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 №5
дисциплины
«Объектно-ориентированное программирование»
Вариант 1

Выполнил:
Бабенко Артём Тимофеевич
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Ассистент департамента цифровых, робототехнических систем и электроники Богданов С.С

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: аннотация типов

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

Порядок выполнения работы:

Пример 1. Для примера 1 лабораторной работы 14 добавьте аннотации типов.

Результат работы программы:

```
>>> help
Список команд:

add - добавить работника;
list - вывести список работников;
select <стаж> - запросить работников со стажем;
load <имя_файла> - загрузить данные из файла;
save <имя_файла> - сохранить данные в файл;
help - отобразить справку;
exit - завершить работу с программой.
>>> add
Фамилия и инициалы? Иванов И.И
Должность? Программист
Год поступления? 1999
>>> lsit
lsit -> Unknown command
>>> list
+------+--------------------------------+----------------------+----------+
|  №   |            Ф.И.О.              |      Должность       |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Иванов И.И                     | Программист          |     1999 |
+------+--------------------------------+----------------------+----------+
>>>
```

Рисунок 1 – Результат работы программы

Код программы:

```
 #!/usr/bin/env python3
 # -*- coding: utf-8 -*
from dataclasses import dataclass, field
from datetime import date
import logging
import sys
from typing import List
import xml.etree.ElementTree as ET

# Класс пользовательского исключения в случае, если неверно
# введен номер года.

class IllegalYearError(Exception):
    def __init__(self, year, message="Illegal year number"):
```

```
        self.year = year
        self.message = message
        super(IllegalYearError, self).__init__(message)

    def __str__(self):
        return f"{self.year} -> {self.message}"


# Класс пользовательского исключения в случае, если введенная
# команда является недопустимой.
class UnknownCommandError(Exception):
    def __init__(self, command, message="Unknown command"):
        self.command = command
        self.message = message
        super(UnknownCommandError, self).__init__(message)
    def __str__(self):
        return f"{self.command} -> {self.message}"
@dataclass(frozen=True)
class Worker:
    name: str
    post: str
    year: int
@dataclass
class Staff:
  workers: List[Worker] = field(default_factory=lambda: [])
  def add(self, name: str, post: str, year: int) -> None:
      # Получить текущую дату.
      today = date.today()
      if year < 0 or year > today.year:
          raise IllegalYearError(year)
      self.workers.append(
          Worker(
              name=name,
              post=post,
              year=year
          )
      )
      self.workers.sort(key=lambda worker: worker.name)
  def __str__(self) -> str:
      # Заголовок таблицы.
      table = []
      line = '+-{}-+-{}-+-{}-+-{}-+'.format(
          '-' * 4,
          '-' * 30,
          '-' * 20,
          '-' * 8
      )
      table.append(line)
      table.append(
          '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
              "№",
              "Ф.И.О.",
```

```
            "Должность",
            "Год"
        )
    )
    table.append(line)
    # Вывести данные о всех сотрудниках.
    for idx, worker in enumerate(self.workers, 1):
        table.append(
            '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                idx,
                worker.name,
                worker.post,
                worker.year
            )
        )
    table.append(line)
    return '\n'.join(table)
def select(self, period: int) -> List[Worker]:
    # Получить текущую дату.
    today = date.today()
    result: List[Worker] = []
    for worker in self.workers:
        if today.year - worker.year >= period:
            result.append(worker)
    return result
def load(self, filename: str) -> None:
  with open(filename, 'r', encoding='utf8') as fin:
    xml = fin.read()
  parser = ET.XMLParser(encoding="utf8")
  tree = ET.fromstring(xml, parser=parser)
  self.workers = []
  for worker_element in tree:
    name, post, year = None, None, None
    for element in worker_element:
      if element.tag == 'name':
        name = element.text
      elif element.tag == 'post':
        post = element.text
      elif element.tag == 'year':
        year = int(element.text)
      if name is not None and post is not None \
          and year is not None:
        self.workers.append(
          Worker(
            name=name,
            post=post,
            year=year
          )
        )
def save(self, filename: str) -> None:
  root = ET.Element('workers')
  for worker in self.workers:
```

```
            worker_element = ET.Element('worker')
            name_element = ET.SubElement(worker_element, 'name')
            name_element.text = worker.name
            post_element = ET.SubElement(worker_element, 'post')
            post_element.text = worker.post
            year_element = ET.SubElement(worker_element, 'year')
            year_element.text = str(worker.year)
            root.append(worker_element)
        tree = ET.ElementTree(root)
        with open(filename, 'wb') as fout:
            tree.write(fout, encoding='utf8', xml_declaration=True)

    if __name__ == '__main__':
        # Выполнить настройку логгера.
        logging.basicConfig(
            filename='workers4.log',
            level=logging.INFO
        )
        # Список работников.
        staff = Staff()
        # Организовать бесконечный цикл запроса команд.
        while True:
            try:
                # Запросить команду из терминала.
                command = input(">>> ").lower()
                # Выполнить действие в соответствие с командой.
                if command == 'exit':
                    break
                elif command == 'add':
                    # Запросить данные о работнике.
                    name = input("Фамилия и инициалы? ")
                    post = input("Должность? ")
                    year = int(input("Год поступления? "))
                    # Добавить работника.
                    staff.add(name, post, year)
                    logging.info(
                        f"Добавлен сотрудник: {name}, {post}, "
                        f"поступивший в {year} году."
                    )
                elif command == 'list':
                    # Вывести список.
                    print(staff)
                    logging.info("Отображен список сотрудников.")
                elif command.startswith('select '):
                    # Разбить команду на части для выделения номера года.
                    parts = command.split(maxsplit=1)
                    # Запросить работников.
                    selected = staff.select(parts[1])
                    # Вывести результаты запроса.
                    if selected:
                        for idx, worker in enumerate(selected, 1):
                            print(
```

```
                    '{:>4}: {}'.format(idx, worker.name)
                )
                logging.info(
                f"Найдено {len(selected)} работников со "
                f"стажем более {parts[1]} лет."
            )
        else:
            print("Работники с заданным стажем не найдены.")
            logging.warning(
                f"Работники со стажем более {parts[1]} лет не найдены."
            )
    elif command.startswith('load '):
        # Разбить команду на части для имени файла.
        parts = command.split(maxsplit=1)
        # Загрузить данные из файла.
        staff.load(parts[1])
        logging.info(f"Загружены данные из файла {parts[1]}.")
    elif command.startswith('save '):
        # Разбить команду на части для имени файла.
        parts = command.split(maxsplit=1)
        # Сохранить данные в файл.
        staff.save(parts[1])
        logging.info(f"Сохранены данные в файл {parts[1]}.")
    elif command == 'help':
        # Вывести справку о работе с программой.
        print("Список команд:\n")
        print("add - добавить работника;")
        print("list - вывести список работников;")
        print("select <стаж> - запросить работников со стажем;")
        print("load <имя_файла> - загрузить данные из файла;")
        print("save <имя_файла> - сохранить данные в файл;")
        print("help - отобразить справку;")
        print("exit - завершить работу с программой.")
    else:
        raise UnknownCommandError(command)
except Exception as exc:
    logging.error(f"Ошибка: {exc}")
    print(exc, file=sys.stderr)
```

Задание 1. Выполнить индивидуальное задание 2 лабораторной работы 2.19, добавив аннотации типов. Выполнить проверку программы с помощью утилиты mypy.

Результат работы программы.

```
D:\Gitlab\O-op\O-op5\.venv\Scripts\python.exe D:\Gitlab\O-op\O-op5\bitstring.py
b1:         10101010
b2:         10101010
Size b1:    8
b1[2]:      1
AND:        10101010
OR:         10101010
XOR:        00000000
NOT b1:     01010101
Shift L 2:  10101000
Shift R 2:  00101010
After count=6: 101010

Process finished with exit code 0
```

Рисунок 2 – Результат работы программы

```
D:\Gitlab\O-op\O-op5>mypy bitstring.py
Success: no issues found in 1 source file

D:\Gitlab\O-op\O-op5>
```

Рисунок 3 – Проверка с помощью mypy

Ответы на контрольные вопросы:

1. Для чего нужны аннотации типов в языке Python?

Аннотация типов нужна для того, чтобы повысить информативность исходного кода, и иметь возможность с помощью сторонних инструментов производить его анализ.

2. Как осуществляется контроль типов в языке Python?

Использование комментариев, аннотаций и утилиты mypy.

3. Какие существуют предложения по усовершенствованию Python для работы с аннотациями типов?

PEP 3107 – Function Annotations. В нем описывается синтаксис использования аннотаций в функциях Python.

PEP 484 – Type Hints. В нем представлены рекомендации по использованию аннотаций типов.

В PEP 526 – Syntax for Variable Annotations приводится описание синтаксиса для аннотации типов переменных (базируется на PEP 484), использующего языковые конструкции, встроенные в Python.

PEP 563 – Postponed Evaluation of Annotations. Данный PEP вступил в силу с выходом Python 3.7. У подхода работы с аннотация до этого PEP'а был ряд проблем, связанных с тем, что определение типов переменных (в функциях, классах и т.п.) происходит во время импорта модуля, и может сложиться такая ситуация, что тип переменной объявлен, но информации об этом типе ещё нет, в таком случае тип указывают в виде строки – в кавычках. В PEP 563 предлагается использовать отложенную обработку аннотаций, это позволяет определять переменные до получения информации об их типах и ускоряет выполнение программы, т.к. при загрузке модулей не будет тратиться время на проверку типов – это будет сделано перед работой с переменными.

4. Как осуществляется аннотирование параметров и возвращаемых значений функций?

Аннотация для аргумента определяется через двоеточие после его имени.

Аннотация, определяющая тип возвращаемого функцией значения, указывается после её имени с использованием символов ->.

Аннотации не имеют семантического значения для интерпретатора Python и предназначены только для анализа сторонними приложениями.

Доступ к использованным в функции аннотациям можно получить через атрибут __annotations__.

5. Как выполнить доступ к аннотациям функций?

Доступ к аннотациям функций в Python можно получить через атрибут функции annotations.

6. Как осуществляется аннотирование переменных в языке Python?

Аннотирование переменных в языке Python осуществляется через двоеточие с пробелом, за которым следует тип/

7. Для чего нужна отложенная аннотация в языке Python?

Отложенная проверка аннотаций в Python позволяет определять переменные до получения информации об их типах и ускоряет выполнение программы.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе с аннотациями типов при написании программ с помощью языка программирования Python версии 3.x.