For Loops

For Loops

For loops can be used to repeat a block of code by iterating through specified values.

The structure of a for loop is as follows:

```
for i in iterator:
    code block to be repeated
```

where `i` is the iteration variable and the iterator represents a sequence of values that `i` will be set to each time the loop is repeated. Here `i` can be treated like a variable and can take on any allowed variable name.

The code block to be repeated must be indented after the `:`. The loop repeats until `i` has run through all of the values in the iterator, or until the loop is broken.

Be careful not to alter the iterator inside the code block being repeated.

Worked Example

As a simple example, let's say we wanted to print out the first n integer squares starting with 0.

This can be achieved manually for small n, for example $n = 5$:

[3]:
```
print('0 squared is', 0**2)
print('1 squared is', 1**2)
print('2 squared is', 2**2)
print('3 squared is', 3**2)
print('4 squared is', 4**2)
print('5 squared is', 5**2)
```

```
0 squared is 0
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

This quickly becomes tedious and produces messy code. To achieve the same goal using a `for` loop we can do the following:

[5]:
```
for i in range(6):
    print(i, 'squared is', i**2)
```

```
0 squared is 0
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

Here we have made use of the `range` function to create our iterator.

The `range()` Function

The `range()` function takes integer arguments and produces a series of integers. As we shall see, the `range()` function's arguments are very similar to slicing.

In the example above we used it with one argument,

`range(stop)`

Here `range()` will produce a series of integers starting at zero and ending just before the `stop` value.

If we were to use range with 2 arguments:

`range(start, stop)`

`range()` will produce a series of integers starting with the `start` value and ending with the `stop` value.

For example:

[1]:
```
for i in range (2, 5):
    print(i)
```

```
2
3
4
```

Lastly if we were to use `range()` with 3 arguments:

`range(start, stop, step)`

`range()` returns a series starting with the `start` value and with step sizes of `step` in between each value until it reaches the value before `stop`.

For example:

[2]:
```
for i in range(2, 10, 3):
    print(i)
```

```
2
5
8
```

The default value for `step` is 1. If you want the series to descend, you can use a negative step size:

[5]:
```python
for i in range(11, 3, -2):
    print(i)
```

```
11
9
7
5
```

Looping Through Sequences

Sequences such as tuples, lists and strings can also be used as iterators. For example:

[6]:
```python
for char in 'This string':
    print(char)
```

```
T
h
i
s
 
s
t
r
i
n
g
```

and

[7]:
```python
for item in [1, 2, 3, 'a', 'b', 'c']:
    print(item)
```

```
1
2
3
a
b
c
```

`enumerate()` To Iterate Through Sequence and Index

Sometimes you want to loop through a sequence but also want to keep track of the index. This can be achieved by using range:

```
[3]: string = 'string'

     for i in range(len(string)):
         print(i, string[i])
```

```
0 s
1 t
2 r
3 i
4 n
5 g
```

but there is a far more convenient way using the `enumerate()` function:

```
[1]: for i,char in enumerate('string'):
         print(i, char)
```

```
0 s
1 t
2 r
3 i
4 n
5 g
```

`zip()` To Iterate Through More Than One Sequence Simultaneously

If you wanted to loop through more than one sequence at a time you could iterate through the index:

```
[5]: list_a = [1,2,3]
     list_b = ['a', 'b', 'c']

     for i in range(len(list_a)):
         print(list_a[i], list_b[i])
```

```
1 a
2 b
3 c
```

but there is a cleaner way using the `zip` function:

```
[7]: for a,b in zip([1,2,3], ['a', 'b', 'c']):
         print(a, b)
```

```
1 a
2 b
3 c
```

Note that the loop will only iterate as much as the shortest sequence.

Looping Through Dictionaries

To loop through the key-value pairs of a dictionary you can use the `dict.items()` method:

[1]:
```
d = {'a' : 54, 'b' : 754, 'c' : 42}

for k,v in d.items():
    print(k, v)
```

```
a 54
b 754
c 42
```