Table of contents

1

1.1

-
-
-
-
-

```
#
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name}    ")

#
class Dog(Animal):  # Animal
    def speak(self):  #
        print(f"{self.name}      ")

#
dog = Dog(" ")
dog.speak()  #
```

1.2

```
class Vehicle:  #
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
```

```
        self.year = year
        self.is_running = False

    def start(self):
        self.is_running = True
        print(f"{self.brand} {self.model}    ")

    def stop(self):
        self.is_running = False
        print(f"{self.brand} {self.model}    ")

    def info(self):
        return f"{self.year}  {self.brand} {self.model}"

class Car(Vehicle):  #
    def __init__(self, brand, model, year, doors):
        super().__init__(brand, model, year)  #
        self.doors = doors  #

    def honk(self):  #
        print("   ")

    def info(self):  #
        base_info = super().info()  #
        return f"{base_info} ({self.doors} )"

#
my_car = Car("Toyota", "Prius", 2023, 4)
print(my_car.info())  # 2023  Toyota Prius (4 )
my_car.start()        # Toyota Prius
my_car.honk()         #
```

1.3 super()

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def work(self):
```

```
        print(f"{self.name}    ")

    def get_annual_salary(self):
        return self.salary * 12

class Manager(Employee):
    def __init__(self, name, salary, team_size):
        super().__init__(name, salary)  #
        self.team_size = team_size

    def work(self):
        super().work()  #
        print(f"  {self.team_size}       ")

    def get_annual_salary(self):
        base_salary = super().get_annual_salary()
        bonus = base_salary * 0.2  # 20%
        return base_salary + bonus

class Developer(Employee):
    def __init__(self, name, salary, programming_language):
        super().__init__(name, salary)
        self.programming_language = programming_language

    def work(self):
        print(f"{self.name} {self.programming_language}       ")

    def code_review(self):
        print(f"{self.name}        ")

#
manager = Manager("  ", 80000, 5)
developer = Developer("    ", 60000, "Python")

manager.work()
print(f" : {manager.get_annual_salary():,} ")

developer.work()
developer.code_review()
```

1.4

```
class Flyable:
    def fly(self):
        print("    ")

class Swimmable:
    def swim(self):
        print("   ")

class Duck(Animal, Flyable, Swimmable):  #
    def __init__(self, name):
        super().__init__(name)

    def speak(self):
        print(f"{self.name}      ")

class Penguin(Animal, Swimmable):  #
    def __init__(self, name):
        super().__init__(name)

    def speak(self):
        print(f"{self.name}      ")

#
duck = Duck("  ")
duck.speak()  #
duck.fly()    #
duck.swim()   #

penguin = Penguin("  ")
penguin.speak()  #
penguin.swim()   #
# penguin.fly()  # AttributeError:
```

1.5 MRO

```
class A:
    def method(self):
```

```
        print("A")

class B(A):
    def method(self):
        print("B")
        super().method()

class C(A):
    def method(self):
        print("C")
        super().method()

class D(B, C):  #
    def method(self):
        print("D")
        super().method()

# MRO Method Resolution Order
print(D.__mro__)
# (<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <
d = D()
d.method()
# D
# B
# C
# A
```

1.6

```
class Shape:
    def area(self):
        raise NotImplementedError("        ")

    def perimeter(self):
        raise NotImplementedError("        ")

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
```

```
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14159 * self.radius ** 2

    def perimeter(self):
        return 2 * 3.14159 * self.radius

#
def print_shape_info(shape):  #  Shape
    print(f" : {shape.area():.2f}")
    print(f" : {shape.perimeter():.2f}")
    print()

#
shapes = [
    Rectangle(5, 3),
    Circle(4),
    Rectangle(2, 8)
]

for shape in shapes:
    print_shape_info(shape)  #
```

1.7 ABC

```
from abc import ABC, abstractmethod

class PaymentProcessor(ABC):
    @abstractmethod
```

```
    def process_payment(self, amount):
        pass

    @abstractmethod
    def validate_payment(self, payment_data):
        pass

    def log_transaction(self, amount):  #
        print(f"  : {amount}  ")

class CreditCardProcessor(PaymentProcessor):
    def process_payment(self, amount):
        print(f"     {amount}  ")
        self.log_transaction(amount)

    def validate_payment(self, payment_data):
        #
        card_number = payment_data.get('card_number', '')
        return len(card_number) == 16

class PayPalProcessor(PaymentProcessor):
    def process_payment(self, amount):
        print(f"PayPal {amount}  ")
        self.log_transaction(amount)

    def validate_payment(self, payment_data):
        # PayPal
        email = payment_data.get('email', '')
        return '@' in email

#
def make_payment(processor, amount, payment_data):
    if processor.validate_payment(payment_data):
        processor.process_payment(amount)
    else:
        print("     ")

credit_processor = CreditCardProcessor()
paypal_processor = PayPalProcessor()

make_payment(credit_processor, 1000, {'card_number': '1234567890123456'})
make_payment(paypal_processor, 500, {'email': 'user@example.com'})
```

```
#
# processor = PaymentProcessor()  # TypeError!
```

1.8

1.9 :

```
from abc import ABC, abstractmethod

class Animal(ABC):
    def __init__(self, name, age, habitat):
        self.name = name
        self.age = age
        self.habitat = habitat
        self.hunger_level = 50  # 0-100
        self.health = 100

    @abstractmethod
    def make_sound(self):
        pass

    @abstractmethod
    def eat(self, food_type):
        pass

    def sleep(self):
        self.health = min(100, self.health + 10)
        print(f"{self.name}      : {self.health}")

    def __str__(self):
        return f"{self.name} ({self.__class__.__name__}) -  : {self.age},   : {self.health}"

class Mammal(Animal):
    def __init__(self, name, age, habitat, fur_color):
        super().__init__(name, age, habitat)
        self.fur_color = fur_color

    def groom(self):
```

```
        print(f"{self.name}      ")

class Bird(Animal):
    def __init__(self, name, age, habitat, wingspan):
        super().__init__(name, age, habitat)
        self.wingspan = wingspan
        self.can_fly = True

    def fly(self):
        if self.can_fly and self.health > 30:
            print(f"{self.name}      : {self.wingspan}cm ")
        else:
            print(f"{self.name}    ")

class Lion(Mammal):
    def __init__(self, name, age, mane_size="medium"):
        super().__init__(name, age, "   ", " ")
        self.mane_size = mane_size

    def make_sound(self):
        print(f"{self.name}      ")

    def eat(self, food_type):
        if food_type == " ":
            self.hunger_level = max(0, self.hunger_level - 30)
            self.health = min(100, self.health + 5)
            print(f"{self.name} {food_type}          ")
        else:
            print(f"{self.name} {food_type}    ")

    def hunt(self):
        if self.hunger_level > 70:
            print(f"{self.name}     ")
            self.hunger_level -= 20
        else:
            print(f"{self.name}        ")

class Elephant(Mammal):
    def __init__(self, name, age, trunk_length=150):
        super().__init__(name, age, "   ", " ")
        self.trunk_length = trunk_length
```

```
    def make_sound(self):
        print(f"{self.name}       ")

    def eat(self, food_type):
        if food_type in [" ", " ", " "]:
            self.hunger_level = max(0, self.hunger_level - 25)
            self.health = min(100, self.health + 3)
            print(f"{self.name} {food_type}    ")
        else:
            print(f"{self.name} {food_type}    ")

    def spray_water(self):
        print(f"{self.name}      ")

class Eagle(Bird):
    def __init__(self, name, age, wingspan=200):
        super().__init__(name, age, " ", wingspan)
        self.hunting_skill = 80

    def make_sound(self):
        print(f"{self.name}        ")

    def eat(self, food_type):
        if food_type in [" ", "  "]:
            self.hunger_level = max(0, self.hunger_level - 35)
            self.health = min(100, self.health + 8)
            print(f"{self.name} {food_type}    ")
        else:
            print(f"{self.name} {food_type}    ")

    def hunt_from_sky(self):
        if self.health > 40:
            print(f"{self.name}       ")
            success = self.hunting_skill > 70
            if success:
                print("   ")
                self.eat("  ")
            else:
                print("  ...")

class Penguin(Bird):
    def __init__(self, name, age):
```

```
        super().__init__(name, age, " ", 80)
        self.can_fly = False  #

    def make_sound(self):
        print(f"{self.name}      ")

    def eat(self, food_type):
        if food_type == " ":
            self.hunger_level = max(0, self.hunger_level - 30)
            self.health = min(100, self.health + 6)
            print(f"{self.name} {food_type}    ")
        else:
            print(f"{self.name} {food_type}    ")

    def swim(self):
        print(f"{self.name}      ")

class Zoo:
    def __init__(self, name):
        self.name = name
        self.animals = []
        self.visitors = 0

    def add_animal(self, animal):
        self.animals.append(animal)
        print(f"{animal.name} {self.name}    ")

    def feed_all_animals(self):
        print(f"\n=== {self.name}     ===")
        food_menu = {
            Lion: " ",
            Elephant: " ",
            Eagle: " ",
            Penguin: " "
        }

        for animal in self.animals:
            food = food_menu.get(type(animal), " ")
            animal.eat(food)

    def animal_show(self):
        print(f"\n=== {self.name}      ===")
```

```
        for animal in self.animals:
            print(f"\n{animal.name}    :")
            animal.make_sound()

            if isinstance(animal, Lion):
                animal.hunt()
            elif isinstance(animal, Elephant):
                animal.spray_water()
            elif isinstance(animal, Eagle):
                animal.hunt_from_sky()
            elif isinstance(animal, Penguin):
                animal.swim()

    def health_check(self):
        print(f"\n=== {self.name}    ===")
        for animal in self.animals:
            print(animal)
            if animal.health < 50:
                print(f"    {animal.name}       ")
            elif animal.health > 80:
                print(f"    {animal.name}   ")

#
zoo = Zoo("    ")

#
lion = Lion(" ", 5, "large")
elephant = Elephant("  ", 12)
eagle = Eagle("  ", 3, 250)
penguin = Penguin("  ", 2)

animals = [lion, elephant, eagle, penguin]
for animal in animals:
    zoo.add_animal(animal)

# 1
print("\n      ")
zoo.health_check()

print("\n     ")
zoo.feed_all_animals()
```

```
print("\n     ")
zoo.animal_show()

print("\n      ")
for animal in zoo.animals:
    animal.sleep()

print("\n      ")
zoo.health_check()
```

1.10

1.
2. **super()**
3.
4.
5. **MRO**

1.11

- ( )
- (async/await)
- (NumPy, Pandas)
- (Web)

1.12

1.12.1

: - Python.org - - Real Python - - Python ABC

1.13

:

:

|

: