|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Кафедра КБ-2 «Прикладные информационные технологии» |

**ПРАКТИЧЕСКАЯ РАБОТА №4 по дисциплине**

**«Технологии и методы программирования»**

***(наименование дисциплины)***

**Тема практической работы   
«Итератор. Посетитель»**

Студент группы: БИСО-01-20 Дутов А.С.

*(шифр учебной группы) (Фамилия И.О)*

Преподаватель: Лесько С.А.

*(Фамилия И.О)*

Москва, 2023

**Практическая работа №4**

**Итератор**

class MyIterator:

def \_\_init\_\_(self, data):

self.data = data

self.index = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.index >= len(self.data):

raise StopIteration

result = self.data[self.index]

self.index += 1

return result

# Пример использования

my\_list = [1, 2, 3, 4, 5]

my\_iter = MyIterator(my\_list)

for item in my\_iter:

print(item)

В этом примере я создал класс MyIterator, который реализует методы iter() и next(). Метод iter() возвращает сам итератор, а метод next() возвращает следующий элемент в последовательности.В методе next() мы проверяем, достигнут ли конец последовательности. Если да, то мы вызываем исключение StopIteration, иначе мы возвращаем следующий элемент в последовательности и увеличиваем переменную index на единицу.В конце, создаем список my\_list и объект итератора my\_iter, и используем оператор for для перебора всех элементов последовательности.

**Посетитель**

class Visitor:

def visit(self, element):

pass

class Element:

def accept(self, visitor):

pass

class ConcreteElementA(Element):

def accept(self, visitor):

visitor.visit(self)

class ConcreteElementB(Element):

def accept(self, visitor):

visitor.visit(self)

class ConcreteVisitor1(Visitor):

def visit(self, element):

print("ConcreteVisitor1 visited ", type(element).\_\_name\_\_)

class ConcreteVisitor2(Visitor):

def visit(self, element):

print("ConcreteVisitor2 visited ", type(element).\_\_name\_\_)

# Пример использования

elements = [ConcreteElementA(), ConcreteElementB()]

visitor1 = ConcreteVisitor1()

visitor2 = ConcreteVisitor2()

for element in elements:

element.accept(visitor1)

for element in elements:

element.accept(visitor2)

В этом примере я создал класс Visitor, определяющий метод visit(), который будет реализован в конкретных посетителях и класс Element, определяющий метод accept(), который будет вызывать соответствующий метод visit() конкретного посетителя.

Затем, создал два класса ConcreteElementA и ConcreteElementB, которые реализуют метод accept() в соответствии со своей спецификой и два класса-посетителя, ConcreteVisitor1 и ConcreteVisitor2, которые реализуют метод visit() для каждого из конкретных элементов.

Список elements содержит объекты классов ConcreteElementA и ConcreteElementB, и объекты посетителей visitor1 и visitor2. В конце мы перебрали объекты elements и вызвали метод accept() для каждого из них, передавая соответствующий объект посетителя.