Obroni Computer Club – Dictionaries and more

Joachim Breitner

Oktober 10th 2006

Last week you suggested that I should be using less prepared slides and more demonstration of coding. So today, we'll try a different way of running this:

I will program something "live" using the projector, explaining what I'm doing while going there. Additionally, I prepared this document, where you can find details of new constructs and concepts, as well as the program that I will be writing. Or at least something similar, as I might change things while programming. After that part you should be experimenting with the new things, either based on the Code I wrote or, even better, on your own ideas.

Today, we'll look at dictionaries, which are a powerful way to managing data in your programs, and while doing so we will turn our little maze into a simple text adventure.

1 Dictionaries in Python

1.1 Creating dictionaries

You might remember how we had a list of lists for the connected rooms in our maze last time.

```
rooms = ['a', 'b', 'c', 'd']
connected = [
    ['b','d'],
    ['c','a'],
    ['d','b'],
    ['a','c'],
]
# later
    room_number = rooms.index(position)
    ways = connected[room_number]
```

This is not nice. We have to "manually" match the rooms to the entries in the connected list, and the code to get the list is ugly.

The problem is that lists always are accessed using numbers. **Dictionaries**, though, are a bit like lists, but you can use strings to address the values. Then, the code looks like:

```
rooms = ['a', 'b', 'c','d']
connected = {
    'a': ['b','d'],
    'b': ['c','a'],
    'c': ['d','b'],
    'd': ['a','c'],
}
# later
    ways = connected[position]
```

Much nicer, isn't it? Now the order in "connected" does not matter (we are using names), and we can get the right list immediately.

1.2 More things to do with dictionaries

There is even more you can do with a dictionary:

```
# Creating a dictionary
rating = { 'Hip Hop': 1, 'Rock' : 9}
# Reading a value
print 'I rate Hip Hop at: ' + str(rating['Hip Hop'])
# Writing a value
rating['Rock'] = 10
# Adding a value
rating['Classic'] = 3
# Checking if we have a rating
if 'RNB' in rating:
    print 'We have rated RNB'
```

Can you also remove a key? Sure you can! Find out yourself, using the

Python Library Reference
http://python.org/doc/2.3/lib/
Bookmark this!

There you can find all kinds of things you can do with python, to explore yourself, including file access, networking, colors on the terminal etc., for your own experiments[1]

[1] You can delete a key using "del rating['Hip Hop']", as described on http://python.org/doc/2.3/lib/typesmapping.html

2 Useful String and List Functions

Often when programming, you want to mess around with strings and lists, modify them, parse them, etc. You can find a lot in the python reference, but here are two functions that we need today, so let's have a look:

2.1 Splitting strings

Often, you have an input that contains various words, and you want to work on these words alone. For that, we can use the method split(), which returns the list of the words in the string:

```
print "This contains a few words".split()
# outputs: ['This', 'contains', 'a', 'few', 'words']
print "OneWord".split()
# outputs: ['OneWord']
print "".split()
# outputs: []
print "Only.Splits.At.Spaces".split()
# outputs: ['Only.Splits.At.Spaces']
```

2.2 Puttings words back together

There is also an inverse operation to splitting, it's joining. We can join a list of strings using, for example, a comma, or something else:

```
print ', '.join(['Apples','Peaches','Bananas'])
# outputs: Apples, Peaches, Bananas
print ', '.join(['Here is only one String'])
# outputs: 'Here is only one String'
print ' '.join('This contians a few words'.split())
# outputs: 'This contians a few words'
```

2.3 Seleting parts of a list

When we have a list, we might want to select only parts of it:

```
list = ['This','is','a','list']
print list      # ['This', 'is', 'a', 'list']
print list[1]   # 'is'
print list[0]   # 'This'
print list[-1]  # 'list'
print list[1:3] # ['is', 'a'], exclusive list[3]!
print list[2:]  # ['a', 'list']
print list[:3]  # ['This', 'is', 'a']
```

2.4 Adding a value to a list

If you want to add a value to a list, you can use the method "append":

```
list = ['This','is','a']
list.append('list')
print list
# outputs: ['This', 'is', 'a', 'list']
```

If you want to add all values of one list to another, use "extend":

```
list2 = ['a', 'list']
wrong = ['This', 'is']
wrong.append(list2)
print wrong
# outputs: ['This', 'is', ['a', 'list']]
right = ['This', 'is']
right.extend(list2)
print right
# outputs: ['This', 'is', 'a', 'list']
```

2.5 Removing a value from a list

Easy: Use "remove":

```
list = ['This','is','a','list']
list.remove('a')
print list
# output: ['This', 'is', 'list']
```

3 The Adventure!

Here is the program that uses all these things. It is based on the maze from our sixth session:

```
1  #!/usr/bin/python
2
3  # Where the user can go to from a given room
4  connections = {
5          'kitchen':     ['hallway','living room'],
6          'bedroom':     ['bathroom','hallway'],
7          'bathroom':    ['bedroom'],
8          'hallway':     ['kitchen','bedroom','living room'],
9          'living room': ['kitchen','hallway'],
10 }
11
```

```
12 # What objects are in givin room at the beginning
13 placed_objects = {
14         'kitchen':      ['knive'],
15         'living room': ['remote control'],
16         'bedroom':     ['balloon'],
17         'hallway':      ['cat','dog','shaving foam'],
18         'bathroom':    [],
19 }
20
21 # Where the user starts, and what he carries
22 start = 'living room'
23 carried_objects = [ 'bedsheets' ]
24
25 # Where he as to go, and what he needs to have
26 end = 'bathroom'
27 needed = ['knive','shaving foam']
28
29 # We trace his steps in the list ''path''
30 path = [start]
31 position = start
32
33 # Telling the user where is is, where he can go, what he can take or drop
34 def show_options():
35         print 'You are in '+position+'.'
36         # Find the possible ways
37         ways = connections[position]
38         print 'You can "goto" these rooms: ' + ', '.join(ways)
39         print 'You can "take" these objects: ' + ', '.join(placed_objects[position])
40         print 'You can "drop" these objects: ' + ', '.join(carried_objects)
41
42 # Going somewhere
43 def goto(where):
44         # We are changing the position, which is a variable that does
45         # not belong to this function, so we have to tell python this.
46         # This is a safety check so that we don't accidentally change
47         # a global variable. Only needed for writing, not for reading.
48         global position
49         if where in connections[position]:
50                 path.append(where) # tracing the steps
51                 position = where # updating the position
52         else:
53                 print "Sorry, but you can't go there from here"
54
55 # User wants to put something down
```

```
56 def drop(what):
57         if what in carried_objects:
58                 placed_objects[position].append(what)
59                 carried_objects.remove(what)
60                 print "You have dropped "+what+"."
61         else:
62                 print "You don't seem to have a "+what+"."
63
64 # User wants to take someting up
65 def take(what):
66         if what in placed_objects[position]:
67                 carried_objects.append(what)
68                 placed_objects[position].remove(what)
69                 print "You carry "+what+" now."
70         else:
71                 print "I looked everywhere, but I dont see a "+what+" here..."
72
73 # Is the puzzle solved? Return ''True'' then, ''False'' otherwise
74 def solved():
75         if position == end:
76                 print "You have found the "+end+"!"
77                 for need in needed:
78                         if need not in carried_objects:
79                                 print "Sorry, you don't have the "+need+"..."
80                                 return False
81                 print "And you brought all you need.."
82                 return True
83         else:
84                 return False
85
86 # This is now our main loop, which we repeat until the puzzle is solved:
87 while not solved():
88         show_options()
89         # We ask for input...
90         choice = raw_input('What do you want to do? ')
91         # ... and parse the result in a command and the command's option
92         words = choice.split()
93         command = words[0]
94         rest = words[1:]
95         what = " ".join(rest)
96
97         # Depending on the command, we run the appropriate function
98         if   command == "goto" or command == "go":
99                 goto(what)
```

```
100         elif command == "take":
101             take(what)
102         elif command == "drop":
103             drop(what)
104         else:
105             print "Sorry, but I did not understand your command"
106 
107         print '--------------------------------------------------'
108 
109 # We are out of the loop, so the user finished the maze. Great!
110 print 'Congratulations, you have finished the maze'
111 print 'You took this path:'
112 print ' -> '.join(path)
```