CS 35L Fall 19 Section 3 Notes Week 3 Monday

Zhaowei Tan

Python Basics:

1. Installation and package management:
   sudo pip install module_name
   sudo pip3 install module_name
   Note that you might not be able to do that if you do not have root permission (i.e., unable to 'sudo')
2. Using interactive shell or script execution:
   python script.py
   python3 script.py
3. Modules are important parts of Python, probably one of the major reasons why people love Python. Use pip to install a new module in your computer. To import a module into the script:
   import module
   import module as module_alias
4. Variable types: You don't have to declare the variable type when you assign it – instead, Python would automatically know what type the variable is. E.g. the following all work:
   a = "123"
   a = 3
   a = 4.0
   Note that:
   1) To transfer from one type to another, use something explicitly like
      a = 4.0
      a = int(a)
   2) In calculation, the Python would not do the auto type transfer for you
      a = 4 / 3 # equals 1 in python 2
      a = 4 / 3 # equals 1.333 in python 3
   3) To check the type, use type(var)
5. Use print (…) to print things to standard output.
   Print can recognize any type of the variable, int, float, list, etc.
   Python 3 print special: sep and end
   print('G','F','G', sep='')
   print("Welcome to" , end = ' ')
6. String:
   a. Access a char: s[index]; index can be negative, s[start:end] for slicing
   b. Split the string: s.split('delimiter')
   c. Find substring within a string: s.find("substring", start_position). Returns index, or negative if not found. Other method: replace, upper, …
   d. Concatenate: c = a + b
7. List:
   To create a list
   my_list = ['a', 4, [1, 3]]
   The items in Python list are not necessarily be of same length or same type.
   Use list[a] to fetch an item. Use list[a:b] to fetch a subset. Index a is inclusive, but b is exclusive.

Useful list functions: append, extend, index, remove, pop, count…

8. Dictionary
   To initiate a list/directory
   ```
   my_dict = {“name”: “Zhaowei”, “major”: “CS”, “status”: 0}
   ```
   Access value by key: my_dict[“name”]
   Useful dictionary functions: keys, values, items…
9. Tuples: Not mutable. Old tradition of keeping heterogeneous stuffs. Can be used as key for dictionary.

10.
    ```
    if condition:
      do something
    ```
    You can use and, or, not to connect the different statements.
    Use indentation to wrap the block.
    To enable else if:
    ```
    if condition1:
      do something
    elif condition2:
      do something else
    else:
      do something different
    ```
    Some special conditions: if key in dict (check if key is in the keys of dict); if ele in list
11. Loops:
    ```
    while condition:
      do something
    ```
    or
    ```
    for i in [list]:
      do something for i
    ```
    Note:
    1) Be careful that Python uses indentation to indicate different levels, instead of using brackets (C) or keywords (Bash). I recommend using spaces instead of tabs as your indentation for compatibility.
    2) You could use range(min, max, step) to generate a number list and use for loop to iterate. The list generated includes min, but not max (similar to list[a:b]). We can use a negative step to generate numbers in a reversed order.
    3) To terminate a while/for loop, use break. To skip the current pass of loop, use continue.
12. Functions in Python:
    ```
    def func_name(arg1, arg2, …):
      do something
      return something
    ```
    Call this function
    ```
    func_name(arg1, arg2, …)
    ```
    You don’t have to specify the type for arguments.
13. Python class
    ```
    class className:
      define your variables
      define your functions
    ```
    A special function is called __init__, which will be called after you instantiate a new variable.

The functions in the class should have an extra argument, self
Inside the function, to call the variable inside the class, use self.varName

When you put double underscore (__) before the variable name, the name of the variable changes from __varName to _className__varName outside this class. This is a common practice to declare the variable as a private variable, however, we could still access this variable using the transferred variable name. But never do that! The same for function here.
A class example:

```
class myClass:
  __num = 0

  def __init__(self, num):
        self.__num = 100
  def printNum (self):
    print self.__num
  def _add_people (self, num):
    self.__num += num

t = myClass(100)
t. printNum()
t._add_people(10)
print t._myClass__num #don't do this in reality!
```

14. Exception handling.
We could use try/except to capture the error and continue the program.

```
try:
   f = open("testfile", "w")
   f.write("This is a test file.")
except IOError:
   print "Error: cannot find file or read data"
else:
   print "Written content in the file successfully"
   f.close()
```

Note: this can only catch runtime error, not syntax error