Practical Computing for Scientists

Armin Sobhani
CSCI 2000U
UOIT – Fall 2015

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Python

Input and Output

by Greg Wilson

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Been using print to see what programs are doing

Been using print to see what programs are doing

How to save data to files?

Been using print to see what programs are doing

How to save data to files?

And read data from them?

Been using print to see what programs are doing

How to save data to files?

And read data from them?

Python's solution looks very much like C's

Been using print to see what programs are doing

How to save data to files?

And read data from them?

Python's solution looks very much like C's

- A file is a sequence of bytes

Been using print to see what programs are doing

How to save data to files?

And read data from them?

Python's solution looks very much like C's

- A file is a sequence of bytes
- But it's often more useful to treat it as a sequence of lines

Sample data file

Three things are certain:
Death, taxes, and lost data.
Guess which has occurred.

Errors have occurred.
We won't tell you where or why.
Lazy programmers.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

A crash reduces
your expensive computer
to a simple stone.

How many characters in a file?

How many ~~characters~~ in a file?

bytes

How many ~~characters~~ in a file?

bytes

Assume each character is one byte for now

How many ~~characters~~ in a file?

bytes

Assume each character is one byte for now

Revisit later

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

Create a file object

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

File to connect to

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

To read

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

Now holds file object

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

Read entire content of file into a string

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

Now has a copy of all the bytes that were in the file

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

Disconnect from the file

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

Disconnect from the file

Not strictly necessary in small programs, but good practice

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

Report how many characters were read

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

Report how many ~~characters~~ were read

bytes

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
```

How many characters in a file?

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
print(len(data))
293
```

If the file might be large, better to read in chunks

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

Read (at most) 64 bytes

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

Read (at most) 64 bytes

Or the empty string

if there is no more data

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

Keep looping as long as the last read returned some data

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

Do something with the data

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

(Try to) reload

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

Should be 0 (or the loop would still be running)

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

```
64
64
64
64
37
0
```

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

```
64
64
64
64
37
0
```

Don't do this unless

If the file might be large, better to read in chunks

```
reader = open('haiku.txt', 'r')
data = reader.read(64)
while data != '':
    print(len(data))
    data = reader.read(64)
print(len(data))
reader.close()
```

```
64
64
64
64
37
0
```

Don't do this unless the file really might be very large (or infinite)

More common to read one line at a time

More common to read one line at a time

```
reader = open('haiku.txt', 'r')
line = reader.readline()
total = 0
count = 0
while line != '':
    count += 1
    total += len(line)
    line = reader.readline()
reader.close()
print('average', total / count)
```

More common to read one line at a time

```
reader = open('haiku.txt', 'r')
line = reader.readline()
total = 0
count = 0
while line != '':
    count += 1
    total += len(line)
    line = reader.readline()
reader.close()
print('average', total / count)
```

Read a single line

More common to read one line at a time

```
reader = open('haiku.txt', 'r')
line = reader.readline()
total = 0
count = 0
while line != '':
    count += 1
    total += len(line)
    line = reader.readline()
reader.close()
print('average', total / count)
```

Keep looping until no more lines in file

More common to read one line at a time

```
reader = open('haiku.txt', 'r')
line = reader.readline()
total = 0
count = 0
while line != '':
    count += 1
    total += len(line)
    line = reader.readline()
reader.close()
print('average', total / count)
```

(Try to) reload

More common to read one line at a time

```
reader = open('haiku.txt', 'r')
line = reader.readline()
total = 0
count = 0
while line != '':
    count += 1
    total += len(line)
    line = reader.readline()
reader.close()
print('average', total / count)
average 19.53333333
```

Often more convenient to read all lines at once

Often more convenient to read all lines at once

```
reader = open('haiku.txt', 'r')
contents = reader.readlines()
reader.close()
total = 0
count = 0
for line in contents:
    count += 1
    total += len(line)
print('average', total / count)
```

Often more convenient to read all lines at once

```
reader = open('haiku.txt', 'r')
contents = reader.readlines()
reader.close()
total = 0
count = 0
for line in contents:
    count += 1
    total += len(line)
print('average', total / count)
```

All lines in file
as list of strings

Often more convenient to read all lines at once

```
reader = open('haiku.txt', 'r')
contents = reader.readlines()
reader.close()
total = 0
count = 0
for line in contents:
    count += 1
    total += len(line)
print('average', total / count)
```

Loop over lines with **for**

Often more convenient to read all lines at once

```
reader = open('haiku.txt', 'r')
contents = reader.readlines()
reader.close()
total = 0
count = 0
for line in contents:
    count += 1
    total += len(line)
print('average', total / count)
average 19.53333333
```

"Read lines as list" + "loop over list" is common idiom

"Read lines as list" + "loop over list" is common idiom

So Python provides "loop over lines in file"

"Read lines as list" + "loop over list" is common idiom

So Python provides "loop over lines in file"

```
reader = open('haiku.txt', 'r')
total = 0
count = 0
for line in reader:
    count += 1
    total += len(line)
reader.close()
print('average', total / count)
```

"Read lines as list" + "loop over list" is common idiom

So Python provides "loop over lines in file"

```
reader = open('haiku.txt', 'r')
total = 0
count = 0
for line in reader:
    count += 1
    total += len(line)
reader.close()
print('average', total / count)
```

Assign lines of text in file to loop variable one by one

"Read lines as list" + "loop over list" is common idiom

So Python provides "loop over lines in file"

```
reader = open('haiku.txt', 'r')
total = 0
count = 0
for line in reader:
    count += 1
    total += len(line)
reader.close()
print('average', total / count)
average 19.53333333
```

Put data in a file using `write` or `writelines`

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

Same function

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

File to write to

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

File to write to

Created if it doesn't exist

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

For writing instead of reading

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

Write a single string

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

Write each string in a list

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

```
elementsHeNeArKr
```

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements')
writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
writer.close()
```

```
elementsHeNeArKr
```

Python only writes what you tell it to

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements\n')
writer.writelines(['He\n', 'Ne\n', 'Ar\n', 'Kr\n'])
writer.close()
```

Have to provide end-of-line characters yourself

Put data in a file using `write` or `writelines`

```
writer = open('temp.txt', 'w')
writer.write('elements\n')
writer.writelines(['He\n', 'Ne\n', 'Ar\n', 'Kr\n'])
writer.close()
```

```
elements
He
Ne
Ar
Kr
```

Often simpler to use `print(args, file=f1)`

Often simpler to use `print(args, file=f1)`

```
writer = open('temp.txt', 'w')
print('elements', file=writer)
for gas in ['He', 'Ne', 'Ar', 'Kr']:
    print(gas, file=writer)
writer.close()
```

Often simpler to use `print(args, file=f1)`

```
writer = open('temp.txt', 'w')
print('elements', file=writer)
for gas in ['He', 'Ne', 'Ar', 'Kr']:
    print(gas, file=writer)
writer.close()
```

Specify open file after

Often simpler to use `print(args, file=f1)`

```
writer = open('temp.txt', 'w')
print('elements', file=writer)
for gas in ['He', 'Ne', 'Ar', 'Kr']:
    print(gas, file=writer)
writer.close()
```

`print` automatically adds the newline

For python 2.x use `print >> f1, args .`

In Python 3.0+, `print(args, file=f1)`

print is a function, which you'd call with `print(...)`

Copy a file

Copy a file

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('temp.txt', 'w')
write.write(data)
writer.close()
```

Copy a file

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('temp.txt', 'w')
write.write(data)
writer.close()
```

Read all

Copy a file

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('temp.txt', 'w')
write.write(data)
writer.close()
```

Write all

Copy a file

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('temp.txt', 'w')
write.write(data)
writer.close()
```

Probably won't work with a terabyte…

Copy a file

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('temp.txt', 'w')
write.write(data)
writer.close()
```

Probably won't work with a terabyte...

...but we probably don't care

Copy a file (version 2)

Copy a file (version 2)

```
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
for line in reader:
    writer.write(line)
reader.close()
writer.close()
```

Copy a file (version 2)

```
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
for line in reader:
    writer.write(line)
reader.close()
writer.close()
```

Assumes the file is text

Copy a file (version 2)

```
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
for line in reader:
    writer.write(line)
reader.close()
writer.close()
```

Assumes the file is text

Or at least that the end-of-line character appears frequently

This version doesn't make an exact copy

This version doesn't make an exact copy

```
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
for line in reader:
    print(line, file=writer)
reader.close()
writer.close()
```

This version doesn't make an exact copy

```
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
for line in reader:
    print(line, file=writer)
reader.close()
writer.close()
```

Python keeps the newline when reading

This version doesn't make an exact copy

```
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
for line in reader:
    print(line, file=writer)
reader.close()
writer.close()
```

Python keeps the newline when reading

print automatically adds a newline

This version doesn't make an exact copy

```
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
for line in reader:
    print(line, file=writer)
reader.close()
writer.close()
```

Python keeps the newline when reading

print automatically adds a newline

Result is double-spaced output

Copy a file (version 3)

Copy a file (version 3)

```
BLOCKSIZE = 1024
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
data = reader.read(BLOCKSIZE)
while len(data) > 0:
    writer.write(data)
    data = reader.read(BLOCKSIZE)
reader.close()
writer.close()
```

Copy a file (version 3)

```
BLOCKSIZE = 1024
reader = open('haiku.txt', 'r')
writer = open('temp.txt', 'w')
data = reader.read(BLOCKSIZE)
while len(data) > 0:
    writer.write(data)
    data = reader.read(BLOCKSIZE)
reader.close()
writer.close()
```

(Needlessly?) harder to understand

Python

Strings

by Greg Wilson

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Strings are sequences of characters

Strings are sequences of characters

No separate character type: just a string of length 1

Strings are sequences of characters

No separate character type: just a string of length 1

Indexed exactly like lists

Strings are sequences of characters

No separate character type: just a string of length 1

Indexed exactly like lists

```
name = 'Darwin'
print(name[0], name[-1])
D n
```

`for` iterates through characters

for iterates through characters

```
name = 'Darwin'
for c in name:
    print(c)
```

```
D
a
r
w
i
n
```

Use either ' or " (as long as they match)

Use either ' or " (as long as they match)

```
print('Alan', "Turing“)
Alan Turing
```

Use either ' or " (as long as they match)

```
print('Alan', "Turing“)
Alan Turing
```

Strings are the same no matter how they're created

Use either ' or " (as long as they match)

```
print('Alan', "Turing“)
Alan Turing
```

Strings are the same no matter how they're created

```
print 'Alan' == "Alan"
True
```

Strings are compared character by character from left to right

Strings are compared character by character from left to right

```
print('a' < 'b')
True
```

Strings are compared character by character from left to right

```
print('a' < 'b')
True
print('ab' < 'abc')
True
```

Strings are compared character by character from left to right

```
print('a' < 'b')
True
print('ab' < 'abc')
True
print('1' < '9')
True
```

Strings are compared character by character from left to right

```
print('a' < 'b')
True
print('ab' < 'abc')
True
print('1' < '9')
True
print('100' < '9')
True
```

Strings are compared character by character from left to right

```
print('a' < 'b')
True
print('ab' < 'abc')
True
print('1' < '9')
True
print('100' < '9')
True
print('A' < 'a')
True
```

Strings are immutable : cannot be changed in place

Strings are immutable : cannot be changed in place

```
name = 'Darwin'
name[0] = 'C'
TypeError: 'str' object does not support item assignment
```

Strings are immutable : cannot be changed in place

```
name = 'Darwin'
name[0] = 'C'
TypeError: 'str' object does not support item assignment
```

Immutability improves performance

Strings are immutable : cannot be changed in place

```
name = 'Darwin'
name[0] = 'C'
TypeError: 'str' object does not support item assignment
```

Immutability improves performance

See later how immutability improves programmers' performance

Use + to concatenate strings

Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print(name)
Charles Darwin
```

Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print(name)
Charles Darwin
```

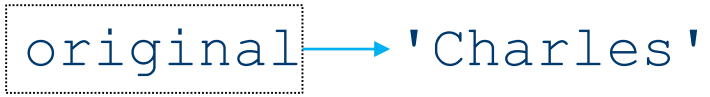
Concatenation always produces a new string

Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print(name)
Charles Darwin
```

Concatenation always produces a new string

```
original = 'Charles'
```

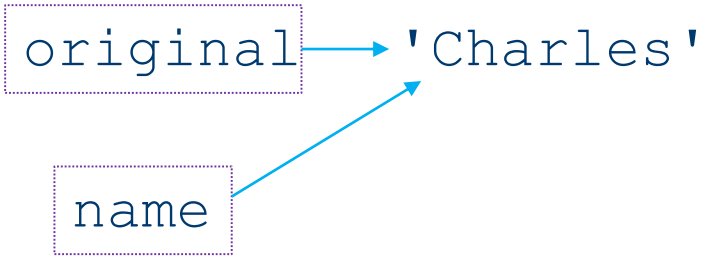


Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print(name)
Charles Darwin
```

Concatenation always produces a new string

```
original = 'Charles'
name = original
```

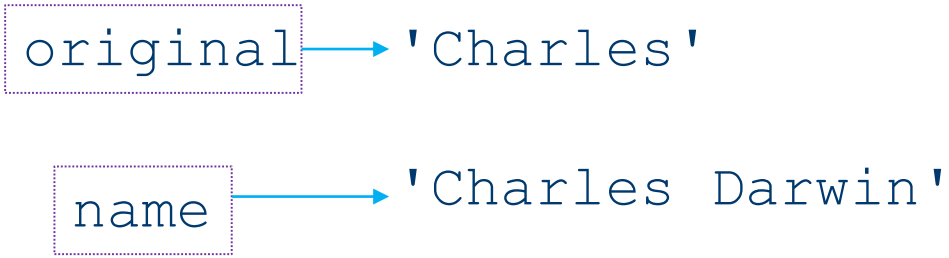


Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print(name)
Charles Darwin
```

Concatenation always produces a new string

```
original = 'Charles'
name = original
name += ' Darwin'
```



Often used to format output

Often used to format output

```
print('reagant: ' + str(reagant_id) + ' produced ' +
      str(percentage_yield) + '% yield')
```

Often used to format output

```
print('reagant: ' + str(reagant_id) + ' produced ' +
      str(percentage_yield) + '% yield')
```

There's a better way…

Use string % value to format output

Use string % value to format output

```
output = 'reagant: %d' % 123
print(output)
reagant: 123
```

Use string % value to format output

```
output = 'reagant: %d' % 123
print(output)
reagant: 123
```

```
percentage_yield = 12.3
print('yield: %6.2f' % percentage_yield)
yield:  12.30
```

And string `% (v1, v2, ...)` for multiple values

And string `% (v1, v2, ...)` for multiple values

```
reagant_id = 123
percentage_yield = 12.3
print('reagant: %d produced %6.2f%% yield' %
      (reagant_id, percentage_yield))
reagant: 123 produced  12.30% yield
```

And string `% (v1, v2, ...)` for multiple values

```
reagant_id = 123
percentage_yield = 12.3
print('reagant: %d produced %6.2f%% yield' %
      (reagant_id, percentage_yield))
reagant: 123 produced  12.30% yield
```

`%` operator turns double '`%%`' into single '`%`'

Use `\n` to represent a newline character

Use `\n` to represent a newline character

Use `\'` for single quote, `\"` for double quote

Use `\n` to represent a newline character

Use `\'` for single quote, `\"` for double quote

```
print('There isn\'t time\nto do it right.')
There isn't time
to do it right.
```

Use `\n` to represent a newline character

Use `\'` for single quote, `\"` for double quote

```
print('There isn\'t time\nto do it right.')
There isn't time
to do it right.
```

```
print("But you said,\n\"There is time to do it over.\"")
But you said,
"There is time to do it over."
```

Use \\ for a literal \ character

Use \\ for a literal \ character

```
print('Most mathematicians write a\\b instead of a%b.')
```

Most mathematicians write a\b instead of a%b.

Use \\ for a literal \ character

```
print('Most mathematicians write a\\b instead of a%b.')
Most mathematicians write a\b instead of a%b.
```

Common pattern with escape sequences

Use \\ for a literal \ character

```
print('Most mathematicians write a\\b instead of a%b.')
Most mathematicians write a\b instead of a%b.
```

Common pattern with escape sequences

- Use a character to mean "what follows is special"

Use \\ for a literal \ character

```
print('Most mathematicians write a\\b instead of a%b.')
Most mathematicians write a\b instead of a%b.
```

Common pattern with escape sequences

- Use a character to mean "what follows is special"
- Double it up to mean "that character itself"

Use triple quotes (either kind) for multi-line strings

Use triple quotes (either kind) for multi-line strings

```
quote = '''We can only see
a short distance ahead,
but we can see plenty there
that needs to be done.'''
```

Use triple quotes (either kind) for multi-line strings

```
quote = '''We can only see
a short distance ahead,
but we can see plenty there
that needs to be done.'''
```

Use triple quotes (either kind) for multi-line strings

```
quote = '''We can only see
a short distance ahead,
but we can see plenty there
that needs to be done.'''


quote = 'We can only see\na short distance ahead,\n'
        + 'but we can see plenty there\nthat needs
        to be done.'
```

Strings have methods

Strings have methods

```
name = 'newTON'
print(name.capitalize(), name.upper(), name.lower(), name)
Newton NEWTON newton newTON
```

Strings have methods

```
name = 'newTON'
print(name.capitalize(), name.upper(), name.lower(), name)
Newton NEWTON newton newTON
dna = 'acggtggtcac'
print(dna.count('g'), dna.count('x'))
4 0
```

Strings have methods

```
name = 'newTON'
print(name.capitalize(), name.upper(), name.lower(), name)
Newton NEWTON newton newTON
dna = 'acggtggtcac'
print(dna.count('g'), dna.count('x'))
4 0
print(dna.find('t'), dna.find('t', 5), dna.find('x'))
4 7 -1
```

Strings have methods

```
name = 'newTON'
print(name.capitalize(), name.upper(), name.lower(), name)
Newton NEWTON newton newTON
dna = 'acggtggtcac'
print(dna.count('g'), dna.count('x'))
4 0
print(dna.find('t'), dna.find('t', 5), dna.find('x'))
4 7 -1
print(dna.replace('t', 'x'), dna)
acggxggxcac acggtggtcac
```

Strings have methods

```
name = 'newTON'
print(name.capitalize(), name.upper(), name.lower(), name)
Newton NEWTON newton newTON
dna = 'acggtggtcac'
print(dna.count('g'), dna.count('x'))
4 0
print(dna.find('t'), dna.find('t', 5), dna.find('x'))
4 7 -1
print(dna.replace('t', 'x'), dna)
acggxggxcac acggtggtcac
print(dna.replace('gt', ''))
acggcac
```

Can chain method calls together

Can chain method calls together

```
element = 'cesium'
print(element.upper().center(10, '.'))
```

Can chain method calls together

```
element = 'cesium'
print(element.upper().center(10, '.'))
```

convert to upper case

Can chain method calls together

```
element = 'cesium'
print(element.upper().center(10, '.'))
```

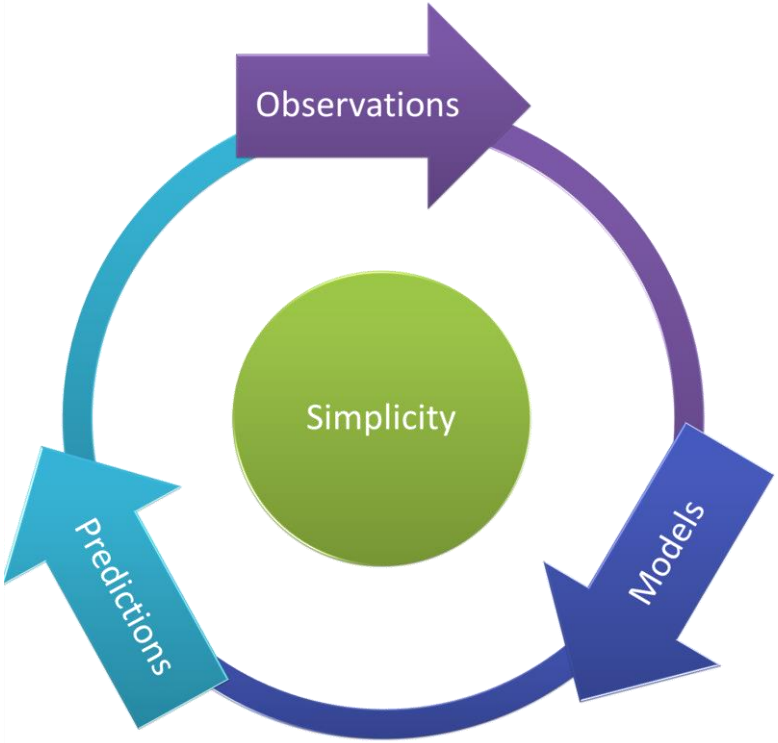
center in a field 10 characters wide

Can chain method calls together

```
element = 'cesium'
print(element.upper().center(10, '.'))
..CESIUM..
```

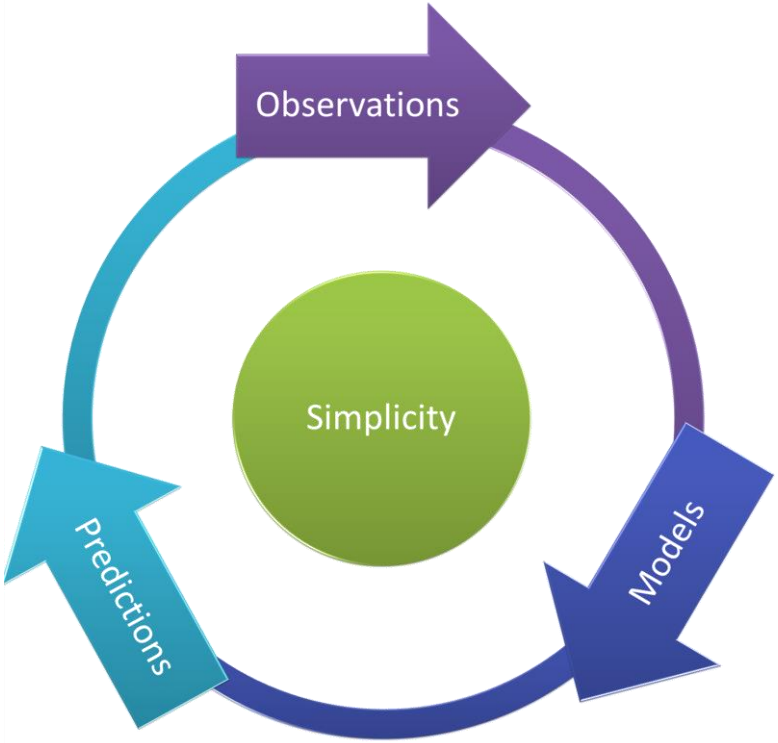
Scientific Method

- Begin with a set of observations



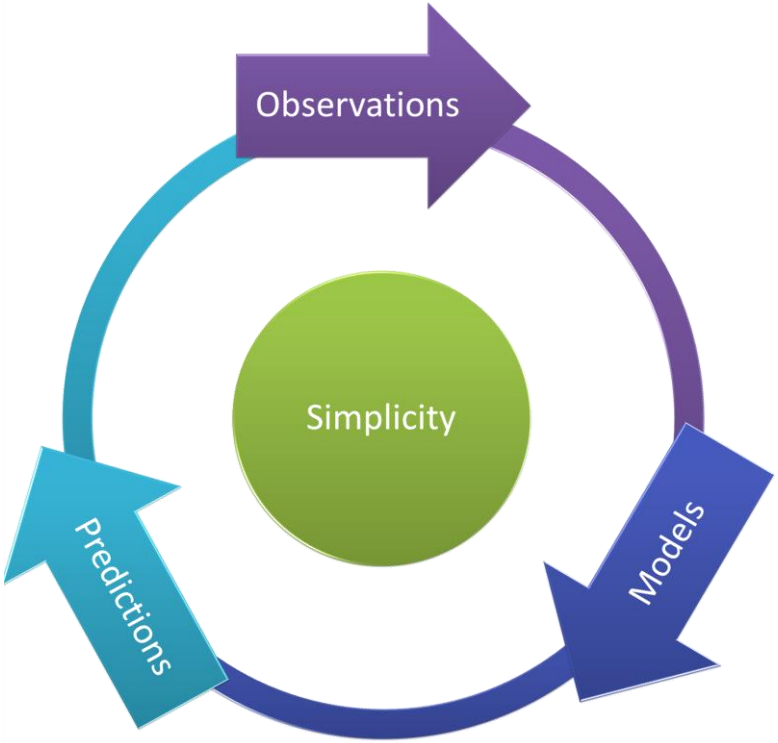
Scientific Method

- Begin with a set of observations
- Create a model to explain the observations



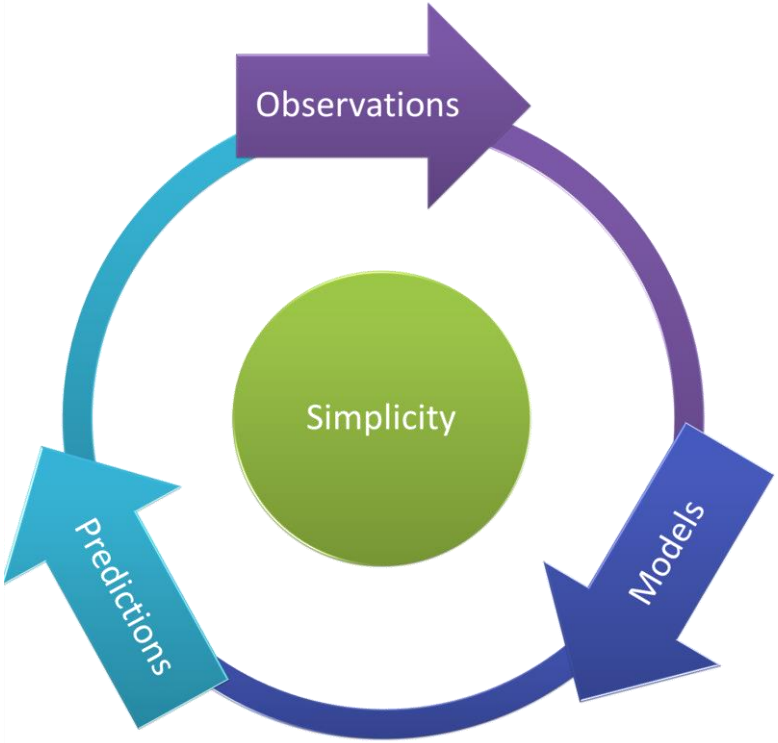
Scientific Method

- Begin with a set of observations
- Create a model to explain the observations
- Make testable predictions using the model



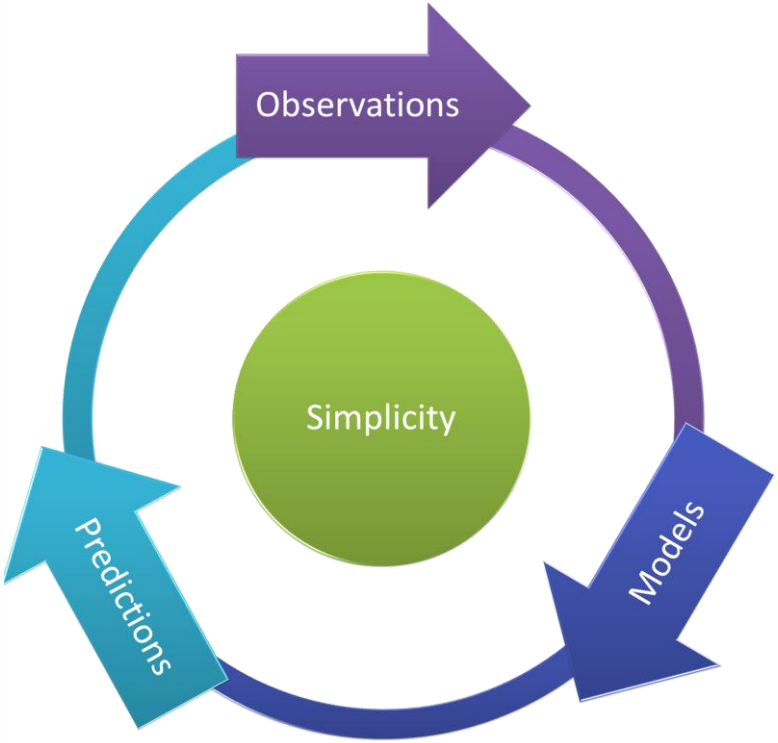
Scientific Method

- Begin with a set of observations
- Create a model to explain the observations
- Make testable predictions using the model
- Compare the predictions with new observations



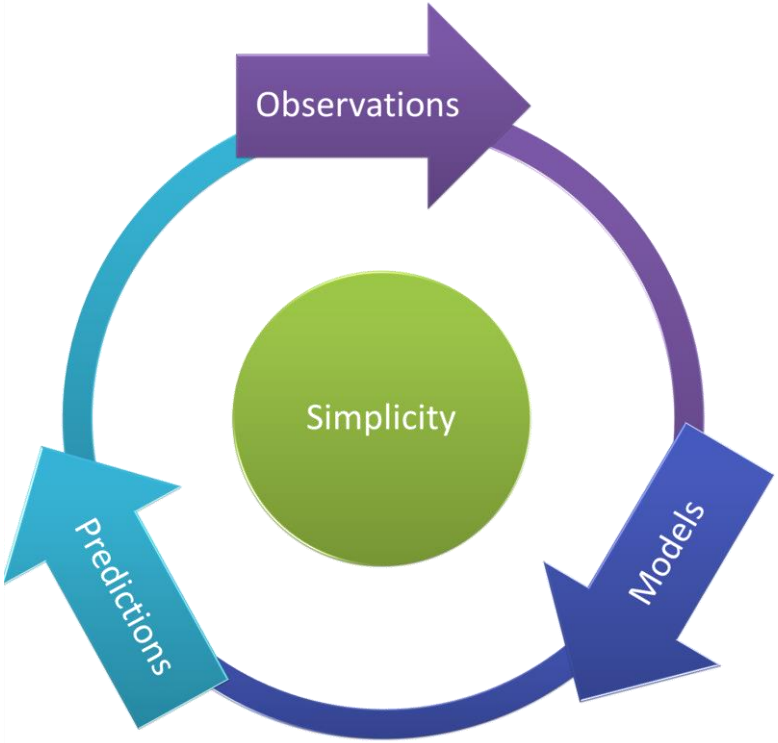
Scientific Method

- Begin with a set of observations
- Create a model to explain the observations
- Make testable predictions using the model
- Compare the predictions with new observations
- Use comparison to asses and modify the model



Scientific Method

- Repeat as required



Scientific Method

- Repeat as required
- Simplicity selects one preferred model from the many possible models that describe any set of observations