A Brief Review of Python

COMPSCI 270 Introduction to AI

Richard (Fangjian) Guo

guo@cs.duke.edu

Department of Computer Science
Go to python.org. Download and install python 2.7 on your system.

Choose an IDE/editor that you feel comfortable with.
- Text editor: Vim, Emacs, Sublime Text, etc.
  - (Make sure you install extension for writing python conveniently)
- IDE: Eclipse + PyDev, Spyder, PyCharm, etc.

Download and unzip search.zip from the course website
- Make sure you can execute the python script from command line/terminal
  - $ python pacman.py
  - $ python pacman.py --help
First Steps

- the points that can potentially be very useful in the pacman assignments.
The Python Style

- Interpreter
  - Interactive running
  - Running a script

```
while True:
    s = input('Enter something: ')
    if s == 'quit':
        break
    print('Length of the string is', len(s))
print('Done')
```

- Program is structured by “:” and indentation
  - Options: TAB, 2 spaces, 4 spaces (usually configurable in your IDE/Editor)
  - Be **consistent throughout** the program!
  - Pacman project: 2 spaces 😊

- Using a variable without declaration
- Dynamic typing
Basic operators

Most are straightforward

+ - / * %

NOTE

Division: integer vs. fractional

1/2 results in 0, 1.0/2 results in 0.5

Trick: a * 1.0 / b

Power: ** (not ^ as in some other languages)

Indexing: [ ]

List, dictionary
Built-in Data Structures

**List**
- Like an array, but can store elements with **different types**
  
  ```python
  >>> a = []  # initialize an empty list
  >>> a = ["hello", 1, 5.5]
  >>> listOfList = [[1,2],[3,4]]
  ```
- Indexed from 0

**Operations**
- `a.append(x)`
- `a.remove()`
  - `del a[0]`
  - if “hello” in `a`: `a.remove("hello")` (**first occurrence of the value**)
- `a.sort()` (**in place**) (**do not forget the brackets, it is a method**)
- `a.reverse()` (**in place**)
- `a + listOfList` (**concatenation**)
Built-in Data Structures

- **Tuple**
  - `x = (1,2,’ok’)`
  - Much like a list, but cannot be changed. *(immutable)*
Dictionary

A hash table that stores a mapping: key -> value

- mydict = {} (initialize)
- mydict[1] = “one” (automatically adding a new key-value pair)

Keys do not have a fixed order

Key must be of an immutable type:

- string, number
- tuple
  - mydict[(-1,0)] = “west”

Operations

- mydict.keys() (in the form of a list)
- mydict.values() (in the form of a list)
- print mydict[(-1,0)]
- del mydict[(-1,0)]

How to get the keys sorted by value?

- for w in sorted(mydict, key=mydict.get):
  - print w, mydict[w]
Set

- An unordered collection of unique elements
- Efficient to test if an element is marked/visited
  - `x in exploredSet`
- Initialization
  - `setOfShapes = set()` *(empty set)*
  - `setOfShapes = set([“circle”, “triangle”, “square”, “circle”])`
  - `setOfShapes = {“circle”, “triangle”, “square”, “circle”} (don’t get confused with dictionary)*
- Operations
  - `setOfShapes.add(“hexagon”)`
  - `setOfShapes.remove(“circle”)`
  - `set1 | set2`
  - `set1 & set2`
  - `set1 – set2`
Other Data Types

- **String**
  - $Z = \text{"chem"} + \text{"is"} + \text{"try"}$

- **Integer**
  - `int_x = \text{int}(x)`
Control Flows

If statement

```python
if x==0:
    print "zero"
    print "wrong input!"
elif x>0:
    print "positive"
else:
    print "negative"
```
For statement

Generally

for x in iterableObject
  Iterable: list, tuple, set, dictionary

myColors = ["yellow", "blue", "red"]
for color in myColors:
  print color

for i in range(0, len(myColors)):
  print myColors[i]
While statement

\[ i = 0 \]

while \( i < 100 \):

\[ i = i + 1 \]
Special clauses in loops
- break
- continue
- else
  - An optional else "block" can be written after "for" or "while" block
  - The "else" block will be executed if no "break" is executed in the loop

```python
for answer in possibleAnswers:
    if isRightAnswer(answer):
        break
    else:
        print "No answer found"
```
Writing a Function

- **Defining a function**
  ```python
  def myadd(x, y):
      z = x+y
      return z  # (would return None if without this line)
  ```

- **Calling functions**
  ```python
  myNumbers = [2, 4]
  print myadd(myNumbers[0], myNumbers[1])
  print myadd(*myNumbers) # (unpacking arguments from list or tuple)
  ```
Object Oriented: Writing a Class

- Defining a class
  - Starting with `def className:`
    ```python
    def classname(baseClassName) (inheritance)
    ```
  - Providing data attributes and methods with `self`
    ```python
    self.title = "a simple class"
    def showTitle(self, repeats=1):
        for t in range(0, repeats):
            print self.title
    ```
  - Initializing the class with method by providing
    ```python
    def __init__(self, someArg):
    ```
  - Making a variable **looking private** by naming with a leading underscore
    ```python
    Unlike C++, Python does not enforce data hiding mechanism
class Queue:
    def __init__(self):
        self.queueList = []

    def push(self, x):
        self.queueList.append(x)

    def pop(self):
        z = self.queueList[0]
        del self.queueList[0]
        return z

    def isEmpty(self):
        return (len(self.queueList) == 0)

This is provided by `util.py`
- import util
- myQueue = util.Queue()

Read `util.py` to learn other utilities that lessens your burden of implementation

Try implementing a `TreeNode` class on your own:
- Parent
- Path from root
- When you are not sure what it is/what went wrong, try printing it out
  - `print` is safe with almost every object 😊

- List comprehension -- mapping a list of input to a list of output with a function
  - `S = [x**2 for x in range(10)]`
  - `S_even = [x for x in S if x%2==0]`
Useful Tricks

Looping with ease

- Iterate with (index, value) pair
  ```python
  for i, v in enumerate(['a', 'b', 'c'])
  print i, v
  ```

- Iterate over two lists of the same length
  ```python
  list_1 = [1, 2, 3, 4]
  list_2 = ['one', 'two', 'three', 'four']
  for u, v in zip(list_1, list_2):
    print u, v
  ```

- Changing the list while iterating
  ```python
  Make a copy at first and iterate over the copy
  for x in list_1[:]:
    list_1.append(x**2)
  ```
Be careful when you make changes to a mutable object (e.g. list, dictionary, set)
```
x = [1,2,3,4]
y = x
y.append(5)
print x  \textit{(what would you expect?)}
```

Even more careful when you pass them to a function
```
x = [1,2,3,4]
def sumUp(z):
  z.append(sum(z))
  return z[-1]
print x  \textit{(what would you expect?)}
```

How to prevent?
```
Make a carbon copy when necessary
y = x.copy()
```
References

- Official tutorial
  - [http://docs.python.org/2/tutorial/](http://docs.python.org/2/tutorial/)

- Python/UNIX tutorial on the course webpage
  - [http://inst.eecs.berkeley.edu/~cs188/fa10/projects/tutorial/tutorial.html#Python](http://inst.eecs.berkeley.edu/~cs188/fa10/projects/tutorial/tutorial.html#Python)

- A Byte of Python

- Python Information and Examples
  - [http://www.secnetix.de/olli/Python/](http://www.secnetix.de/olli/Python/)

- Learning Python (O’Reilly)

- Python Pocket Reference (O’Reilly)