Pythonlecture2 LanguageBasics

January 18, 2018

1 Python Language Basics

1.1 Imports library/module into memory

In Python a module is simply a file with the .py extension containing Python code. Suppose that we had the following module:

```
In [ ]: # some_module.py
        PI = 3.14159
        def f(x):
            return x + 2
        def g(a, b):
            return a + b
```

If we wanted to access the variables and functions defined in some_module.py, from another file in the same directory we could do:

```
In [ ]: import some_module
        result = some_module.f(5)
        pi = some_module.PI
        #or
        from some_module import f, g, PI
        result = g(5, PI)
```

By using the as keyword you can give imports **different variable names/nicknames**:

```
In [ ]: import some_module as sm
        from some_module import PI as pi, g as gf
        r1 = sm.f(pi)
        r2 = gf(6, pi)
```

1.2 Binary operators and comparisons

Most of the binary math operations and comparisons are the same as in math

```
In [ ]: year = 2018
        diff = 2
        print(year+diff)
        print(year*diff)
```

2 Summary of Binary Operations

Operation Description 1. a + b Add a and b 2. a - b Subtract b from a 3. a * b Multiply a by b 4. a / b Divide a by b 5. a // b Floor-divide a by b, dropping any fractional remainder 6. a ** b Raise a to the b power 7. a & b True if both a and b are True; for integers, take the bitwise AND 8. a | b True if either a or b is True; for integers, take the bitwise OR 9. a ^ b For booleans, True if a or b is True, but not both; for integers, take the bitwise EXCLUSIVE-OR 10. a == b True if a equals b; note use **==** instead of **=** 11. a != b True if a is not equal to b 12. a <= b, a < b True if a is less than (less than or equal) to b 13. a > b, a >= b True if a is greater than (greater than or equal) to b 14. a is b True if a and b reference the same Python object 15. a is not b True if a and b reference different Python objects

To check if two references refer to the same object, use the is keyword. is not is also perfectly valid if you want to check that two objects are not the same:

```
In [ ]: a = [2017, 2018, 2019]
        b =a
        c=list(a)
        print(a is b)
        print(a is c)
```

Since list always creates a new Python list (i.e., a copy), we can be sure that c is distinct from a. Comparing with is is not the same as the == operator, because in this case we have:

```
In [ ]: print(a==c)
```

A very common use of is and is not is to check if a variable is None, since there is only one instance of None:

```
In [ ]: a = None
        print(a is None)
```

2.1 Mutable and immutable objects

Most objects in Python, such as **lists, dicts, NumPy arrays, and most user-defined types (classes)**, are mutable, meaning you can change their content without changing their identity. This means that the object or values that they contain can be modified:

Others, like **strings and tuples**, are immutable, which means they are constants cannot be changed!

```
In [ ]: #operations on list []
        a_list =[2, "year", [4,5,6]]
        print(a_list)
        a_list[1] =(2018)
        print(a_list)
        #operations on tuple ()
        a_tuple = (2, "year", [4,5,6])
        a_tuple[1]= (2018)
```

2.2 Scalar Types

Python along with its standard library has a small set of built-in types for handling numerical data, strings, boolean (True or False) values, and dates and time. These "single value" types are sometimes called scalar types and we refer to them in this book as scalars. See Table below for a list of the main scalar types. Date and time handling will be discussed separately, as these are provided by the datetime module in the standard library.

Standard Python scalar types Type Description

1. None The Python "null" value (only one instance of the None object exists)
2. str String type; holds Unicode (UTF-8 encoded) strings
3. bytes Raw ASCII bytes (or Unicode encoded as bytes)
4. float Double-precision (64-bit) floating-point number (note there is no separate double type)
5. bool A True or False value
6. int Arbitrary precision signed integer

2.2.1 Numeric types

The primary Python types for numbers are int and float. An int can store arbitrarily large numbers:

```
In [ ]: i = 2018
        print(i**5)
        from math import factorial
        factorial(2018)
```

Floating-point numbers are represented with the Python float type. Under the hood each one is a double-precision (64-bit) value. They can also be expressed with scientific notation:

```
In [ ]: fval = 2018.0111
        print(fval)
        f2 = 1e-10
        print(f2)
```

Integer division not resulting in a whole number will always yield a floating-point number To get C-style integer division (which drops the fractional part if the result is not a whole number), use the floor division operator //

```
In [ ]: a= 10
        b=3
        print(a/b)
        print(a//b)
```

2.3 Srings

1. You can write string literals using either single quotes ' or double quotes
2. For multiline strings with line breaks, you can use triple quotes, either ''' or """:
3. Python strings are immutable; you cannot modify a string
4. Many Python objects can be converted to a string using the str function