THE MAGIC OF PYTHON UNDERSCORES

PyData Bristol, 26.09.2019

THE SINGLE UNDERSCORE

- = “I don’t care about this variable”

- Example:

```
>>> magic_var = [‘I’, ‘am’, 'magic’]
>>> _,  _, v = magic_var
>>> _
‘am’
>>> v
‘magic’
```

THE SINGLE UNDERSCORE

- = return the last seen expression

- Example:

```
>>> 1+1
2
>>> _
2
>>> []
[]
>>> _
[]
```

(cont'd)

```
>>> _.append('i')
>>> _.append ('am')
>>> _.append ('magic!')
>>> _
 >>> ['I','am','magic']
 2
>>> _
2
```

THE SINGLE UNDERSCORE

- = return the last seen calculation…not always!

- Exception 🤯🤯🤯

```
>>> magic_var = ['I', 'am', 'magic']
>>> _, _, v = magic_var
>>> 1+1
2
>>> _
'magic'
```

THE SINGLE LEADING UNDERSCORE

- = “This variable is for internal use”
  - Just a programmer’s convention
  - There are exceptions… 👾 👾 👾
- Example:

```
>>> class Magic:
        def __init__(self):
            self.normal_magic = 10
            self._mega_magic = 1000000
>>> magic = Magic()
>>> magic.normal_magic
10
>>> magic._mega_magic
1000000
```

THE SINGLE LEADING UNDERSCORE

- = “This function is for internal use” a.k.a. “the wildcard conundrum”

- Exception 🤯🤯🤯

leading_underscore.py

```
>>> def external_magic():
        return "I am external. My magic is safe.”
>>> external_magic()
        'I am external. My magic is safe.’
>>> def _internal_magic():
        return "I am internal. Dangerous magic!!!”
>>> _internal_magic()
        'I am internal. Dangerous magic!!!’
```

leading_underscore_test1.py

```
>>> from leading_underscore import *

>>> external_magic()
     'I am external. My magic is safe’
>>> _internal_magic()
    ---------------------------------------------------------------
    NameError                 Traceback (most recent call last)
    <ipython-input-3-79deaafbdcae> in <module>()
    ----> 1 _internal_magic()
    NameError: name '_internal_magic' is not defined
```

THE SINGLE LEADING UNDERSCORE

- = “This function is for internal use” …. neah! No wildcard = no worries!

- Exception 🤯🤯🤯

leading_underscore.py

```
>>> def external_magic():
        return "I am external. My magic is safe.”
>>> external_magic()
        'I am external. My magic is safe.’
>>> def _internal_magic():
        return "I am internal. Dangerous magic!!!”
>>> _internal_magic()
        'I am internal. Dangerous magic!!!’
```

leading_underscore_test2.py

```
>>> from leading_underscore import external_magic
>>> from leading_underscore import _internal_magic

>>> external_magic()
      'I am external. My magic is safe’
>>> _internal_magic()
      'I am internal. Dangerous magic!!!’
```

THE SINGLE LEADING UNDERSCORE

- = “This function is for internal use” …. neah! No wildcard = no worries!

- Exception 🤯🤯🤯

leading_underscore.py

```
>>> def external_magic():
        return "I am external. My magic is safe. "
>>> external_magic()
        'I am external. My magic is safe. '
>>> def _internal_magic():
        return "I am internal. Dangerous magic!!! "
>>> _internal_magic()
        'I am internal. Dangerous magic!!! '
```

leading_underscore_test3.py

```
>>> import leading_underscore

>>> leading_underscore.external_magic()
      'I am external. My magic is safe'
>>> leading_underscore._internal_magic()
      'I am internal. Dangerous magic!!!'
```

THE SINGLE TRAILING UNDERSCORE

- = “How to make peace with internal Python naming conventions”

- Example:

```
>>> def check_if_class_is_magic(class, magic_level):
        if magic_level > 5:
            print('Class *%s* is magic!' % (class))
        else:
            print('No magic here... ')
File "<ipython-input-1-cf6bdd89d2f1>", line 1
  def check_if_class_is_magic(class, magic_level):
                              ^
SyntaxError: invalid syntax
```

THE SINGLE TRAILING UNDERSCORE

- = “How to make peace with internal Python naming conventions”

- Example:

```
>>> def check_if_class_is_magic(class_, magic_level):
        if magic_level > 5:
            print('Class *%s* is magic!' % (class_))
        else:
            print('No magic here... ')
>>> check_if_class_is_magic('Wizard of Oz', 10)
Class *Wizard of Oz* is magic!
```

THE DOUBLE LEADING UNDERSCORE

- = “I am nice. I will take care of name mangling for you”
  - *name mangling* = a variable will be renamed by the interpreter in order to avoid conflicts when the class will be extended later
- Example:

```
>>> class NoMagic:
        def __init__(self):
            self.really = 'yes'
            self._really_really = 'well... '
            self.__core = 'just a bit more magic'
>>> magic_in_denial = NoMagic()
>>> dir(magic_in_denial)
['_NoMagic__core', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', '_really_really', 'really']
>>> magic_in_denial._NoMagic__core
'just a bit more magic'
```

THE DOUBLE LEADING UNDERSCORE

- = “I am nice. I will take care of name mangling for you”
- Example:

```
>>> class MagicDestructor(NoMagic):
        def __init__(self):
            super().__init__()
            self.really = 'no more magic'
            self._really_really = 'really no more magic'
            self.__core = 'magic = 1%'
>>> no_magic = MagicDestructor()
>>> dir(no_magic)
['_MagicDestructor__core', '_NoMagic__core', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__','__ge__','__getattribute__','__gt__','__hash__','__init__','__init_subclass__','__le__','__lt__',
'__module__','__ne__','__new__','__reduce__','__reduce_ex__','__repr__','__setattr__','__sizeof__','__str__',
'__subclasshook__', '__weakref__', '_really_really', 'really']
>>> no_magic._MagicDestructor__core
'magic = 1%'
 >>> no_magic._NoMagic__core
'just a bit more magic'
```

THE DUNDER

- = “Better not touch it. Real Python magic”
  - Reserved for special Python methods, e.g. , __init__, __add__, __len__ etc.
- Example:

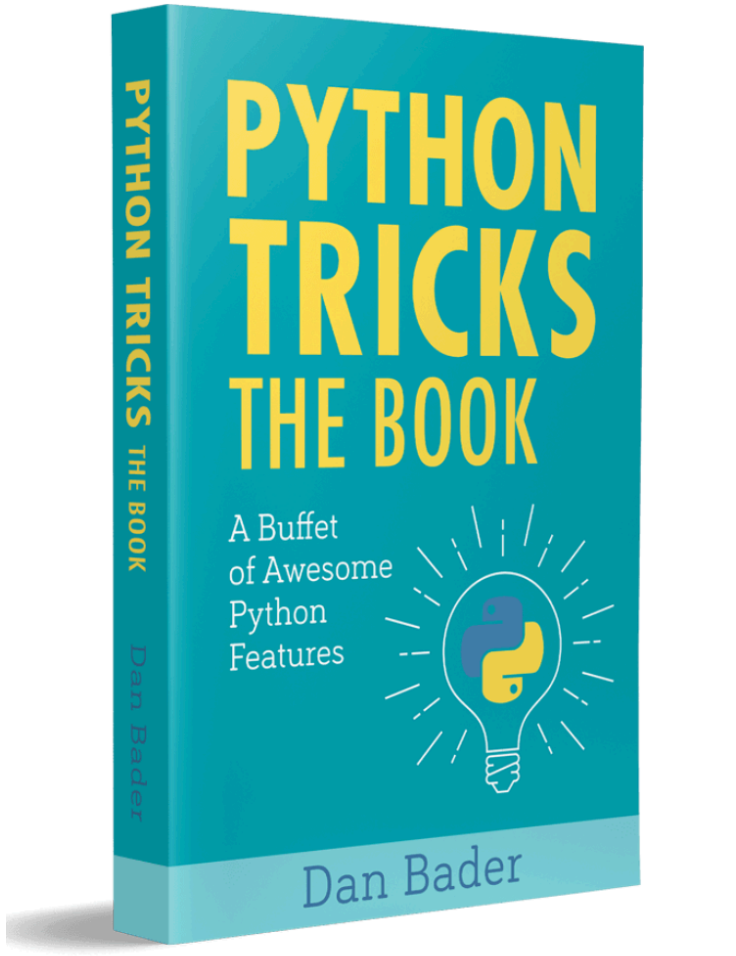
```
>>> class Dunder():
        def __init__(self, key):
            self.key = "__" + key + "__”
>>> d0 = Dunder('d0')
>>> d1 = Dunder('d1')
>>> d0 + d1
---------------------------------------------------------------------------
TypeError Traceback (most recent call last)
<ipython-input-4-f0e974d77b61> in <module>()
      1 d0 = Dunder('d0’)
      2 d1 = Dunder('d1’)
----> 3 d0+d1
TypeError: unsupported operand type(s) for +: 'Dunder' and 'Dunder'
```

THE DUNDER

- = “Better not touch it. Real Python magic”
- Example:

```
>>> class Dunder():
        def __init__(self, key):
            self.key = "__" + key + "__"
        def __add__(self, new):
            return self.key + '+' + new.key
>>> d0 = Dunder('d0')
>>> d1 = Dunder('d1')
>>> d0 + d1
'__d0__+__d1__’
>>> d1 + d0
‘__d1__+__d0__'
```

MORE PYTHON TRICKS



THAT'S ALL, FOLKS!