УДК 004.426 **Информационные технологии**

Халевин Тимофей Анатольевич, *студент факультета информатики и вычислительной техники, Хакасский государственный университет имени Н.Ф. Катанова, г. Абакан, Россия*

Голубничий Артем Александрович, *научный руководитель, старший преподаватель кафедры ПОВТиАС, Хакасский государственный университет имени Н.Ф. Катанова, г. Абакан, Россия*

АННОТАЦИИ ТИПОВ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON

Аннотация: В основном аннотации используются для функций, чтобы упроситить понимание кода программисту который будет с ним работать, но иногда они используются и для переменных. В данной статье описан механизм аннотирования данных в языке программирования Python.

Ключевые слова: Python, аннотации типов, подсказки типа.

Annotation: Annotations are mostly used for functions to make it easier for the programmer who will be working with them to understand the code, but sometimes they are used for variables as well. This article describes the mechanism of annotating data in the Python programming language.

Keywords: Python, annotating data, type hints.

Python – язык программирования с динамической строгой типизацией. Это означает что нам нет необходимости явно указывать к какому типу данных принадлежит переменная, Python поймет это сам. Для примера мы можем создать сначала целочисленную переменную integer и поместить в нее целое число, а в

следующей строке передать в неё строковое значение и ошибок это не вызовет. Пример представлен на рис. 1.

```
1  integer = 15
2  print(integer)  # Выведет число 15
3  integer = 'Hello!'
4  print(integer)  # Выведет строку 'Hello!'
```

Рисунок 1 – Передача разных типов в одну переменную

Это удобно для небольших проектов, где все переменные на виду и их не очень много, но что делать если проект очень большой, и программист может просто не помнить какого типа должна быть та или иная переменная? В этом нам и помогут аннотации типов.

Аннотации типов позволяют добавлять подсказки о типах переменных. Они используются, чтобы информировать читателя кода, каким должен быть тип переменной. Их так же называют Type Hints – подсказки типа. Начиная с версии 3.6 языка Python были добавлены аннотации типов.

Давайте рассмотрим пример кода где мы укажем через аннотацию типов какого типа данных должна быть переменная. Пример представлен уже с использованием аннотации (рис. 2).

```
1  integer: int = 15
2  print(integer)  # Выведет число 15
3  integer = 'Hello!'
4                  Expected type 'int', got 'str' instead
5  print(integer)  # Выведет строку 'Hello!'
```

Рисунок 2 –Аннотация типа int

Как видим IDE подсказывает нам, что переменная *integer* ожидалась целочисленной, но вместо целого числа была получена строка. При этом работа

программы не поменяется, она точно так же выведет сначала число *15*, а затем строку *'Hello!'*.

Аннотации типов для переменных не означают что мы не можем присвоить в переменную другой тип данных, они лишь показывают какой тип данных должен быть у этой переменной [1].

Теперь давайте разберем пример аннотации с функцией. Для этого напишем функцию *add* которая в качестве входных данных будет принимать две переменные, *a* и *b* и возвращать нам сумму этих переменных. То есть мы реализуем функцию сложения двух переменных:

```
def add(a, b):
    return a + b
```

Особенность заключается в том, что данная функция одинаково отработает с целочисленными, вещественными, строковыми переменными. Даже если мы передадим две переменных типа *list*, то функция отработает нормально. Пример вызова функции с разными данными (рис. 3).

```
1  def add(a, b):
2      return a + b
3
4
5  print(add(3, 5))  # Выведет 8
6  print(add(5.5, 9.5))  # Выведет 15.0
7  print(add('Hello,', ' world!'))  # Выведет строку 'Hello, world!'
8  print(add([1, 2, 3], [4, 5]))  # Выведет список [1, 2, 3, 4, 5]
```

Рисунок 3 – Вызов функции add с разными входными типами данных

В этом и заключается неоднозначность такой функции. Мы можем передать несколько типов данных, и функция отработает без ошибок.

Чтобы мы понимали какой тип данных ожидается в функции мы можем сделать аннотацию типов. Подсветка кода с аннотацией функции (рис. 4).

```python
def add(a: int, b: int):
    return a + b


print(add(3, 5))  # Выведет 8
print(add(5.5, 9.5))  # Выведет 15.0

print(add('Hello,', ' world!'))  # Выведет строку 'Hello, world!'

print(add([1, 2, 3], [4, 5]))  # Выведет список [1, 2, 3, 4, 5]
```

Expected type 'int', got 'list[int]' instead

Рисунок 4 – Подсветка IDE при аннотации

Как мы видим подсвечиваются все вызовы функции переменные в которых не являются целочисленными, при этом функция также будет возвращать все значения. В примере нам пишет что ожидался целочисленный тип, а получен тип *list* с целочисленными значениями.

Давайте посмотрим, где аннотации хранятся внутри функции. Для этого вызовем *__annotations__* нашей функции

```
def add(a: int, b: int):
    return a + b
print(add.__annotations__)
```

Представлен вывод аннотации внутри функции (рис. 5).

```python
def add(a: int, b: int):
    return a + b


print(add.__annotations__)  # Выведет {'a': <class 'int'>, 'b': <class 'int'>}
```

Рисунок 5 – Аннотации типов внутри функции

При вызове *__annotations__* мы получаем словарь, в котором указано, что переменная *a* и переменная *b* аннотированы целочисленным типом данных.

Так же можно аннотировать вывод функции, делается это следующим образом:

def add(a: int, b: int) –> int:

    return a + b

Для аннотации вывода после указания всех аргументов принимаемых функцией нужно поставить –> и после этого указать тип данных который планируется возвращать.

Так же теперь у нас изменился вывод *__annotations__*. В словаре добавился return, который проаннотирован целочисленным типом. Вывод *__annotations__* представлен на рис. 6.

```
1  def add(a: int, b: int) -> int:
2      return a + b
3
4
5  print(add.__annotations__)  # Выведет {'a': <class 'int'>, 'b': <class 'int'>, 'return': <class 'int'>}
```

Рисунок 6 – Аннотация вывода в функции

Разберем так же пример со списками. Для примера создадим функцию *list_int_to_list_str* которая будет принимать одну переменную и проаннотируем её:

def list_int_to_list_str(lst: list[int]) –> list[str]:

    return list(map(str, lst))

Эта функция будет принимать список целочисленных значений и возвращать список состоящий из строк. То есть каждый элемент вместо целого числа станет строкой. Примеры работы с этой функцией (рис. 7).

```
1  def list_int_to_list_str(lst: list[int]) -> list[str]:
2      return list(map(str, lst))
3
4
5  print(list_int_to_list_str.__annotations__)  # Выведет {'lst': list[int], 'return': list[str]}
6  print(list_int_to_list_str([1, 2, 3, 4, 5]))  # Выведет список ['1', '2', '3', '4', '5']
7  print(list_int_to_list_str([True]))  # Выведет список ['True']
8  print(list_int_to_list_str(['1', '2', '3', '4', 5]))    # Выведет список ['1', '2', '3', '4', '5']
9  print(list_int_to_list_str(['1', '2', '3', '4', '5']))    # Выведет список ['1', '2', '3', '4', '5']
10
   Expected type 'list[int]', got 'list[str]' instead
```

Рисунок 7 – Пример аннотаций для списка

Как мы видим IDE подсвечивает нам только один вызов, в котором все элементы являются строковыми. *True* в языке Python является единицей, по этому IDE его не подсвечивает, но это уже особенности языка. В третьем вызове функции у нас не весь список состоит из строковых значений, по этому он не подсвечивается.

Возможно мы захотим чтобы функция принимала не только список с целочисленными значениями, но еще и строковыми, для этого нам достаточно указать тип данных через символ | [2].

Вот пример такой функции:

```
def list_int_to_list_str(lst: list[int | str]) –> list[str]:
    return list(map(str, lst))
```

Заключение

Важно сказать, что аннотации типов не гарантируют что в функцию или переменную придет то, что нам нужно. Они лишь подсказывают программисту который использует эти функции или переменные о том, какой тип данных там должен быть. Так же необходимо понимать, что если ваша программа не большая, то аннотировать переменные или функции не обязательно.

Библиографический список:

1. Бэрри П. Изучаем программирование на Python [Текст] / П. Бэрри. – М.: Вильямс, 2014. – 243 с.

2. Гэддис Т. Начинаем программировать на Python [Текст] / Т. Гэддис. – СПб.: БХВ-Петербург, 2021. – 768 с.