Лабораторная работа 4.5

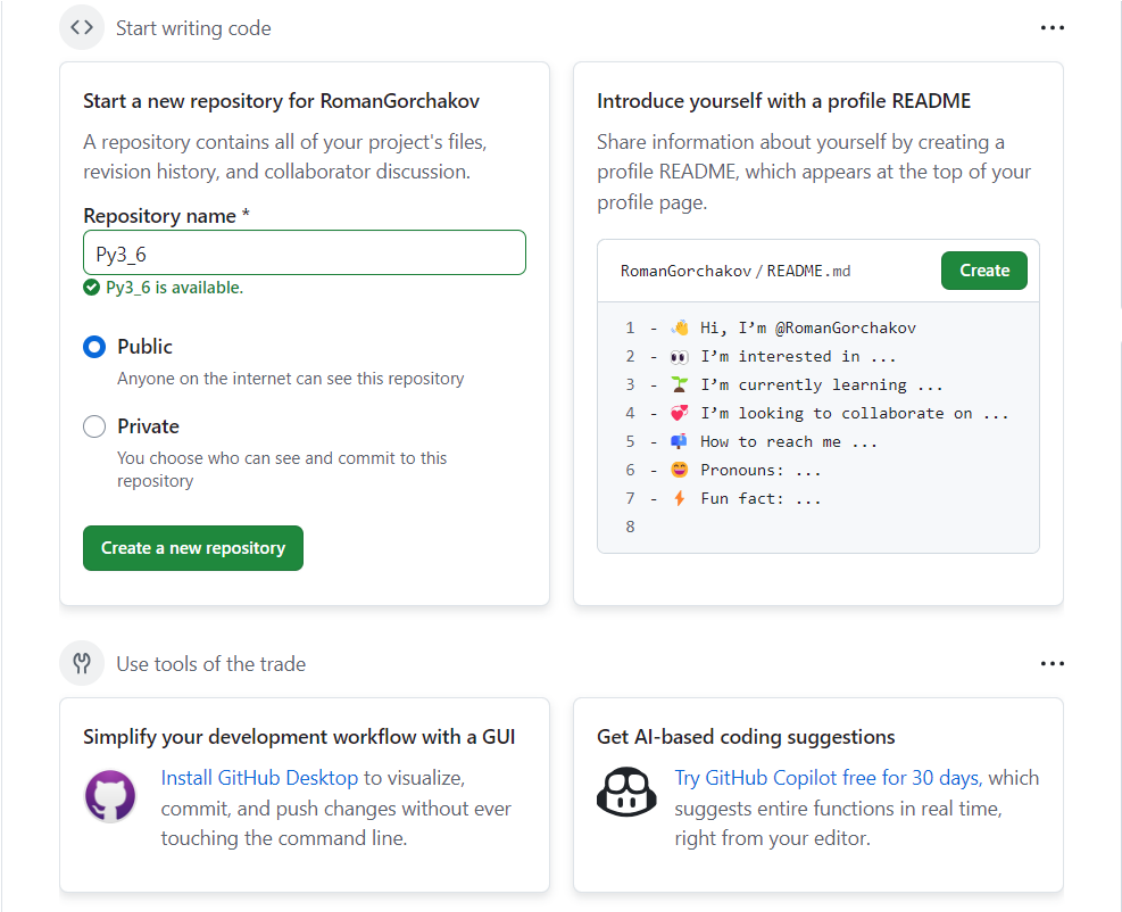
Тема: Аннотация типов

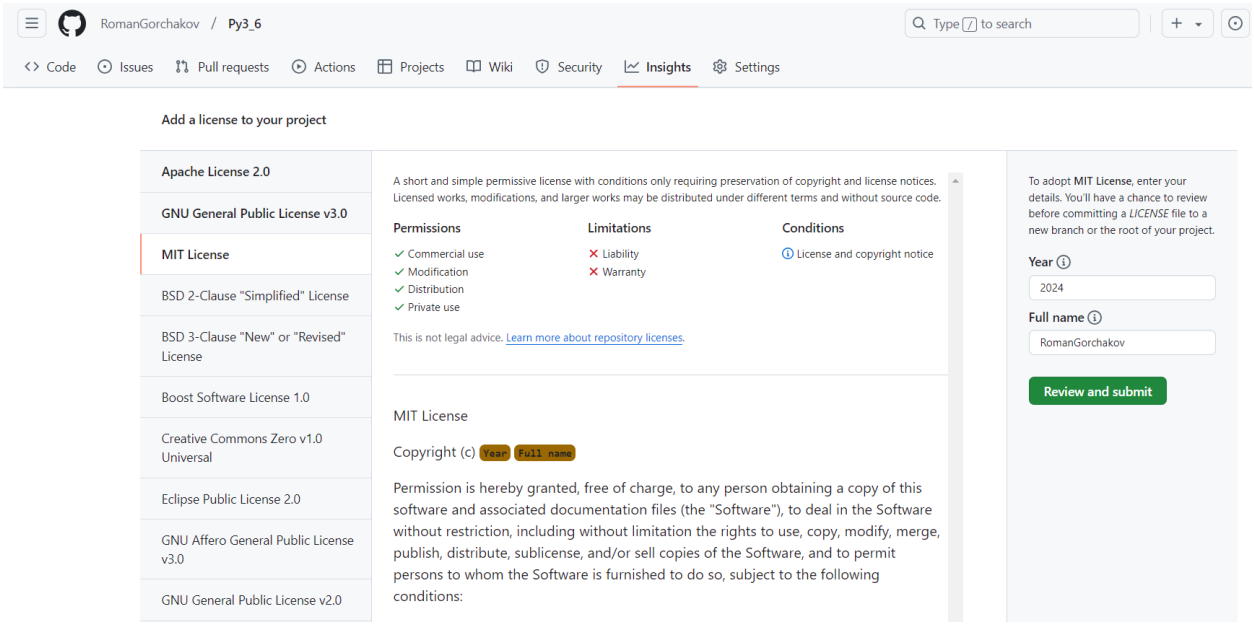
Цель работы: приобретение навыков по работе с аннотациями типов при написании программ с помощью языка программирования Python версии 3.x. Рассмотрен вопрос контроля типов переменных и функций с использованием комментариев и аннотаций. Приведено описание PEP‘ов, регламентирующих работу с аннотациями, и представлены примеры работы с инструментом mypy для анализа Python кода.

Ссылка на GitHub: https://github.com/RomanGorchakov/Py3_6

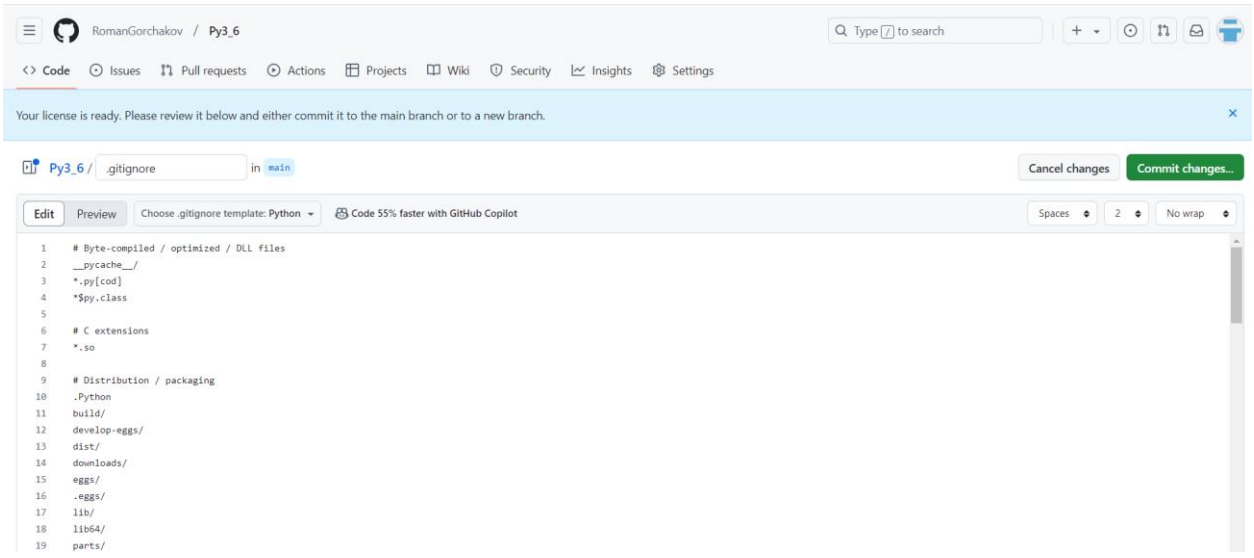
Порядок выполнения работы

1. Создаём новый общедоступный репозиторий, в котором будет использована лицензия MIT и язык программирования Python.

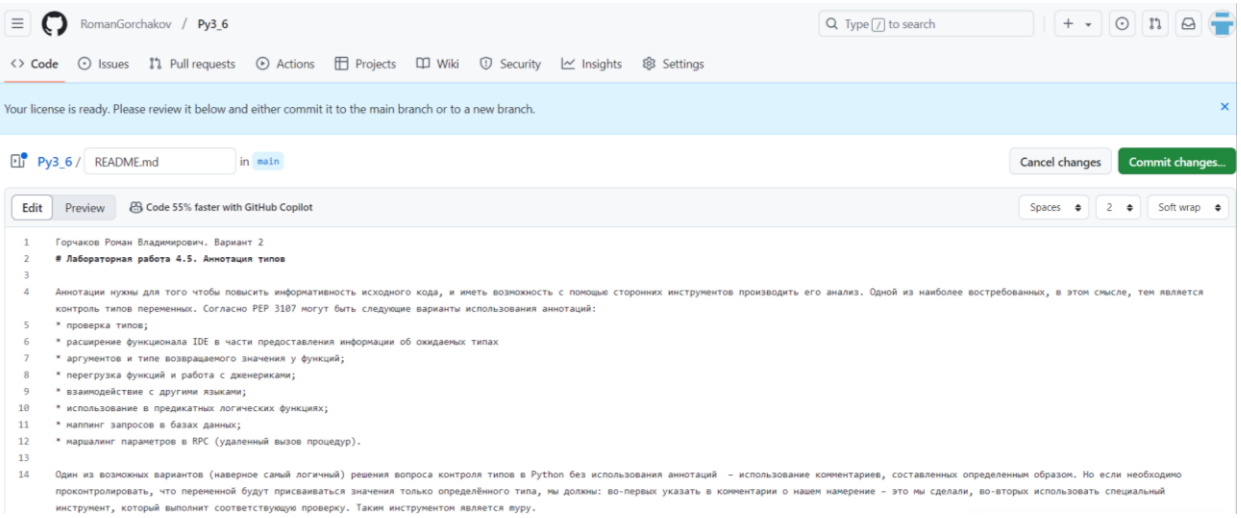




2. Теперь необходимо дополнить файл .gitignore с необходимыми правилами для языка программирования Python. Для этого переходим по ссылке «https://github.com/github/gitignore» и скачиваем оттуда файл «Python.gitignore».



3. Теперь создаём файл «README.md», где вносим ФИО и теоретический конспект лекции. Сохраняем набранный текст через кнопку «Commit changes».



4. В окне «Codespace» выбираем опцию «Create codespace on main». Откроется терминал, куда мы введём команду «git clone», чтобы клонировать свой репозиторий. После этого организуем репозиторий в соответствие с моделью ветвления Git-flow. Для этого введём в терминал команды: «git checkout –b develop» для создания ветки разработки; «git branch feature_branch» для создания ветки функций; «git branch release/1.0.0» для создания ветки релиза; «git checkout main» и «git branch hotfix» для создания веток hotfix. Устанавливаем библиотеки isort, black и flake8 и создаём файлы .pre-commit-config.yaml и environment.yml.

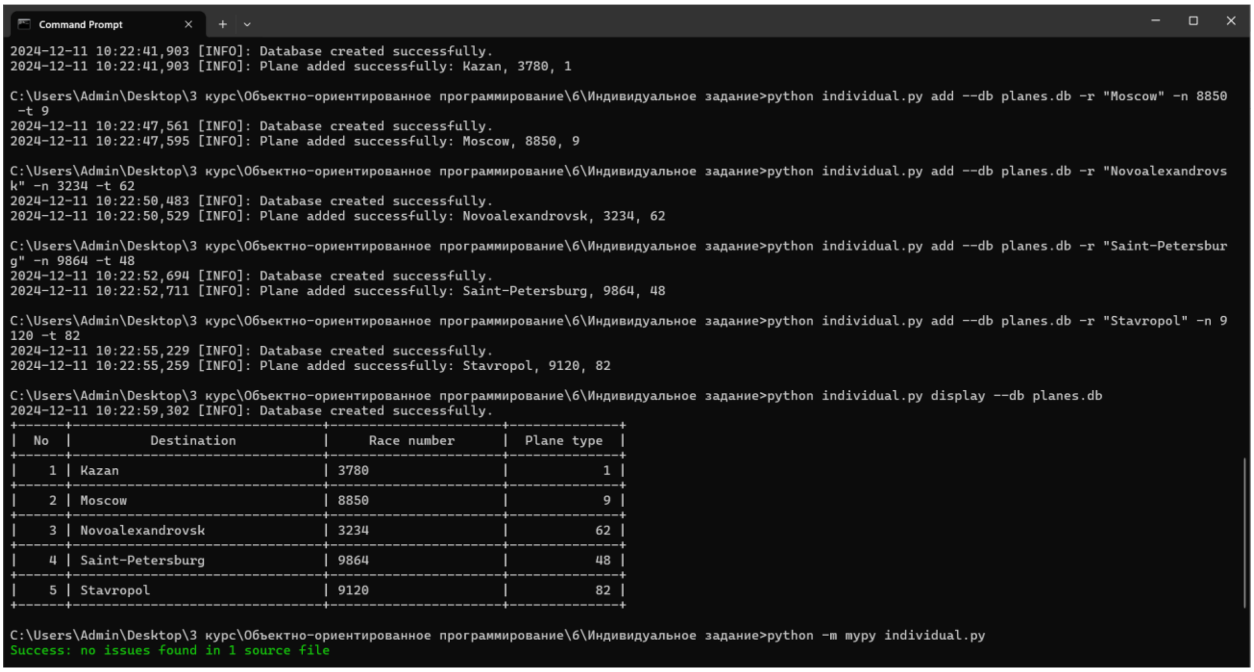
```
@RomanGorchakov →/workspaces/Py3_6 (main) $ git checkout -b develop
Switched to a new branch 'develop'
@RomanGorchakov →/workspaces/Py3_6 (develop) $ git branch feature_branch
@RomanGorchakov →/workspaces/Py3_6 (develop) $ git branch release/1.0.0
@RomanGorchakov →/workspaces/Py3_6 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
@RomanGorchakov →/workspaces/Py3_6 (main) $ git branch hotfix
@RomanGorchakov →/workspaces/Py3_6 (main) $ git checkout develop
Switched to branch 'develop'
@RomanGorchakov →/workspaces/Py3_6 (develop) $
```

```
Collecting black
  Downloading black-24.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl.metadata (79 kB)
Collecting click>=8.0.0 (from black)
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting mypy-extensions>=0.4.3 (from black)
  Downloading mypy_extensions-1.0.0-py3-none-any.whl.metadata (1.1 kB)
Requirement already satisfied: packaging>=22.0 in /home/codespace/.local/lib/python3.12/site-packages (from black) (24.1)
Collecting pathspec>=0.9.0 (from black)
  Downloading pathspec-0.12.1-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: platformdirs>=2 in /home/codespace/.local/lib/python3.12/site-packages (from black) (4.3.6)
Downloading black-24.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 55.3 MB/s eta 0:00:00
Downloading click-8.1.7-py3-none-any.whl (97 kB)
Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Downloading pathspec-0.12.1-py3-none-any.whl (31 kB)
Installing collected packages: pathspec, mypy-extensions, click, black
Successfully installed black-24.10.0 click-8.1.7 mypy-extensions-1.0.0 pathspec-0.12.1
@RomanGorchakov →/workspaces/Py3_6 (develop) $ pip install flake8
Collecting flake8
  Downloading flake8-7.1.1-py2.py3-none-any.whl.metadata (3.8 kB)
Collecting mccabe<0.8.0,>=0.7.0 (from flake8)
  Downloading mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting pycodestyle<2.13.0,>=2.12.0 (from flake8)
  Downloading pycodestyle-2.12.1-py2.py3-none-any.whl.metadata (4.5 kB)
Collecting pyflakes<3.3.0,>=3.2.0 (from flake8)
  Downloading pyflakes-3.2.0-py2.py3-none-any.whl.metadata (3.5 kB)
Downloading flake8-7.1.1-py2.py3-none-any.whl (57 kB)
Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Downloading pycodestyle-2.12.1-py2.py3-none-any.whl (31 kB)
Downloading pyflakes-3.2.0-py2.py3-none-any.whl (62 kB)
Installing collected packages: pyflakes, pycodestyle, mccabe, flake8
Successfully installed flake8-7.1.1 mccabe-0.7.0 pycodestyle-2.12.1 pyflakes-3.2.0
@RomanGorchakov →/workspaces/Py3_6 (develop) $ pre-commit sample-config > .pre-commit.config.yaml
@RomanGorchakov →/workspaces/Py3_6 (develop) $ conda env export > environment.yml
@RomanGorchakov →/workspaces/Py3_6 (develop) $
```

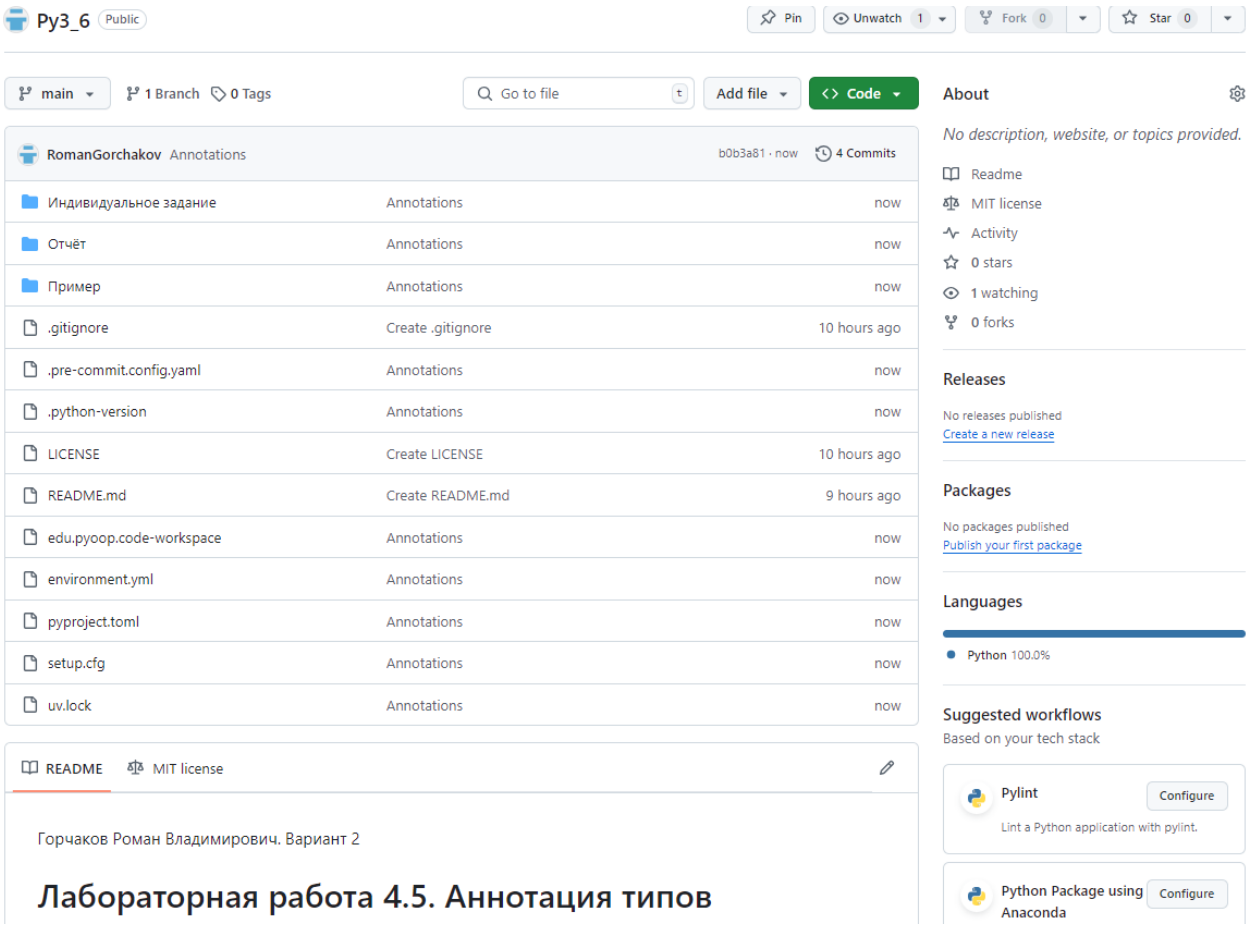
5. Создаём файл «example.py», в котором нужно добавить аннотации типов для примера 1 лабораторной работы 14.

```
Command Prompt
C:\Users\Admin>cd C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\6\Пример

C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\6\Пример>python example.py
>>> add
Фамилия и инициалы? Иванов И.И.
Должность? Директор
Год поступления? 2007
>>> add
Фамилия и инициалы? Петров П.П.
Должность? Бухгалтер
Год поступления? 2010
>>> add
Фамилия и инициалы? Сидоров С.С.
Должность? Главный инженер
Год поступления? 2012
>>> list
+------+----------------------------------+----------------------+----------+
|  №   |             Ф.И.О.               |      Должность       |   Год    |
+------+----------------------------------+----------------------+----------+
|    1 | Иванов И.И.                      | Директор             |     2007 |
|    2 | Петров П.П.                      | Бухгалтер            |     2010 |
|    3 | Сидоров С.С.                     | Главный инженер      |     2012 |
+------+----------------------------------+----------------------+----------+
>>> exit

C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\6\Пример>
```

6. Создаём файл «individual.py», в котором нужно добавить аннотацию типов в индивидуальном заданим 2 лабораторной работы 2.19. Выполняем проверку программы с помощью утилиты mypy.



7. Выполняем коммит файлов в репозиторий Git в ветку разработки, сливаем её с веткой main и отправляем изменения на сервер GitHub.



Контрольные вопросы

1. Для чего нужны аннотации типов в языке Python?

Аннотации нужны для того чтобы повысить информативность исходного кода, и иметь возможность с помощью сторонних инструментов производить его анализ.

2. Как осуществляется контроль типов в язяке Python?

Один из возможных вариантов (наверное самый логичный) решения вопроса контроля типов в Python без использования аннотаций – использование комментариев, составленных определенным образом. Но если необходимо проконтролировать, что переменной будут присваиваться значения только определённого типа, мы должны: во-первых указать в комментарии о нашем намерение – это мы сделали, во-вторых использовать специальный инструмент,

который выполнит соответствующую проверку. Таким инструментом является mypy.

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

PEP 3107 – Function Annotations, является исторически первым из перечисленных выше документов. В нём описывается синтаксис использования аннотаций в функциях Python. Важным является то, что аннотации не имеют никакого семантического значения для интерпретатора Python и предназначены только для анализа сторонними приложениями. Аннотировать можно аргументы функции и возвращаемое ей значение.

PEP 484 – Type Hints. В нём представлены рекомендации по использованию аннотаций типов. Аннотация типов упрощает статический анализ кода, рефакторинг, контроль типов в рантайме и кодогенерацию, использующую информацию о типах.

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 сложится такая ситуация, что тип переменной объявлен, но информации об этом типе ещё нет, в таком случае тип указывают в виде строки – в кавычках.

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 ее имени с использованием символов ->.

5. Как выполнить доступ к аннотациям функций?

Доступ к использованным в функции аннотациям можно получить через атрибут __annotations__, в котором аннотации представлены в виде словаря, где ключами являются атрибуты, а значениями – аннотации. Возвращаемое функцией значение хранится в записи с ключом return.

6. Как осуществляется аннотирование переменных в языке Python?

```
var = value # type: annotation
var: annotation; var = value
var: annotation = value
```

7. Для чего нужна отложенная аннотация в языке Python?

Отложенная аннотация в языке Python нужна для ускорения выполнения программы. Это достигается за счёт того, что переменные определяются до получения информации об их типах, а проверка типов не тратится при загрузке модулей, а делается перед работой с переменными.