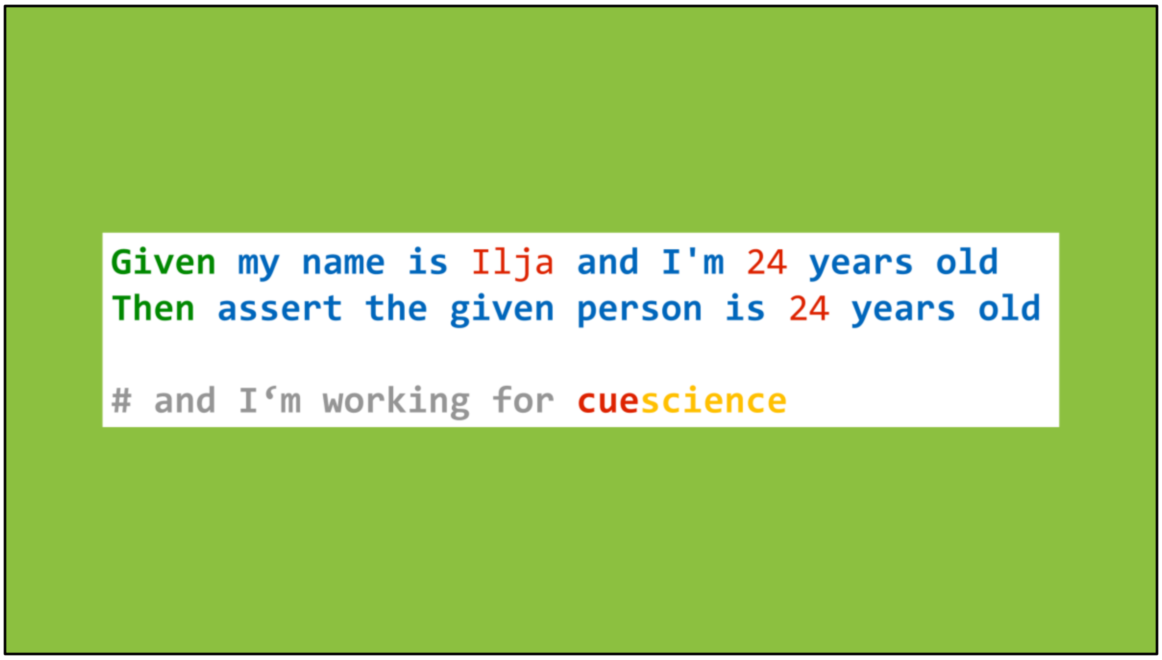
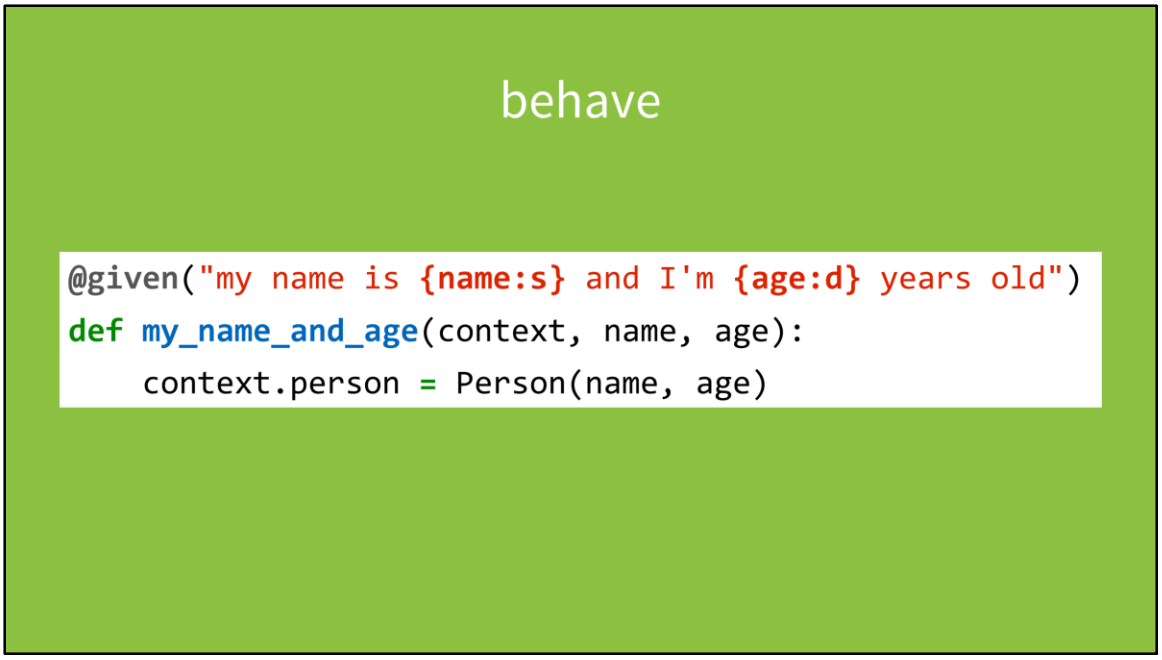


```
Given my name is Ilja and I'm 24 years old
Then assert the given person is 24 years old

# and I‘m working for cuescience
```

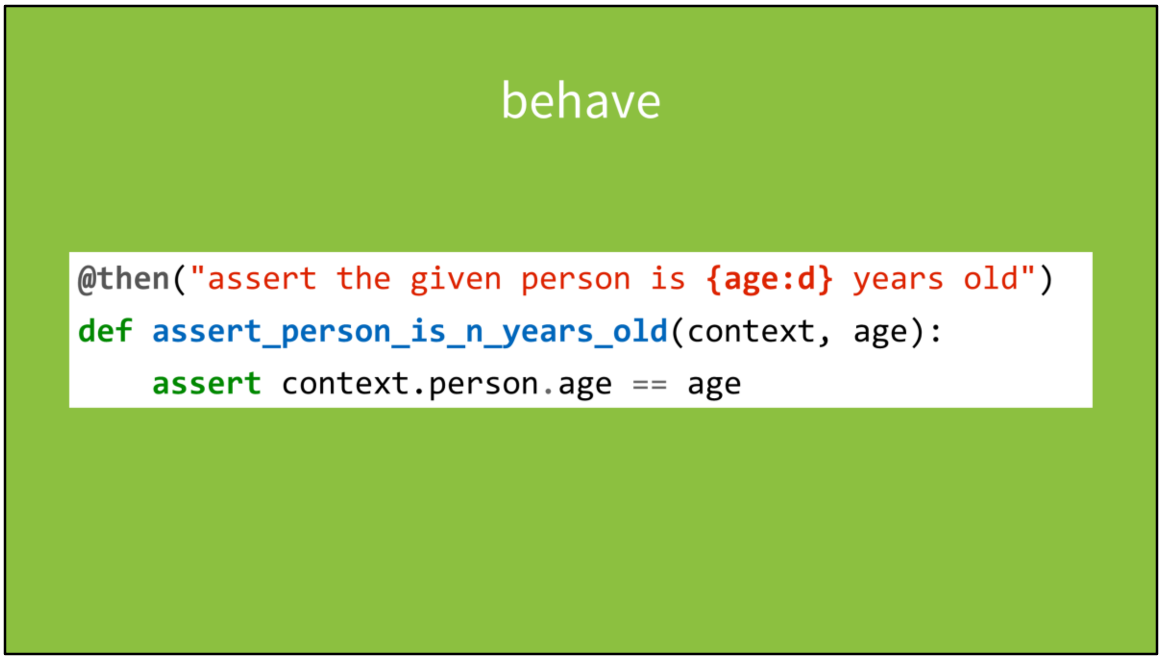
Hi everyone,I want to show you one of my projects which I wrote for my daily work.
But at first, I provide you with some context:
What you see here is actually a test. Its written in natural language. This style of writing tests is often called behaviour driven development.



behave

```
@given("my name is {name:s} and I'm {age:d} years old")
def my_name_and_age(context, name, age):
    context.person = Person(name, age)
```

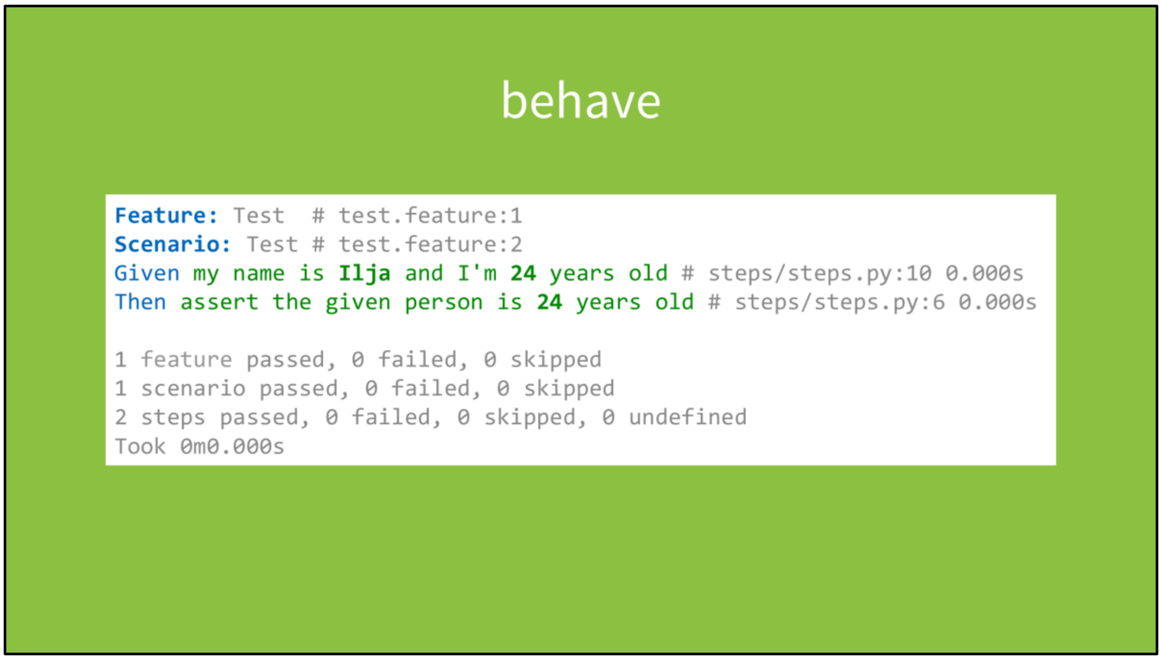
To map this text to actual code, we need a library which is called behave.
What we are doing here is to create a person instance with the given arguments which are injected by behave. After the person is created we put it into the context, which is like a dictionary for sharing data between sentences.



behave

```
@then("assert the given person is {age:d} years old")
def assert_person_is_n_years_old(context, age):
    assert context.person.age == age
```

The second sentence contains the assertion that the age of the person created is set correctly.I quite simple example here.

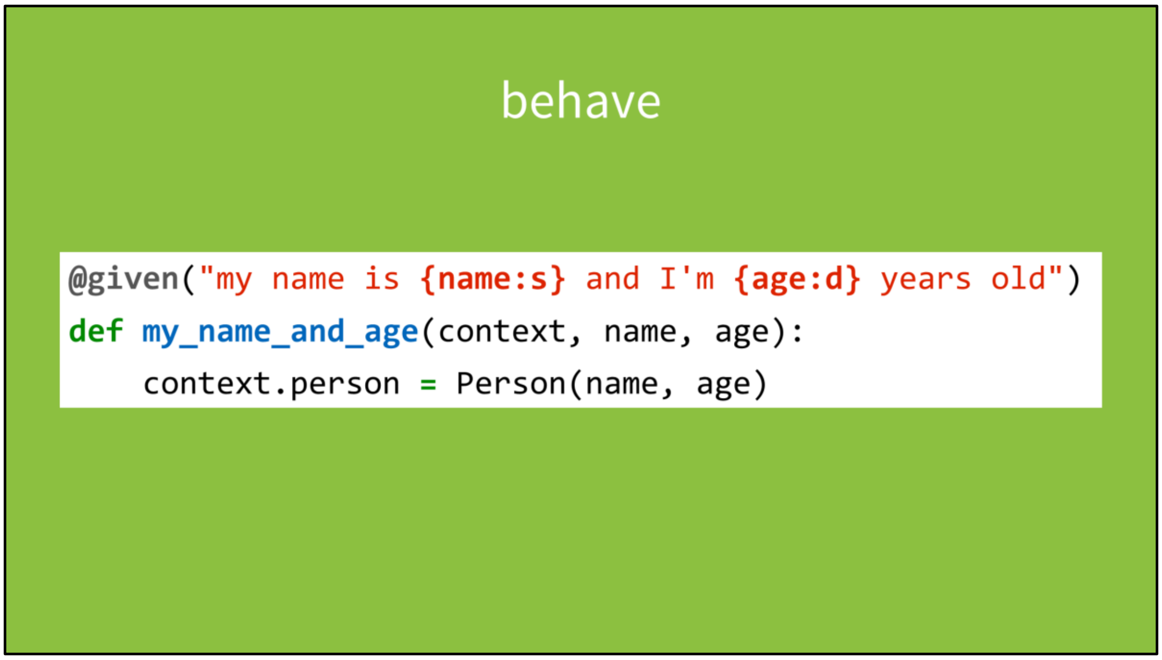


behave

```
Feature: Test  # test.feature:1
Scenario: Test # test.feature:2
Given my name is Ilja and I'm 24 years old # steps/steps.py:10 0.000s
Then assert the given person is 24 years old # steps/steps.py:6 0.000s

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
2 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.000s
```

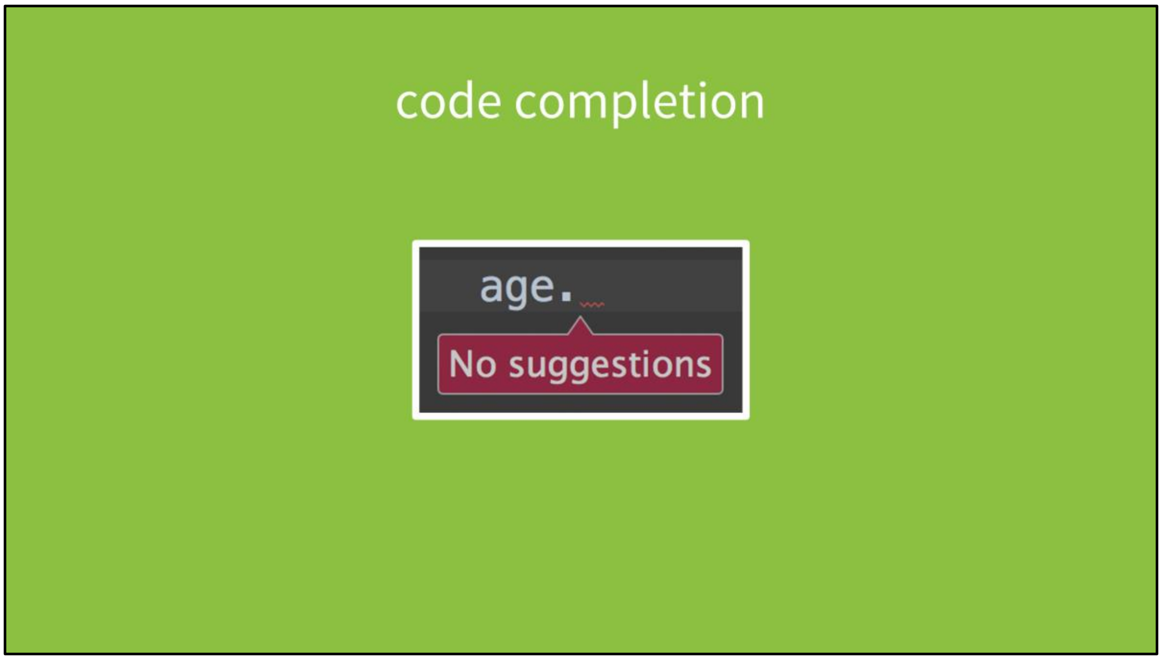
After we done this, we can run this natural language test and see that it will pass.



behave

```python
@given("my name is {name:s} and I'm {age:d} years old")
def my_name_and_age(context, name, age):
    context.person = Person(name, age)
```

The problem with this definition here, that we annotate the type of the arguments in this string here.



code completion

```
age.
```

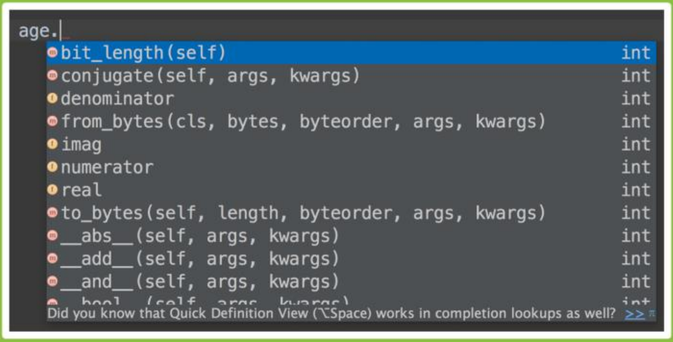
No suggestions

This means that we will not have code completion working.

Python 3

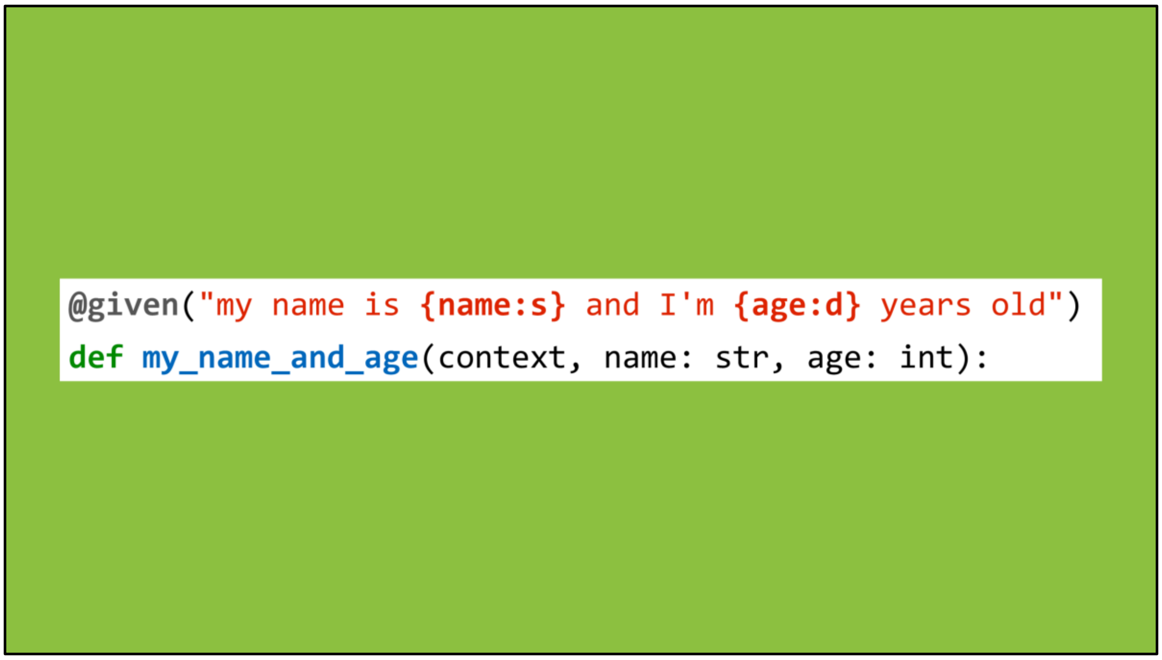
```
@given("my name is {name:s} and I'm {age:d} years old")
def my_name_and_age(context, name: str, age: int):
```

Python 3 to the rescue. Lets add some function annotations to the function.

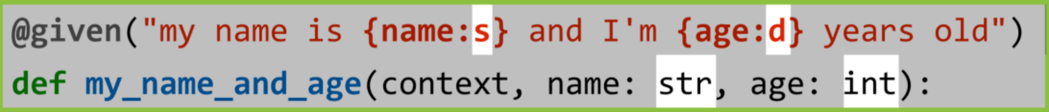


```
age.|
bit_length(self)                                          int
conjugate(self, args, kwargs)                             int
denominator                                               int
from_bytes(cls, bytes, byteorder, args, kwargs)           int
imag                                                      int
numerator                                                 int
real                                                      int
to_bytes(self, length, byteorder, args, kwargs)           int
__abs__(self, args, kwargs)                               int
__add__(self, args, kwargs)                               int
__and__(self, args, kwargs)                               int
__bool__(self, args, kwargs)                              int
Did you know that Quick Definition View (⌥Space) works in completion lookups as well? >>
```

So here we go: Code completion is working. Thank you JetBrains.



```python
@given("my name is {name:s} and I'm {age:d} years old")
def my_name_and_age(context, name: str, age: int):
```

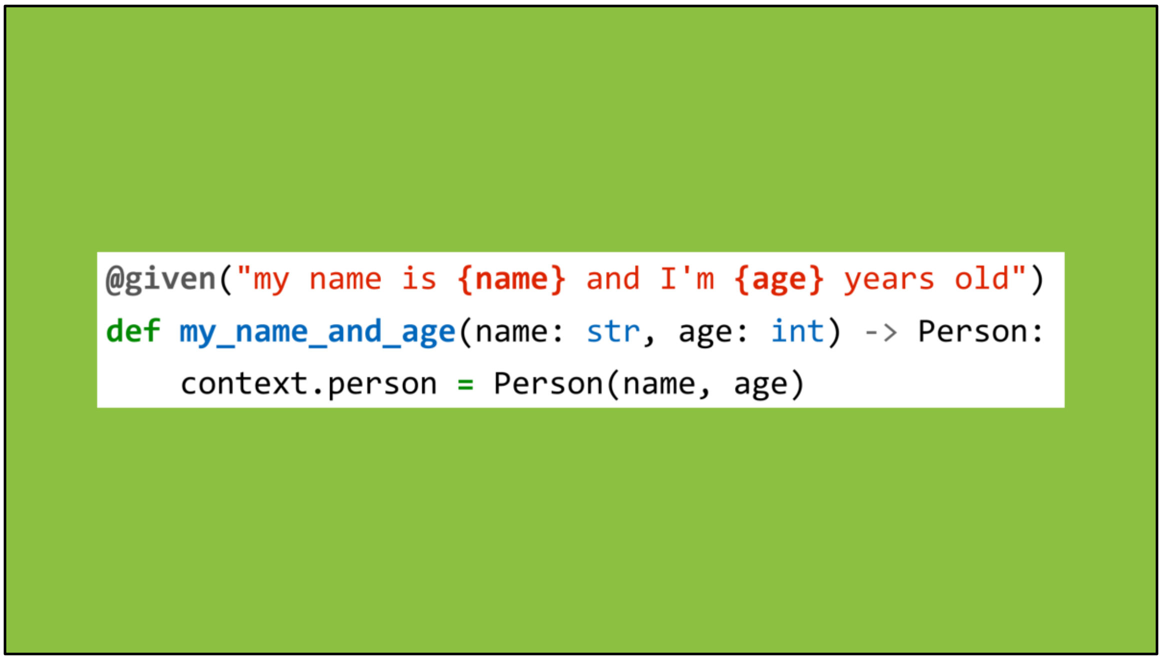


```
@given("my name is {name:s} and I'm {age:d} years old")
def my_name_and_age(context, name: str, age: int):
```

But if we look at this code, there is a kind of redundancy here.

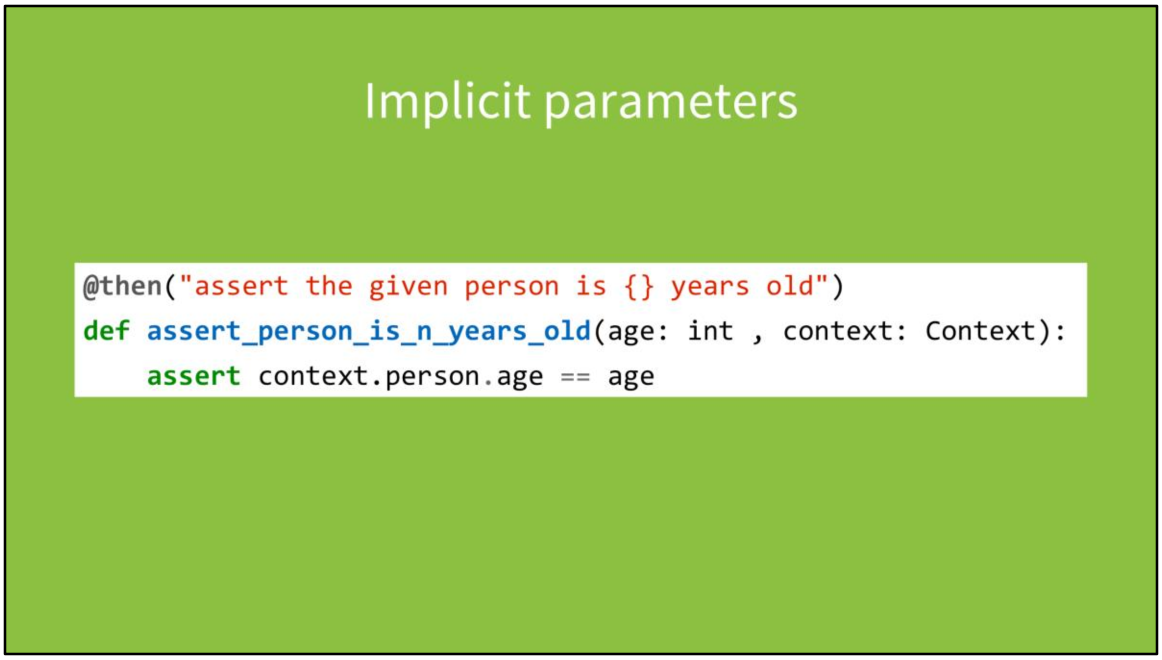
Goat

And this was the moment I started goat. Goat is an plugin for behave and will work only with python 3.



```python
@given("my name is {name} and I'm {age} years old")
def my_name_and_age(name: str, age: int) -> Person:
    context.person = Person(name, age)
```

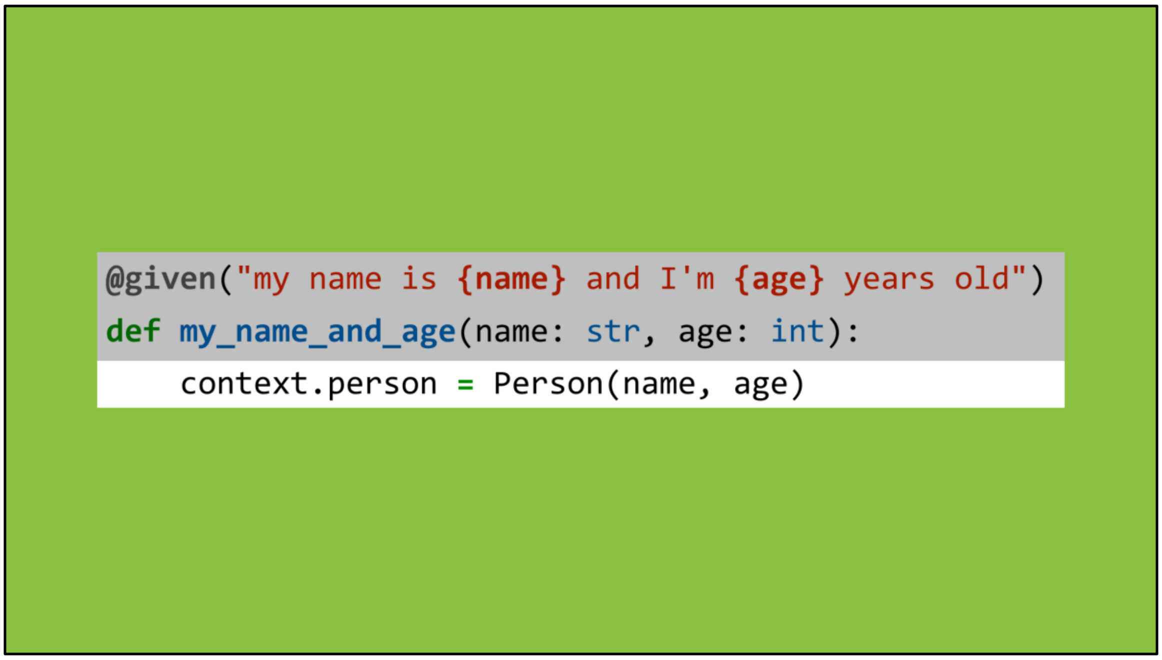
With goat we can remove the types from the string.



Implicit parameters

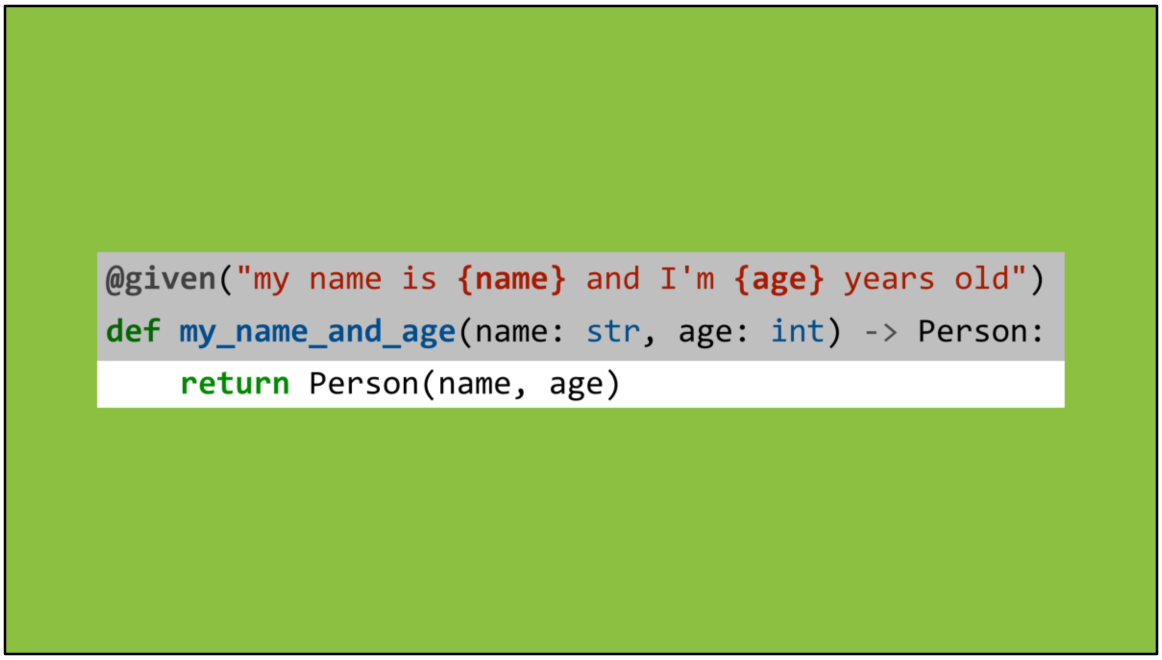
```
@then("assert the given person is {} years old")
def assert_person_is_n_years_old(age: int , context: Context):
    assert context.person.age == age
```

If we look at the second function, we see that I also annotated the context argument with the correct type.



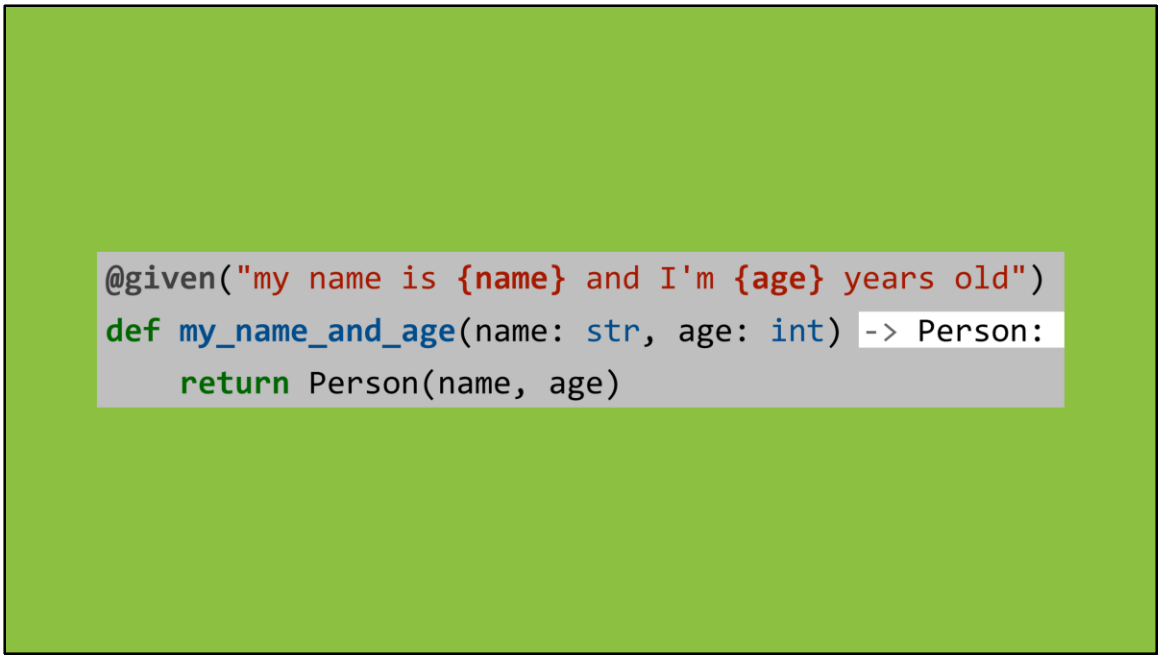
```python
@given("my name is {name} and I'm {age} years old")
def my_name_and_age(name: str, age: int):
    context.person = Person(name, age)
```

But this is somehow still not good enough... Wouldn't it be nicer to replace this line, with something like this?



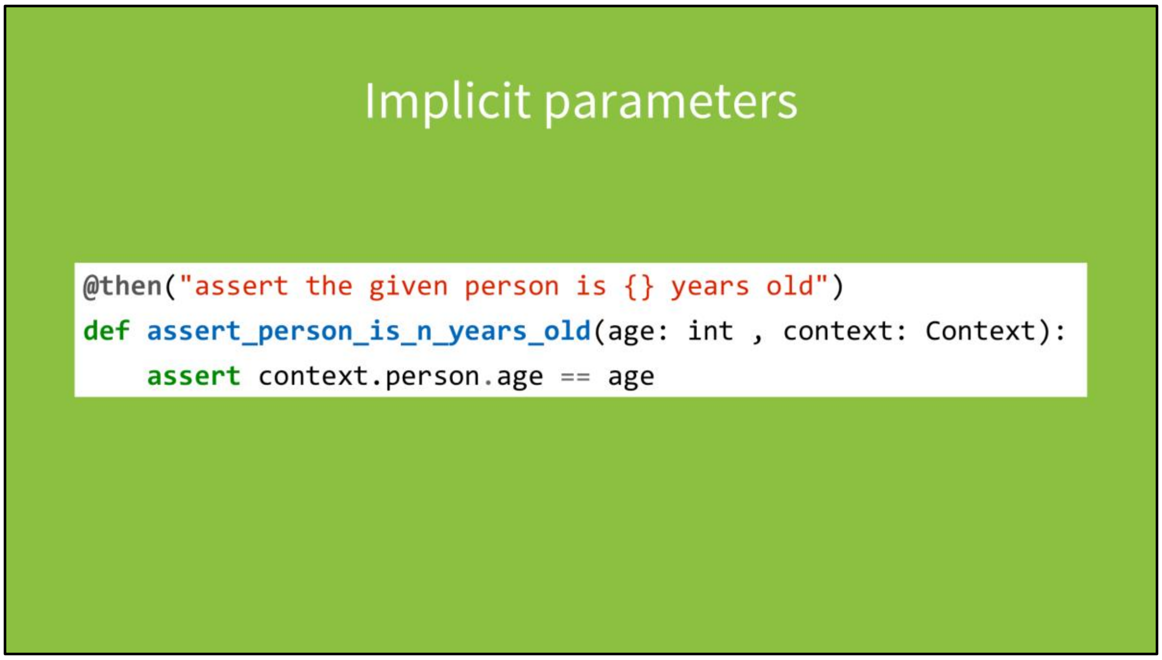
```
@given("my name is {name} and I'm {age} years old")
def my_name_and_age(name: str, age: int) -> Person:
    return Person(name, age)
```

And just return the person we created.



```
@given("my name is {name} and I'm {age} years old")
def my_name_and_age(name: str, age: int) -> Person:
    return Person(name, age)
```

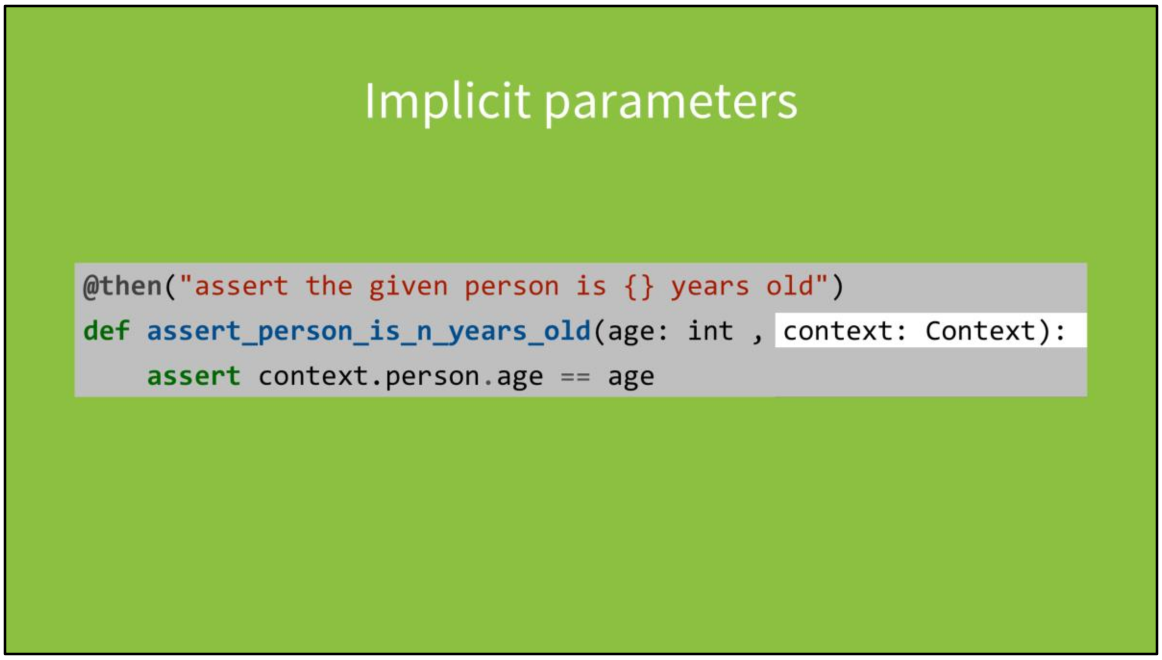
Also add the the return type here...



Implicit parameters

```python
@then("assert the given person is {} years old")
def assert_person_is_n_years_old(age: int , context: Context):
    assert context.person.age == age
```

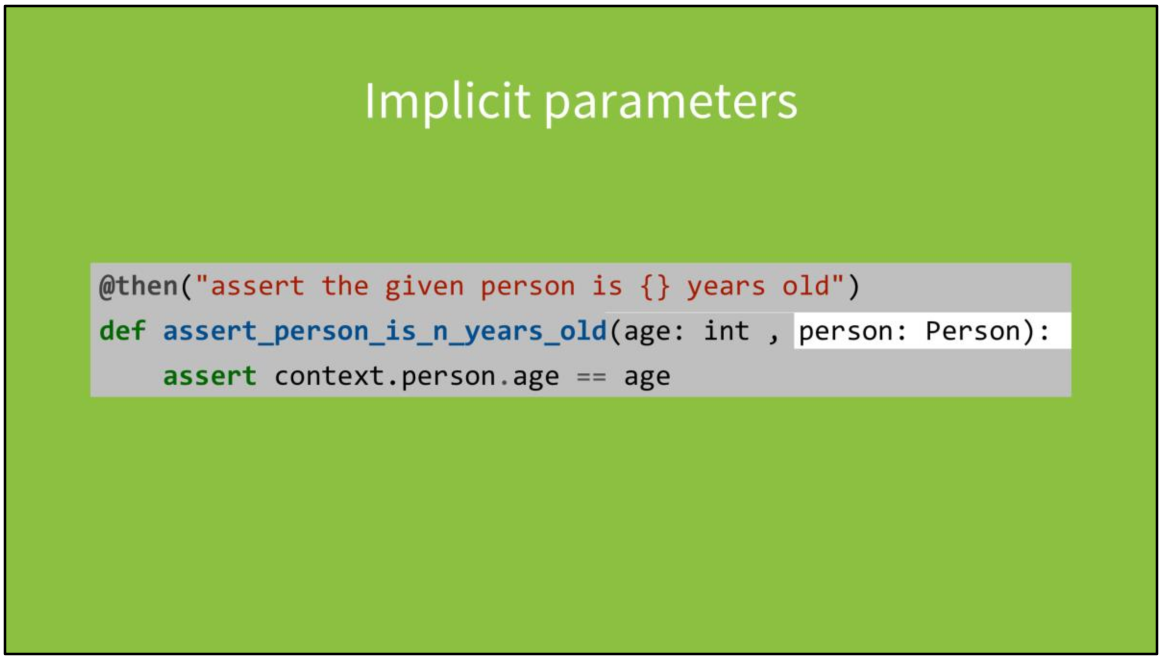
With goat we can use the returned object in this function



Implicit parameters

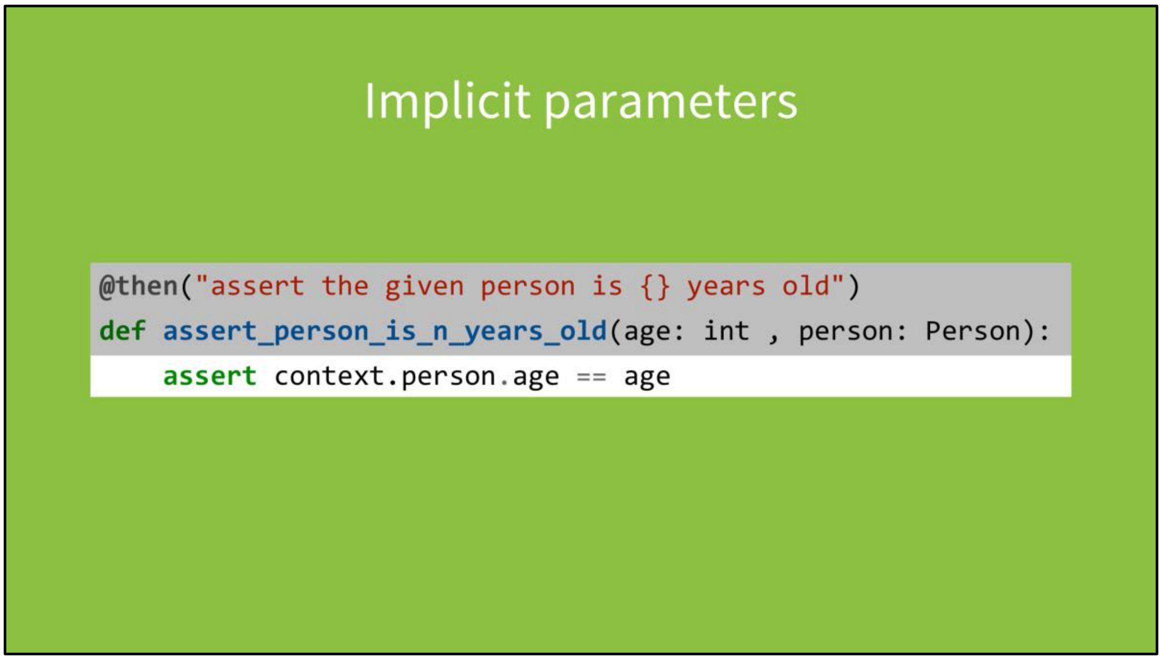
```
@then("assert the given person is {} years old")
def assert_person_is_n_years_old(age: int , context: Context):
    assert context.person.age == age
```

by replacing the context with a person argument. Goat will take care of injecting the person object here.



Implicit parameters

```
@then("assert the given person is {} years old")
def assert_person_is_n_years_old(age: int , person: Person):
    assert context.person.age == age
```



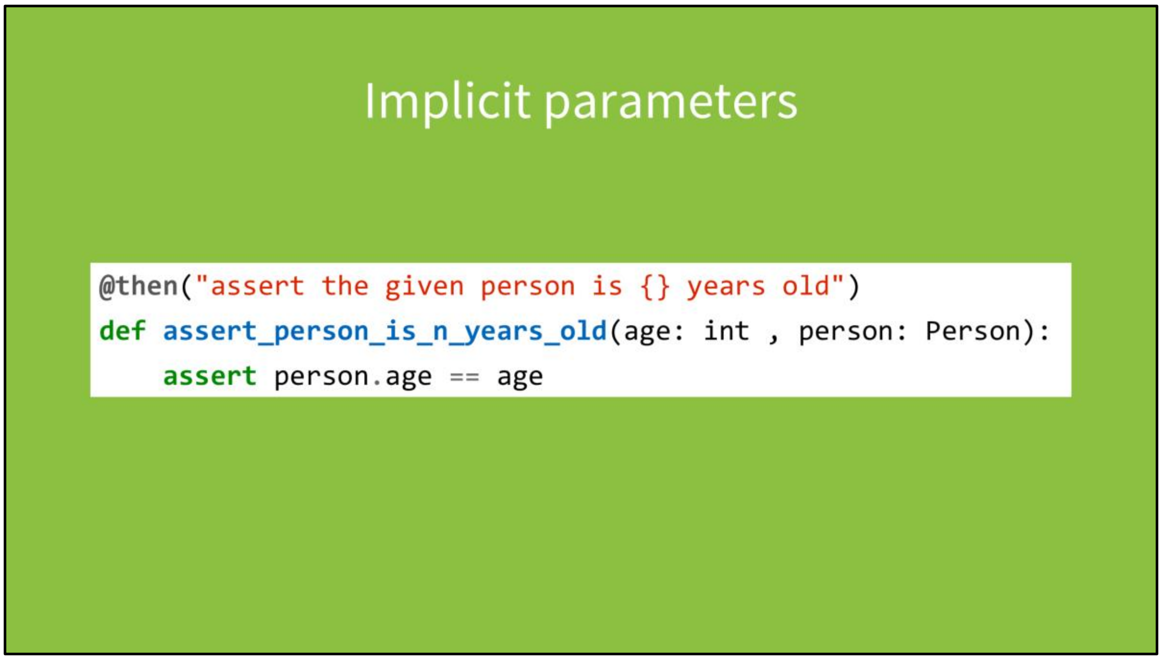
Implicit parameters

```python
@then("assert the given person is {} years old")
def assert_person_is_n_years_old(age: int , person: Person):
    assert context.person.age == age
```

Now we can also replace this line with this one.

Implicit parameters

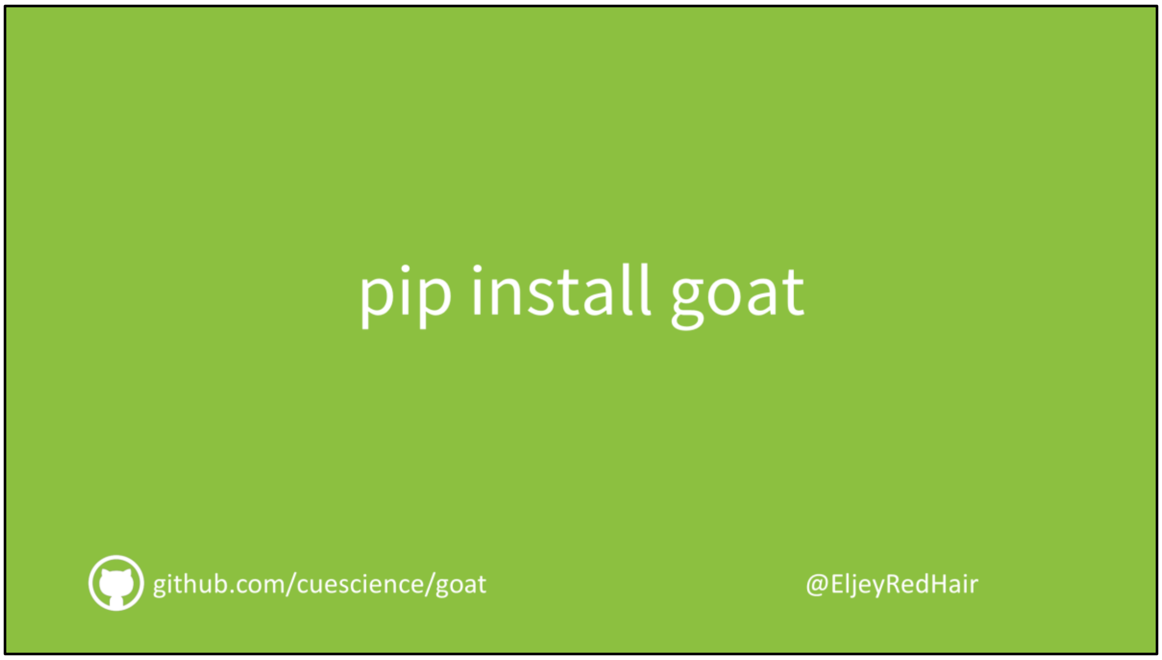
```
@then("assert the given person is {} years old")
def assert_person_is_n_years_old(age: int , person: Person):
    assert person.age == age
```



Implicit parameters

```python
@then("assert the given person is {} years old")
def assert_person_is_n_years_old(age: int , person: Person):
    assert person.age == age
```

Much better now.



pip install goat

github.com/cuescience/goat

@EljeyRedHair

This is goat. Please give it a try. Report issues, write feedback, create pull requests. Help me, where you can. Thank you