MAX
BRENNER
DEVOPS
FREELANCER

Python's str() vs. repr()

1 min read - April 10, 2015 - [python]

Ever wondered what happens when you call Python's built-in str(X), with X being any object you want? The return value of this function depends on the two magic methods __str__ being the first choice and __repr__ as a fallback. But what's the difference between them? When having a look at the docs

```
>>> help(str)
'Create a new string object from the given object.'
>>> help(repr)
'Return the canonical string representation of the object.'
```

they seem to be fairly similar. Let's see them in action:

```
>>> str(123)
'123'
>>> repr(123)
'123'
```

Alright no difference for now.

```
>>> str('Python')
'Python'
>>> repr('Python')
"'Python'"
```

A second pair of quotes around our string. Why?
With the return value of *repr()* it should be possible to recreate our object using eval(). This function takes a string and evaluates it's content as Python code. In our case passing *"'Python'"* to it works, whereas *'Python'* leads to an error cause it's interpreted as the variable *Python* which is of course undefined. Let's move on…

```
>>> import datetime
>>> now = datetime.datetime.now()
>>> str(now)
'2015-04-04 20:51:31.766862'
>>> repr(now)
'datetime.datetime(2015, 4, 4, 20, 51, 31, 766862)'
```

This is some significant difference. While *str(now)* computes a string containing the value of *now*, *repr(now)* again returns the Python code needed to rebuild our *now* object.</br>

The following clues might help you to decide when to use which:

str()
- make object readable
- generate output for end user

repr()
- need code that reproduces object
- generate output for developer

These points should also be considered when writing __str__ or __repr__ for your classes.

Comments

If you have questions or want to give feedback feel free to contact me.