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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины
«Объектно-ориентированное программирование»
Вариант № 18

Выполнил:
Текеева Мадина Азрет-Алиевна
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)

Проверил:
Доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии Воронкин Р.А

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Аннотации типов в языке Python.

Цель работы: приобретение навыков по работе с аннотациями типов при написании программ с помощью языка программирования Python версии 3.x.

Репозиторий: https://github.com/bickrosss/TM_oop_lab4

Порядок выполнения работы:

1. Демонстрация базового синтаксиса аннотаций типов.

```
def total_price(price, tax):
    return price + price * tax

def total_price(price: float, tax: float) -> float:
    return price + price * tax
```

2. Аннотации для коллекций и различных структур данных.

```
from typing import List, Dict, Tuple, Set

numbers: List[int] = [1, 2, 3]
grades: Dict[str, float] = {"math": 4.5, "physics": 5.0}
record: Tuple[str, int] = ("John", 25)
tags: Set[str] = {"python", "typing"}

numbers: list[int] = [1, 2, 3]
grades: dict[str, float] = {"math": 5.0}
```

3. Работа с необязательными типами и объединениями.

```
from typing import Union

def normalize(value: Union[int, float, str]) -> float:
    if isinstance(value, str):
        value = value.replace(",", "")
    return float(value)

def normalize(value: int | float | str) -> float:
    if isinstance(value, str):
        value = value.replace(",", "")
    return float(value)

from typing import Optional

def find_user_id(name: str) -> Optional[int]:
    users = {"Alice": 1, "Bob": 2}
    return users.get(name)
```

```
uid = find_user_id("Charlie")
print(uid + 1)
```

4. Различные варианты аннотаций для функций.

```
from typing import Tuple

def min_max(values: list[int]) -> Tuple[int, int]:
    return min(values), max(values)

def min_max(values: list[int]) -> tuple[int, int]:
    return min(values), max(values)

def log_message(msg: str) -> None:
    print(f"LOG {msg}")

def add(a: int, b: int) -> int:
    return a + b

print(add.__annotations__)
```

5. Аннотации типов в классах.

```
class Student:
    name: str
    grades: list[int]

    def __init__(self, name: str, grades: list[int]):
        self.name = name
        self.grades = grades

    def average(self) -> float:
        return sum(self.grades) / len(self.grades)

class Person:
    name: str
    age: int

class Employee(Person):
    position: str
    salary: float

from typing import Literal

def set_status(status: Literal["new", "in_progress", "done"]) -> None:
    print(f"Текущий статус: {status}")
```

6. Работа с универсальными типами (дженериками).

```python
from typing import TypeVar, List

T = TypeVar("T")
def first(items: List[T]) -> T:
    return items[0]

from typing import Generic

T = TypeVar("T")
class Box(Generic[T]):
    def __init__(self, value: T):
        self.value = value
    def get(self) -> T:
        return self.value

b1 = Box[int](10)
b2 = Box[str]("Hello")
print(b1.get())
print(b2.get())
```

7. Специальные типы Any и NoReturn.

```python
from typing import Any

def show(value: Any) -> None:
    print(f"Получено значение: {value}")

from typing import NoReturn
import sys

def fatal_error(msg: str) -> NoReturn:
    print(f"Фатальная ошибка: {msg}")
    sys.exit(1)

fatal_error("Ошибка конфигурации!")
```

8. Примеры для статического анализатора mypy.

```python
def sq_sum(a: int, b: int) -> int:
    return a*2 + b*2

print(sq_sum(3, 4))

def sq_sum(a: int, b: int) -> int:
    return a*2 + b*2

print(sq_sum(3, "4"))

result = process_data(data)
```

9. Решение проблемы циклических зависимостей.

```
class Rectangle:
    def __init__(self, width: int, height: int, color: Color):
        self.width = width
        self.height = height
        self.color = color

class Rectangle:
    def __init__(self, width: int, height: int, color: "Color") -> None:
        self.width = width
        self.height = height
        self.color = color

from __future__ import annotations

class Rectangle:
    def __init__(self, width: int, height: int, color: Color):
        self.width = width
        self.height = height
        self.color = color

class Color:
    def __init__(self, r: int, g: int, b: int):
        self.r = r
        self.g = g
        self.b = b
```

10. Выполнили задание 1. Проверка аннотаций у функции с кортежем.

Создать функцию: def get_coords(point: tuple[float, float]) -> str: return f"x={point[0]}, y={point[1]}"

Перед вызовом проверить, что оба элемента кортежа имеют тип float.

Листинг программы:

```
def get_coords(point: tuple[float, float]) -> str:
    return f"x={point[0]}, y={point[1]}"

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

from typing import get_type_hints
from coords import get_coords

def main():
    print("Функция get_coords")

    try:
```

```
            x = float(input("Введите координату x: "))
            y = float(input("Введите координату y: "))
        except ValueError:
            print("Ошибка: введите числовые значения для координат")
            return

        point_val = (x, y)
        result = get_coords(point_val)
        func_hints = get_type_hints(get_coords)
        actual_point = type(point_val)
        actual_return = type(result)

        print(f"\nРезультат вызова функции get_coords: {result}")
        print(f"Тип аргумента 'point': ожидается {func_hints['point']}, получен
{actual_point}")
        print(f"Тип возвращаемого значения: ожидается {func_hints['return']}, получен
{actual_return}")

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

```
Функция get_coords
Введите координату x: 6
Введите координату y: 9

Результат вызова функции get_coords: x=6.0, y=9.0
Тип аргумента 'point': ожидается tuple[float, float], получен <class 'tuple'>
Тип возвращаемого значения: ожидается <class 'str'>, получен <class 'str'>
```

Рисунок 1. Результат выполнения программы

11. Решили задание 2. Универсальная функция объединения словарей.

Создать функцию, которая принимает два словаря одинаковой структуры и возвращает их объединение. Типы ключей и значений должны быть параметризованы.

Листинг программы:

```
from typing import Dict, TypeVar

K = TypeVar('K')
V = TypeVar('V')

def merge_dicts(dict1: Dict[K, V], dict2: Dict[K, V]) -> Dict[K, V]:
    result = dict1.copy()
    result.update(dict2)
    return result
#!/usr/bin/env python3
```

```python
# -*- coding: utf-8 -*-

from typing import get_type_hints
from merge_dicts_package.merge_dicts import merge_dicts

def input_dict(prompt: str) -> dict:
    print(prompt)
    result = {}
    while True:
        key = input("Введите ключ (или пустую строку для завершения): ")
        if not key:
            break
        value = input(f"Введите значение для ключа '{key}': ")
        result[key] = value
    return result

def main():
    print("Функция merge_dicts")

    print("\nВвод первого словаря:")
    dict1 = input_dict("Первый словарь:")

    print("\nВвод второго словаря:")
    dict2 = input_dict("Второй словарь:")

    result = merge_dicts(dict1, dict2)
    func_hints = get_type_hints(merge_dicts)
    actual_dict1 = type(dict1)
    actual_dict2 = type(dict2)
    actual_return = type(result)

    print(f"\nПервый словарь: {dict1}")
    print(f"Второй словарь: {dict2}")
    print(f"Результат объединения: {result}")
    print(f"Тип аргумента 'dict1': ожидается {func_hints['dict1']}, получен
{actual_dict1}")
    print(f"Тип аргумента 'dict2': ожидается {func_hints['dict2']}, получен
{actual_dict2}")
    print(f"Тип возвращаемого значения: ожидается {func_hints['return']}, получен
{actual_return}")

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

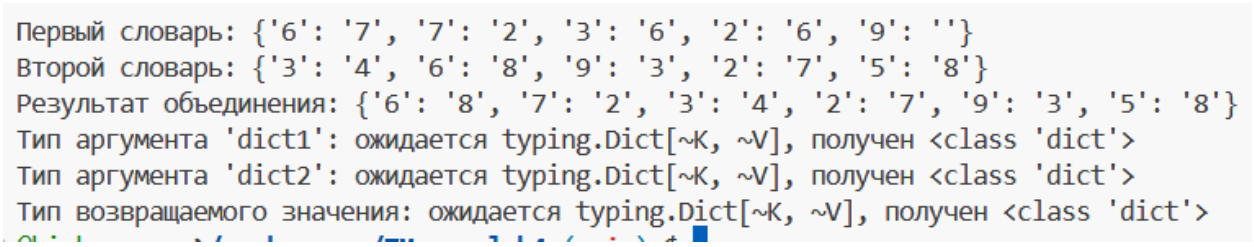

Рисунок 2. Результат

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе с аннотациями типов при написании программ с помощью языка программирования Python версии.