While Loops

While Loops

For loops are useful if you know what you want to iterate over, but what if you wanted to keep looping until a certain condition is met? `while` loops are the tool for this job.

The syntax for a `while` loop is:

```
while condition:
    block of code to be repeated
```

where `condition` is/evaluates to a boolean value. The loop will keep repeating, executing the block of code indented after the `:` as long as `condition` evaluates to `True`. When `condition` evaluates to `False` the loop will no longer be repeated and control will progress to the code after the loop. Note that if `condition` starts as `False`, the code inside the loop will never be executed.

Worked Example

Let's consider the following problem where we can make use of a `while` loop. Consider the recursive series:

$$T_n = T_{n-1}^{3/4} \tag{1}$$
$$T_0 = 100 \tag{2}$$

We want to know when this series drops below 2 (what is the first value of n for which $T_n < 2$). One solution is:

[3]:
```
T = 100 #T_0 term

n = 0

while T >= 2:
    T = T**(3/4.) #T_{n+1} term
    n += 1

print('T_n is less than 2 for n =', n)
```

```
T_n is less than 2 for n = 7
```

Notice how the condition is `T >= 2` and not `T < 2`. That is because the loop continues **while** the condition is true and we want the loop to stop when `T < 2` is `True` (and the converse `T >= 2` is `False`).

Avoiding Infinite Recursion

Something to be careful of when using `while` loops is a loop that doesn't stop looping. If `condition` never evaluates to `False`, or if you never break out of the loop in another way, control will never leave the loop. Sometimes it is useful to use a maximum number of loop iterations to avoid this:

```
counter = 0

while condition and counter < max_count:
    block of code
```

where `max_count` is the chosen maximum number of recursions (normally chosen as a very large number).

Replacing For Loops

`while` loops can be used to replace for `loops`, for example:

[4]:
```python
## For loop
print('for loop')

for i in range(5):
    print(i)

## While loop
print('')
print('while loop')

i = 0

while i < 5:
    print(i)
    i+=1
```

```
for loop
0
1
2
3
4

while loop
0
1
2
```

```
3
4
```

As you can see the `while` loop is a bit less convenient than the `for` loop in this case. The `while` loop becomes even less convenient when looping through a collection:

[5]:
```
string = 'a string'

## For loop
print('for loop')

for char in string:
    print(char)

## While loop
print('')
print('while loop')

index = 0

while index < len(string):
    print(string[index])
    index += 1
```

```
for loop
a

s
t
r
i
n
g

while loop
a

s
t
r
i
n
g
```