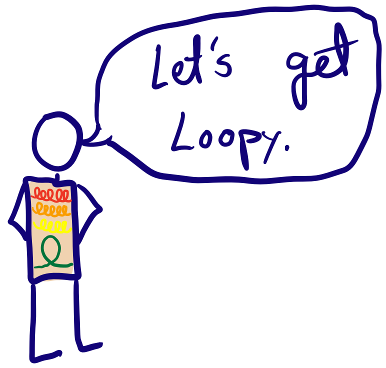
Iterations in Python



Whenever we want to do something many times in Python we should think about iteration.

First we should talk about the idea that some objects in Python are iterable. That means it is possible to get each value in the object and do something with it. Let's start with a for loop that iterates on an iterable:

```
for item in iterable:
    do something with item
```

} the loop body is indented.

A list is an explicit iterable.
The for loop assigns each item in an iterable to a variable name that you can use in the loop body

```
for item in [3, 1, 2]:        =>    3
    print(item)                     1
                                    2
```

There are also iterables that are defined by generators.

this is a generator
it is like [0, 1, 2]

```
for i in range(3):        =>    0
    print(i)                    1
                                2
```

This is helpful to do a task a certain number of times.

Sometimes you want to iterate on an iterable, and you need the index (position) and the items. Use **enumerate** for this.

the index of item

the item in each iteration

```
for i, item   in  enumerate(iterable):
    do something with i and item.
```

```
for i, val  in  enumerate([3,1,2]):        0, 3
    print (i, val)                  =>     1, 1
                                           2, 2
```

Iterating over multiple iterables

Say you have two (or more) lists and you want to iterate over pairs of each element? Use the zip function!

from iterable1

from iterable 2

```
for a, b in zip(iterable1, iterable2):
    do something with a and b
```

```
A = [1, 3, 5]

B = [2, 4, 6]

for a,b in zip(A,B):
    print (a + b)
```

$\Rightarrow$

```
3
7
11
```

List comprehensions

Often, you will want to save the output of each step of the iteration, which usually is a new list.

List comprehension is a convenient syntax for this.

```
[expression of var  for  var  in  iterable]
```

Say   A = [1, 2, 3]

```
B = [2*x for x in A]
```

⇒ [2, 4, 6]

While loops

for loops are finite loops, i.e. a fixed number of iterations. There are times you don't know how many iterations you need in advance. In that case we have the while loop

```
while condition_is_true:
    loop body
```

Here is an example

```
a = 10
while a >1:
    a= a/ 2
    print(a)
```

$\Rightarrow$

```
5
2.5
1.25
0.625
```

A while loop may run forever if the condition is always True! You can use a break statement to exit a loop.

```
a = 10
while a >0:
    a = a/2
    print(a)
    if a < 1:
        break
print('done')
```

Break out of this loop

$\Rightarrow$

```
5
2.5
1.25
0.625
done
```

You should always make sure there is a step that will terminate a while loop!

Note break only exits the current loop!

Nested loops

A nested loop runs inside another loop.
For example if you want to loop over every column in a row, you need two loops

$$M = \begin{array}{c|ccc} & \text{col } 0 & \text{col } 1 & \text{col } 2 \\ \text{row } 0 & [[1 & 3 & 5], \\ \text{row } 1 & [7 & 9 & 4], \\ \text{row } 2 & [2 & 0 & 1]] \end{array}$$

```
for row in M:
    for col in row:
        print(col)
```

⇒

```
1
3
5
7
9
4
2
0
1
```

M is iterable so we can directly get one row at a time.
A row is also iterable, so we can get each element in the for loop to do something.