Python Magic Methods with Examples

---

1. Object Creation & Initialization

```
class Example:
    def __new__(cls, *args, **kwargs):
        print("Creating instance with __new__")
        instance = super().__new__(cls)
        return instance

    def __init__(self, value):
        print("Initializing instance with __init__")
        self.value = value

    def __del__(self):
        print(f"Destroying instance with value {self.value}")

obj = Example(10)
del obj
```

---

2. String & Representation

```
class Person:
    def __init__(self, name, age):
        self.name, self.age = name, age

    def __str__(self):      # Human-friendly
        return f"{self.name}, {self.age} years old"
```

```
    def __repr__(self):     # Developer-friendly
        return f"Person(name='{self.name}', age={self.age})"

    def __format__(self, spec):
        return f"{self.name:{spec}} ({self.age})"

    def __bytes__(self):
        return bytes(f"{self.name}:{self.age}", "utf-8")

p = Person("Alice", 25)
print(str(p))       # Alice, 25 years old
print(repr(p))      # Person(name='Alice', age=25)
print(f"{p:>10}")   # Right aligned name (format spec)
print(bytes(p))     # b'Alice:25'
```

---

3. Comparison Operators

```
class Box:
    def __init__(self, volume):
        self.volume = volume

    def __eq__(self, other): return self.volume == other.volume
    def __ne__(self, other): return self.volume != other.volume
    def __lt__(self, other): return self.volume < other.volume
    def __le__(self, other): return self.volume <= other.volume
    def __gt__(self, other): return self.volume > other.volume
```

```
    def __ge__(self, other): return self.volume >= other.volume


b1, b2 = Box(10), Box(20)
print(b1 < b2)   # True
print(b1 == b2)  # False
print(b1 != b2)  # True
```

---

4. Arithmetic Operators

```
class Number:
    def __init__(self, value):
        self.value = value

    def __add__(self, other): return Number(self.value + other.value)
    def __sub__(self, other): return Number(self.value - other.value)
    def __mul__(self, other): return Number(self.value * other.value)
    def __truediv__(self, other): return Number(self.value / other.value)
    def __floordiv__(self, other): return Number(self.value // other.value)
    def __mod__(self, other): return Number(self.value % other.value)
    def __pow__(self, other): return Number(self.value ** other.value)

    # Reflected
    def __radd__(self, other): return Number(other + self.value)

    # In-place
    def __iadd__(self, other):
        self.value += other.value
```

```
        return self

    def __repr__(self): return f"Number({self.value})"

a, b = Number(10), Number(3)
print(a + b)   # Number(13)
print(5 + a)   # Number(15) -> __radd__
a += b
print(a)       # Number(13) -> __iadd__
```

---

5. Unary Operators

```
class Vector:
    def __init__(self, x): self.x = x
    def __neg__(self): return Vector(-self.x)
    def __pos__(self): return Vector(+self.x)
    def __abs__(self): return abs(self.x)
    def __invert__(self): return ~self.x
    def __repr__(self): return f"Vector({self.x})"

v = Vector(5)
print(-v)       # Vector(-5)
print(+v)       # Vector(5)
print(abs(v))   # 5
print(~v)       # -6 (bitwise NOT of 5)
```

---

6. Container & Sequence Protocol

```
class MyList:
    def __init__(self, data):
        self.data = data

    def __len__(self): return len(self.data)
    def __getitem__(self, index): return self.data[index]
    def __setitem__(self, index, value): self.data[index] = value
    def __delitem__(self, index): del self.data[index]
    def __iter__(self): return iter(self.data)
    def __contains__(self, item): return item in self.data
    def __reversed__(self): return reversed(self.data)

lst = MyList([1, 2, 3])
print(len(lst))       # 3
print(lst[1])         # 2
lst[1] = 20
print(lst[1])         # 20
del lst[0]
print(list(lst))      # [20, 3]
print(20 in lst)      # True
print(list(reversed(lst)))  # [3, 20]
```

---

7. Attribute Access

```
class Demo:
    def __init__(self):
        self.x = 10
```

```
    def __getattr__(self, name):   # Only if missing
        return f"{name} not found"

    def __getattribute__(self, name):   # Always
        print(f"Accessing {name}")
        return super().__getattribute__(name)

    def __setattr__(self, name, value):
        print(f"Setting {name} = {value}")
        super().__setattr__(name, value)

    def __delattr__(self, name):
        print(f"Deleting {name}")
        super().__delattr__(name)

    def __dir__(self):
        return ['x', 'y', 'z']

d = Demo()
print(d.x)       # Logs attribute access
print(d.unknown) # "unknown not found"
d.y = 50         # Logs setting
del d.x          # Logs deleting
print(dir(d))    # ['x', 'y', 'z']
```

---

8. Callable Objects

```
class Greeter:
    def __call__(self, name):
        return f"Hello, {name}!"

g = Greeter()
print(g("Alice"))  # Hello, Alice!
```

---

9. Context Managers

```
class FileManager:
    def __init__(self, filename):
        self.filename = filename

    def __enter__(self):
        self.file = open(self.filename, "w")
        return self.file

    def __exit__(self, exc_type, exc_value, traceback):
        self.file.close()
        print("File closed")

with FileManager("test.txt") as f:
    f.write("Hello!")
```

---

10. Descriptor Protocol

```
class Descriptor:
```

```
    def __get__(self, instance, owner):
        return instance._value
    def __set__(self, instance, value):
        instance._value = value
    def __delete__(self, instance):
        del instance._value

class MyClass:
    attr = Descriptor()

obj = MyClass()
obj.attr = 42
print(obj.attr)   # 42
del obj.attr
```

---

11. Other Useful Magic Methods

```
import copy, sys

class Example:
    def __init__(self, value): self.value = value
    def __hash__(self): return hash(self.value)
    def __bool__(self): return bool(self.value)
    def __index__(self): return int(self.value)
    def __copy__(self): return Example(self.value)
    def __deepcopy__(self, memo): return Example(copy.deepcopy(self.value, memo))
    def __sizeof__(self): return sys.getsizeof(self.value)
```

```
e = Example(10)

print(hash(e))         # Hash value

print(bool(e))         # True

print(hex(e))          # Uses __index__

print(copy.copy(e))    # Shallow copy

print(sys.getsizeof(e)) # Custom size
```

Section A: Theory (10 questions)

1. What is the purpose of the __init__ method in a Python class?
2. How does __str__ differ from __repr__?
3. Which magic method allows an object to be called like a function?
4. What is the difference between __add__ and __radd__?
5. When does Python call the __del__ method, and why should it be used carefully?
6. Which magic method is triggered when you use the in keyword (item in obj)?
7. Explain the role of __enter__ and __exit__ in context managers.
8. How does __getattr__ differ from __getattribute__?
9. Which magic method would you implement to make your class compatible with len()?
10. What is the purpose of the __hash__ magic method?

Section B: Multiple Choice (5 questions)

11. Which method is used for object creation (before initialization)?
    a) __init__
    b) __new__
    c) __create__
    d) __start__
12. Which of the following is **not** a comparison magic method?
    a) __lt__
    b) __eq__

c) __gt__
d) __sum__

13. What does the __iter__ method return?
    a) A boolean
    b) An iterator
    c) A string
    d) A dictionary

14. Which magic method allows slicing and indexing (obj[0])?
    a) __getitem__
    b) __setitem__
    c) __contains__
    d) __index__

15. If you want to customize truth value testing (if obj:), which method should you override?
    a) __bool__
    b) __truth__
    c) __true__
    d) __len__ only

Section C: Coding (5 questions)

16. Write a class Point that supports addition (+) of two points (x, y).

17. Write a class Person that returns "Person(name, age)" when printed (using __str__).

18. Write a class Counter that supports len(obj) to return the number of items.

19. Write a class Greeter where g("Alice") prints "Hello, Alice!".

20. Write a class ReverseList that behaves like a list but always iterates in reverse order.

🚀 Capstone Projects on Python Magic Methods

---

1. Mini Vector Mathematics Library

🔷 **Goal**: Build a Vector class that behaves like a mathematical vector.

Requirements:

- Support arithmetic operators (+, -, *, //, /, %, **).

- Implement comparison operators (==, <, >, etc.) based on vector magnitude.
- Implement __len__ to return the dimension of the vector.
- Implement __getitem__ and __setitem__ for indexing.
- Implement __str__ and __repr__ for clean output.

Example Usage:

```
v1 = Vector([1, 2, 3])

v2 = Vector([4, 5, 6])

print(v1 + v2)        # Vector([5, 7, 9])

print(len(v1))        # 3

print(v1 < v2)        # True (based on magnitude)

print(v1[0])          # 1
```

---

2. Custom Dictionary with Logging

🔷 **Goal**: Create a LoggedDict class that works like a Python dictionary but logs every access, update, and deletion.

Requirements:

- Override __getitem__, __setitem__, __delitem__.
- Implement __contains__ for in checks.
- Implement __iter__ to iterate keys.
- Override __str__ for pretty-printing the dictionary.
- Bonus: Support context management (with LoggedDict() as d: → automatically saves to file).

Example Usage:

```
d = LoggedDict()

d["a"] = 10   # Logs: Setting a=10

print(d["a"]) # Logs: Getting key a
```

```
print("a" in d)  # True
del d["a"]   # Logs: Deleting key a
```

---

3. Smart Bank Account System

🔷 **Goal**: Create a BankAccount class with rich behavior.

Requirements:

- Implement __add__ to allow transferring money between accounts (acc1 + acc2 merges balances).
- Implement __iadd__ for deposits (acc += amount).
- Implement __sub__ for withdrawals (acc - amount).
- Implement __eq__ and __lt__ to compare accounts by balance.
- Implement __str__ and __repr__ for display.
- Implement __call__ to quickly check balance (acc() returns balance).
- Implement __enter__ and __exit__ so accounts can be used in with (e.g., auto-close session).

Example Usage:

```
acc1 = BankAccount("Alice", 1000)
acc2 = BankAccount("Bob", 500)

acc1 += 200        # Deposit
acc1 - 100         # Withdraw
print(acc1 == acc2) # Compare balances
print(acc1())      # Check balance
with acc1:
   print("Bank session open")
```

👉 These projects combine **multiple magic methods** at once, just like Python's built-in types (list, dict, int).