

特殊方法（魔术方法）

以两个下划线开头且以两个下划线结尾的方法。

在特定情况下，它会被自动调用，不需要我们手动调用该方法。

`__init__(self [, ...])`

- 初始化方法，在实例化过程中调用

```
class Ex:

    def __init__(self, arg1, arg2):
        print(f"__init__被调用, arg1:{arg1},
arg2:{arg2}")

Ex("a", "b")  # 实例化
```

`__call__(self [, ...])`

- 当实例对象像函数那样被“调用”时，会调用该方法

```
class Ex:

    def __call__(self, arg1, arg2):
        print(f"__call__被调用, arg1:{arg1},
arg2:{arg2}")

e = Ex()
e("a", "b")
```

`__getitem__(self, key)`

- 当执行 `self[key]` 操作时，会调用该方法

```
class Ex:

    def __getitem__(self, key):
        print(f"__getitem__被调用, key: {key}")
        print(["a", "b", "c"][key])
        print({0: "零", 1: "壹", 2: "贰"}[key])

e = Ex()
e[2]
```

`__repr__(self) / __str__(self)`

- 实例对象转字符串时，会调用该方法，要求必需返回字符串类型

```
class Ex:

    def __repr__(self):
        return "__repr__被调用"

    # def __str__(self):
    #     return "__str__被调用"

e = Ex()
print(str(e))
print(f"{e}")
print(e) # print会转成字符串再输出
```

`__add__(self, other)`

- 实例对象进行加法操作时会调用该方法，要求只要加法左边有当前类的实例对象即可

```
class Number:

    def __init__(self, num):
        self.num = num

    def __add__(self, other):
        return self.num + other

n = Number(6)
print(n + 7)  # 实例对象在左边
```

`__radd__(self, other)`

- 实例对象进行加法操作时会调用该方法，要求加法右边有当前类的实例对象且左边没有

```
class Number:

    def __init__(self, num):
        self.num = num

    def __radd__(self, other):
        return other + self.num

n = Number(6)
print(7 + n)  # 实例对象在右边
```

`__sub__(self, other)`

- 实例对象进行减法操作时会调用该方法，要求只要减法左边有当前类的实例对象即可

```
class Number:

    def __init__(self, num):
        self.num = num

    def __sub__(self, other):
        return self.num - other

n = Number(6)
print(n - 4)  # 实例对象在左边
```

`__rsub__(self, other)`

- 实例对象进行减法操作时会调用该方法，要求减法右边有当前类的实例对象且左边没有

```
class Number:

    def __init__(self, num):
        self.num = num

    def __rsub__(self, other):
        return other - self.num

n = Number(6)
print(4 - n)  # 实例对象在右边
```

`__mul__(self, other)`

- 实例对象进行乘法操作时会调用该方法，要求只要乘法左边有当前类的实例对象即可

```
class Number:

    def __init__(self, num):
        self.num = num

    def __mul__(self, other):
        return self.num * other

n = Number(6)
print(n * 4)  # 实例对象在左边
```

`__rmul__(self, other)`

- 实例对象进行乘法操作时会调用该方法，要求乘法右边有当前类的实例对象且左边没有

```
class Number:

    def __init__(self, num):
        self.num = num

    def __rmul__(self, other):
        return other * self.num

n = Number(6)
print(4 * n)  # 实例对象在右边
```

`__truediv__(self, other)`

- 实例对象进行除法操作时会调用该方法，要求只要除法左边有当前类的实例对象即可

```
class Number:

    def __init__(self, num):
        self.num = num

    def __truediv__(self, other):
        return self.num / other

n = Number(6)
print(n / 3)  # 实例对象在左边
```

`__rtruediv__(self, other)`

- 实例对象进行除法操作时会调用该方法，要求除法右边有当前类的实例对象且左边没有

```
class Number:

    def __init__(self, num):
        self.num = num

    def __rtruediv__(self, other):
        return other / self.num

n = Number(6)
print(3 / n)  # 实例对象在右边
```


案例：实现分数运算

```
def get_gcd(a, b):
    for i in range(min(abs(a), abs(b)), 0, -1):
        if not a % i and not b % i:
            return -i if a < 0 and b < 0 else i
    return b

def get_frac(obj):
    if isinstance(obj, Fraction):
        return obj
    elif isinstance(obj, int):
        return Fraction(obj, 1)
    elif isinstance(obj, float):
        b = 10**(len(str(obj).split(".")[1]))
        return Fraction(int(obj*b), b)
    raise TypeError("类型错误")

class Fraction:

    def __init__(self, a, b):
        gcd = get_gcd(a, b)
        self.a = a // gcd
        self.b = b // gcd

    def __repr__(self):
        if self.b == 1:
            return str(self.a)
        elif self.b == -1:
            return str(-self.a)
```

```

        return f"{self.a}/{self.b}"

    def __add__(self, other):
        other = get_frac(other)
        return
        Fraction(self.a*other.b+other.a*self.b,
        self.b*other.b)

    def __sub__(self, other):
        other = get_frac(other)
        return Fraction(self.a*other.b-
        other.a*self.b, self.b*other.b)

    def __mul__(self, other):
        other = get_frac(other)
        return Fraction(self.a*other.a,
        self.b*other.b)

    def __truediv__(self, other):
        other = get_frac(other)
        return Fraction(self.a*other.b,
        self.b*other.a)

    def __radd__(self, other):
        other = get_frac(other)
        return
        Fraction(other.a*self.b+other.b*self.a,
        other.b*self.b)

    def __rsub__(self, other):
        other = get_frac(other)
        return Fraction(other.a*self.b-
        other.b*self.a, other.b*self.b)

```

```
def __rmul__(self, other):  
    other = get_frac(other)  
    return Fraction(other.a*self.a,  
other.b*self.b)
```

```
def __rtruediv__(self, other):  
    other = get_frac(other)  
    return Fraction(other.a*self.b,  
other.b*self.a)
```

```
f1 = Fraction(3, 4)  
f2 = Fraction(2, 3)  
print(f1 + f2)  
print(f1 - f2)  
print(f1 * f2)  
print(f1 / f2)  
print(f1 + 3)  
print(f1 - 3.2)  
print(f1 * False)  
print(f1 / True)  
print(3 + f1)  
print(3.2 - f1)  
print(True * f1)  
print(False / f1)
```