Lists and the '`for`' loop

Lists

Lists are an ordered collection of objects

```
>>> data = []
>>> print data
[]
>>> data.append("Hello!")
>>> print data
['Hello!']
>>> data.append(5)
>>> print data
['Hello!', 5]
>>> data.append([9, 8, 7])
>>> print data
['Hello!', 5, [9, 8, 7]]
>>> data.extend([4, 5, 6])
>>> print data
['Hello!', 5, [9, 8, 7], 4, 5, 6]
>>>
```

Make an empty list

“append” == “add to the end”

You can put different objects in the same list

“extend” appends each element of the new list to the old one

Lists and strings are similar

Strings

```
>>> s = "ATCG"
>>> print s[0]
A
>>> print s[-1]
G
>>> print s[2:]
CG
>>> print "C" in s
True
>>> s * 3
'ATCGATCGATCG'
>>> s[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of
range
>>>
```

Lists

```
>>> L = ["adenine", "thymine", "cytosine",
"guanine"]
>>> print L[0]
adenine
>>> print L[-1]
guanine
>>> print L[2:]
['cytosine', 'guanine']
>>> print "cytosine" in L
True
>>> L * 3
['adenine', 'thymine', 'cytosine', 'guanine',
'adenine', 'thymine', 'cytosine', 'guanine',
'adenine', 'thymine', 'cytosine', 'guanine']
>>> L[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
>>>
```

But lists are mutable

Lists can be changed. Strings are immutable.

```
>>> s = "ATCG"
>>> print s
ATCG
>>> s[1] = "U"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
>>> s.reverse()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: 'str' object has no attribute
'reverse'
>>> print s[::-1]
GCTA
>>> print s
ATCG
>>>
```

```
>>> L = ["adenine", "thymine", "cytosine",
"guanine"]
>>> print L
['adenine', 'thymine', 'cytosine', 'guanine']
>>> L[1] = "uracil"
>>> print L
['adenine', 'uracil', 'cytosine', 'guanine']
>>> L.reverse()
>>> print L
['guanine', 'cytosine', 'uracil', 'adenine']
>>> del L[0]
>>> print L
['cytosine', 'uracil', 'adenine']
>>>
```

Lists can hold any object

```
>>> L = ["", 1, "two", 3.0, ["quatro", "fem", [6j], []]]
>>> len(L)
5
>>> print L[-1]
['quatro', 'fem', [6j], []]
>>> len(L[-1])
4
>>> print L[-1][-1]
[]
>>> len(L[-1][-1])
0
>>>
```

A few more methods

```
>>> L = ["thymine", "cytosine", "guanine"]
>>> L.insert(0, "adenine")
>>> print L
['adenine', 'thymine', 'cytosine', 'guanine']
>>> L.insert(2, "uracil")
>>> print L
['adenine', 'thymine', 'uracil', 'cytosine', 'guanine']
>>> print L[:2]
['adenine', 'thymine']
>>> L[:2] = ["A", "T"]
>>> print L
['A', 'T', 'uracil', 'cytosine', 'guanine']
>>> L[:2] = []
>>> print L
['uracil', 'cytosine', 'guanine']
>>> L[:] = ["A", "T", "C", "G"]
>>> print L
['A', 'T', 'C', 'G']
>>>
```

Turn a string into a list

```
>>> s = "AAL532906 aaaatagtcaaatatatcccaattcagtatgcgctgagta"
>>> i = s.find(" ")
>>> print i
9
>>> print s[:i]
AAL532906
>>> print s[i+1:]
aaaatagtcaaatatatcccaattcagtatgcgctgagta
>>>
```

} Complicated

```
>>> fields = s.split()
>>> print fields
['AAL532906', 'aaaatagtcaaatatatcccaattcagtatgcgctgagta']
>>> print fields[0]
AAL532906
>>> print len(fields[1])
40
>>>
```

Easier!

More split examples

```
>>> protein = "ALA PRO ILU CYS"
>>> residues = protein.split()
>>> print residues
['ALA', 'PRO', 'ILU', 'CYS']
>>>
>>> protein = " ALA     PRO    ILU CYS  \n"
>>> print protein.split()
['ALA', 'PRO', 'ILU', 'CYS']


>>> print "HIS-GLU-PHE-ASP".split("-")
['HIS', 'GLU', 'PHE', 'ASP']
>>>
```

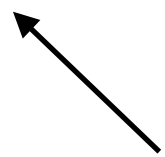
split() uses ‘whitespace’ to find each word

split(c) uses that character to find each word

Turn a list into a string

`join` is the opposite of `split`

```
>>> L1 = ["Asp", "Gly", "Gln", "Pro", "Val"]
>>> print "-".join(L1)
Asp-Gly-Gln-Pro-Val
>>> print "**".join(L1)
Asp**Gly**Gln**Pro**Val
>>> print "\n".join(L1)
Asp
Gly
Gln
Pro
Val
>>>
```



The order is confusing.

- string to join is first
- list to be joined is second

The ‘`for`’ loop

Lets you do something to each element in a list

```
>>> for name in ["Andrew", "Tsanwani", "Arno", "Tebogo"]:
...     print "Hello,", name
...
Hello, Andrew
Hello, Tsanwani
Hello, Arno
Hello, Tebogo
>>>
```

The ‘`for`’ loop

Lets you do something to
each element in a list

```
>>> for name in ["Andrew", "Tsanwani", "Arno", "Tebogo"]:
...     print "Hello,", name
...
Hello, Andrew
Hello, Tsanwani
Hello, Arno
Hello, Tebogo
>>>
```

a new code block

it must be indented

IDLE indents automatically when
it sees a ‘:’ on the previous line

A two line block

All lines in the same code block
must have the same indentation

```
>>> for name in ["Andrew", "Tsanwani", "Arno", "Tebogo"]:
...     print "Hello,", name
...     print "Your name is", len(name), "letters long"
...
Hello, Andrew
Your name is 6 letters long
Hello, Tsanwani
Your name is 8 letters long
Hello, Arno
Your name is 4 letters long
Hello, Tebogo
Your name is 6 letters long
>>>
```

When indentation does not match

```
>>> a = 1
>>>   a = 1
  File "<stdin>", line 1
    a = 1
    ^
SyntaxError: invalid syntax
>>> for name in ["Andrew", "Tsanwani", "Arno", "Tebogo"]:
...     print "Hello,", name
...         print "Your name is", len(name), "letters long"
  File "<stdin>", line 3
    print "Your name is", len(name), "letters long"
    ^
SyntaxError: invalid syntax
>>> for name in ["Andrew", "Tsanwani", "Arno", "Tebogo"]:
...     print "Hello,", name
...   print "Your name is", len(name), "letters long"
  File "<stdin>", line 3
    print "Your name is", len(name), "letters long"
                                                   ^
IndentationError: unindent does not match any outer indentation level
>>>
```

‘for’ works on strings

A string is similar to a list of letters

```
>>> seq = "ATGCATGTCGC"
>>> for letter in seq:
...   print "Base:", letter
...
Base: A
Base: T
Base: G
Base: C
Base: A
Base: T
Base: G
Base: T
Base: C
Base: G
Base: C
>>>
```

Numbering bases

```
>>> seq = "ATGCATGTCGC"
>>> n = 0
>>> for letter in seq:
...     print "base", n, "is", letter
...     n = n + 1
...
base 0 is A
base 1 is T
base 2 is G
base 3 is C
base 4 is A
base 5 is T
base 6 is G
base 7 is T
base 8 is C
base 9 is G
base 10 is C
>>>
>>> print "The sequence has", n, "bases"
The sequence has 11 bases
>>>
```

The `range` function

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(8)
[0, 1, 2, 3, 4, 5, 6, 7]
>>> range(2, 8)
[2, 3, 4, 5, 6, 7]
>>> range(0, 8, 1)
[0, 1, 2, 3, 4, 5, 6, 7]
>>> range(0, 8, 2)
[0, 2, 4, 6]
>>> range(0, 8, 3)
[0, 3, 6]
>>> range(0, 8, 4)
[0, 4]
>>> range(0, 8, -1)
[]
>>> range(8, 0, -1)
[8, 7, 6, 5, 4, 3, 2, 1]
>>>
```

```
>>> help(range)
Help on built-in function range:
```

range(...)
range([start,] stop[, step]) -> list of integers

Return a list containing an arithmetic progression of integers.
range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to 0.
When step is given, it specifies the increment (or decrement).
For example, range(4) returns [0, 1, 2, 3]. The end point is omitted!
These are exactly the valid indices for a list of 4 elements.

Do something ‘N’ times

```
>>> for i in range(3):
...     print "If I tell you three times it must be true."
...
If I tell you three times it must be true.
If I tell you three times it must be true.
If I tell you three times it must be true.
>>>
>>> for i in range(4):
...     print i, "squared is", i*i, "and cubed is", i*i*i
...
0 squared is 0 and cubed is 0
1 squared is 1 and cubed is 1
2 squared is 4 and cubed is 8
3 squared is 9 and cubed is 27
>>>
```

Exercise 1

Write a program that asks for a sequence (use the raw_input function) then prints it 10 times. Include the loop count in the output

```
Enter a sequence: TACG
0 TACG
1 TACG
2 TACG
3 TACG
4 TACG
5 TACG
6 TACG
7 TACG
8 TACG
9 TACG
```

Exercise 2

Write a program that asks for a sequence then numbers each base, one base per line.

```
Enter a sequence: GTTCAG
base 0 is G
base 1 is T
base 2 is T
base 3 is C
base 4 is A
base 5 is G
```

Can you modify your program to start with base 1 instead of 0?

Exercise 3

Here is a Python list of restriction site patterns

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

Write a program that prints each pattern.

```
GAATTC is a restriction site
GGATCC is a restriction site
AAGCTT is a restriction site
```

Note: there is no input for this exercise,
just print the items in the list.

Exercise 4

Modify the program from Exercise 3 to ask for a sequence then say whether each restriction site is or is not present

```
Enter a sequence: AGAATTC
GAATTC is in the sequence: True
GGATCC is in the sequence: False
AAGCTT is in the sequence: False
```

Hint from yesterday's lecture on strings - use '`in`':

```
>>> print "AT" in "GATTACA"
True
>>> print "GG" in "GATTACA"
False
>>>
```