes__

*Rationale: The chapter explicitly advises implementing __repr__ if only one is provided.*

**Q29.**

By default, instances of user-defined classes are considered truthy unless which methods alter that behavior?

**A)** __repr__ or __str__
**B)** __iter__ or __contains__
**C) __bool__ or __len__ ✓ Correct**
**D)** __hash__ or __eq__

*Rationale: Truth value testing for user-defined objects is affected by __bool__ or, failing that, __len__.*

**Q30.**

If __bool__ is not implemented, Python determines truthiness by trying which special method next?

**A)** __index__
**B) __len__ ✓ Correct**
**C)** __iter__
**D)** __call__

*Rationale: The chapter explains that Python falls back to __len__ when __bool__ is absent.*

**Q31.**

In truth value testing, what result does bool(x) return if x.__len__() returns zero and __bool__ is not defined?

**A)** True
**B) False ✓ Correct**
**C)** None
**D)** TypeError

*Rationale: A zero length makes the object falsy when __bool__ is not implemented.*

**Q32.**

Which abstract base class, introduced in Python 3.6, unifies Iterable, Sized, and Container?

**A)** Sequence
**B)** Reversible
**C) Collection ✓ Correct**
**D)** Mapping

*Rationale: The chapter identifies collections.abc.Collection as the ABC that unifies those three interfaces.*

**Q33.**

Which of the following is listed as one of the three important specializations of Collection?

**A)** Generator
**B) Mapping ✓ Correct**
**C)** Callable
**D)** Descriptor

*Rationale: The chapter lists Sequence, Mapping, and Set as key specializations of Collection.*

**Q34.**

Why is only Sequence described as Reversible among the major Collection specializations?

**A) Because only sequences support arbitrary ordering of contents ✓ Correct**
**B)** Because mappings and sets have no length
**C)** Because sequences alone support hashing
**D)** Because only sequences can implement __iter__

*Rationale: The chapter explains that only sequences support arbitrary ordering, unlike mappings and sets.*

**Q35.**

According to the chapter, what does dict being officially “ordered” since Python 3.7 mean?

**A)** Its keys are automatically sorted alphabetically
**B)** Its values can be reversed with reversed(dict)
**C) Its key insertion order is preserved ✓ Correct**
**D)** Its items support arbitrary user-defined rearrangement

*Rationale: The text clarifies that ordered dict means preserving insertion order, not arbitrary rearrangement.*

**Q36.**

In the Set abstract base class, infix operators such as a & b are implemented by what kind of methods?

**A)** Context manager methods
**B) Special methods for operators ✓ Correct**
**C)** Descriptor methods
**D)** Attribute access hooks

*Rationale: The chapter states that Set ABC methods implement infix operators such as intersection via __and__.*

**Q37.**

Which special method listed in the chapter supports asynchronous iteration by producing the next item?

**A)** __await__
**B)** __aiter__
**C) __anext__ ✓ Correct**
**D)** __next__

*Rationale: __anext__ is the asynchronous counterpart used to get the next item in async iteration.*

**Q38.**

Which special method was added in Python 3.6 as a class customization hook?

**A)** __prepare__
**B)** __class_getitem__
**C)** __mro_entries__
**D) __init_subclass__ ✓ Correct**

*Rationale: The chapter identifies __init_subclass__ as the class customization hook added in Python 3.6.*

**Q39.**

Which operator was added in Python 3.5 to support matrix multiplication?

**A)** #
**B) @ ✓ Correct**
**C)** ~
**D)** ^

*Rationale: The chapter notes that @ was added as the infix operator for matrix multiplication.*

**Q40.**
Which special method corresponds to the unary abs() operation?

**A)** __neg__
**B)** __pos__
**C) __abs__ ✓ Correct**
**D)** __round__

*Rationale: The abs built-in maps to the __abs__ special method.*

**Q41.**
Which special method corresponds to the rich comparison operator == ?

**A)** __lt__
**B) __eq__ ✓ Correct**
**C)** __ne__
**D)** __ge__

*Rationale: The equality operator == is implemented by __eq__.*

**Q42.**
When the first operand's arithmetic special method cannot be used, Python may try which method on the second operand?

**A)** A descriptor method
**B) A reversed operator special method ✓ Correct**
**C)** An augmented assignment method
**D)** A context manager method

*Rationale: The chapter explains that Python calls the reversed operator special method on the second operand in that case.*

**Q43.**
Augmented assignment is best described in the chapter as what?

**A)** A metaclass customization hook
**B) A shortcut combining an infix operator with variable assignment ✓ Correct**
**C)** A special case of Boolean coercion
**D)** A debugging-only representation feature

*Rationale: The text defines augmented assignment as a shortcut combining an infix operator with assignment, such as +=.*

**Q44.**
According to Raymond Hettinger's explanation summarized in the chapter, why is len not a method?

**A)** Because methods cannot return integers efficiently
**B) Because practicality and performance for built-in types outweigh pure method syntax ✓ Correct**
**C)** Because len applies only to strings and lists
**D)** Because the Zen of Python forbids methods on collections

*Rationale: The chapter says len gets special treatment for efficiency with built-in types, reflecting practicality over purity.*

**Q45.**
Which statement from The Zen of Python is explicitly quoted in the discussion of len?

**A)** Readability counts
**B)** Explicit is better than implicit
**C) Practicality beats purity ✓ Correct**
**D)** Simple is better than complex

*Rationale: The explanation of len not being a method cites "practicality beats purity."*

**Q46.**
The chapter suggests that thinking of abs and len as what may make their functional syntax seem more acceptable?

**A)** Decorators
**B) Unary operators ✓ Correct**
**C)** Generators
**D)** Descriptors

*Rationale: The text says that viewing abs and len as unary operators helps justify their functional form.*

**Q47.**
In the ABC language mentioned in the chapter, which symbol served as the equivalent of len?

**A)** @
**B)** &
**C) # ✓ Correct**
**D)** $

*Rationale: The chapter notes that ABC used the # operator as the equivalent of len.*

**Q48.**
What term does the chapter use as a synonym for interface when discussing a metaobject protocol?

**A)** Descriptor
**B) Protocol ✓ Correct**
**C)** Adapter
**D)** Contractor

*Rationale: The chapter explicitly says that in this context, protocol is a synonym for interface.*

**Q49.**
According to the chapter's "Soapbox" section, many authors would more commonly call the Python Data Model the Python what?

**A)** Package index
**B)** Compilation pipeline
**C) Object model ✓ Correct**
**D)** Import graph

*Rationale: The author notes that many authors refer to the Python Data Model as the Python object model.*

**Q50.**
Which framework is cited as an important example of aspect-oriented programming support in Python?

**A)** asyncio
**B) zope.interface ✓ Correct**
**C)** tkinter
**D)** unittest

*Rationale: The chapter names zope.interface as the most important example in that discussion.*

**Q51.**
A health analytics team creates a custom QueryResult class to hold rows returned from an internal reporting engine. They want analysts to use len(result) without learning a custom API. Which special method should the class implement?

**A)** __iter__
**B)** __contains__
**C) __len__ ✓ Correct**
**D)** __repr__

*Rationale: Implementing __len__ allows the built-in len() function to return the number of items in the custom object.*

**Q52.**
A pharmacy informatics developer wants MedicationQueue objects to support bracket access such as queue[0] and queue[-1]. Which special method is required?

**A) __getitem__ ✓ Correct**
**B)** __getattr__
**C)** __call__
**D)** __index__

*Rationale: The interpreter uses __getitem__ to support item access with bracket syntax.*

**Q53.**
A nursing education app uses a custom FlashcardDeck class that only implements __len__ and __getitem__. Which additional capability will it gain automatically from those methods?

**A)** Database persistence
**B) Implicit sequence-style iteration ✓ Correct**
**C)** Network serialization
**D)** Thread synchronization

*Rationale: A class with __getitem__ can be iterated by Python even without an explicit __iter__ method.*

**Q54.**
A clinical software engineer wants a custom VitalsVector object to display as VitalsVector(120, 80) in debugging sessions. Which special method is most appropriate?

**A)** __str__
**B)** __bytes__
**C) __repr__ ✓ Correct**
**D)** __format__

*Rationale: __repr__ provides the inspection-oriented representation used by the console and debugger.*

**Q55.**
A developer is documenting a custom object and refers to __getitem__ as a “dunder” method. In this context, “dunder” means the method name has what characteristic?

**A)** It is inherited from a data class
**B) It has leading and trailing double underscores ✓ Correct**
**C)** It is called only by decorators
**D)** It is available only in asynchronous code

*Rationale: “Dunder” is shorthand for names with double underscores before and after the identifier.*

**Q56.**
A biomedical engineer defines __bool__ for a SensorVector class. What is Python expecting that method to return?

**A)** Any numeric magnitude
**B)** Any nonempty string
**C) A Boolean value ✓ Correct**
**D)** A tuple of truth states

*Rationale: __bool__ is expected to return True or False.*

**Q57.**
A health IT instructor asks why Python uses len(obj) instead of obj.len() for standard collections. Which explanation best matches the chapter?

**A)** It avoids object-oriented design entirely
**B) It allows CPython to optimize built-in variable-sized collections efficiently ✓ Correct**
**C)** It is required only for asynchronous objects
**D)** It prevents subclasses from overriding length behavior

*Rationale: len() gets special treatment so built-in collections can provide length efficiently, often without a method call.*

**Q58.**
A custom scheduling class implements __len__ but not __bool__. How will Python determine truthiness for an instance?

**A)** It will always be True because all objects are truthy
**B)** It will call __repr__ and check whether the string is empty
**C) It will use __len__ and treat zero length as False ✓ Correct**
**D)** It will raise a TypeError in Boolean contexts

*Rationale: If __bool__ is absent, Python falls back to __len__, with zero meaning False.*

**Q59.**
A health data science team wants a custom Matrix class to support the @ operator. Which special method corresponds to that operator?

**A)** __mul__
**B) __matmul__ ✓ Correct**
**C)** __and__
**D)** __pow__

*Rationale: The @ operator is implemented by the __matmul__ special method.*

**Q60.**
A PMHNP informatics student asks which collection abstract base class combines Iterable, Sized, and Container. Which is correct?

**A)** Sequence
**B)** Mapping
**C)** Set
**D) Collection ✓ Correct**

*Rationale: Collection unifies the essential interfaces of Iterable, Sized, and Container.*

**Q61.**
A hospital reporting module stores a format template in a configuration table because it must be reused in multiple places. Which formatting approach from the chapter remains especially useful in that situation?

**A)** Only f-strings
**B) The str.format() method ✓ Correct**
**C)** Only %-formatting
**D)** The repr() built-in

*Rationale: str.format() is still useful when the format string must be defined separately from where formatting occurs.*

**Q62.**
A clinical developer writes if result_set: for a custom class that has neither __bool__ nor __len__. What outcome should be expected?

**A) The object is considered truthy by default ✓ Correct**
**B)** The object is considered falsy by default
**C)** Python calls __iter__ to decide truthiness
**D)** Python raises an AttributeError

*Rationale: User-defined instances are truthy by default unless __bool__ or __len__ changes that behavior.*

**Q63.**
A laboratory application uses print(sample_vector) for end-user display and repr(sample_vector) during debugging. Which statement best distinguishes these operations?

**A)** Both always call __str__
**B)** print uses __repr__ only, while repr uses __format__
**C) str/print target user-friendly display, while repr targets unambiguous inspection ✓ Correct**
**D)** repr is for localization, while str is for serialization

*Rationale: __str__ is intended for end-user display, whereas __repr__ is for inspection and debugging.*

**Q64.**
A healthcare developer manually writes my_object.__len__() throughout the codebase instead of len(my_object). Based on the chapter, what is the better practice?

**A)** Use __len__ directly because it is more explicit
**B) Use len(my_object) because built-ins are the intended interface and may provide extra benefits ✓ Correct**
**C)** Use my_object.length() because it is more Pythonic
**D)** Use bool(my_object) to infer length

*Rationale: The related built-in function is the preferred interface because it is idiomatic and may be optimized.*

**Q65.**
A custom InsuranceDeck class delegates __getitem__ to an internal list. A claims analyst uses deck[:3]. Why does slicing work?

**A)** Because __len__ automatically creates slices
**B) Because __getitem__ receives slice objects as well as integer indexes ✓ Correct**
**C)** Because slicing is available only on namedtuple instances
**D)** Because reversed() enables slice semantics

*Rationale: Slicing works because __getitem__ can be called with a slice, which the delegated list handles.*

**Q66.**
A population health tool defines a Vector class with __add__ and __mul__. During code review, which behavior is most consistent with Python infix operators?

**A)** The left operand should always be modified in place
**B)** Both operands should be deleted after calculation
**C) A new object should be returned without modifying the operands ✓ Correct**
**D)** The operator should return None after updating internal state

*Rationale: Infix operators are expected to create and return new objects rather than mutate their operands.*

**Q67.**
A healthcare simulation uses abs(v) on a custom Vector object to compute magnitude. Which special method should provide that behavior?

**A)** __index__
**B) __abs__ ✓ Correct**
**C)** __float__
**D)** __round__

*Rationale: The abs() built-in invokes the __abs__ special method.*

**Q68.**
An EHR integration class represents records as simple bundles of fields with no custom methods. Which construct from the chapter is presented as suitable for that purpose?

**A) collections.namedtuple ✓ Correct**
**B)** collections.deque
**C)** collections.abc.Mapping
**D)** memoryview

*Rationale: namedtuple is appropriate for simple attribute bundles such as record-like objects.*

**Q69.**
A health app developer wants a custom class to support the in operator efficiently. Which special method directly formalizes that interface?

**A) __contains__ ✓ Correct**
**B)** __iter__
**C)** __len__
**D)** __setitem__

*Rationale: __contains__ is the special method associated with membership testing using in.*

**Q70.**
A custom formulary collection does not implement __contains__, but it does support iteration. What happens when code checks item in formulary?

**A) Python performs a sequential scan through the iterable ✓ Correct**
**B)** Python refuses membership tests without __contains__
**C)** Python checks only the first element
**D)** Python converts the object to a dict before searching

*Rationale: Without __contains__, Python can fall back to scanning the iterable sequentially for membership.*

**Q71.**
A hospital inventory class implements __getitem__ and __len__ but not a shuffle-related mutation method. The team finds random.shuffle fails. According to the chapter, what is the most relevant reason?

**A)** The object lacks __repr__
**B) The object behaves as immutable because positions cannot be changed through the public interface ✓ Correct**
**C)** The object must inherit from Sequence explicitly
**D)** The object cannot be iterated in reverse

*Rationale: Without a way to assign items, the sequence is effectively immutable and cannot be shuffled in place.*

**Q72.**
A coding bootcamp for HIM professionals compares f-strings with older formatting styles. Which statement best reflects the chapter's guidance?

**A) f-strings are generally more readable and convenient ✓ Correct**
**B)** The % operator is always preferred for custom classes
**C)** str.format() has replaced repr()
**D)** f-strings cannot use conversions like !r

*Rationale: The chapter notes that f-strings are often more readable and convenient than older formatting approaches.*

**Q73.**
A data engineer wants a custom object to display attribute values in a way that distinguishes numeric 1 from string '1' during debugging. Which technique from the chapter supports that goal?

**A) Use !r in the f-string inside __repr__ ✓ Correct**
**B)** Use print() inside __init__
**C)** Use __str__ only
**D)** Convert every attribute to bytes

*Rationale: Using !r in __repr__ shows the standard representation of attributes, preserving distinctions like numbers versus strings.*

**Q74.**
A telehealth scheduling API includes a class that must support async for iteration over streamed events. Which special method category from the chapter is directly relevant?

**A) Asynchronous iteration methods such as __aiter__ and __anext__ ✓ Correct**
**B)** Bitwise operator methods such as __and__
**C)** String conversion methods such as __bytes__
**D)** Descriptor methods such as __set_name__

*Rationale: Async iteration relies on asynchronous special methods including __aiter__ and __anext__.*

**Q75.**
A nurse informaticist reads that only one major collection specialization is Reversible in the chapter's overview. Which one is it?

**A)** Mapping
**B)** Set
**C) Sequence ✓ Correct**
**D)** Collection

*Rationale: Only Sequence is described as Reversible because sequences have arbitrary ordering of contents.*

**Q76.**
A developer stores patient tasks in a custom class and wants reversed(tasks) to work through sequence behavior. Which existing capability most directly supports that possibility in the chapter's example?

**A)** A custom __dir__ implementation
**B) Indexed access through __getitem__ ✓ Correct**
**C)** A custom __hash__ implementation
**D)** A __call__ implementation

*Rationale: The example shows reverse iteration working because the object behaves like a sequence through indexed access.*

**Q77.**
A public health analyst sorts a custom deck-like collection with sorted(deck, key=rank_func). Which design choice from the chapter makes this easy without writing a custom sort method on the class?

**A) Implementing sequence behavior so standard library functions can operate on the object ✓ Correct**
**B)** Subclassing bool to control comparisons
**C)** Defining __delitem__ to remove unsorted items
**D)** Using __new__ instead of __init__

*Rationale: By behaving like a standard sequence, the object can be used directly with standard library tools such as sorted().*

**Q78.**
A biomedical software team is debating whether a custom sequence must inherit from collections.abc.Sequence to work with len() and indexing. Which answer is most accurate?

**A)** Yes, inheritance from the ABC is mandatory
**B) No, implementing the relevant special methods is sufficient ✓ Correct**
**C)** Yes, but only in Python 3.7 and later
**D)** No, because Python ignores special methods on user classes

*Rationale: Python does not require inheritance from the ABCs if the class implements the needed special methods.*

**Q79.**
A custom RiskVector class defines __mul__ but not __rmul__. Which limitation should the team expect based on the chapter?

**A)** vector * number will fail, but number * vector will work
**B)** Both multiplication forms will fail
**C) vector * number may work, but number * vector may not ✓ Correct**
**D)** Multiplication will mutate the vector in place

*Rationale: Without __rmul__, right-hand scalar multiplication may be unsupported even if left-hand multiplication works.*

**Q80.**
A machine learning engineer in radiology manually calls dataset.__iter__() in routine code instead of iter(dataset). What guidance from the chapter best applies?

**A)** Direct calls are preferred because they bypass the interpreter
**B) Use iter(dataset), since built-ins are the intended public interface ✓ Correct**
**C)** Use len(dataset) to start iteration
**D)** Use __getitem__ directly because it is always faster

*Rationale: The chapter advises using the built-in function rather than directly invoking special methods.*

**Q81.**
A quality improvement team creates a custom collection for incident reports. They want tuple unpacking and for-loops to work idiomatically. Which interface from Collection is most directly involved?

**A)** Container
**B)** Sized
**C) Iterable ✓ Correct**
**D)** Hashable

*Rationale: Iterable is the interface that supports for-loops, unpacking, and other forms of iteration.*

**Q82.**
A software architect says dict is ordered in modern Python, so it should be treated like a freely reorderable sequence. Why is that reasoning flawed according to the chapter?

**A) dict preserves insertion order but does not support arbitrary rearrangement like a sequence ✓ Correct**
**B)** dict ordering exists only in interactive mode
**C)** dict ordering requires __reversed__ to be defined manually
**D)** dict is a Set specialization, not a Mapping

*Rationale: The chapter notes that ordered dict means insertion order is preserved, not that keys can be rearranged arbitrarily.*

**Q83.**
A developer is choosing between implementing only __str__ or only __repr__ on a custom clinical object. If only one is implemented, which should be chosen?

**A)** __str__
**B) __repr__ ✓ Correct**
**C)** __format__
**D)** __bytes__

*Rationale: The chapter explicitly recommends implementing __repr__ if only one of the two is provided.*

**Q84.**
A custom object is shown in the interactive console immediately after evaluation. Which representation mechanism is typically used there?

**A)** __str__ via print semantics
**B) __repr__ via repr semantics ✓ Correct**
**C)** __format__ via f-string semantics
**D)** __bytes__ via encoding semantics

*Rationale: The interactive console typically displays the repr of evaluated expressions.*

**Q85.**
A healthcare startup wants its custom cache object to support with statements for managed resource handling. Which special methods are relevant?

**A) __enter__ and __exit__ ✓ Correct**
**B)** __iter__ and __next__
**C)** __getattr__ and __setattr__
**D)** __add__ and __iadd__

*Rationale: Managed contexts using with rely on __enter__ and __exit__.*

**Q86.**
A genomics pipeline object needs to support await in asynchronous workflows. Which special method category from the chapter applies?

**A)** Callable execution via __call__ only
**B) Coroutine execution via __await__ ✓ Correct**
**C)** Collection emulation via __getitem__
**D)** Comparison via __lt__

*Rationale: The await expression relies on the __await__ special method.*

**Q87.**
A student claims special methods are “magic” because they are undocumented internal tricks. Which response is most aligned with the chapter?

**A)** They are intentionally hidden and unsupported
**B) They form a documented metaobject protocol that users can implement ✓ Correct**
**C)** They are available only to CPython core developers
**D)** They are valid only for numeric classes

*Rationale: The chapter emphasizes that special methods are documented interfaces of Python’s data model, not hidden tricks.*

**Q88.**
A healthcare operations platform needs a custom class whose instances can be called like functions to trigger a workflow. Which special method should be implemented?

**A) __call__ ✓ Correct**
**B)** __invoke__
**C)** __trigger__
**D)** __exec__

*Rationale: Implementing __call__ allows instances to be invoked with function-call syntax.*

**Q89.**
A custom sequence-like class is built by composing an internal list and delegating behavior to it. Which design principle from the FrenchDeck example does this illustrate?

**A)** Inheritance from built-in list is mandatory
**B) Composition can provide sequence behavior by forwarding to an internal object ✓ Correct**
**C)** Metaclasses are required for indexing
**D)** Descriptors are necessary for slicing

*Rationale: The chapter highlights composition, with special methods delegating work to an internal list.*

**Q90.**
A hospital app defines a __repr__ that returns '<Task object>'. During debugging, engineers complain it is not very useful. Which improvement best follows the chapter’s guidance?

**A)** Return a string that is ambiguous but shorter
**B) Return a string matching, when possible, the source needed to recreate the object ✓ Correct**
**C)** Return the same as __bool__
**D)** Return only the memory address

*Rationale: A good __repr__ should be unambiguous and ideally resemble code that can recreate the object.*

**Q91.**
A custom vector class in a rehabilitation app defines __bool__ as return bool(abs(self)). What business rule does that encode?

**A)** Vectors are always truthy after initialization
**B)** Only vectors with negative coordinates are falsy
**C) A zero-magnitude vector is falsy and a nonzero vector is truthy ✓ Correct**
**D)** Truthiness depends on the string representation

*Rationale: Converting the magnitude to bool makes zero vectors False and nonzero vectors True.*

**Q92.**
A performance-conscious developer proposes changing Vector.__bool__ from bool(abs(self)) to bool(self.x or self.y). Why might this be faster?

**A) It avoids computing magnitude through squares and square root ✓ Correct**
**B)** It adds caching automatically
**C)** It converts the vector to an integer first
**D)** It bypasses attribute access entirely

*Rationale: Checking x or y avoids the more expensive magnitude calculation path.*

**Q93.**
A data governance team wants a custom policy object to participate in membership tests, length checks, and iteration. Which trio of interfaces is unified by Collection?

**A)** Callable, Awaitable, Reversible
**B) Iterable, Sized, Container ✓ Correct**
**C)** Mapping, Sequence, Set
**D)** Hashable, Comparable, Numeric

*Rationale: Collection combines the Iterable, Sized, and Container interfaces.*

**Q94.**
A student reviewing operator methods asks when Python uses a reversed operator such as __radd__. Which answer is correct?

**A)** Only when both operands are built-in types
**B) When the first operand's corresponding method cannot be used ✓ Correct**
**C)** Whenever augmented assignment is written
**D)** Only after __iadd__ succeeds

*Rationale: Python tries the reversed operator method on the second operand when the first operand's method cannot handle the operation.*

**Q95.**
A healthcare finance system uses += on a custom MoneyVector object. In the chapter's operator overview, this kind of syntax is classified as what?

**A)** Rich comparison
**B)** Unary numeric operation
**C) Augmented assignment ✓ Correct**
**D)** Reversed arithmetic

*Rationale: Operators like += are augmented assignment forms.*

**Q96.**
A developer building a custom set-like class for terminology matching wants support for expressions such as a & b. Which special method corresponds to that operator?

**A)** __or__
**B)** __xor__
**C) __and__ ✓ Correct**
**D)** __contains__

*Rationale: The & operator is implemented by the __and__ special method.*

**Q97.**
A research platform defines __getitem__ but not __iter__. During a for loop, what interpreter behavior from the chapter may still allow iteration?

**A)** The loop calls len() repeatedly until it reaches zero
**B) The loop invokes iter(x), which may fall back to x.__getitem__() ✓ Correct**
**C)** The loop converts the object to a set automatically
**D)** The loop requires __contains__ and otherwise fails

*Rationale: Python iteration may fall back to __getitem__ when __iter__ is unavailable.*

**Q98.**
A software engineer wants to use undocumented __*__ names for custom internal hooks because they look similar to special methods. What caution from the chapter is most relevant?

**A) Undocumented __*__ names may break without warning ✓ Correct**
**B)** Undocumented names are automatically optimized by CPython
**C)** Undocumented names are reserved only for abstract base classes
**D)** Undocumented names are safer than public methods

*Rationale: The language reference warns that undocumented uses of __*__ names are subject to breakage without warning.*

**Q99.**
A custom patient queue implements __len__ and __getitem__. An engineer argues that most of its useful behavior is inherited from object. Which response best matches the chapter's explanation of FrenchDeck-like classes?

**A) Most useful collection behavior comes from leveraging the data model, not from object inheritance ✓ Correct**
**B)** All sequence behavior is inherited automatically from object
**C)** The class works only because namedtuple injects methods into it
**D)** Its behavior depends primarily on metaclass customization

*Rationale: The chapter explains that sequence-like behavior comes from implementing special methods and composition, not from inheriting behavior from object.*

**Q100.**

A hospital innovation lab is deciding whether Python's data model is best thought of as an API for core language constructs. Which interpretation is most consistent with the chapter?

**A)** It is only a naming convention for private methods

**B) It is a framework-like interface describing how objects interact with language features ✓ Correct**

**C)** It is limited to numeric overloading only

**D)** It applies only to built-in classes written in C

*Rationale: The chapter describes the Python Data Model as a framework-like API for core language building blocks and behaviors.*