Table of contents

1

1.1

-
- -
-
-

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

    def start(self):          #
        print(f"{self.brand} {self.model}      ")

#     =
my_car = Car("Toyota", "Prius")
my_car.start()  # Toyota Prius
```

1.2

```
class Person:
    #
    species = "Homo sapiens"

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

```
    def introduce(self):
        #
        return f"   {self.name}  {self.age}  "

    def have_birthday(self):
        self.age += 1
        print(f"      {self.age}  ")

#
person1 = Person(" ", 25)
person2 = Person(" ", 30)

print(person1.introduce())  #       25
person1.have_birthday()     #       26
```

1.3 init

```
class BankAccount:
    def __init__(self, account_number, initial_balance=0):
        self.account_number = account_number
        self.balance = initial_balance
        self.transaction_history = []

        #
        if initial_balance > 0:
            self.transaction_history.append(f"  : {initial_balance} ")

    def __str__(self):
        #
        return f"  : {self.account_number},  : {self.balance} "

    def __repr__(self):
        #
        return f"BankAccount('{self.account_number}', {self.balance})"

#
account = BankAccount("123-456", 1000)
print(account)        #   : 123-456,  : 1000
print(repr(account))  # BankAccount('123-456', 1000)
```

1.4

```
class Calculator:
    def __init__(self):
        self.history = []

    def add(self, a, b):
        result = a + b
        self._record_operation(f"{a} + {b} = {result}")
        return result

    def multiply(self, a, b):
        result = a * b
        self._record_operation(f"{a} × {b} = {result}")
        return result

    def _record_operation(self, operation):
        #          _
        self.history.append(operation)

    def show_history(self):
        print("  :")
        for i, op in enumerate(self.history, 1):
            print(f"  {i}. {op}")

calc = Calculator()
calc.add(5, 3)
calc.multiply(4, 7)
calc.show_history()
```

1.5 vs

```
class Student:
    #     -
    school = "Python "
    total_students = 0

    def __init__(self, name, student_id):
        #       -
```

```
        self.name = name
        self.student_id = student_id
        self.grades = []

        #
        Student.total_students += 1

    @classmethod
    def get_total_students(cls):
        #
        return cls.total_students

    @staticmethod
    def is_passing_grade(grade):
        #
        return grade >= 60

#
student1 = Student(" ", "S001")
student2 = Student(" ", "S002")

print(f" : {Student.school}")                    # Python
print(f"  : {Student.get_total_students()}")   # 2
print(f"  : {Student.is_passing_grade(75)}")    # True
```

1.6 getter/setter

```
class Temperature:
    def __init__(self, celsius=0):
        self._celsius = celsius

    @property
    def celsius(self):
        """   getter"""
        return self._celsius

    @celsius.setter
    def celsius(self, value):
        """   setter"""
        if value < -273.15:
```

```
            raise ValueError("         ")
        self._celsius = value

    @property
    def fahrenheit(self):
        """       """
        return self._celsius * 9/5 + 32

    @property
    def kelvin(self):
        """        """
        return self._celsius + 273.15

#
temp = Temperature(25)
print(f" : {temp.celsius}°C")      # 25°C
print(f" : {temp.fahrenheit}°F")   # 77.0°F
print(f"  : {temp.kelvin}K")     # 298.15K

temp.celsius = 30  # setter
# temp.celsius = -300  # ValueError!
```

1.7 Magic Methods

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"Vector({self.x}, {self.y})"

    def __add__(self, other):
        # +
        return Vector(self.x + other.x, self.y + other.y)

    def __mul__(self, scalar):
        # *
        return Vector(self.x * scalar, self.y * scalar)
```

```
    def __len__(self):
        # len()
        return int((self.x**2 + self.y**2)**0.5)

    def __eq__(self, other):
        # ==
        return self.x == other.x and self.y == other.y

#
v1 = Vector(3, 4)
v2 = Vector(1, 2)

print(v1 + v2)    # Vector(4, 6)
print(v1 * 2)     # Vector(6, 8)
print(len(v1))    # 5
print(v1 == v2)   # False
```

1.8

1.9 :

```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.is_borrowed = False
        self.borrower = None

    def __str__(self):
        status = "  " if self.is_borrowed else "  "
        return f"{self.title} - {self.author} [{status}]"

class Library:
    def __init__(self, name):
        self.name = name
        self.books = []
        self.members = []
```

```
    def add_book(self, book):
        self.books.append(book)
        print(f"    : {book.title}")

    def register_member(self, member_name):
        if member_name not in self.members:
            self.members.append(member_name)
            print(f"    : {member_name}")

    def borrow_book(self, isbn, member_name):
        if member_name not in self.members:
            print("    ")
            return False

        for book in self.books:
            if book.isbn == isbn:
                if not book.is_borrowed:
                    book.is_borrowed = True
                    book.borrower = member_name
                    print(f"  : {book.title} → {member_name}")
                    return True
                else:
                    print(f"{book.title}    ")
                    return False

        print("  ISBN    ")
        return False

    def return_book(self, isbn):
        for book in self.books:
            if book.isbn == isbn and book.is_borrowed:
                borrower = book.borrower
                book.is_borrowed = False
                book.borrower = None
                print(f"  : {book.title} ← {borrower}")
                return True
        print("      ")
        return False

    def list_books(self):
        print(f"\n{self.name}   :")
        for book in self.books:
```

```
            print(f"  {book}")

#
library = Library("   ")

#
book1 = Book("Python ", "   ", "978-1234567890")
book2 = Book("    ", "   ", "978-0987654321")

library.add_book(book1)
library.add_book(book2)

#
library.register_member("   ")
library.register_member("   ")

#
library.borrow_book("978-1234567890", "   ")
library.list_books()
library.return_book("978-1234567890")
library.list_books()
```

1.10 : RPG

```
import random

class Character:
    def __init__(self, name, hp=100, attack=20, defense=10):
        self.name = name
        self.max_hp = hp
        self.hp = hp
        self.attack = attack
        self.defense = defense
        self.level = 1
        self.experience = 0

    def take_damage(self, damage):
        actual_damage = max(0, damage - self.defense)
        self.hp = max(0, self.hp - actual_damage)
```

```
        print(f"{self.name} {actual_damage}       HP: {self.hp} ")

        if self.hp <= 0:
            print(f"{self.name}  ...")
            return True
        return False

    def attack_target(self, target):
        #
        is_critical = random.random() < 0.1  # 10%
        damage = self.attack

        if is_critical:
            damage *= 2
            print(f"{self.name}      ")

        print(f"{self.name}  ")
        return target.take_damage(damage)

    def heal(self, amount):
        old_hp = self.hp
        self.hp = min(self.max_hp, self.hp + amount)
        healed = self.hp - old_hp
        print(f"{self.name} {healed}HP    HP: {self.hp} ")

    def gain_experience(self, exp):
        self.experience += exp
        print(f"{self.name} {exp}    ")

        #
        exp_needed = self.level * 100
        if self.experience >= exp_needed:
            self.level_up()

    def level_up(self):
        self.level += 1
        self.max_hp += 10
        self.hp = self.max_hp
        self.attack += 5
        self.defense += 3
        print(f"  {self.name}  {self.level}   ")
        print(f"   HP: {self.max_hp},   : {self.attack},   : {self.defense}")
```

```
    def __str__(self):
        return f"{self.name} (Lv.{self.level}) HP:{self.hp}/{self.max_hp}"

class Warrior(Character):
    def __init__(self, name):
        super().__init__(name, hp=120, attack=25, defense=15)
        self.special_attacks = 3

    def shield_bash(self, target):
        if self.special_attacks > 0:
            self.special_attacks -= 1
            damage = self.attack * 1.5
            print(f"{self.name}       {self.special_attacks} ")
            return target.take_damage(damage)
        else:
            print("        ")
            return self.attack_target(target)

class Mage(Character):
    def __init__(self, name):
        super().__init__(name, hp=80, attack=30, defense=5)
        self.magic_points = 50

    def fireball(self, target):
        if self.magic_points >= 10:
            self.magic_points -= 10
            damage = self.attack * 2
            print(f"{self.name}      MP: {self.magic_points} ")
            return target.take_damage(damage)
        else:
            print("MP    ")
            return self.attack_target(target)

    def heal_spell(self):
        if self.magic_points >= 15:
            self.magic_points -= 15
            self.heal(30)
            print(f"  MP: {self.magic_points} ")
        else:
            print("MP    ")

#
```

```python
def battle(char1, char2):
    print(f"\n  {char1.name} vs {char2.name}  ")

    turn = 1
    while char1.hp > 0 and char2.hp > 0:
        print(f"\n---    {turn} ---")
        print(f"{char1} vs {char2}")

        #    1
        if isinstance(char1, Warrior) and char1.special_attacks > 0:
            char1.shield_bash(char2)
        elif isinstance(char1, Mage) and char1.magic_points >= 10:
            char1.fireball(char2)
        else:
            char1.attack_target(char2)

        if char2.hp <= 0:
            print(f"\n  {char1.name}  ")
            char1.gain_experience(50)
            break

        #    2
        char2.attack_target(char1)

        if char1.hp <= 0:
            print(f"\n  {char2.name}  ")
            char2.gain_experience(50)
            break

        turn += 1

#
warrior = Warrior(" ")
mage = Mage("  ")

print(warrior)
print(mage)

battle(warrior, mage)
```

1.11

1. = =
2. **init**
3. **self**
4.
5. Python

1.12

- ( )
- ( )
- ( )
- ( )

1.13

1.13.1

: - Python.org - - Real Python - OOP - Python Tutor

1.14

:

:

|

: