Наследование. Шпаргалка

Наследование

```
class Employee:

    raise_amt = 1.04

    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay

    def fullname(self):
        return f'{self.first} {self.last}'

    def apply_raise(self):
        self.pay = int(self.pay * self.raise_amt)

class Developer(Employee):
    raise_amt = 1.10

dev_1 = Developer('Ivan', 'Ivanov', 60000)
dev_1.apply_raise()
print(dev_1.pay)
print(dev_1.fullname())
```

Функция super()

```python
class Employee:

    raise_amt = 1.04

    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay

    def fullname(self):
        return f'{self.first} {self.last}'

    def apply_raise(self):
        self.pay = int(self.pay * self.raise_amt)


class Developer(Employee):
    raise_amt = 1.10
    # Переопределяем метод базового класса
    def __init__(self, first, last, pay, prog_lang):
        # Вызываем метод базового класса
        super().__init__(first, last, pay)
        # Дополнительный код
        self.prog_lang = prog_lang

dev_1 = Developer('Ivan', 'Ivanov', 60000, 'Python')
dev_2 = Developer('Petr', 'Petrov', 70000, 'Java')
```

Функция isinstance()

```
 class Developer(Employee):
    raise_amt = 1.10

    def __init__(self, first, last, pay, prog_lang):
        super().__init__(first, last, pay)
        self.prog_lang = prog_lang

    def __add__(self, other):
        if not isinstance(other, Employee):
            raise ValueError('Складывать можно только объекты Employee и дочерние от них.')
        return self.pay + other.pay

dev_1 = Developer('Ivan', 'Ivanov', 60000, 'Python')
dev_2 = Developer('Petr', 'Petrov', 70000, 'Java')
print(dev_1 + dev_2)

emp_1 = Employee('Petr', 'Petrov', 50000)
print(dev_1 + emp_1)

print(dev_2 + 10000)
```

Функция issubclass()

```
class Employee:
    """ Базовый класс сотрудника """

    def set_salary(self):
        """ Метод для начисления зарплаты """
        print('Pay salary')


class Developer(Employee):
    """ Класс для разработчика """
    pass


class Client:
    """ Класс клиента """

    def get_payment(self):
        """ Метод получения оплаты от клиента """
        print('Get payment')


system_users = [Employee(), Developer(), Developer(), Client()]

for user in system_users:
    if issubclass(user.__class__, Employee):
        # Если класс объекта сотрудник, то начисляем зарплату
        user.set_salary()
    else:
        # В случае, если объект не является сотрудником, то запрашиваем оплату
        user.get_payment()
```