Lecture 3 Notes

November 8, 2019

Lecture 3: Loops and Arrays, Loops and Arrays, Loops. . .

Introduction to Arrays

Arrays are another way to store data. In a nutshell, an array contains a series of variables that are ordered with numbers from 0 to some number (integer) n. Note that if you want to get the 1st element of an array, you must access the 0th element. This is called **0-indexing**. A list can contain multiple different types of variables in Python but this is different in other programming languages.

Syntax of an Array

Some examples of arrays:

```
some_numbers = [1, 2, 3, 4, 5]
some_array = [True, 23.21, "HI!"]
alphabet = ['a', 'b', 'c', 'd',]
```

These arrays can be accessed with '[]', like this:

```
some_numbers[3] # "4th" element in the array => 4
some_array[2]   # => "HI!"
alphabet[0]     # => 'a'
```

So how are arrays useful?

Lets say that we have a program that calculates the average of any number of numbers you give it. So if we give it the numbers 1, 2, and 3, it would give the average of 2. Without arrays, making this would be doable but quite annoying. With arrays, we can just add the number that the user inputs to the end of the array and after the user is done inputting numbers, we take the average.

e.g

In other words: Instead of having

```
user_input_1 = raw_input()
user_input_2 = raw_input()
user_input_3 = raw_input()
user_input_4 = raw_input()
user_input_5 = raw_input()
...
```

We can neatly organise it in an array

```
user_inputs = []
```

Using an array makes the code much more clear and easier to modify.

Advanced Arrays

As already said, arrays are basically a collection of variables in a variable. So what about arrays inside arrays? Array-ception.

Of course, this can be done in Python! These are called **nested arrays**. They can be very useful in many computing problems.

An example of a 2-D array would be:

```
two_dee = [["this", "is a", "nested array"], [1,2,3,4],
    [True, True, False]]
```

A very important thing to realise here is that what a '2-D' list *really* is just a list of lists

Modifying Arrays

In order to add elements to an array you can do

```
myArray = []
myArray.append(1)
myArray.append(4)
myArray.append(41)
# myArray = [1, 4, 41]
myArray.remove(4)
# myArray = [1, 41]
```

'.append(a)' adds a value of 'a' into the array at the end.

'.remove(a)' removes a value 'a' from the array, if it is not found it returns an error.

There is also a function called '.insert(i, elem)' which takes an index of 'i' and puts 'elem' into that location.

Joining Lists

Joining two lists is easy in Python! Here is an example:

```
array1 = ["Hello ", "World! "]
array2 = ["Bye ", "Space!"]
print(array1 + array2)
```

Guess what this program will print.

Very useful interactions with lists

```
arr = [1,3,2,4,5]
len(arr)          # => 5
sum(arr)          # => 15
max(arr)          # => 5
min(arr)          # => 1
arr.sort()        # => [1,2,3,4,5]
arr.pop()         # => [1,2,3,4]
arr.clear()       # => []
```

Slashing and Slicing Lists

Lets say that we have python list like below

```
our_array = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
```

Of course, to get *one* element from the list, e.g 5, we do 'our_array[4]'. But lets say that we want a *range* of values. For example, we could want a *list* of numbers from 2 to 10 from our_array variable. In order to do this we use slicing:

```
our_array[1:10]     # => [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Similar to the good old array[a] which returns the value at index a but array[a:b] returns the list of values in between the indices of a and b (not including b).

Introduction to Loops

Loops are a way of repeating certain code snippets.

For example the following code snippet would print the numbers from 1 to 99 on the screen.

```
for i in range(0,100):
    print(i)
```

This specific example is a 'for' loop. It takes a variable, in this case i, that changes every time the loop goes on to another iteration.

Note: range is a 'function' that produces a list of number that fulfil $0 \leq x < 100$

While Loops

Another kind of loop would be a 'while' loop. Instead of changing a variable, it takes a statement like in an 'if' statement and loops as long as this statement is true. For example this loop would print "hello" infinitely as the statement is always 'True':

```
while True:
    print("hello")
```

This is usually bad practice in programming as there is no real way to quit the program.

However you can call a keyword called 'break' which essentially breaks the loop and continues with the code below.

```
while True:
    if input() == "":
        break
```

this snippet accepts input but only 'exits' when nothing is given

Loops with arrays

Lets say that you want to take a sum of some array filled with numbers. For this you can use the 'in' keyword.

```
num_array = [...]
total = 0
for num in num_array:
    total += num
print(total)
```

This snippet will take some array called 'num_array' and sum all of it and then print the sum to the user.

The 'in' keyword basically goes through all the element in an array and is very useful compared to writing the for loop with indexes.

Coming back to loops and arrays

Remember the example before on why array are more useful than normal, single variables?

It might be unclear on how to *populate* an array with user data, so here is a simple method on asking the user for every single input to the array

```
arr = []

while True:
    user_input = input()
    if user_input = "":
        break
    arr.append(user_input)
```

Try this and give it some data and probably print the array after the user has given the data to the array.

Challenges!

Challenge no. 1

Make a Python program that that asks the user to input any number of numbers from the user and then calculates the sum, the average, and the range of input. As an additional requirement, make sure that the user is actually inputting numbers instead of nonsense.

Challenge no. 2

Take 10 numbers from the user into an array and then: 1. Print the first number given 2. Print the last number given 3. Print the array, but without the first element nor the last 4. Print a sorted version of the array

Challenge no. 3

A classic: "fizzbuzz"

Iterate through the numbers from 1 to 100 and do the following:

1. Print "fizz" if the number is divisible by 3
2. Print "buzz" if the number is divisible by 5
3. Print "fizzbuzz" if the number is divisible by 3 AND 5

Hint: remember the modulus operator: %