Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4
дисциплины «Объектно-ориентированное программирование»
Вариант 7

Выполнил:
Мотовилов Вадим Борисович
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А. ктн доцент
департамента перспективной
инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

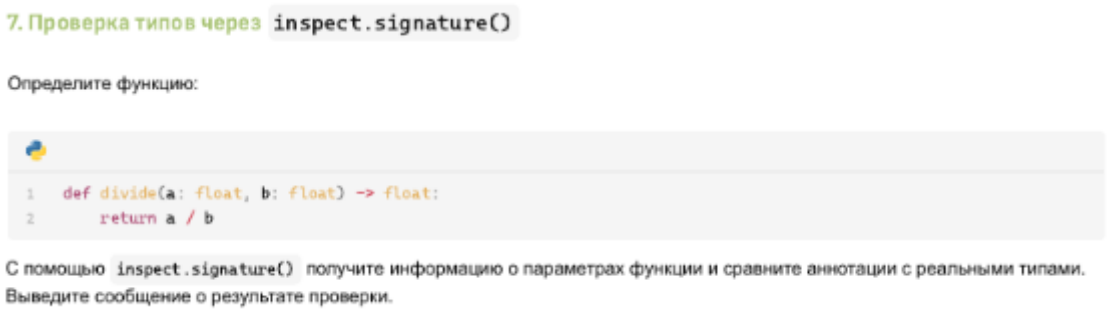
Ставрополь, 2026 г.

Тема: Элементы объектно-ориентированного программирования в языке Python.

Цель работы: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

Задание 1.



Код программы:

```
import inspect

def divide(a: float, b: float) -> float:
    return a / b

def check_function_types(func):
    """
    Проверяет аннотации типов функции и сравнивает их с реальными типами аргументов
    """
    print(f"Проверка функции: {func.__name__}")
    print("-" * 40)

    # Получаем сигнатуру функции
    sig = inspect.signature(func)

    # Получаем аннотации функции
    annotations = func.__annotations__

    print("Информация из inspect.signature():")
    print(f"  Сигнатура: {sig}")

    if annotations:
        print(f"  Аннотации: {annotations}")
    else:
        print("  Аннотации: отсутствуют")
        return

    print("\nПараметры функции:")
    for param_name, param in sig.parameters.items():
```

```
        param_info = f"  {param_name}: "

        # Получаем аннотацию типа
        if param.annotation != inspect.Parameter.empty:
            param_info += f"аннотация = {param.annotation.__name__}"
        else:
            param_info += "аннотация отсутствует"

        # Информация о параметре
        if param.default != inspect.Parameter.empty:
            param_info += f", значение по умолчанию = {param.default}"

        print(param_info)

    # Информация о возвращаемом значении
    if 'return' in annotations:
        print(f"\nВозвращаемое значение: аннотация = {annotations['return'].__name__}")
    else:
        print("\nВозвращаемое значение: аннотация отсутствует")

    print("\n" + "=" * 40)

def test_function_with_arguments(func, test_cases):
    """
    Тестирует функцию с различными аргументами и проверяет типы
    """
    print("Тестирование функции с различными аргументами:")
    print("-" * 40)

    sig = inspect.signature(func)
    annotations = func.__annotations__

    for i, (args, kwargs) in enumerate(test_cases, 1):
        print(f"\nТест {i}:")
        print(f"  Аргументы: {args}")

        try:
            # Вызываем функцию
            result = func(*args, **kwargs)
            print(f"  Результат: {result}")
            print(f"  Тип результата: {type(result).__name__}")

            # Сравниваем с аннотацией
            if 'return' in annotations:
                expected_type = annotations['return']
                actual_type = type(result)

                # Проверяем, соответствует ли тип аннотации
                if isinstance(result, expected_type):
                    print(f"  ✓ Тип результата ({actual_type.__name__}) соответствует аннотации
({expected_type.__name__})")
                else:
```

```
                print(f"  ✗ Тип результата ({actual_type.__name__}) НЕ соответствует аннотации ({expected_type.__name__})")

        except Exception as e:
            print(f"  Ошибка при выполнении: {e}")

    print("=" * 40)

if __name__ == "__main__":
    # Проверяем функцию divide
    check_function_types(divide)

    # Тестовые случаи для функции divide
    test_cases = [
        ((10, 2), {}),          # оба int
        ((10.5, 2.5), {}),      # оба float
        ((10, 2.5), {}),        # int и float
        ((10.5, 2), {}),        # float и int
        ((0, 5), {}),           # ноль в числителе
        ((10, 0), {}),          # деление на ноль (будет ошибка)
    ]

    test_function_with_arguments(divide, test_cases)

    # Дополнительная информация о модуле inspect
    print("\nДополнительная информация о функции через inspect:")
    print("-" * 40)

    # Получаем исходный код функции
    try:
        source = inspect.getsource(divide)
        print("Исходный код функции:")
        print(source)
    except:
        print("Не удалось получить исходный код")

    # Получаем информацию о модуле
    module = inspect.getmodule(divide)
    print(f"\nМодуль функции: {module}")

    # Получаем информацию о строке определения
    line_no = inspect.getsourcelines(divide)[1]
    print(f"Номер строки определения: {line_no}")

    print("\n" + "=" * 40)
    print("Проверка завершена!")
```

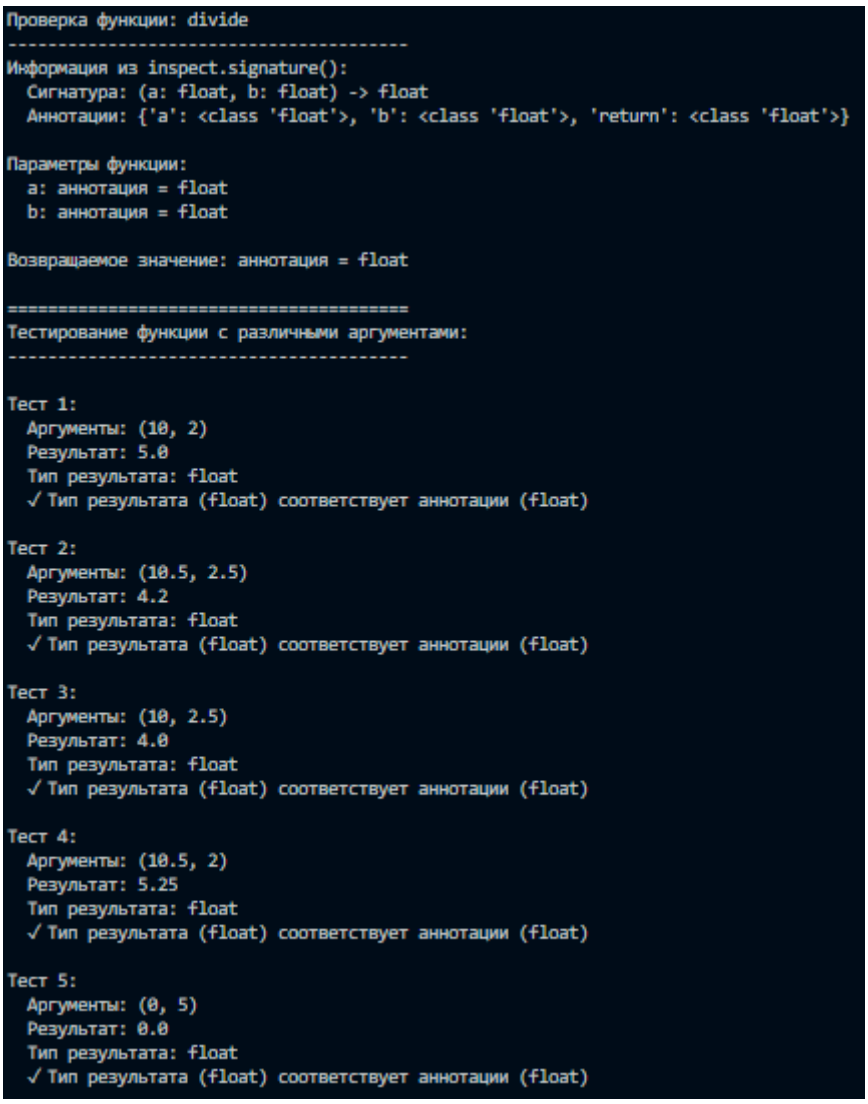


Рисунок 1. Пример работы кода

Задание 2.

7. Универсальный стек

Создайте обобщённый класс стека с методами `push`, `pop` и `is_empty`.
Тип элементов должен быть параметризован с помощью `TypeVar`.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from typing import TypeVar, Generic, List, Optional

# Создаем TypeVar для параметризации типа элементов стека
T = TypeVar('T')

class Stack(Generic[T]):
    """Универсальный стек (LIFO - Last In, First Out)"""

    def __init__(self) -> None:
        """Инициализация пустого стека"""
```

```
        self._items: List[T] = []

    def push(self, item: T) -> None:
        """Добавить элемент на вершину стека"""
        self._items.append(item)

    def pop(self) -> T:
        """Удалить и вернуть элемент с вершины стека"""
        if self.is_empty():
            raise IndexError("Попытка извлечь элемент из пустого стека")
        return self._items.pop()

    def is_empty(self) -> bool:
        """Проверить, пуст ли стек"""
        return len(self._items) == 0

    def peek(self) -> Optional[T]:
        """Посмотреть элемент на вершине стека без его удаления"""
        if self.is_empty():
            return None
        return self._items[-1]

    def size(self) -> int:
        """Получить количество элементов в стеке"""
        return len(self._items)

    def clear(self) -> None:
        """Очистить стек"""
        self._items.clear()

    def __str__(self) -> str:
        """Строковое представление стека"""
        return f"Stack({self._items})"

    def __repr__(self) -> str:
        """Представление для отладки"""
        return str(self)

    def __len__(self) -> int:
        """Количество элементов в стеке"""
        return len(self._items)

def demonstrate_stack() -> None:
    """Демонстрация работы универсального стека с разными типами данных"""
    # 1. Стек целых чисел
    print("1. Стек целых чисел (Stack[int]):")
    int_stack: Stack[int] = Stack()

    print(f"   Стек пуст? {int_stack.is_empty()}")

    # Добавляем элементы
    for i in range(1, 6):
```

```
    int_stack.push(i * 10)
    print(f"   push({i * 10}) -> {int_stack}")

print(f"   Размер стека: {int_stack.size()}")
print(f"   Элемент на вершине (peek): {int_stack.peek()}")
print(f"   Стек пуст? {int_stack.is_empty()}")

# Извлекаем элементы
while not int_stack.is_empty():
    item = int_stack.pop()
    print(f"   pop() -> {item}, стек: {int_stack}")

print(f"   Стек пуст? {int_stack.is_empty()}")

# 2. Стек строк
print("\n2. Стек строк (Stack[str]):")
str_stack: Stack[str] = Stack()

words = ["Hello", "World", "Python", "Stack"]
for word in words:
    str_stack.push(word)

print(f"   Исходный стек: {str_stack}")

# Обратный порядок слов
reversed_words = []
while not str_stack.is_empty():
    reversed_words.append(str_stack.pop())

print(f"   Слова в обратном порядке: {reversed_words}")

# 3. Стек с пользовательскими объектами
print("\n3. Стек с пользовательскими объектами:")

class Person:
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

    def __repr__(self):
        return f"Person('{self.name}', {self.age})"

person_stack: Stack[Person] = Stack()

person_stack.push(Person("Alice", 25))
person_stack.push(Person("Bob", 30))
person_stack.push(Person("Charlie", 35))

print(f"   Стек людей: {person_stack}")
print(f"   Извлекаем: {person_stack.pop()}")
print(f"   Осталось: {person_stack}")
```

```
# 4. Обработка ошибок
print("\n4. Обработка ошибок:")
empty_stack: Stack[float] = Stack()

try:
    empty_stack.pop()
except IndexError as e:
    print(f"   Ошибка при pop() из пустого стека: {e}")

# 5. Использование методов clear и len
print("\n5. Дополнительные методы:")

test_stack: Stack[int] = Stack()
for i in range(5):
    test_stack.push(i)

print(f"   Стек: {test_stack}")
print(f"   Размер через len(): {len(test_stack)}")
print(f"   Размер через size(): {test_stack.size()}")

test_stack.clear()
print(f"   После clear(): {test_stack}")
print(f"   is_empty(): {test_stack.is_empty()}")

# 6. Математическое выражение (проверка скобок)
print("\n6. Проверка корректности скобок в выражении:")

def check_parentheses(expression: str) -> bool:
    """Проверяет корректность расстановки скобок"""
    stack: Stack[str] = Stack()
    pairs = {')': '(', ']': '[', '}': '{'}

    for char in expression:
        if char in '([{':
            stack.push(char)
        elif char in ')]}':
            if stack.is_empty():
                return False
            if stack.pop() != pairs[char]:
                return False

    return stack.is_empty()

expressions = [
    "(a + b) * (c - d)",      # корректно
    "[(x + y) * {z - w}]",    # корректно
    "(a + b] * c",            # некорректно
    "((a + b) * c",           # некорректно
    "a + b) * c",             # некорректно
    "{[()]}",                 # корректно
]
```

```
    for expr in expressions:
        result = check_parentheses(expr)
        status = "✓ корректно" if result else "✗ некорректно"
        print(f"   '{expr}' -> {status}")

if __name__ == "__main__":
    demonstrate_stack()
```

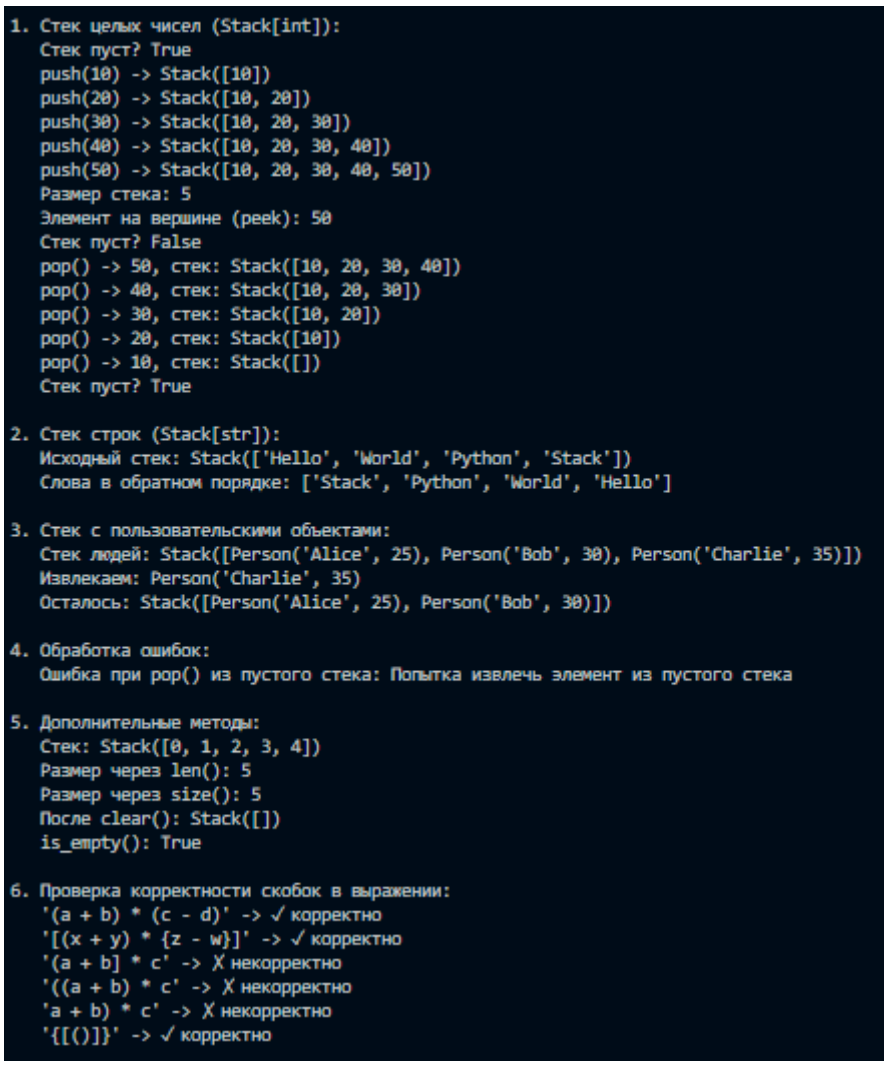


Рисунок 2. Пример работы кода

Вывод: приобрел навыки по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.