Pythonlecture1Peliminaries

January 16, 2018

1 Chapter 1 preliminaries

Python is very popular 1. The 7 Most In-Demand Programming Languages of 2018

2. TIOBE Index for January 2018

3. Most Popular and Influential Programming Languages of 2018

Python 2 or Python 3 1. Python 2.x is legacy, Python 3.x is the present and future of the language

2. PYTHON 3 VS PYTHON 2: IT'S DIFFERENT THIS TIME

Installation and Setup 1. Download Python 3.6 at Anaconda 2. Integrated Development Environments (IDEs): Spyder (free)

1.1 Python Language Basics

1.1.1 The Python Interpreter

1. Click the Anaconda Navagator from the startmenu
2. click Python Interpreter

```
In [1]: # print the message
        print("Hello, World!")
```

```
Hello, World!
```

```
In [3]: #Calculator
        print(2+3*5)
```

```
17
```

1.1.2 Jupyter Notebook

1. Support many populer languages such as Python and R etc
2. Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.
3. Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.
4. Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, TensorFlow.

To start Jupyter Notebook, you need to 1. Click the Anaconda Navigator 2. Click the Jupyter Notebook 3. Click the right cornner and select the language you need, Python 4. Double click the text on the top to the left of Jupyter logo, choose the file name 5. Enter the Python command in the cell and make sure the drop down box is Code 6. Run the code by pressing Shift-Enter or from the Cell Menu

```
In [6]: print("Hello, World!")
```

```
Hello, World!
```

1.2 Python Language Basics

1. Python is case sensitive
2. Indentation, not braces Python uses whitespace (tabs or spaces) to structure code instead of using braces as in many other languages like R, C++, Java, and Perl. Consider a for loop from a sorting algorithm:

```
In [35]:     Year = 2018
             year = 2019
             YEAR =2020
             print(Year)
             print(year)
             print(YEAR)
             temperature = float(input('What is the temperature? '))
             if temperature > 70:
                 print('Wear shorts.')
             else:
                 print('Wear long pants.')
             print('Get some exercise outside.')
```

```
2018
2019
2020
What is the temperature? 12
Wear long pants.
Get some exercise outside.
```

A colon denotes the start of an indented code block after which all of the code must be indented by the same amount until the end of the block

strongly recommend using four spaces as your default indentation and replacing tabs with four spaces

1.3 Everything is an object

An important characteristic of the Python language is the consistency of its object model. Every number, string, data structure, function, class, module, and so on exists in the Python interpreter in its own "box," which is referred to as a Python object. Each object has an associated type (e.g., string or function) and internal data. In practice this makes the language very flexible, as even functions can be treated like any other object.

1.4 Comments

Any text preceded by the hash mark (pound sign) # is ignored by the Python interpreter. This is often used to add comments to code. At times you may also want to exclude certain blocks of code without deleting them. An easy solution is to comment out the code:

```
In [19]: temperature = float(input('What is the temperature? '))#ask user enter the message
         if temperature > 70:
             print('Wear shorts.')
         #   else:
         #       print('Wear long pants.')
         #   print('Get some exercise outside.')
```

```
What is the temperature? 12
```

Comments can also occur after a line of executed code. While some programmers prefer comments to be placed in the line preceding a particular line of code, this can be useful at times: print("Reached this line") # Simple status report

1.5 Variables and argument passing

Python pass variable **by reference** instead by value

Understanding the semantics of references in Python and when, how, and why data is copied is especially critical when you are working with larger datasets in Python.

Assignment is also referred to as binding, as we are binding a name to an object. Variable names that have been assigned may occasionally be referred to as bound variables.

When you pass objects as arguments to a function, new local variables are **created referencing the original objects without any copying**

When a parameter is passed **by reference (in Python language)**, the caller and the callee use **the same variable for the parameter**. If the callee modifies the parameter variable, the effect is visible to the caller's variable.

When a parameter is passed **by value (in R language)**, the caller and callee have **two independent variables** with the same value. If the callee modifies the parameter variable, the effect is not visible to the caller.

```
In [5]: # assign  a list
        a = [2017,2018,2019]
        # assign a to a new var b
        b =a
        #print out a and b
        print("a=", a)
        print("b=", b)
        #next, we modify b by changing b[2]
        b[2] = 2020
        #print out a and b
        print("a=", a)
        print("b=", b)
```

```
a= [2017, 2018, 2019]
b= [2017, 2018, 2019]
a= [2017, 2018, 2020]
b= [2017, 2018, 2020]
```

1.6 Dynamic references, strong types

In contrast with many compiled languages, such as Java and C++, object references in Python have no type associated with them. There is no problem with the following

```
In [12]: year = 2018
         print(type(year))
         month = "January"
         print(type(month))
         ## String formatting, to be visited later
         print('year is {0}, month is {1}'.format(type(year), type(month)))
```

```
<class 'int'>
<class 'str'>
year is <class 'int'>, month is <class 'str'>
```

Knowing the type of an object is important, and it's useful to be able to write functions that can handle many different kinds of input. You can check that an object is an instance of a particular type using the isinstance function:

```
In [15]: year = 2018
         isinstance(year, int)
         #isinstance can accept a tuple of types if you want to check that an objects type is
         #among those present in the tuple
         isinstance(year, (int, float))
```

```
Out[15]: True
```

1.7 Attributes and methods

Objects in Python typically have both attributes (other Python objects stored “inside” the object) and methods (functions associated with an object that can have access to the object’s internal data). Both of them are accessed via the syntax

1. obj.attribute_name:
2. getattr function

```
In [22]: Date = "January 16, 2018"
         newDate = Date.upper()
         print(newDate)
         getattr(Date, 'split')

JANUARY 16, 2018


Out[22]: <function str.split>
```

1.8 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 [24]: # 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 [26]: import some_module
         result = some_module.f(5)
         pi = some_module.PI
         #or
         from some_module import f, g, PI
         result = g(5, PI)


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

        ImportError                               Traceback (most recent call last)

        <ipython-input-26-15baaa74778f> in <module>()
    ----> 1 import some_module
          2 result = some_module.f(5)
          3 pi = some_module.PI
```

```
      4 #or
      5 from some_module import f, g, PI


    ImportError: No module named 'some_module'
```

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

```
In [27]: import some_module as sm
         from some_module import PI as pi, g as gf
         r1 = sm.f(pi)
         r2 = gf(6, pi)
```

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

    ImportError                               Traceback (most recent call last)

    <ipython-input-27-806ac02eb597> in <module>()
----> 1 import some_module as sm
      2 from some_module import PI as pi, g as gf
      3 r1 = sm.f(pi)
      4 r2 = gf(6, pi)


    ImportError: No module named 'some_module'
```

1.9 Binary operators and comparisons

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

```
In [29]: year = 2018
         diff = 2
         print(year+diff)
         print(year*diff)
```

```
2020
4036
```

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 perfecty valid if you want to check that two objects are not the same:

```
In [30]: a = [2017, 2018, 2019]
         b =a
         c=list(a)
         print(a is b)
         print(a is c)
```

```
True
False
```

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

```
True
```

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

```
True
```

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 [39]: #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, 'year', [4, 5, 6]]
[2, 2018, [4, 5, 6]]
```

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

        TypeError                                 Traceback (most recent call last)

        <ipython-input-39-ffc431071f53> in <module>()
          6 #operations on tuple ()
          7 a_tuple = (2, "year", [4,5,6])
    ----> 8 a_tuple[1]= (2018)


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

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

```
33466154331649568
```

```
Out[44]: 9466919507910613341020889551960700807076653812644652943192353826812674257313418471182589
```

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

```
2018.0111
1e-10
```

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

```
3.3333333333333335
3
```

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

```
In [54]: s1 = "this is my string"
         print(s1)
         s2 = 'this is the second string'
         print(s2)
         s3 = """ this is the first row
                 this is the seond row
                 this is the thrid row
         """
         print(s3)
         #Python strings are immutable; you cannot modify a string
         #s2[10] ='a'
         f1 = 2018.10
         s3 =str(f1)
         print(type(f1))
         print(type(s3))
```

```
this is my string
this is the second string
 this is the first row
        this is the seond row
```