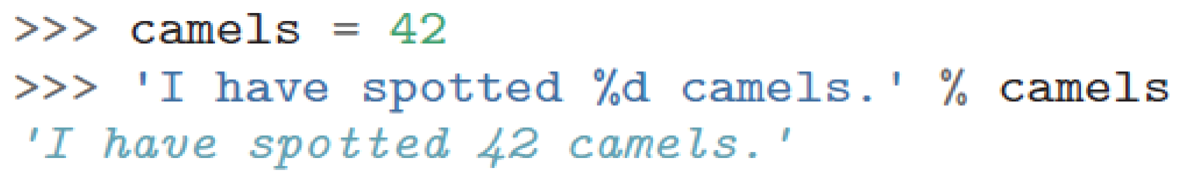
Format operator

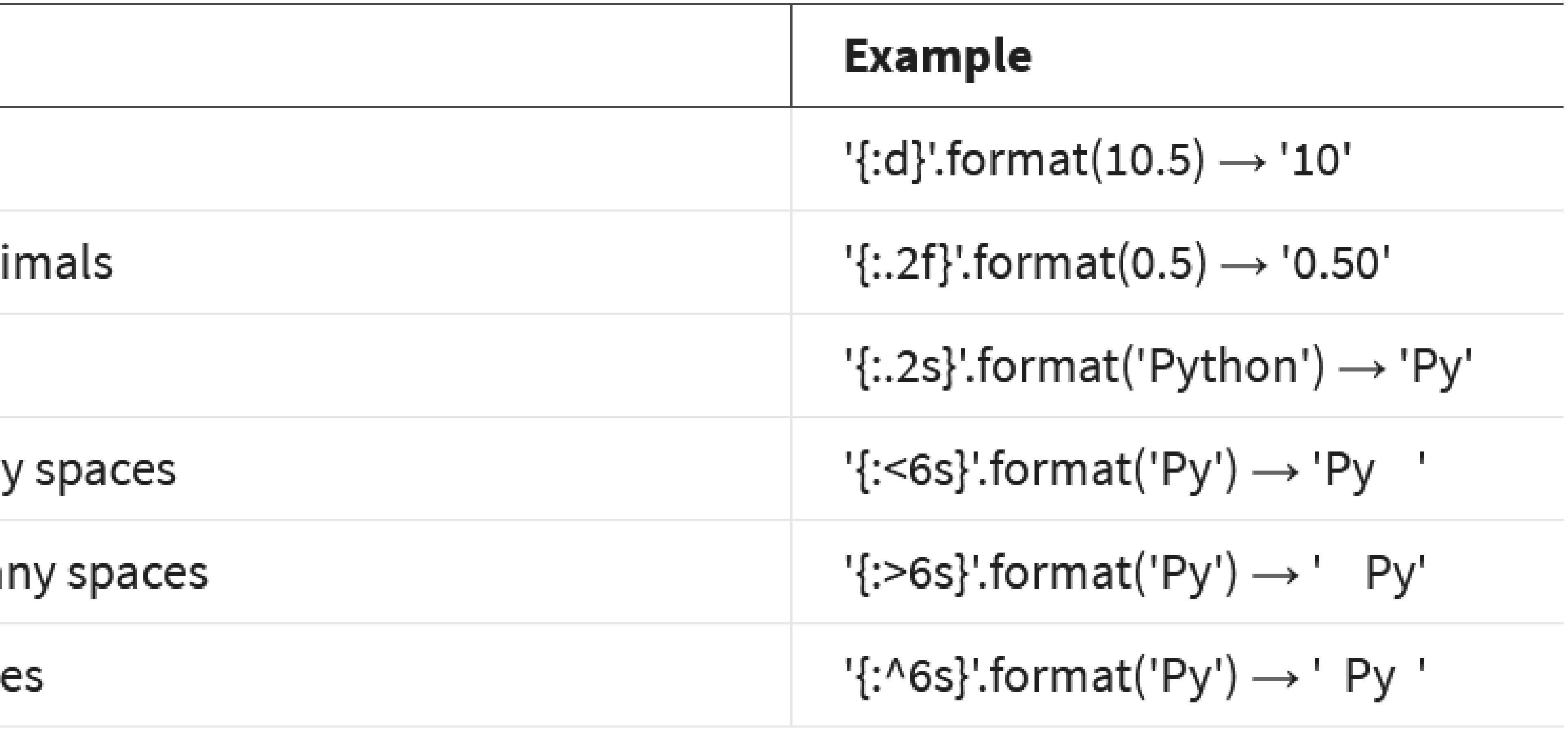
- The format operator, % allows us to construct strings, replacing parts of the strings with the data stored in variables

- When applied to integers, % is the modulus operator. But when the first operand is a string, % is the format operator



```
>>> camels = 42
>>> 'I have spotted %d camels.' % camels
'I have spotted 42 camels.'
```

Format expressions



	Example
	'{:d}'.format(10.5) → '10'
imals	'{:.2f}'.format(0.5) → '0.50'
	'{:.2s}'.format('Python') → 'Py'
y spaces	'{:<6s}'.format('Py') → 'Py '
ny spaces	'{:>6s}'.format('Py') → ' Py'
es	'{:^6s}'.format('Py') → ' Py '

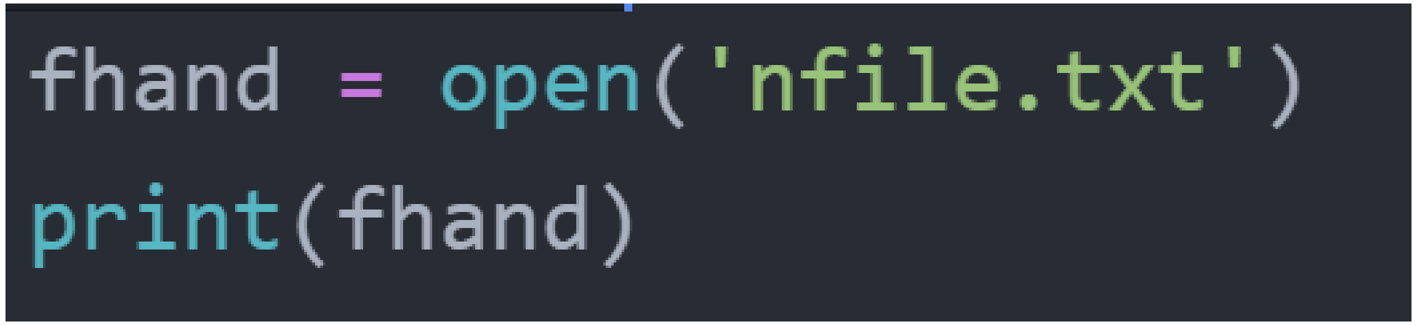
Opening files

- When we want to read or write a file

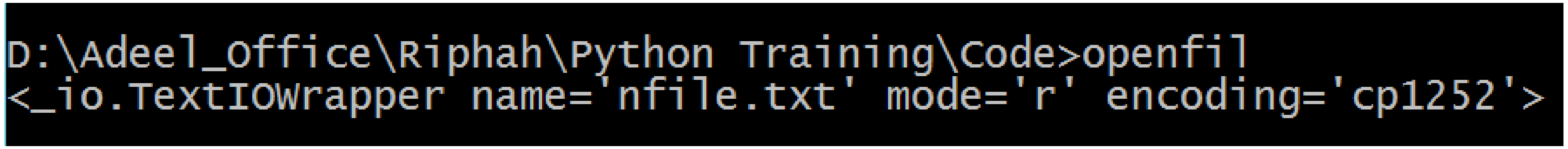
- Opening the file communicates with your operating system, which knows where the data for each file is stored

- When you open a file, you are asking the operating system to find the file by name and make sure the file exists

Opening files



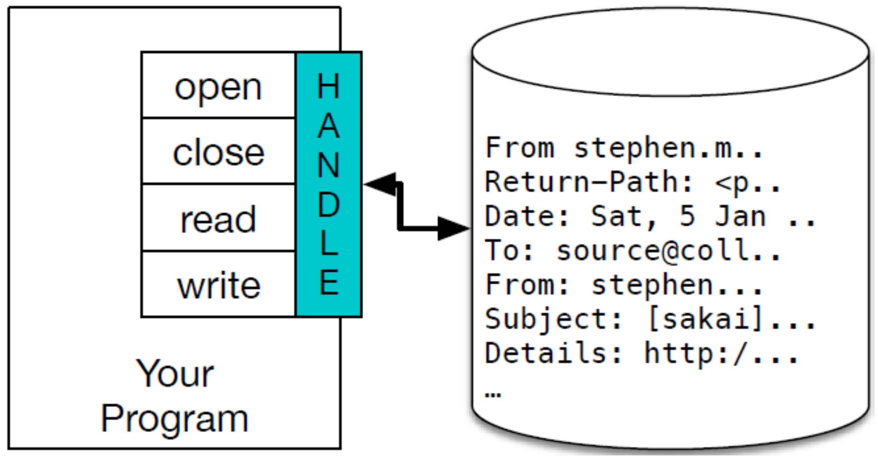
```python
fhand = open('nfile.txt')
print(fhand)
```



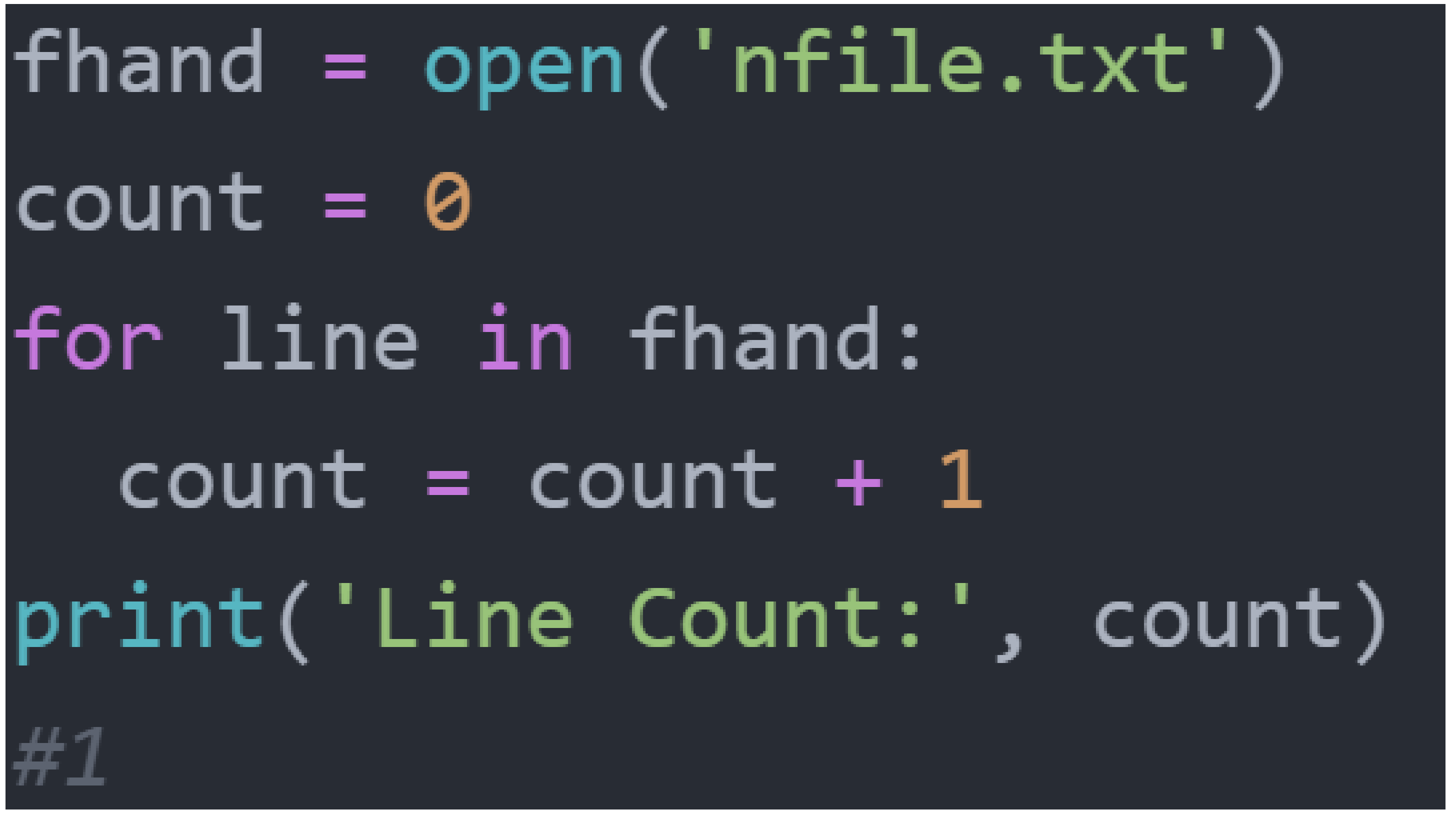
```
D:\Adeel_Office\Riphah\Python Training\Code>openfil
<_io.TextIOWrapper name='nfile.txt' mode='r' encoding='cp1252'>
```

File handle

- If the open is successful, the operating system returns us a file handle
- The file handle is not the actual data contained in the file, but instead it is a "handle" that we can use to read the data

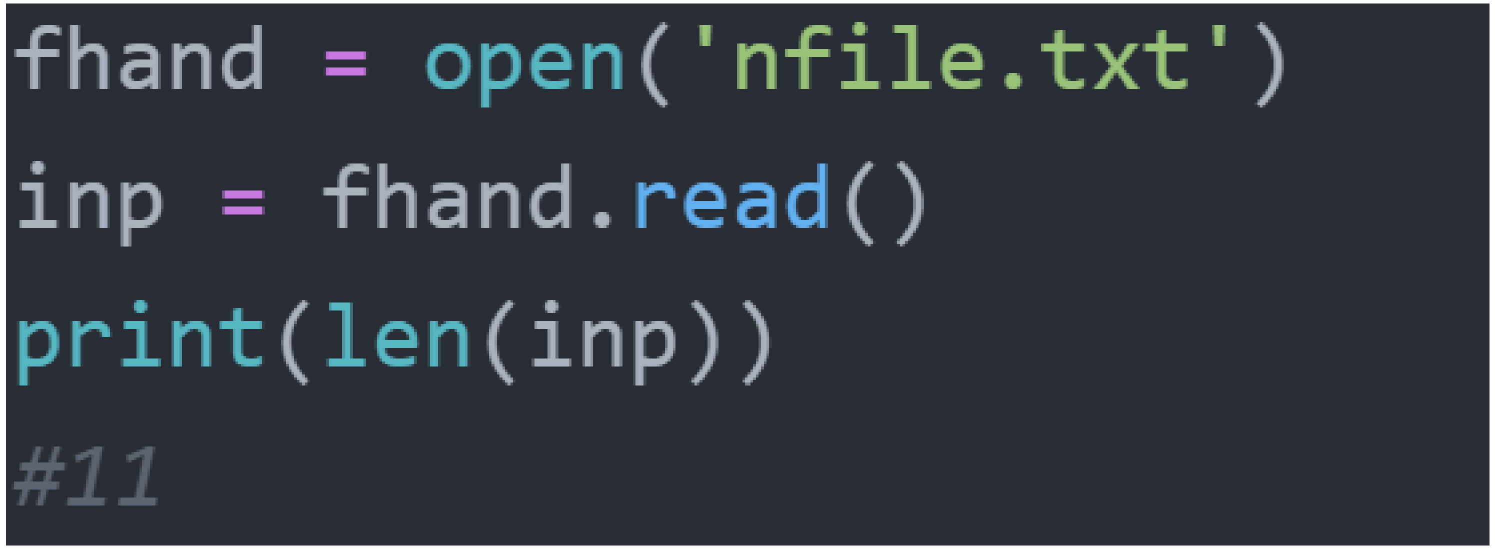


Reading files



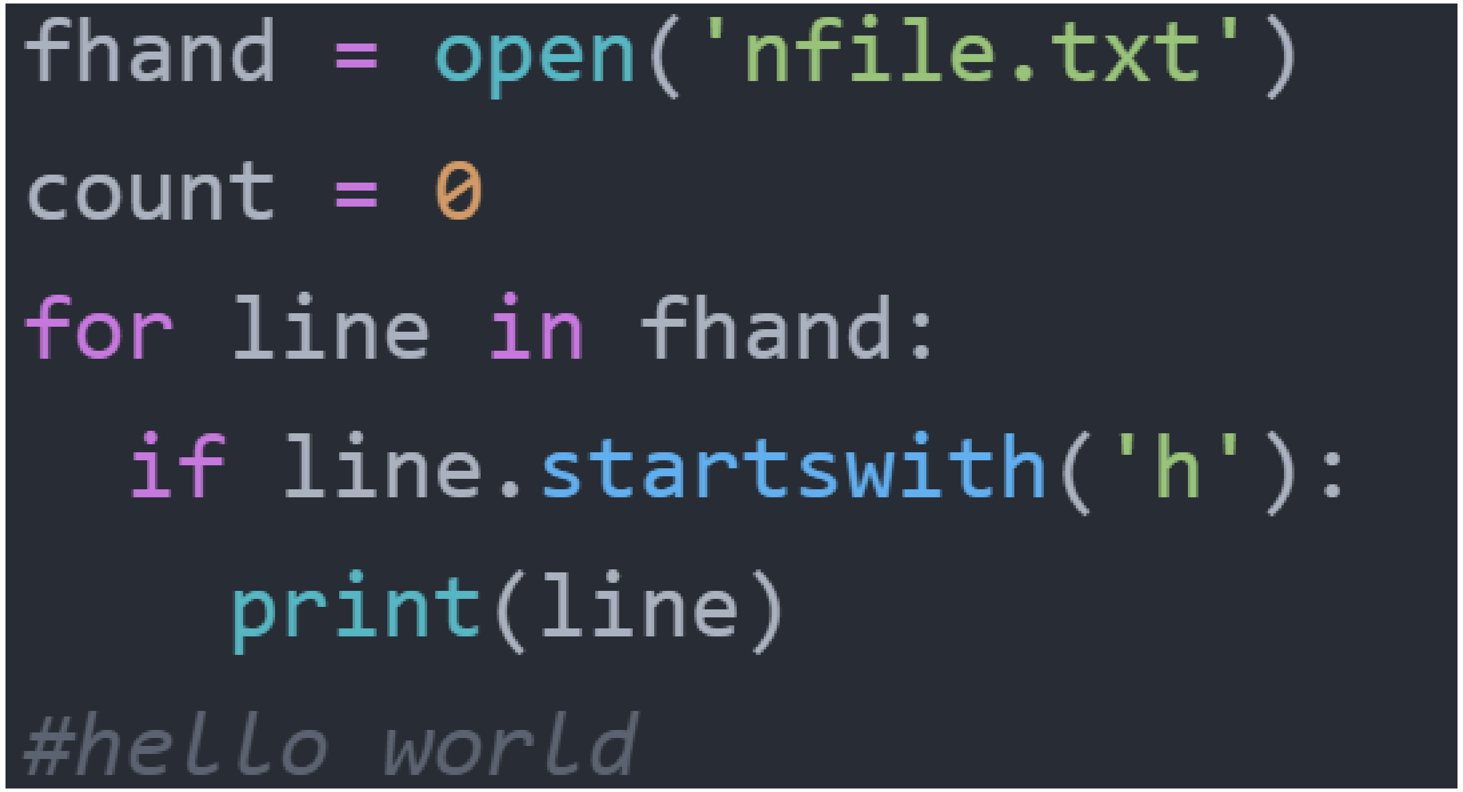
```python
fhand = open('nfile.txt')
count = 0
for line in fhand:
  count = count + 1
print('Line Count:', count)
#1
```

Reading files



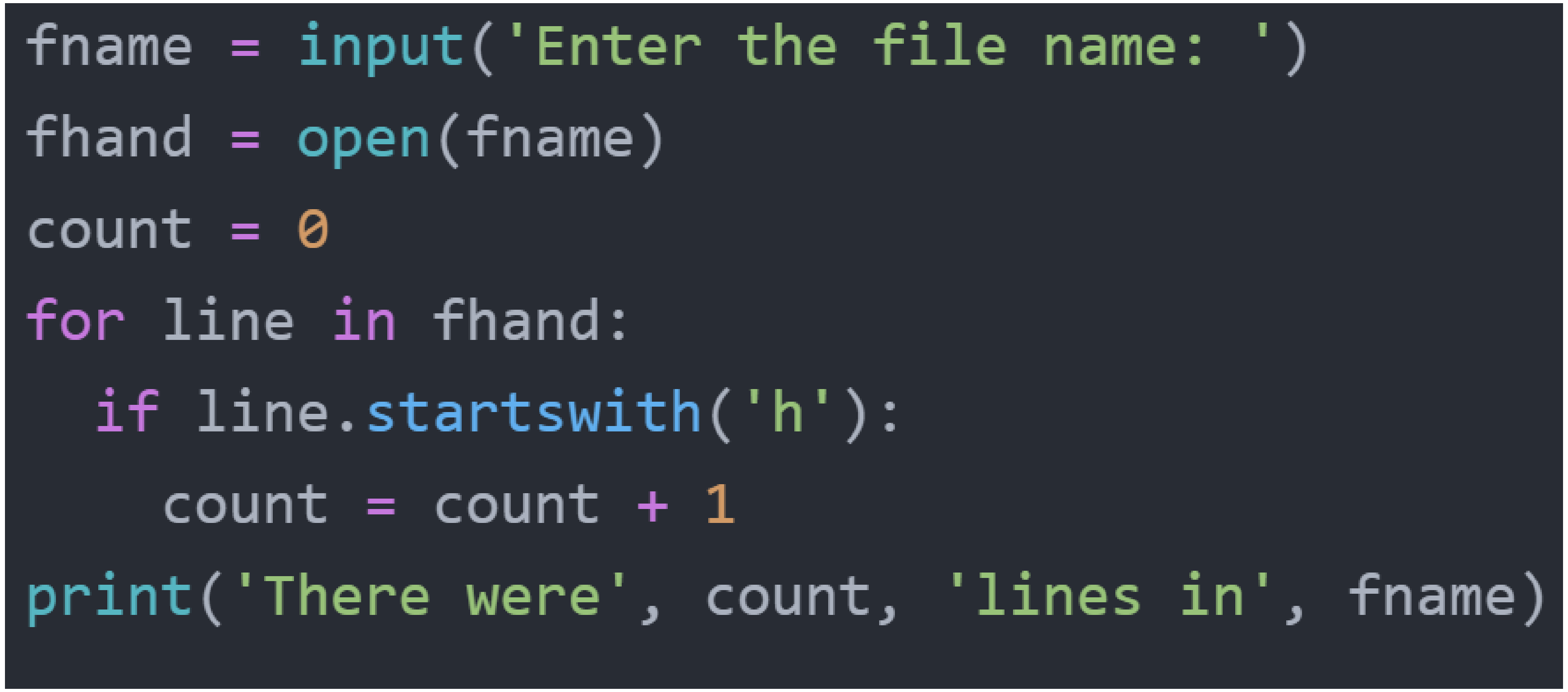
```
fhand = open('nfile.txt')
inp = fhand.read()
print(len(inp))
#11
```

Search files



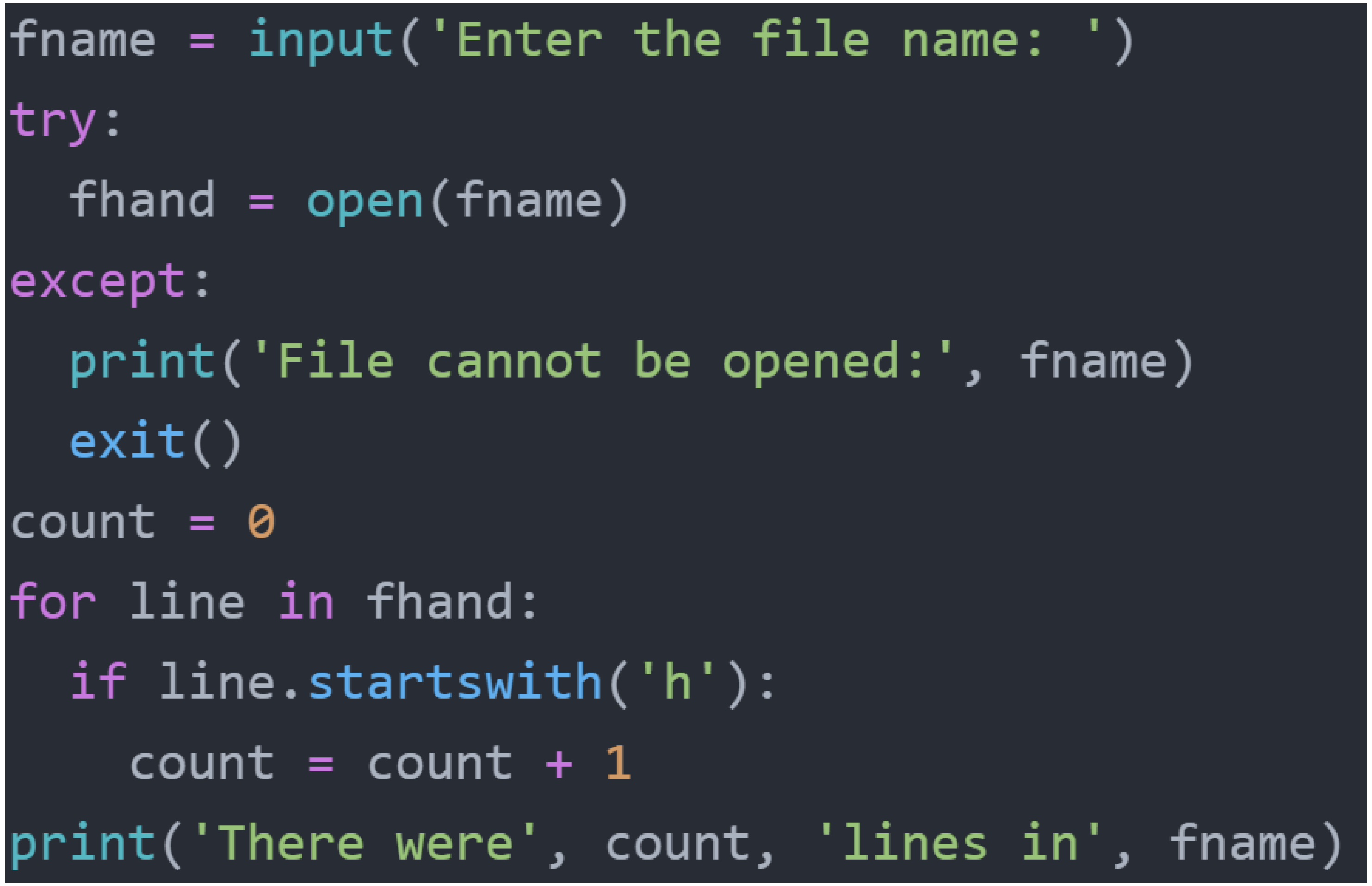
```python
fhand = open('nfile.txt')
count = 0
for line in fhand:
  if line.startswith('h'):
    print(line)
#hello world
```

Letting the user choose the file name



```python
fname = input('Enter the file name: ')
fhand = open(fname)
count = 0
for line in fhand:
  if line.startswith('h'):
    count = count + 1
print('There were', count, 'lines in', fname)
```

Try-Except



```
fname = input('Enter the file name: ')
try:
  fhand = open(fname)
except:
  print('File cannot be opened:', fname)
  exit()
count = 0
for line in fhand:
  if line.startswith('h'):
    count = count + 1
print('There were', count, 'lines in', fname)
```

Writing files

- To write a file, you have to open it with mode “w” as a second parameter:

```
>>> fout = open('output.txt', 'w')
>>> print(fout)
<_io.TextIOWrapper name='output.txt' mode='w' encoding='cp1252'>
```

Writing files

- If the file already exists, opening it in write mode clears out the old data and starts fresh, so be careful! If the file doesn't exist, a new one is created

```
>>> line1 = "This here's the wattle,\n"
>>> fout.write(line1)
24
```

Activity - 13

- **Write a program to read through a file and print the contents of the file (line by line) all in upper case. Executing the program will look as follows:**

```
python shout.py
Enter a file name: mbox-short.txt
FROM STEPHEN.MARQUARD@UCT.AC.ZA SAT JAN 5 09:14:16 2008
RETURN-PATH: <POSTMASTER@COLLAB.SAKAIPROJECT.ORG>
RECEIVED: FROM MURDER (MAIL.UMICH.EDU [141.211.14.90])
BY FRANKENSTEIN.MAIL.UMICH.EDU (CYRUS V2.3.8) WITH LMTPA;
SAT, 05 JAN 2008 09:14:16 -0500
```

Note: You need to first create this file in the same directory

Exam

- Python Training-Exam
- Total Questions: 17
- Time: 2 Hrs.

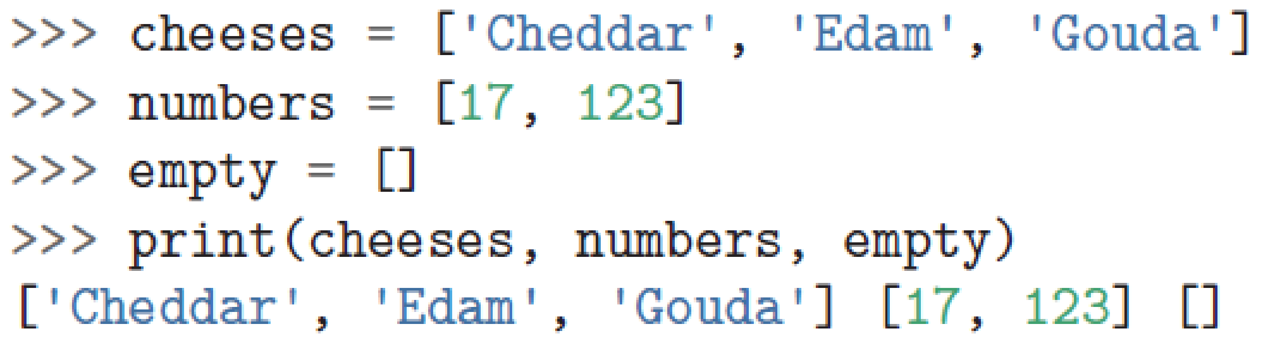
Note: Please don’t use internet. Use BRAIN!!!

Lists

- Like a string, a list is a sequence of values. In a string, the values are characters; in a list, they can be any type
- There are several ways to create a new list; the simplest is to enclose the elements in square brackets (“[" and “]”):

```
[10, 20, 30, 40]
['crunchy frog', 'ram bladder', 'lark vomit']
```

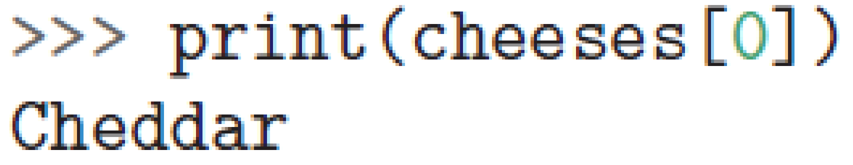
Lists



```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [17, 123]
>>> empty = []
>>> print(cheeses, numbers, empty)
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

Lists are mutable

- The syntax for accessing the elements of a list is the same as for accessing the characters of a string: the bracket operator
- The expression inside the brackets specifies the index. Remember that the indices start at 0:



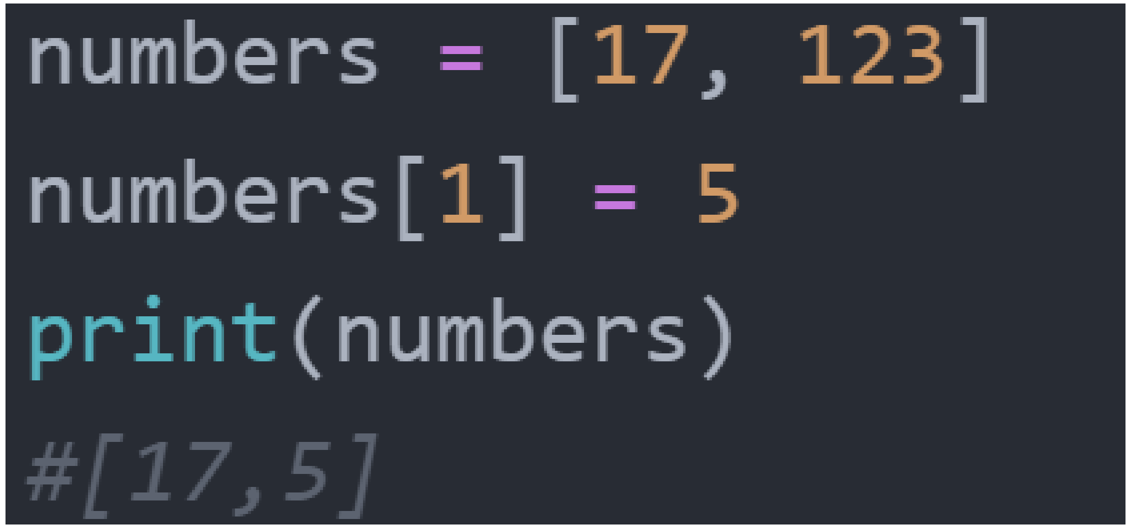
```
>>> print(cheeses[0])
Cheddar
```

Lists are mutable

- Unlike strings, lists are mutable because you can change the order of items in a list or reassign an item in a list

- When the bracket operator appears on the left side of an assignment, it identifies the element of the list that will be assigned

Lists are mutable



```
numbers = [17, 123]
numbers[1] = 5
print(numbers)
#[17,5]
```

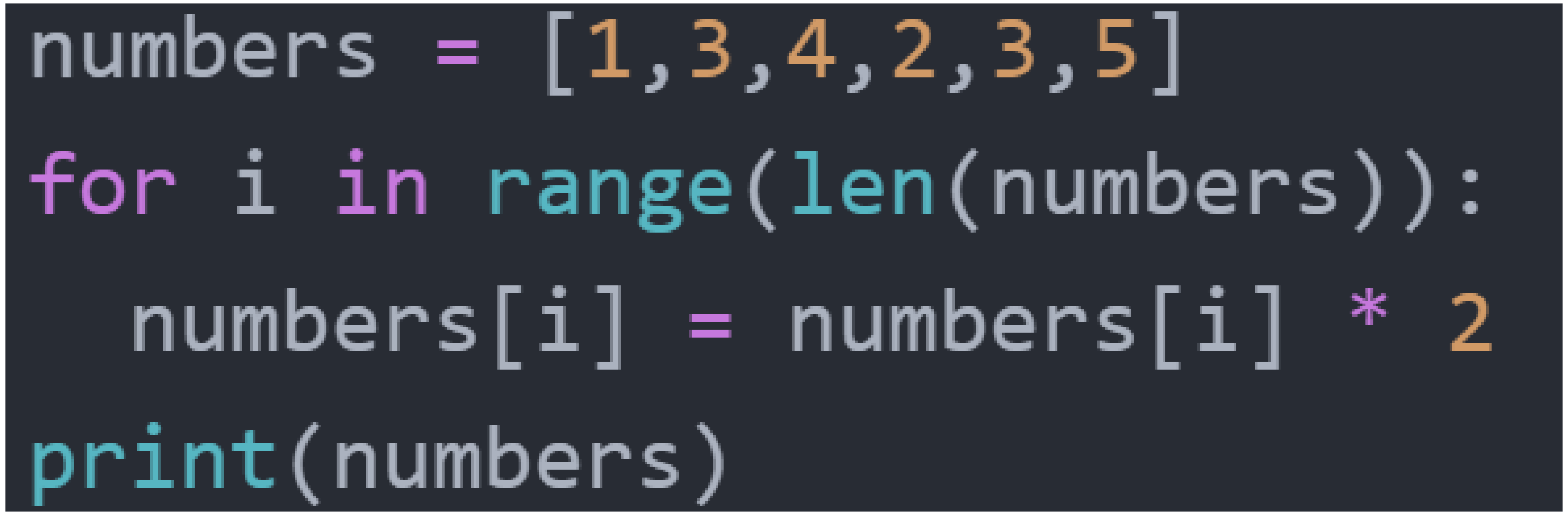
Traversing a list

```
numbers = [2,4,6,8,10,12]
for i in numbers:
    print(i)
```

Output:

```
2
4
6
8
10
12
```

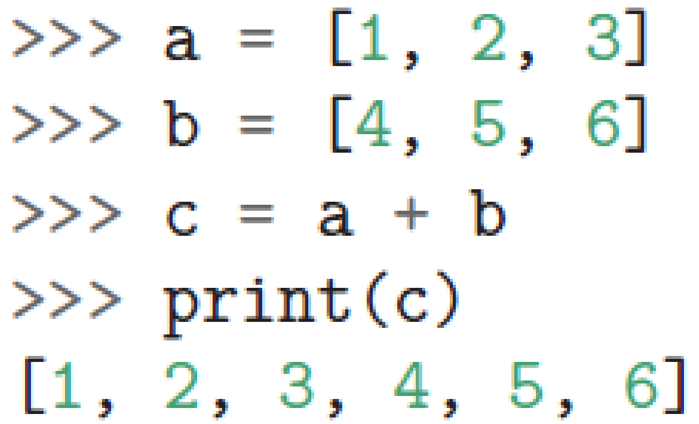
Traversing a list



```
numbers = [1,3,4,2,3,5]
for i in range(len(numbers)):
  numbers[i] = numbers[i] * 2
print(numbers)
```

List operations

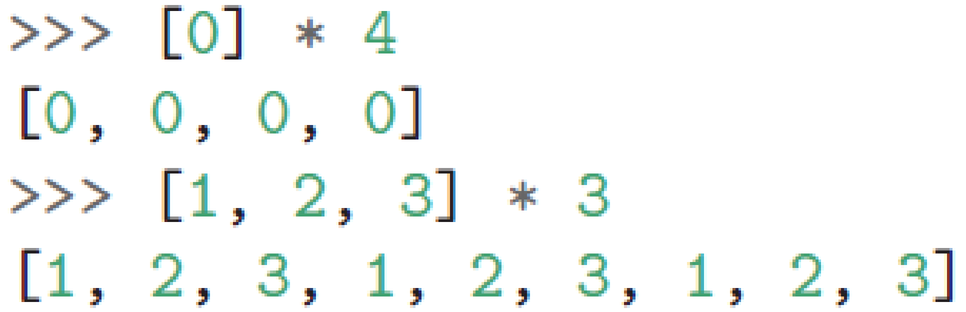
- The + operator concatenates lists:



```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
```

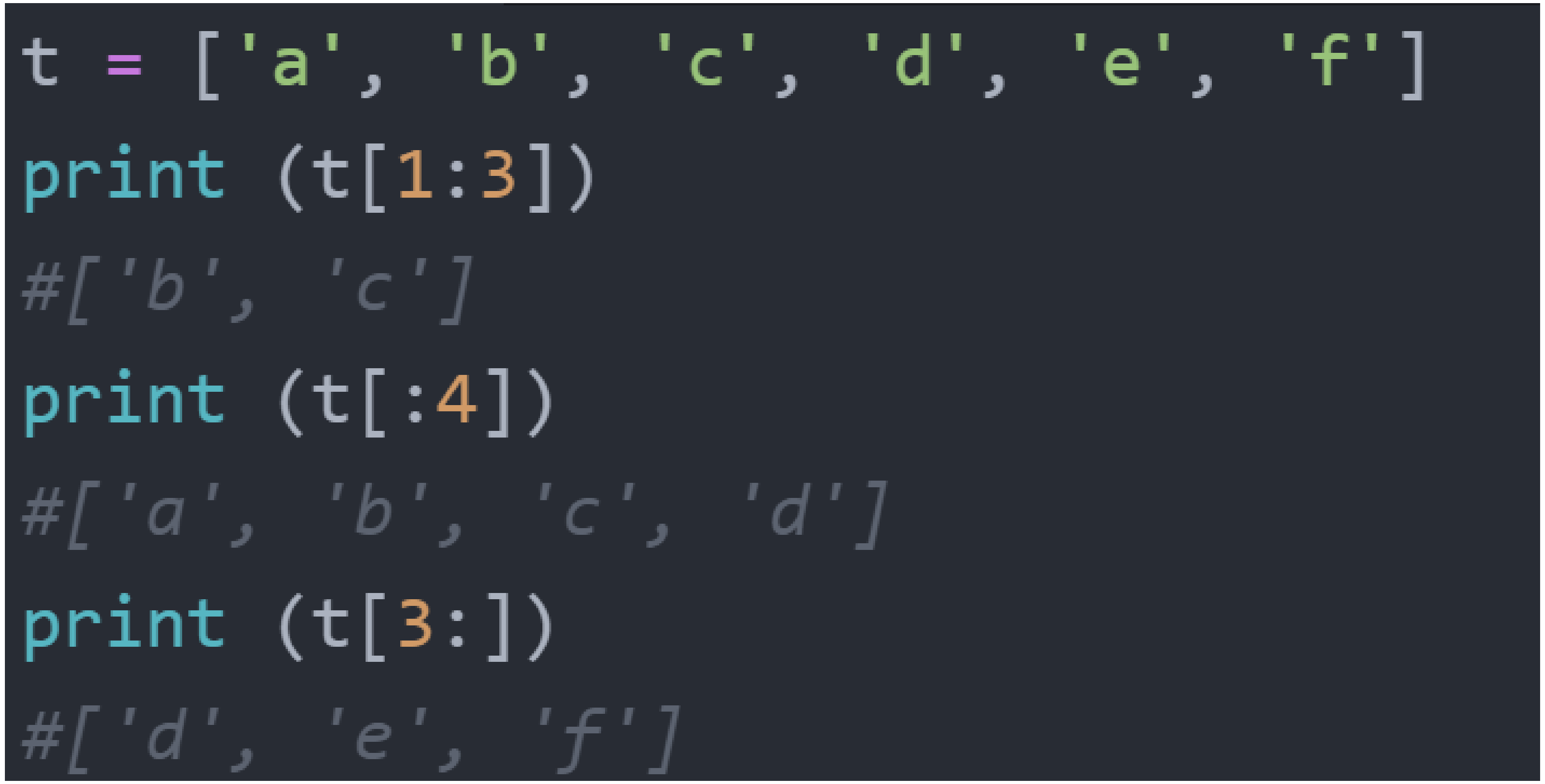
List operations

- Similarly, the * operator repeats a list a given number of times:



```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

List slices



```
t = ['a', 'b', 'c', 'd', 'e', 'f']
print (t[1:3])
#['b', 'c']
print (t[:4])
#['a', 'b', 'c', 'd']
print (t[3:])
#['d', 'e', 'f']
```

List methods

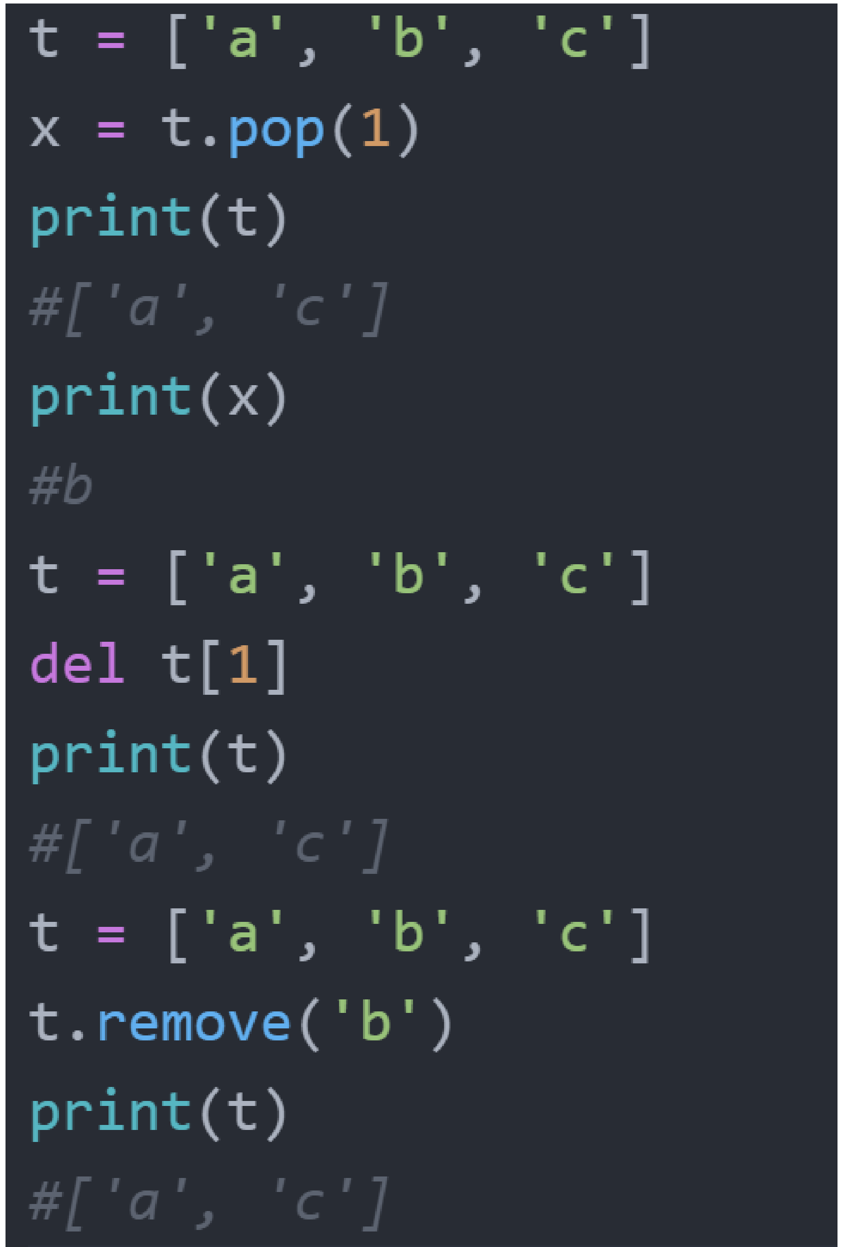
```
c']

'e']
```

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print(t)
['a', 'b', 'c', 'd', 'e']
```

Deleting elements

- There are several ways to delete elements from a list
  - If you know the index of the element you want, you can use **pop**
  - If you don’t need the removed value, you can use the **del** operator
  - If you know the element you want to remove (but not the index), you can use **remove**

Deleting elements



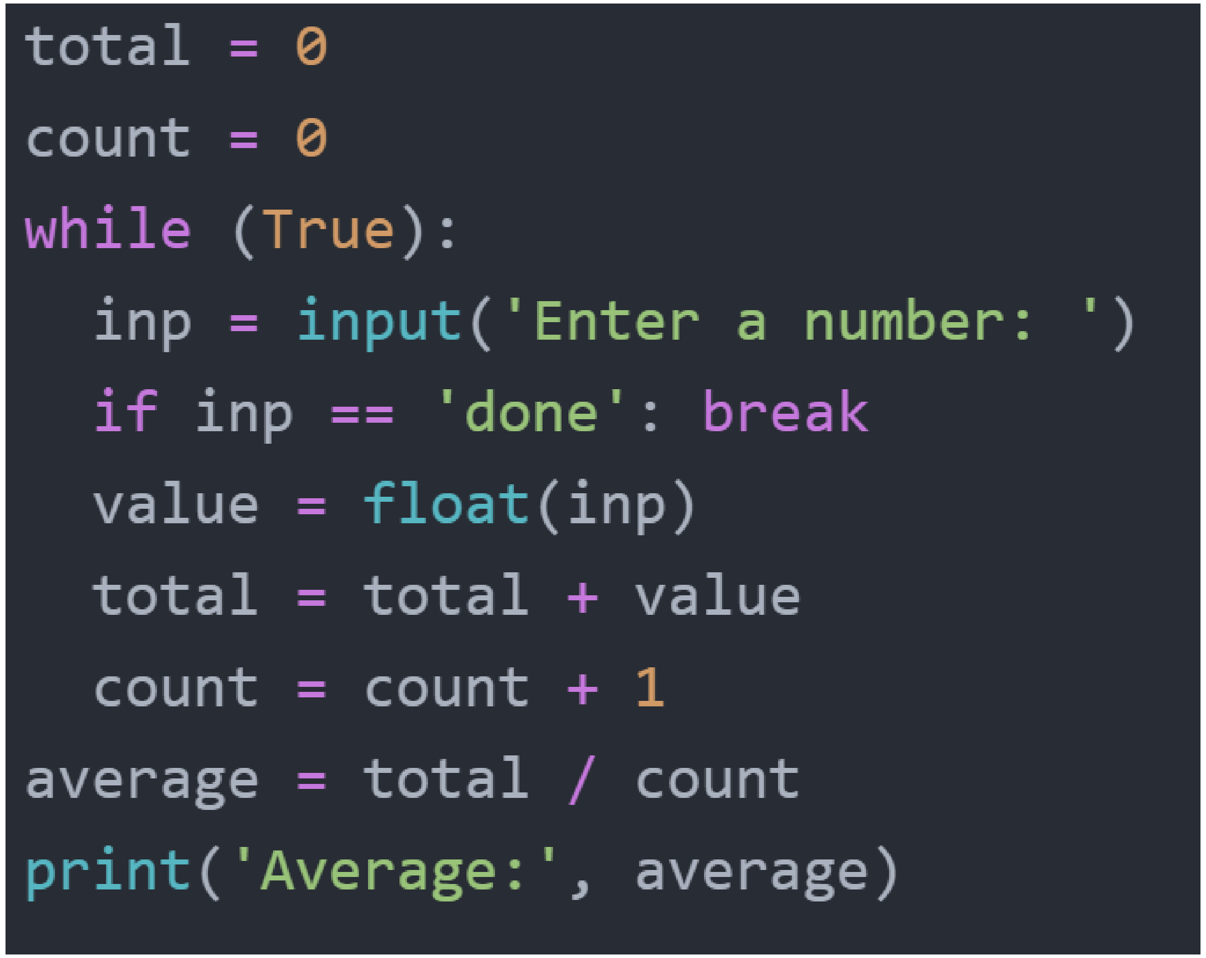
```python
t = ['a', 'b', 'c']
x = t.pop(1)
print(t)
#['a', 'c']
print(x)
#b
t = ['a', 'b', 'c']
del t[1]
print(t)
#['a', 'c']
t = ['a', 'b', 'c']
t.remove('b')
print(t)
#['a', 'c']
```

Lists and functions

- There are a number of built-in functions that can be used on lists that allow you to quickly look through a list without writing your own loops

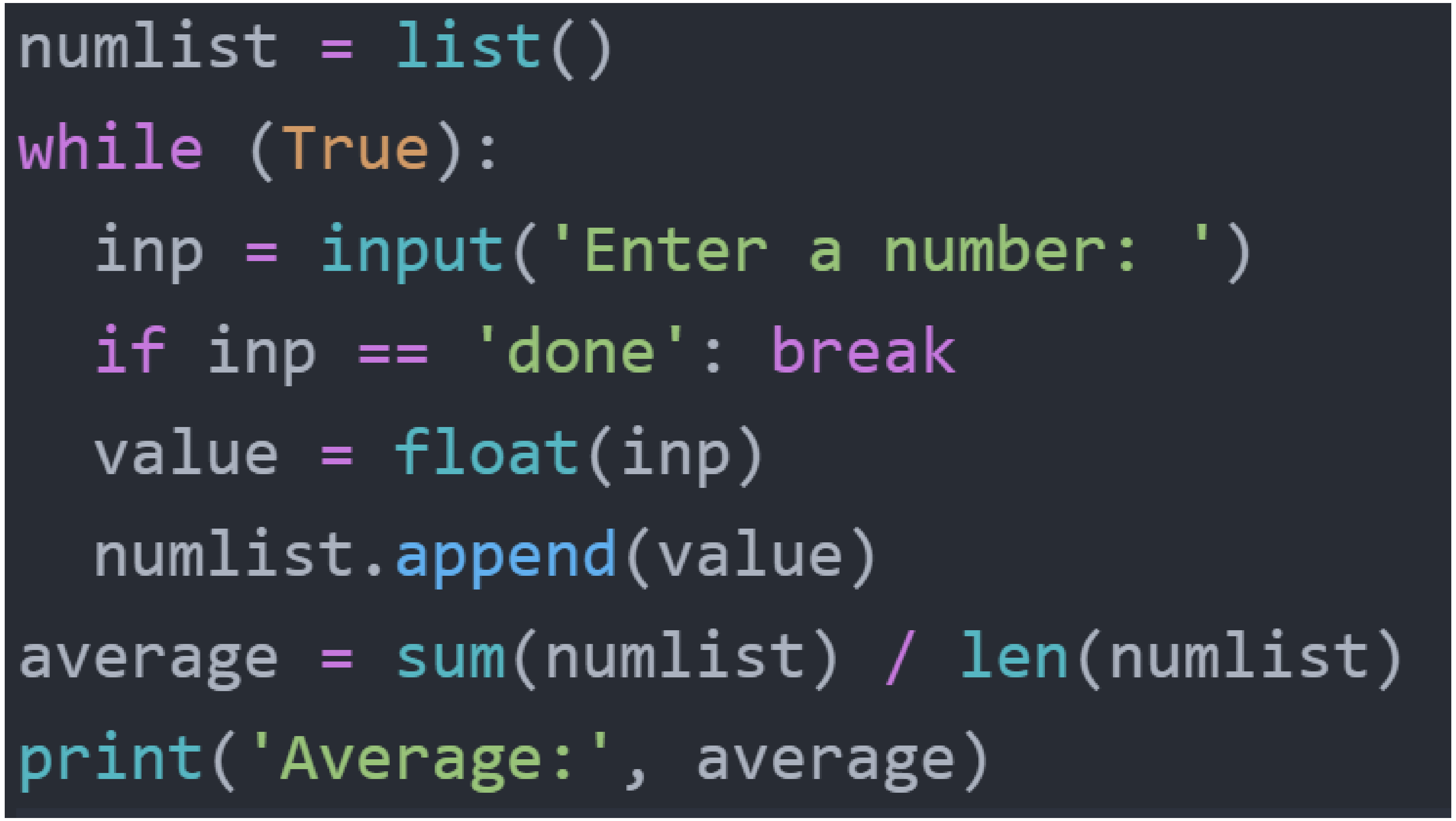
```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25
```

Calculating average without list



```
total = 0
count = 0
while (True):
  inp = input('Enter a number: ')
  if inp == 'done': break
  value = float(inp)
  total = total + value
  count = count + 1
average = total / count
print('Average:', average)
```

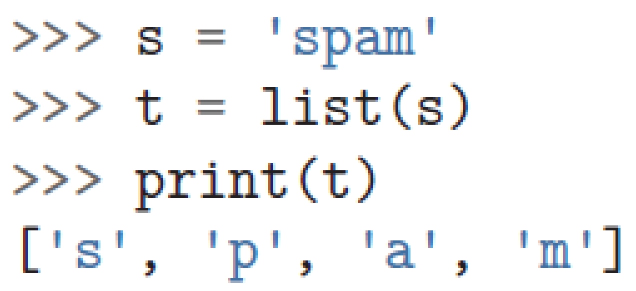
Calculating average with list



```python
numlist = list()
while (True):
  inp = input('Enter a number: ')
  if inp == 'done': break
  value = float(inp)
  numlist.append(value)
average = sum(numlist) / len(numlist)
print('Average:', average)
```

Lists and strings

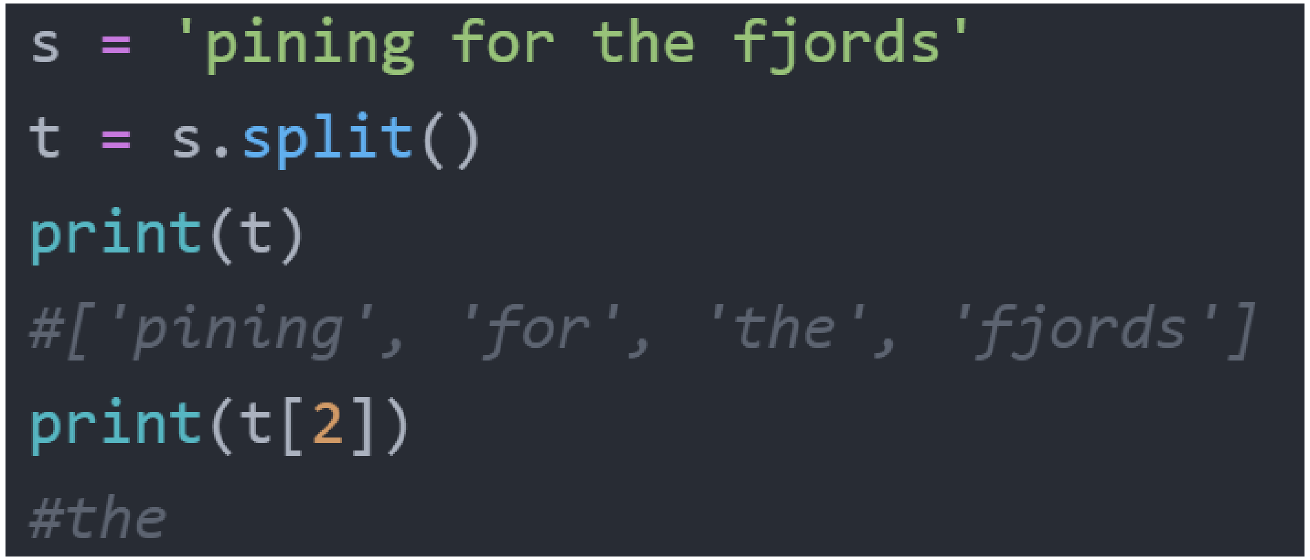
- A string is a sequence of characters and a list is a sequence of values, but a list of characters is not the same as a string
- To convert from a string to a list of characters, you can use list:



```
>>> s = 'spam'
>>> t = list(s)
>>> print(t)
['s', 'p', 'a', 'm']
```

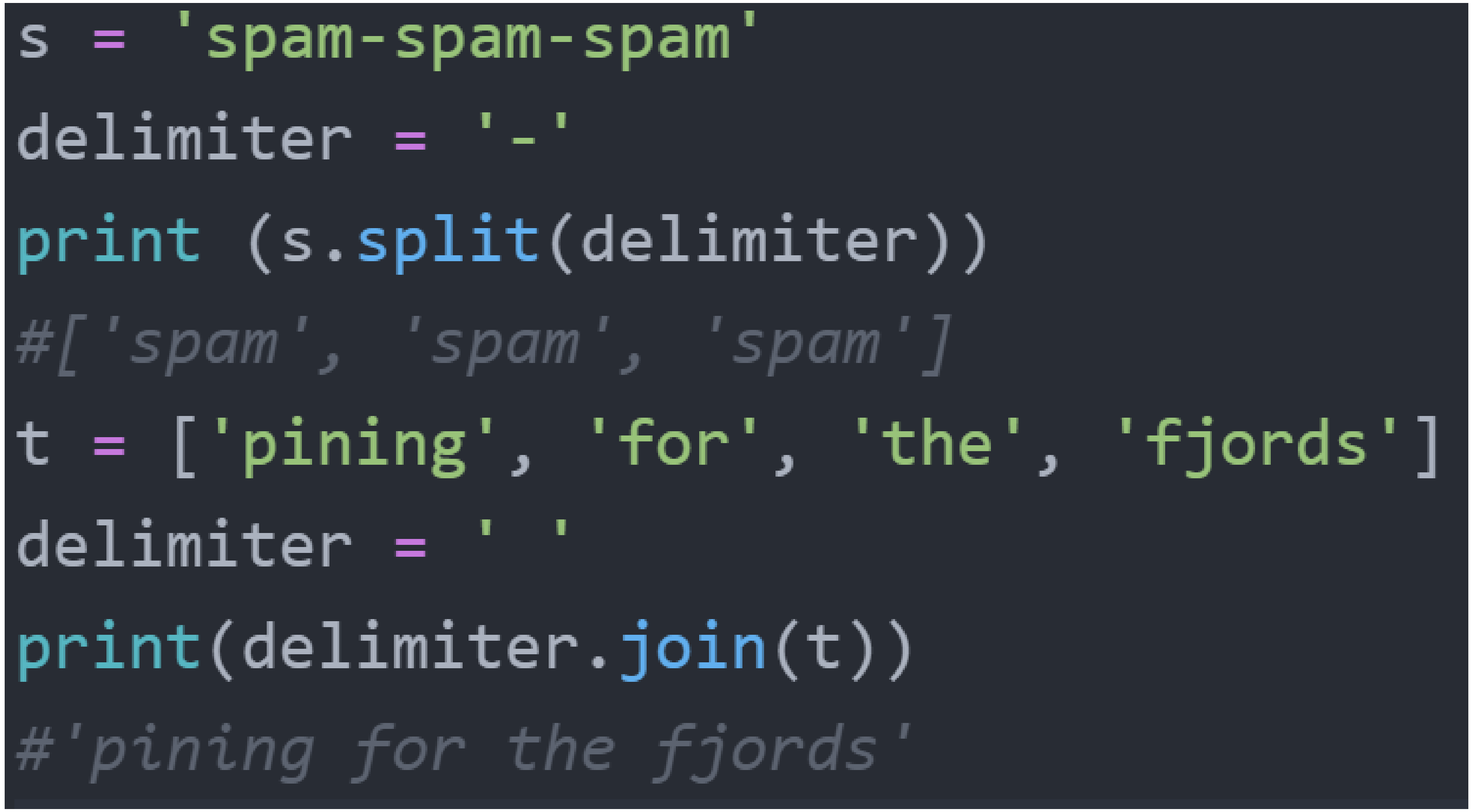
Lists and strings

- The list function breaks a string into individual letters. If you want to break a string into words, you can use the split method:



```
s = 'pining for the fjords'
t = s.split()
print(t)
#['pining', 'for', 'the', 'fjords']
print(t[2])
#the
```

Delimiters



```python
s = 'spam-spam-spam'
delimiter = '-'
print (s.split(delimiter))
#['spam', 'spam', 'spam']
t = ['pining', 'for', 'the', 'fjords']
delimiter = ' '
print(delimiter.join(t))
#'pining for the fjords'
```