Example 1

Iterating through a list

In [1]:

# We'll learn how to automate this sort of list in the next lecture

list1 = [1,2,3,4,5,6,7,8,9,10]

In [2]:

for num in list1:

print(num)

1

2

3

4

5

6

7

8

9

10

Great! Hopefully this makes sense. Now let's add an <code>if</code> statement to check for even numbers. We'll first introduce a new concept here--the modulo.

### Modulo

The modulo allows us to get the remainder in a division and uses the % symbol. For example:

Great! Hopefully this makes sense. Now let's add an if statement to check for even numbers. We'll first introduce a new concept here--the modulo.

Modulo¶

The modulo allows us to get the remainder in a division and uses the % symbol. For example:

In [3]:

17 % 5

This makes sense since 17 divided by 5 is 3 remainder 2. Let's see a few more quick examples:

This makes sense since 17 divided by 5 is 3 remainder 2. Let's see a few more quick examples:

In [4]:

# 3 Remainder 1

10 % 3

In [5]:

# 2 Remainder 4

18 % 7

In [6]:

# 2 no remainder

4 % 2

Notice that if a number is fully divisible with no remainder, the result of the modulo call is 0. We can use this to test for even numbers, since if a number modulo 2 is equal to 0, that means it is an even number!

Back to the <code>for</code> loops!

## Example 2

Let's print only the even numbers from that list!

Notice that if a number is fully divisible with no remainder, the result of the modulo call is 0. We can use this to test for even numbers, since if a number modulo 2 is equal to 0, that means it is an even number!

Back to the for loops!

Example 2¶

Let's print only the even numbers from that list!

In [7]:

for num in list1:

if num % 2 == 0:

print(num)

2

4

6

8

10

We could have also put an <code>else</code> statement in there:

We could have also put an else statement in there:

In [8]:

for num in list1:

if num % 2 == 0:

print(num)

else:

print('Odd number')

Odd number

2

Odd number

4

Odd number

6

Odd number

8

Odd number

10

## Example 3

Another common idea during a <code>for</code> loop is keeping some sort of running tally during multiple loops. For example, let's create a <code>for</code> loop that sums up the list:

Example 3¶

Another common idea during a for loop is keeping some sort of running tally during multiple loops. For example, let's create a for loop that sums up the list:

In [9]:

# Start sum at zero

list\_sum = 0

for num in list1:

list\_sum = list\_sum + num

print(list\_sum)

55

Great! Read over the above cell and make sure you understand fully what is going on. Also we could have implemented a <code>+=</code> to perform the addition towards the sum. For example:

Great! Read over the above cell and make sure you understand fully what is going on. Also we could have implemented a += to perform the addition towards the sum. For example:

In [10]:

# Start sum at zero

list\_sum = 0

for num in list1:

list\_sum += num

print(list\_sum)

55

## Example 4

We've used <code>for</code> loops with lists, how about with strings? Remember strings are a sequence so when we iterate through them we will be accessing each item in that string.

Example 4¶

We've used for loops with lists, how about with strings? Remember strings are a sequence so when we iterate through them we will be accessing each item in that string.

In [11]:

for letter in 'This is a string.':

print(letter)

T

h

i

s

i

s

a

s

t

r

i

n

g

.

## Example 5

Let's now look at how a <code>for</code> loop can be used with a tuple:

Example 5¶

Let's now look at how a for loop can be used with a tuple:

In [12]:

tup = (1,2,3,4,5)

for t in tup:

print(t)

1

2

3

4

5

## Example 6

Tuples have a special quality when it comes to <code>for</code> loops. If you are iterating through a sequence that contains tuples, the item can actually be the tuple itself, this is an example of \*tuple unpacking\*. During the <code>for</code> loop we will be unpacking the tuple inside of a sequence and we can access the individual items inside that tuple!

Example 6¶

Tuples have a special quality when it comes to for loops. If you are iterating through a sequence that contains tuples, the item can actually be the tuple itself, this is an example of tuple unpacking. During the for loop we will be unpacking the tuple inside of a sequence and we can access the individual items inside that tuple!

In [13]:

list2 = [(2,4),(6,8),(10,12)]

In [14]:

for tup in list2:

print(tup)

(2, 4)

(6, 8)

(10, 12)

In [15]:

# Now with unpacking!

for (t1,t2) in list2:

print(t1)

2

6

10

Cool! With tuples in a sequence we can access the items inside of them through unpacking! The reason this is important is because many objects will deliver their iterables through tuples. Let's start exploring iterating through Dictionaries to explore this further!

Cool! With tuples in a sequence we can access the items inside of them through unpacking! The reason this is important is because many objects will deliver their iterables through tuples. Let's start exploring iterating through Dictionaries to explore this further!

## Example 7

Example 7¶

In [16]:

d = {'k1':1,'k2':2,'k3':3}

In [17]:

for item in d:

print(item)

k1

k2

k3

Notice how this produces only the keys. So how can we get the values? Or both the keys and the values?

We're going to introduce three new Dictionary methods: \*\*.keys()\*\*, \*\*.values()\*\* and \*\*.items()\*\*

In Python each of these methods return a \*dictionary view object\*. It supports operations like membership test and iteration, but its contents are not independent of the original dictionary – it is only a view. Let's see it in action:

Notice how this produces only the keys. So how can we get the values? Or both the keys and the values?

We're going to introduce three new Dictionary methods: .keys(), .values() and .items()

In Python each of these methods return a dictionary view object. It supports operations like membership test and iteration, but its contents are not independent of the original dictionary – it is only a view. Let's see it in action:

In [18]:

# Create a dictionary view object

d.items()

Out[18]:

dict\_items([('k1', 1), ('k2', 2), ('k3', 3)])

Since the .items() method supports iteration, we can perform \*dictionary unpacking\* to separate keys and values just as we did in the previous examples.

Since the .items() method supports iteration, we can perform dictionary unpacking to separate keys and values just as we did in the previous examples.

In [19]:

# Dictionary unpacking

for k,v in d.items():

print(k)

print(v)

k1

1

k2

2

k3

3

If you want to obtain a true list of keys, values, or key/value tuples, you can \*cast\* the view as a list:

If you want to obtain a true list of keys, values, or key/value tuples, you can cast the view as a list:

In [20]:

list(d.keys())

Out[20]:

['k1', 'k2', 'k3']

Remember that dictionaries are unordered, and that keys and values come back in arbitrary order. You can obtain a sorted list using sorted():

Remember that dictionaries are unordered, and that keys and values come back in arbitrary order. You can obtain a sorted list using sorted():

In [21]:

sorted(d.values())

Out[21]:

[1, 2, 3]