

# The `if` statement and files

# The `if` statement

Do a code block only when something is True

```
if test:  
    print "The expression is true"
```

# Example

```
if "GAATTC" in "ATCTGGAATTCATCG":  
    print "EcoRI site is present"
```

**if** the test is true...

```
if "GAATTC" in "ATCTGGAATTCATCG":  
    print "EcoRI site is present"
```



The test is: "GAATTC" in "ATCTGGAATTCATCG"

# Then **print** the message

```
if "GAATTC" in "ATCTGGAATTCATCG":  
    print "EcoRI site is present"
```

Here is it done in the Python shell

```
>>> if "GAATTC" in "ATCTGGAATTCATCG":  
...     print "EcoRI is present"  
...  
EcoRI is present  
>>>
```

# What if you want the false case?

There are several possibilities; here's two

1) Python has a **not in** operator

```
if "GAATTC" not in "AAAAAAAAA":  
    print "EcoRI will not cut the sequence"
```

2) The **not** operator switches true and false

```
if not "GAATTC" in "AAAAAAAAA":  
    print "EcoRI will not cut the sequence"
```

# In the Python shell

```
>>> x = True
>>> x
True
>>> not x
False
>>> not not x
True
>>> if "GAATTC" not in "AAAAAAAAA":
...     print "EcoRI will not cut the sequence"
...
EcoRI will not cut the sequence
>>> if not "GAATTC" in "ATCTGGAATTCATCG":
...     print "EcoRI will not cut the sequence"
...
>>> if not "GAATTC" in "AAAAAAAAA":
...     print "EcoRI will not cut the sequence"
...
EcoRI will not cut the sequence
>>>
```

# else:

What if you want to do one thing when the test is true and another thing when the test is false?

Do the first code block (after the `if:`) if the test is true

```
if "GAATTC" in "ATCTGGAATTCATCG":  
    print "EcoRI site is present"
```

```
else:
```

```
    print "EcoRI will not cut the sequence"
```

Do the second code block (after the `else:`) if the test is false



# Examples with `else`

```
>>> if "GAATTC" in "ATCTGGAATTCATCG":  
...     print "EcoRI site is present"  
... else:  
...     print "EcoRI will not cut the sequence"  
...  
EcoRI site is present  
>>> if "GAATTC" in "AAAAC TCGT":  
...     print "EcoRI site is present"  
... else:  
...     print "EcoRI will not cut the sequence"  
...  
EcoRI will not cut the sequence  
>>>
```

# Where is the site?

The 'find' method of strings returns the index of a substring in the string, or -1 if the substring doesn't exist

```
>>> seq = "ATCTGGAATTCATCG"
```

```
>>> seq.find("GAATTC")
```

```
5
```

```
>>> seq.find("GGCGC")
```

```
-1
```

```
>>>
```

} There is a GAATTC  
at position 5

} But there is no GGCGC  
in the sequence

# But where is the site?

```
>>> seq = "ATCTGGAATTCATCG"
>>> pos = seq.find("GAATTC")
>>> if pos == -1:
...     print "EcoRI does not cut the sequence"
... else:
...     print "EcoRI site starting at index", pos
...
EcoRI site starting at index 5
>>>
```

Start by creating the string “ATCTGGAATTCATCG”  
and assigning it to the variable with name ‘seq’

```
→ seq = "ATCTGGAATTCATCG"  
pos = seq.find("GAATTC")  
if pos == -1:  
    print "EcoRI does not cut the sequence"  
else:  
    print "EcoRI site starting at index", pos
```

Using the `seq` string, call the method named `find`. This looks for the string `"GAATTC"` in the `seq` string

```
seq = "ATCTGGAATTCATCG"  
→ pos = seq.find("GAATTC")  
if pos == -1:  
    print "EcoRI does not cut the sequence"  
else:  
    print "EcoRI site starting at index", pos
```

The string “GAATC” is at position 5 in the seq string.  
Assign the 5 object to the variable named **pos**.

```
seq = "ATCTGGAATTCCATCG"  
→ pos = seq.find( "GAATTC" )  
if pos == -1:  
    print "EcoRI does not cut the sequence"  
else:  
    print "EcoRI site starting at index", pos
```

The variable name “**pos**” is often used for positions.  
Common variations are “pos1”, “pos2”,  
“start\_pos”, “end\_pos”

Do the test for the `if` statement

Is the variable `pos` equal to `-1`?

```
seq = "ATCTGGAATTCATCG"  
pos = seq.find("GAATTC")  
→ if pos == -1:  
    print "EcoRI does not cut the sequence"  
else:  
    print "EcoRI site starting at index", pos
```

Since pos is 5 and 5 is not equal to -1,  
this test is false.

```
seq = "ATCTGGAATTCATCG"  
pos = seq.find("GAATTC")  
→ if pos == -1: ← The test is False  
    print "EcoRI does not cut the sequence"  
else:  
    print "EcoRI site starting at index", pos
```



Skip the first code block  
(that is only run if the test is True)

Instead, run the code block after the `else`:

```
seq = "ATCTGGAATTCATCG"  
pos = seq.find("GAATTC")  
if pos == -1:  
    print "EcoRI does not cut the sequence"  
else:  
→ print "EcoRI site starting at index", pos
```

This is a `print` statement.  
Print the index of the start position

```
seq = "ATCTGGAATTCATCG"  
pos = seq.find("GAATTC")  
if pos == -1:  
    print "EcoRI does not cut the sequence"  
else:  
→ print "EcoRI site starting at index", pos
```

This prints

```
EcoRI site starting at index 5
```

There are no more statements so Python stops.

```
seq = "ATCTGGAATTCATCG"  
pos = seq.find("GAATTC")  
if pos == -1:  
    print "EcoRI does not cut the sequence"  
else:  
    print "EcoRI site starting at index", pos
```



# A more complex example

## Using `if` inside a `for`

```
restriction_sites = [  
    "GAATTC",      # EcoRI  
    "GGATCC",      # BamHI  
    "AAGCTT",      # HindIII  
]  
  
seq = raw_input("Enter a DNA sequence: ")  
  
for site in restriction_sites:  
    if site in seq:  
        print site, "is a cleavage site"  
    else:  
        print site, "is not present"
```

# Nested code blocks

```
restriction_sites = [  
    "GAATTC",    # EcoRI  
    "GGATCC",    # BamHI  
    "AAGCTT",    # HindIII  
]
```

```
seq = raw_input("Enter a DNA sequence: ")
```

```
for site in restriction_sites:
```

```
    if site in seq:  
        print site, "is a cleavage site"  
    else:  
        print site, "is not present"
```

} This is the code  
block for the  
for statement

```
restriction_sites = [  
    "GAATTC",    # EcoRI  
    "GGATCC",    # BamHI  
    "AAGCTT",    # HindIII  
]  
  
seq = raw_input("Enter a DNA sequence: ")  
  
for site in restriction_sites:  
    if site in seq:  
        print site, "is a cleavage site"  
    else:  
        print site, "is not present"
```

This is the code  
block for the  
True part of the  
if statement

```
restriction_sites = [  
    "GAATTC",    # EcoRI  
    "GGATCC",    # BamHI  
    "AAGCTT",    # HindIII  
]  
  
seq = raw_input("Enter a DNA sequence: ")  
  
for site in restriction_sites:  
    if site in seq:  
        print site, "is a cleavage site"  
    else:  
        print site, "is not present" }
```

This is the code  
block for the  
False part of the  
if statement

# The program output

Enter a DNA sequence: **AATGAATTCTCTGGAAGCTTA**

GAATTC is a cleavage site

GGATCC is not present

AAGCTT is a cleavage site



# Read lines from a file

- `raw_input()` asks the user for input
- Most of the time you'll get data from a file. (Or would you rather type in the sequence every time?)
- To read from a file you need to tell Python to open that file.

# The open function

```
>>> infile = open("/usr/coursehome/dalke/10_sequences.seq")
>>> print infile
<open file '/usr/coursehome/dalke/10_sequences.seq', mode 'r' at 0x817ca60>
>>>
```

open returns a new object of type *file*

A file can't be displayed like a number or a string. It is useful because it has methods for working with the data in the file.

# the readline() method

```
>>> infile = open("/usr/coursehome/dalke/10_sequences.seq")
>>> print infile
<open file '/usr/coursehome/dalke/10_sequences.seq', mode 'r' at 0x817ca60>
>>> infile.readline()
'CCTGTATTAGCAGCAGATTTCGATTAGCTTTACAACAATTCAATAAAATAGCTTCGCGCTAA\n'
>>>
```

readline returns one line from the file

The line includes the end of line character (represented here by “\n”)

(Note: the last line of some files may not have a “\n”)

# readline finishes with ""

```
>>> infile = open("/usr/coursehome/dalke/10_sequences.seq")
>>> print infile
<open file '/usr/coursehome/dalke/10_sequences.seq', mode 'r' at 0x817ca60>
>>> infile.readline()
'CCTGTATTAGCAGCAGATTTCGATTAGCTTTACAACAATTCAATAAAATAGCTTCGCGCTAA\n'
>>> infile.readline()
'ATTTTTAACTTTTCTCTGTCGTCGCACAATCGACTTTCTCTGTTTTCTTGGGTTTACCGGAA\n'
>>> infile.readline()
'TTGTTTCTGCTGCGATGAGGTATTGCTCGTCAGCCTGAGGCTGAAAATAAAATCCGTGGT\n'
>>> infile.readline()
'CACACCAATAAGTTAGAGAGAGTACTTTGACTTGGAGCTGGAGGAATTTGACATAGTCGAT\n'
>>> infile.readline()
'TCTTCTCCAAGACGCATCCACGTGAACCGTTGTA ACTATGTTCTGTGC\n'
>>> infile.readline()
'CCACACCAAAAAA ACTTTCCACGTGAACCGAAAACGAAAGTCTTTGGTTTTAATCAATAA\n'
>>> infile.readline()
'GTGCTCTCTTCTCGGAGAGAGAAGGTGGGCTGCTTGTCTGCCGATGTACTTTATTAATCCAATAA\n'
>>> infile.readline()
'CCACACCAAAAAA ACTTTCCACGTGTGA ACTTACTCCAAAAACGAAGTATTGGTTTATCATAA\n'
>>> infile.readline()
'TCTGAAAAGTGCAAAGAACGATGATGATGATGATAGAGGAACCTGAGCAGCCATGTCTGAACCTATAGC\n'
>>> infile.readline()
'GTATTGGTTCGTCGTGCGACTAAATTAGGTA AAAAAAGTAGTTCTAAGAGATTTTGATGATTCAATGCAAAGTTCTATTAATCGTTCAATTG\n'
>>> infile.readline()
''
>>>
```

When there are no more lines,  
readline returns the empty string

# Using `for` with a file

A simple way to read lines from a file

```
>>> filename = "/usr/coursehome/dalke/10_sequences.seq"
>>> for line in open(filename):
...     print line[:10]
...
CCTGTATTAG
ATTTTAACT
TTGTTTCTGC
CACACCCAAT
TCTTCTCCAA
CCACACCAA
GTGCTCTCTT
CCACACCAA
TCTGAAAAGT
GTATTGGTCG
>>>
```

`for` starts with the first line in the file ...  
then the second ...  
then the third ...  
...  
and finishes with the last line.

# A more complex task

List the sequences starting with a cytosine

```
>>> filename = "/usr/coursehome/dalke/10_sequences.seq"
>>> for line in open(filename):
...     line = line.rstrip()
...     if line.startswith("C"):
...         print line
...
...
CCTGTATTAGCAGCAGATTCGATTAGCTTTACAACAATTCAATAAAATAGCTTCGCGCTAA
CACACCCAATAAGTTAGAGAGAGTACTTTGACTTGGAGCTGGAGGAATTTGACATAGTCGAT
CCACACCAAAAAAATTTCCACGTGAACCGAAAACGAAAGTCTTTGGTTTTAATCAATAA
CCACACCAAAAAAATTTCCACGTGTGAACTATACTCCAAAAACGAAGTATTGGTTTATCATAA
>>>
```

← Use `rstrip` to get rid of the “`\n`”

# Exercise 1

Get a sequence from the user. If there is an A in the sequence, print the number of times it appears in the sequence. Do the same for T, C and G. If a base does not exist, don't print anything.

Enter a sequence: **ACCAGGCA**

A count: 3

C count: 3

G count: 2

---

Test input #1:

Enter a sequence: **TTTTTGGGG**

T count: 5

G count: 4

Test input #2:

# Excercise 2

Get a sequence from the user. If there is an A in the sequence, print the number of times it appears in the sequence. If it does not exist, print “A not found”. Do the same for T, C and G.

Test input #1:

```
Enter a sequence: ACCAGGCA  
A count: 3  
T not found  
C count: 3  
G count: 2
```

---

Test input #2:

```
Enter a sequence: TTTTTGGGG  
A not found  
T count: 5  
C not found  
G count: 4
```



# Exercise 3

## number lines in a file

Read the file `/usr/coursehome/dalke/10_sequences.seq` .  
Print out the line number (starting with 1) then the line.  
Remember to use `rstrip()` to remove the extra newline.

The output should look like this

```
1 CCTGTATTAGCAGCAGATTCGATTAGCTTTACAACAATTCAATAAAATAGCTTCGCGCTAA
2 ATTTTTAACTTTTCTCTGTCGTCGCACAATCGACTTTCTCTGTTTTCTTGGGTTTACCGGAA
3 TTGTTTCTGCTGCGATGAGGTATTGCTCGTCAGCCTGAGGCTGAAAATAAAATCCGTGGT
4 CACACCCAATAAGTTAGAGAGAGTACTTTGACTTGGAGCTGGAGGAATTTGACATAGTCGAT
5 TCTTCTCCAAGACGCATCCACGTGAACCGTTGTAACCTATGTTCTGTGC
6 CCACACCAAAAAAATTTCCACGTGAACCGAAAACGAAAGTCTTTGGTTTTAATCAATAA
7 GTGCTCTCTTCTCGGAGAGAGAAGGTGGGCTGCTTGTCTGCCGATGTACTTTATTAAATCCAATAA
8 CCACACCAAAAAAATTTCCACGTGTGAACCTATACTCCAAAAACGAAGTATTGGTTTATCATAA
9 TCTGAAAAGTGCAAAGAACGATGATGATGATGATAGAGGAACCTGAGCAGCCATGTCTGAACCTATAGC
10 GTATTGGTCGTCGTGCGACTAAATTAGGTAAAAAAGTAGTTCTAAGAGATTTTGATGATTC AATGCAAAGTTCTATTAATCGTTCAATTG
```

# Exercise 4

List the sequences in `/usr/coursehome/dalke/I0_sequences.seq` which have the pattern CTATA.

Hint: You should find two of them.

Once that works, print the index of the first time that pattern is found.

# Exercise 5 - Filtering

Using `/usr/coursehome/dalke/sequences.seq`

- A. How many sequences are in that file?
- B. How many have the pattern CTATA?
- C. How many have more than 1000 bases?
- D. How many have over 50% GC composition?
- E. How many have more than 2000 bases and more than 50% GC composition?

Note: for %GC use float to convert the counts into floats before doing the division for percentage.