practicePython

April 24, 2019

1 Keywords and Identifiers

```
In [6]: import keyword

        print(keyword.kwlist)
        print(keyword.iskeyword('True'))
        print(len(keyword.kwlist)) # keywords defined by the interpreter
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue',
True
35
```

Identifiers:

```
1) Identifiers can consist - a-z, A-Z, 0-9, _
2) cannot start with number
3) cannot use keywords as identifier
```

```
In [7]: global = 1
        a@ = 2
```

```
          File "<ipython-input-7-4e489388dec1>", line 1
        global = 1
               ^
    SyntaxError: invalid syntax
```

2 Comments, Indentation & Statement

1) Single line comment - #

2) multi line comments - """ or ′′′

```
In [9]: """
            multi
            line
            comment
        """
```

```
Out[9]: '\n    multi\n    line\n    comment\n'
```

Doc string *(Documentation string)* **:** String that occurs in the first statement of a module, function, class or method definition *used as **doc**

```
In [10]: def double(n):
             """
             function to double the number
             """
             return 2*n;
         print(double(11))
         print(double.__doc__)
```

```
22

    function to double the number
```

Indentation:

```
1) Maintian consistency in indentation through out the block
2) ';' can be used as terminator
```

Python Statement:

```
1) instructions that a python interpreter can execute are called python statements
```

MultiLine Statements:

```
1) ()
2) \
3) ; -> multiple line statements in single line
```

```
In [21]: print(range(10))

         for i in range(10):
             print(i)
             i = i + 2
         print(i)
```

```
range(0, 10)
0
1
```

```
2
3
4
5
6
7
8
9
11
```

3 Var & Datatypes

In [33]:
```python
a = 10
print(type(a))
b = 5.5
print(type(b))
c = "ML"
print(type(c))

# Multiple assignments

a, b, c = 10, 5.5, "ML"

# Storage Locations

print(id(a))
a1 = 10
print(id(a1))
a1 = 15
print(id(a1))
print(id(a))

j= 10.0
k = float(10)
print(id(j))
print(id(k))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
140710936309792
140710936309792
140710936309952
140710936309792
2655974007480
2655974007384
```

4 Data Types

1) Everything is an object in python:

    1) data types are classes
    2) variables are instances of the classes

2) All functions have inbuilt attribute doc - returns doc string

3) sys(System specific parameters and functions) module has path attribute and other

4) Objects here are subclassable(we can inherit them from class Foo(object):pass)

5) In python, object definition is looser(some objects neither have attributes nor methods and not all objects are subclassable)

6) Everything is an object means it can be assigned to avar or passed as an argument to a function

7) looser - creating and using classes and objects are easy with immense flexibility provided by python

Numbers:

```
1) integer : int class
2) float   : float class
3) complex : complex class
```

type() - to know which class variable
isinstance() - to check if it belongs to particular class

```
In [36]: class Foo:
             a = 5
         fooInstance = Foo()

         print(isinstance(fooInstance, Foo))
         print(isinstance(fooInstance, (list, tuple)))
         print(isinstance(fooInstance, (list, tuple, Foo)))
```

```
True
False
True
```

```
In [41]: numbers = [1,2,3]
         result = isinstance(numbers, list)
         print(result)
         a = 1+2j
         print(isinstance(a, complex))
```

```
True
True
```

```
In [40]: help(float)
```

```
Help on class float in module builtins:

class float(object)
 |  float(x=0, /)
 |
 |  Convert a string or number to a floating point number, if possible.
 |
 |  Methods defined here:
 |
 |  __abs__(self, /)
 |      abs(self)
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __bool__(self, /)
 |      self != 0
 |
 |  __divmod__(self, value, /)
 |      Return divmod(self, value).
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __float__(self, /)
 |      float(self)
 |
 |  __floordiv__(self, value, /)
 |      Return self//value.
 |
 |  __format__(self, format_spec, /)
 |      Formats the float according to format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __int__(self, /)
 |      int(self)
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __neg__(self, /)
 |      -self
 |
 |  __pos__(self, /)
 |      +self
 |
 |  __pow__(self, value, mod=None, /)
 |      Return pow(self, value, mod).
 |
 |  __radd__(self, value, /)
 |      Return value+self.
 |
 |  __rdivmod__(self, value, /)
 |      Return divmod(value, self).
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rfloordiv__(self, value, /)
 |      Return value//self.
 |
 |  __rmod__(self, value, /)
 |      Return value%self.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __round__(self, ndigits=None, /)
 |      Return the Integral closest to x, rounding half toward even.
 |
 |      When an argument is passed, work like built-in round(x, ndigits).
 |
 |  __rpow__(self, value, mod=None, /)
 |      Return pow(value, self, mod).
 |
 |  __rsub__(self, value, /)
 |      Return value-self.
 |
 |  __rtruediv__(self, value, /)
 |      Return value/self.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  __sub__(self, value, /)
 |      Return self-value.
 |
 |  __truediv__(self, value, /)
 |      Return self/value.
 |
 |  __trunc__(self, /)
 |      Return the Integral closest to x between 0 and x.
 |
 |  as_integer_ratio(self, /)
 |      Return integer ratio.
 |
 |      Return a pair of integers, whose ratio is exactly equal to the original float
 |      and with a positive denominator.
 |
 |      Raise OverflowError on infinities and a ValueError on NaNs.
 |
 |      >>> (10.0).as_integer_ratio()
 |      (10, 1)
 |      >>> (0.0).as_integer_ratio()
 |      (0, 1)
 |      >>> (-.25).as_integer_ratio()
 |      (-1, 4)
 |
 |  conjugate(self, /)
 |      Return self, the complex conjugate of any float.
 |
 |  hex(self, /)
 |      Return a hexadecimal representation of a floating-point number.
```

```
|
|      >>> (-0.1).hex()
|      '-0x1.999999999999ap-4'
|      >>> 3.14159.hex()
|      '0x1.921f9f01b866ep+1'
|
|  is_integer(self, /)
|      Return True if the float is an integer.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __getformat__(typestr, /) from builtins.type
|      You probably don't want to use this function.
|
|        typestr
|          Must be 'double' or 'float'.
|
|      It exists mainly to be used in Python's test suite.
|
|      This function returns whichever of 'unknown', 'IEEE, big-endian' or 'IEEE,
|      little-endian' best describes the format of floating point numbers used by the
|      C type named by typestr.
|
|  __set_format__(typestr, fmt, /) from builtins.type
|      You probably don't want to use this function.
|
|        typestr
|          Must be 'double' or 'float'.
|        fmt
|          Must be one of 'unknown', 'IEEE, big-endian' or 'IEEE, little-endian',
|          and in addition can only be one of the latter two if it appears to
|          match the underlying C reality.
|
|      It exists mainly to be used in Python's test suite.
|
|      Override the automatic determination of C-level floating point type.
|      This affects how floats are converted to and from binary strings.
|
|  fromhex(string, /) from builtins.type
|      Create a floating-point number from a hexadecimal string.
|
|      >>> float.fromhex('0x1.ffffp10')
|      2047.984375
|      >>> float.fromhex('-0x1p-1074')
|      -5e-324
|
|  ----------------------------------------------------------------------
```

```
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  imag
 |      the imaginary part of a complex number
 |
 |  real
 |      the real part of a complex number
```

```
In [42]: # Boolean
         a = True
         print(type(a))
```

```
<class 'bool'>
```

Python Strings

```
1) string is sequence of unicode characters
2) multi line string can be denoted using triple quotes - ''' or """ or \
3) indexing starts from 0
```

```
In [53]: a = "thisis deepa learning ai"
         print(len(a))
         print(a[1:])
         print(a[-1])
         print(a[:5]) # 0 to 4
         print(a[5: -1]) # before -1
         print(a[len(a) - 1])
         print(a[-25])
```

```
24
hisis deepa learning ai
i
thisi
s deepa learning a
i
```

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

```
    IndexError                                Traceback (most recent call last)

    <ipython-input-53-724f54baf981> in <module>()
      6 print(a[5: -1]) # before -1
      7 print(a[len(a) - 1])
----> 8 print(a[-25])


    IndexError: string index out of range
```

Python Lists

```
1) Order sequence of items
2) objects can be multiple data types
3) index starts from 0
4) mutable
```

```
In [54]: a = [10, 2.5, "Deepa"]
         print(a[1])
         print(type(a))
```

```
2.5
<class 'list'>
```

Python Tuple

```
1) ordered sequnce of items
2) objects can be multiple data types
3) index starts from 0
4) immutable
```

```
In [58]: a = (1, 1,234, 346.7, "asf", 1 + 2j)
         a[4]
         a[4] = 12
```

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

    TypeError                                 Traceback (most recent call last)

    <ipython-input-58-288ddc83e195> in <module>()
      1 a = (1, 1,234, 346.7, "asf", 1 + 2j)
      2 a[4]
----> 3 a[4] = 12


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

Python Set

```
1) no indexing
2) unordered collection
3) unique items
4) data structure
5) multiple data types can be stored
```

```
In [60]: a = {10, "hello"}
         print(a)
```

```
{'hello', 10}
```

```
In [61]: a[1]
```

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

        TypeError                                 Traceback (most recent call last)

        <ipython-input-61-8bc71255a22e> in <module>()
    ----> 1 a[1]


        TypeError: 'set' object does not support indexing
```

Python Dictionary