Python Course

May 17, 2018

1 Python Course

1.1 Course goals

- The audience could implement a specific feature in a elegant and clean way.
- The audience could use the right tools in order to make life easier and happier.

1.2 Lexical analysis

Although it's easy to start writing Python code with merely basic syntax knowledge, I'd like to start our course from the lexical analysis of Python to make sure we didn't miss anything important.

1.2.1 Encoding declarations

```
# -*- coding: <encoding-name> -*-
```

If a comment in the first or second line of the Python script matches the regular expression `coding[=:]\s*([-\w.]+)`, this comment is processed as an encoding declaration; the first group of this expression names the encoding of the source code file. The encoding declaration must appear on a line of its own. If it is the second line, the first line must also be a comment-only line.

UTF-8 is the default source encoding for Python 3, just as ASCII was the default for Python 2 (starting with 2.5).

The default encoding behavior is defined in the function `decoding_fgets` which is defined in the Parser/tokenizer.c file. This function was added in the commit below:

- Patch #534304: Implement phase 1 of PEP 263. https://github.com/python/cpython/commit/00f1e3f5a54a
30b8266a4285de981f8b1b82a8cc6231

Before Python 2.5, there is only `fgets` available to read lines from source file.

- Mass checkin of universal newline support. https://github.com/python/cpython/commit/7b8c7546ebc1f
30b8266a4285de981f8b1b82a8cc6231

```
char *
Py_UniversalNewlineFgets(char *buf, int n, FILE *stream, PyObject *fobj)
{
    return fgets(buf, n, stream);
}
```

1.2.2 Indentation

In Python 2, one table is an equivalence of eight spaces. In Python 3, can not use a mixture of tabs and spaces in indentation.

Make tabs always 8 spaces wide – it's more portable. https://github.com/python/cpython/commit/4fe872988b3dd9edf004160c44076df839f14516#diff-30b8266a4285de981f8b1b82a8cc6231

```
In [4]: for i in range(3):
                print(i) # 8 spaces this line.
                print(i) # 1 tab this line.
```

```
0
0
1
1
2
2
```

Never ever use a mixture of tabs and spaces.

1.2.3 Reserved classes of identifiers

`_*`

Not imported by `from module import *`. The special identifier _ is used in the interactive interpreter to store the result of the last evaluation; it is stored in the `__builtin__` module. When not in interactive mode, _ has no special meaning and is not defined.

`__*__`

System-defined names. These names are defined by the interpreter and its implementation (including the standard library). Current system names are discussed in the Special method names section and elsewhere. More will likely be defined in future versions of Python. Any use of `__*__` names, in any context, that does not follow explicitly documented use, is subject to breakage without warning.

`__*`

Class-private names. Names in this category, when used within the context of a class definition, are re-written to use a mangled form to help avoid name clashes between "private" attributes of base and derived classes.

1.3 Data structure

In this section, I will introduce some basic Python types which come from the `types` module.

```
In [1]: import types
        dir(types)
```

```
Out[1]: ['BooleanType',
         'BufferType',
         'BuiltinFunctionType',
         'BuiltinMethodType',
         'ClassType',
         'CodeType',
         'ComplexType',
         'DictProxyType',
         'DictType',
         'DictionaryType',
         'EllipsisType',
         'FileType',
         'FloatType',
         'FrameType',
         'FunctionType',
         'GeneratorType',
         'GetSetDescriptorType',
         'InstanceType',
         'IntType',
         'LambdaType',
         'ListType',
         'LongType',
         'MemberDescriptorType',
         'MethodType',
         'ModuleType',
         'NoneType',
         'NotImplementedType',
         'ObjectType',
         'SliceType',
         'StringType',
         'StringTypes',
         'TracebackType',
         'TupleType',
         'TypeType',
         'UnboundMethodType',
         'UnicodeType',
         'XRangeType',
         '__all__',
         '__builtins__',
         '__doc__',
         '__file__',
         '__name__',
         '__package__']
```

In order to discuss them in a proper way, we will classify them into several groups:

- Numeric Types
- Sequence Types
- Set Types and Mapping Types
- Other types

1.3.1 Numeric Types

Built-in numeric types There are some numeric types which can be defined as a literal. Please note that `bool` type is a numeric type.

```
In [17]: # built-in numeric types
         {
             type(True): True,
             type(1): 1,
             type(1.0): 1.0,
             type(1+1j): 1+1j,
             type(1L): 1L,
         }

Out[17]: {bool: True, complex: (1+1j), float: 1.0, int: 1, long: 1L}
```

```
In [19]: issubclass(bool, int)

Out[19]: True
```

Other numeric types There are some other numeric types in Python Standard Library, for example, Decimal and Fraction.

Besides the regular use cases of the numbers, there are some singular cases to which we need to pay attention.

```
In [15]: # other numeric types
         from decimal import Decimal
         from fractions import Fraction

         Fraction(2, 3), Decimal('1.000000001')

Out[15]: (Fraction(2, 3), Decimal('1.000000001'))
```

Number tricks Almost all platforms map Python floats to IEEE-754 "double precision". In that way, the represent of the floats will encounter the precision issue.

https://docs.python.org/2/tutorial/floatingpoint.html

```
In [15]: from fractions import Fraction
         print(0.1 + 0.2, Fraction(0.1 + 0.2))

(0.30000000000000004, Fraction(1351079888211149, 4503599627370496))
```

By using Decimal, which is a fixed point, the precision issue can be solved.

```
In [1]: from decimal import Decimal
        from fractions import Fraction
        # Don't use float number to construct the Decimal.
        Decimal("0.1") + Decimal("0.2"), Fraction(Decimal("0.1") + Decimal("0.2"))
```

```
Out[1]: (Decimal('0.3'), Fraction(3, 10))
```

Although Decimal is a handy class to use, but it seems that in some cases Decimal cannot be used directly.

```
In [18]: import numbers
         from decimal import Decimal

         issubclass(Decimal, numbers.Real), issubclass(Decimal, numbers.Number)

Out[18]: (False, True)

In [17]: import json
         from decimal import Decimal

         json.dumps({1: Decimal("1")})


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-17-9ea8cead75a8> in <module>()
          2 from decimal import Decimal
          3
    ----> 4 json.dumps({1: Decimal("1")})


        /usr/local/Cellar/python@2/2.7.15/Frameworks/Python.framework/Versions/2.7/lib/python2
        242         cls is None and indent is None and separators is None and
        243         encoding == 'utf-8' and default is None and not sort_keys and not kw):
    --> 244         return _default_encoder.encode(obj)
        245     if cls is None:
        246         cls = JSONEncoder


        /usr/local/Cellar/python@2/2.7.15/Frameworks/Python.framework/Versions/2.7/lib/python2
        205         # exceptions aren't as detailed.  The list call should be roughly
        206         # equivalent to the PySequence_Fast that ''.join() would do.
    --> 207         chunks = self.iterencode(o, _one_shot=True)
        208         if not isinstance(chunks, (list, tuple)):
        209             chunks = list(chunks)


        /usr/local/Cellar/python@2/2.7.15/Frameworks/Python.framework/Versions/2.7/lib/python2
        268                 self.key_separator, self.item_separator, self.sort_keys,
        269                 self.skipkeys, _one_shot)
    --> 270         return _iterencode(o, 0)
        271
```

```
272 def _make_iterencode(markers, _default, _encoder, _indent, _floatstr,


/usr/local/Cellar/python@2/2.7.15/Frameworks/Python.framework/Versions/2.7/lib/python2
182
183         """
--> 184         raise TypeError(repr(o) + " is not JSON serializable")
185
186     def encode(self, o):


TypeError: Decimal('1') is not JSON serializable
```

As a part of IEEE-754, there are nan and inf. Although nan may be not very useful in our real world, but the inf is still a useful concept.

```
In [25]: inf = float('inf')
         inf + inf, inf - inf, inf * inf, inf / inf, -inf

Out[25]: (inf, nan, inf, nan, -inf)
```

1.3.2 Sequence Types

Mutable sequences vs. Immutable sequences An object of an immutable sequence type cannot change once it is created.

Mutable sequences can be changed after they are created.

```
In [5]: import collections

        print "name\tmutable?"
        for t in [str, list, tuple]:
            print "%s\t%s" % (t.__name__, issubclass(t, collections.MutableSequence))

name        mutable?
str        False
list        True
tuple        False
```

If the object contains references to other objects, these other objects may be mutable and may be changed; however, the collection of objects directly referenced by an immutable object cannot change.

```
In [8]: a = (1, [], 'Hello')
        a[1].append(2) # The refered object can be changed.
        print(a)
        a[1] = [] # The reference itself in the immutable sequence can not be changed.
```

```
(1, [2], 'Hello')
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-8-6c0266194b77> in <module>()
          2 a[1].append(2) # The refered object can be changed.
          3 print(a)
    ----> 4 a[1] = [] # The reference itself in the immutable sequence can not be changed.


        TypeError: 'tuple' object does not support item assignment
```

List Comprehensions

```
In [10]: # Variable leaking
         l = [x for x in range(10)]
         print(l)
         print(x)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
9
```

2 References

<img class="reference" src="../../references/fluent-python.jpg">
<img class="reference" src="../../references/python-language-reference.jpg">

3 Videos

```
In [22]: from IPython.display import YouTubeVideo
         YouTubeVideo('7Zlp9rKHGD4')
```

```
   Out[22]:
```

```
In [27]: from decimal import Decimal
         True == 1 == 1+0j == 1.0 == Decimal("1")

Out[27]: True

In [6]: from decimal import Decimal
        {True: 1, 1: 2, 1 + 0j: 3, 1.0: 4, Decimal(1): 5}

Out[6]: {True: 5}
```

- global https://git.garena.com/beepos/pos_python_server/merge_requests/906/diffs#f5cacdfe36190a6548
- Why should we NOT use sys.setdefaultencoding("utf-8") in a py script? https://stackoverflow.com/questions/3828723/why-should-we-not-use-sys-setdefaultencodingutf-8-in-a-py-script

Python compare. is, ==, bool