conditionals and flow control (week 2)

Ben Bolker

19:39 17 September 2019

# Lists and indexing (PP chapter 8)

*reference*: [Python intro](https://docs.python.org/3/tutorial/introduction.html) section 3.1.3

## Lists

* Use square brackets [] to set up a **list**
* Lists can contain anything but usually homogeneous
* Put other variables into lists
* range() makes a **range** but you can turn it into a list with list()
	+ *Set up a list that runs from 101 to 200*
* Indexing and slicing lists works almost the same way as indexing and slicing …
* Put lists into lists! (“yo dawg …”)
	+ difference between an *item from a list* (indexing, x[0]) and a *one-element list* (slicing, x[0:1])



## Other list operations

* Lots of things you can do with lists!
* Lists are **mutable**

x = [1,2,3]
y = x
y[2] = 17
print(x)

## [1, 2, 17]

Check it out at [Python Tutor](http://pythontutor.com/visualize.html#mode=edit)

* *operators* vs. *functions* vs. *methods* x+y vs. foo(x,y) vs. x.foo(y)
	+ list [*methods*](http://www.linuxtopia.org/online_books/programming_books/python_programming/python_ch14s07.html)
	+ appending and extending:

x = [1,2,3]
y = [4,5]
x.append(y)
print(x)

## [1, 2, 3, [4, 5]]

x = [1,2,3] # reset x
y = [4,5]
x.extend(y)
print(x)

## [1, 2, 3, 4, 5]

Can use + and += as shortcut for extending:

x = [1,2,3]
y = [4,5]
z = x+y
print(z)

## [1, 2, 3, 4, 5]

## list methods

* x.insert(position,value): inserts (or x=x[0:position]+[value]+x[position+1:len(x)])
* x.remove(value): removes *first* value
* x.pop(position) (or del x[position] or x=x[0:position]+x[position+1:len(x)])
* x.reverse() (or x[::-1])
* x.sort(): what it says
* x.count(value): number of occurrences of value
* x.index(value): first occurrence of value
* value in x: does value occur in x? (or logical(x.count(value)==0))
* len(x): length

**Note**: [pythonicity](http://blog.startifact.com/posts/older/what-is-pythonic.html) vs. [TMTOWTDI](http://en.wikipedia.org/wiki/There%27s_more_than_one_way_to_do_it)

# Conditionals and flow control

* **Conditionals**: Do something *if* something else is true
* **Flow control**: Go to different places in the code: especially, repeat calculations
* Everything we need for interesting programs ([“the rest is commentary”](http://en.wikipedia.org/wiki/Hillel_the_Elder))
* Technically we can compute *anything*: [Turing machines](http://en.wikipedia.org/wiki/Turing_machine) ([xkcd](http://xkcd.com/205/))

## Conditionals

* Do something *if* something is true
* if statement ([reference](https://docs.python.org/3/tutorial/controlflow.html))

if False:
 print("no")

* else-if (elif) and else [clauses](http://xkcd.com/764/)

if (x<=0):
 print("what??")
elif(x==1):
 print("one")
elif(x==2):
 print("two")
else:
 print("many")

* not too much else to say
* we can do more than one thing; use a *code block*
* indentation is crucial

## codingbat examples

* [CodingBat date\_fashion problem](http://codingbat.com/prob/p129125)
* [CodingBat alarm clock problem](http://codingbat.com/prob/p119867)

## while

* repeat code many times, *while* some logical statement is true ([reference](https://docs.python.org/3/reference/compound_stmts.html#while))

For example: 

x = 17
while x>1:
 x = x/2

Maybe we want to know how many steps that took:

x = 17
n = 0
while x>1:
 x = x/2
 n = n+1

* **What is the answer?**
* Can you get the same answer using import math and math.log(x,2) (and maybe round() or math.floor)?
* We can use logical operators to combine

x = 17
n = 0
while x>1 and n<3:
 x = x/2
 n = n+1

## for loops

* what if we want to repeat a fixed number of times? We could use something like

n = 0
while n<n\_max:
 # do stuff
 n = n+1

Or we could use a for loop:

for n in range(0,n\_max):
 # do stuff

* does this repeat n\_max or n\_max+1 times? (hint: try it out, and/or use list(range(...)) …)
* more generally, we can use for to iterate over *any list*.



for loop

## for loop examples

* [CodingBat > string-2 > countHi](https://codingbat.com/prob/p167246)
* [CodingBat > string-2 > catDog](https://codingbat.com/prob/p164876)
* [CodingBat > Array-2 > bigDiff](http://codingbat.com/prob/p196640)

Another example: a change-writing program.

*Given an amount of money, return a list of length 5 that gives the (smallest) number of coins of each unit (toonies, loonies, quarters, dimes, and nickels) required to make up that amount.*

total=5.73
toonies = 5.73 // 2 ## integer division
total = total - 2\*toonies

total = 5.73
res = [] # empty list
denoms = list(2,1,0.25,0.1,0.05)
for d in denoms:
 # do stuff

* start with total, use denoms above
1. program to see how many pennies are left (how could we do this much more easily?)
2. **or** print out change as we go along
3. **or** save results as an array

## Coin counting continued

Before coding up a solution, first describe it at a high level and then refine it:

* Initialization phase
	+ initialize the variables that will be used, such as variables to hold the total amount of money, the list of coin denominations being used, and a list of the results.
* Loop. For each denomination d in our list:
	+ determine how many coins of denomination d are needed.
	+ update our result list with this amount.
	+ update the total amount of money left.
* Print out the results

## Prime walk

Now let’s look at the [prime walk program](../code/primewalk.py) again …

* Initialization phase
	+ retrieve a list of primes
	+ initialize the variables that will be used:
		- variables to hold the lists of the x and y coordinates of the points visited on the walk
		- the current direction of the walk
		- the number of steps taken on the walk so far
* Loop. For each step of the walk:
	+ update the x and y coordinate lists with the coordinates of the next step
	+ change the walk direction.
* display the walk.

## More CodingBat examples:

* List-2 > count\_evens
* List-2 >sum13
* List-2 > bigdiff
* reverse a list (not using slicing)?

## break

break is a way to get out of a while or for loop early:

for i in range(0,10):
 if i>5:
 break

## nested for loops

We can look at (e.g.) all the combinations of i and j via:

for i in range(0,3):
 for j in range(0,3):
 print([i,j])

## matrix addition

We can store matrices as a **list of lists**: represents a 2 $×$ 3 matrix. We can loop over rows and columns to operate on every element, or combine the elements in some way:

## initialization
m = [[1,2,3], [2,7,9]]
nrows = len(m)
ncols = len(m[0])
total = 0
## loop
for i in range(nrows):
 for j in range(ncols):
 total += m[i][j]
print(total)

## 24

## Loops and indices

From [Secret Weblog](http://blog.startifact.com/posts/older/what-is-pythonic.html): all of the following are equivalent …

i = 0
while i < mylist\_length:
 do\_something(mylist[i])
 i += 1 ## or i=i+1

vs.

for i in range(mylist\_length):
 do\_something(mylist[i])

(this form is useful if we need to combine two lists, or otherwise index element i of several different things …)

vs.

for element in mylist:
 do\_something(element)

## Criteria

* speed
* memory use
* simplicity (code length)
* simplicity (avoid modules)
* simplicity (avoid abstractions)
* pythonicity