An Introduction to Python

Day 1

Simon Mitchell

Simon.Mitchell@ucla.edu

Beautiful is better than ugly. **Explicit** is better than implicit. **Simple** is better than complex. **Complex** is better than complicated. **Flat** is better than nested. **Sparse** is better than dense. **Readability** counts. *Special cases* aren't special enough to break the rules. Although **practicality** beats purity. *Errors* should never pass silently. Unless **explicitly** silenced. In the face of *ambiguity*, **refuse** the temptation to guess. There should be **one** — and preferably only one — obvious way to do it. Although that way may not be obvious at first *unless you're Dutch*. **Now** is better than never. Although never is **often** better than *right* now. If the implementation is *hard* to explain, it's a **bad** idea. If the implementation is *easy* to explain, it *may* be a **good** idea. **Namespaces** are one *honking great* idea — let's do more of those!

Beautiful is better than ugly. **Explicit** is better than implicit. **Simple** is better than complex. **Complex** is better than complicated. **Flat** is better than nested. **Sparse** is better than dense. **Readability** counts. *Special cases* aren't special enough to break the rules. Although **practicality** beats purity. *Errors* should never pass silently. Unless **explicitly** silenced. In the face of *ambiguity*, **refuse** the temptation to guess. There should be **one** — and preferably only one — obvious way to do it. Although that way may not be obvious at first *unless you're Dutch*. **Now** is better than never. Although never is **often** better than *right* now. If the implementation is *hard* to explain, it's a **bad** idea. If the implementation is *easy* to explain, it *may* be a **good** idea. **Namespaces** are one *honking great* idea — let's do more of those!

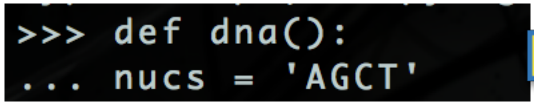
python™

Why Python?

* Clear code
* Great beginner language
* Powerful text manipulation
* Wrangle large data files
* Great compliment to other languages
* Large user group
* Supports many advanced features

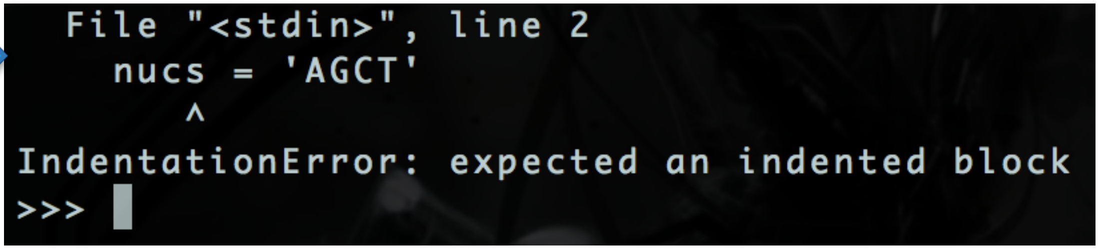
Warning: Spacing is important!

Wrong:



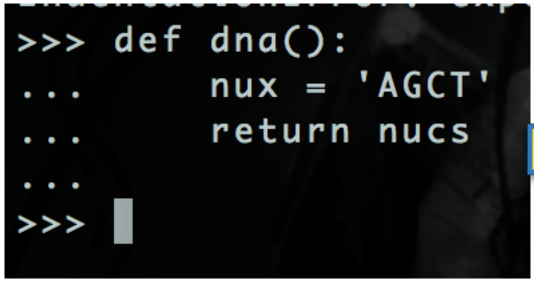
```
>>> def dna():
... nucs = 'AGCT'
```

Error:



```
  File "<stdin>", line 2
    nucs = 'AGCT'
       ^
IndentationError: expected an indented block
>>> 
```

Correct:



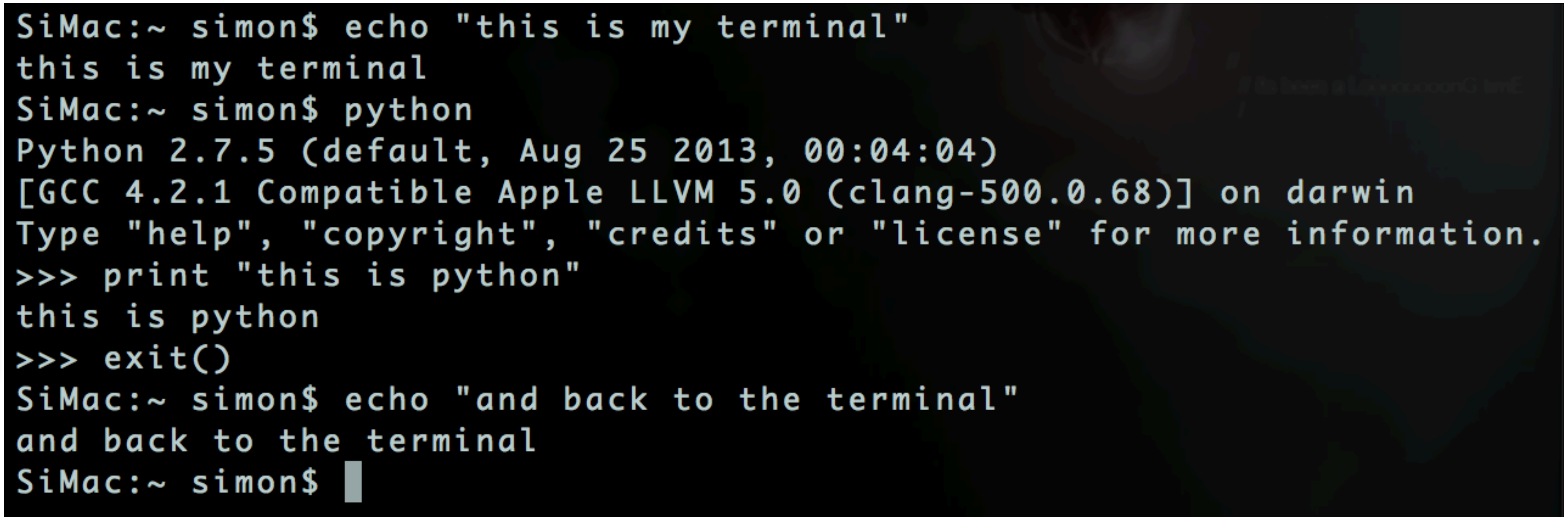
```
>>> def dna():
...     nux = 'AGCT'
...     return nucs
...
>>> 
```

No Error:

Open A Terminal

- Open a terminal:
  - Mac: cmd + space then type terminal and press enter
  - Windows: Start -> Program Files -> Accessories -> Command Prompt.
- Type "python" (no quotes). Exit() to exit python.

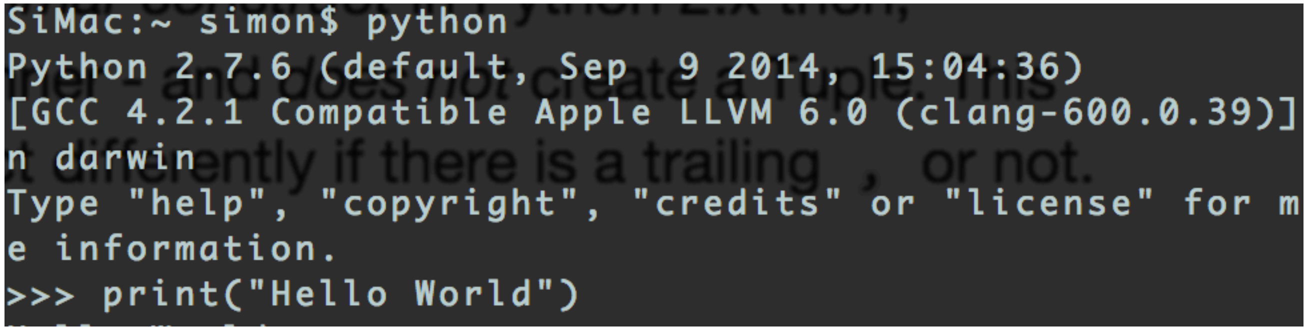
This is python



```
SiMac:~ simon$ echo "this is my terminal"
this is my terminal
SiMac:~ simon$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "this is python"
this is python
>>> exit()
SiMac:~ simon$ echo "and back to the terminal"
and back to the terminal
SiMac:~ simon$ 
```

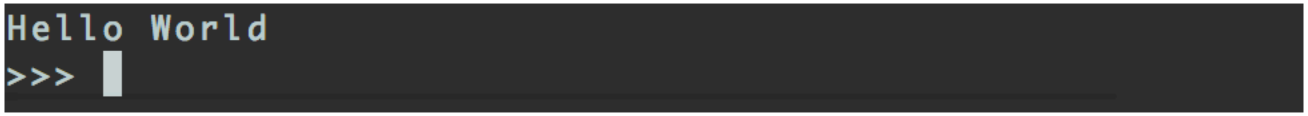
Hello World

Launch python



```
SiMac:~ simon$ python
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)]
n darwin
Type "help", "copyright", "credits" or "license" for m
e information.
>>> print("Hello World")
```

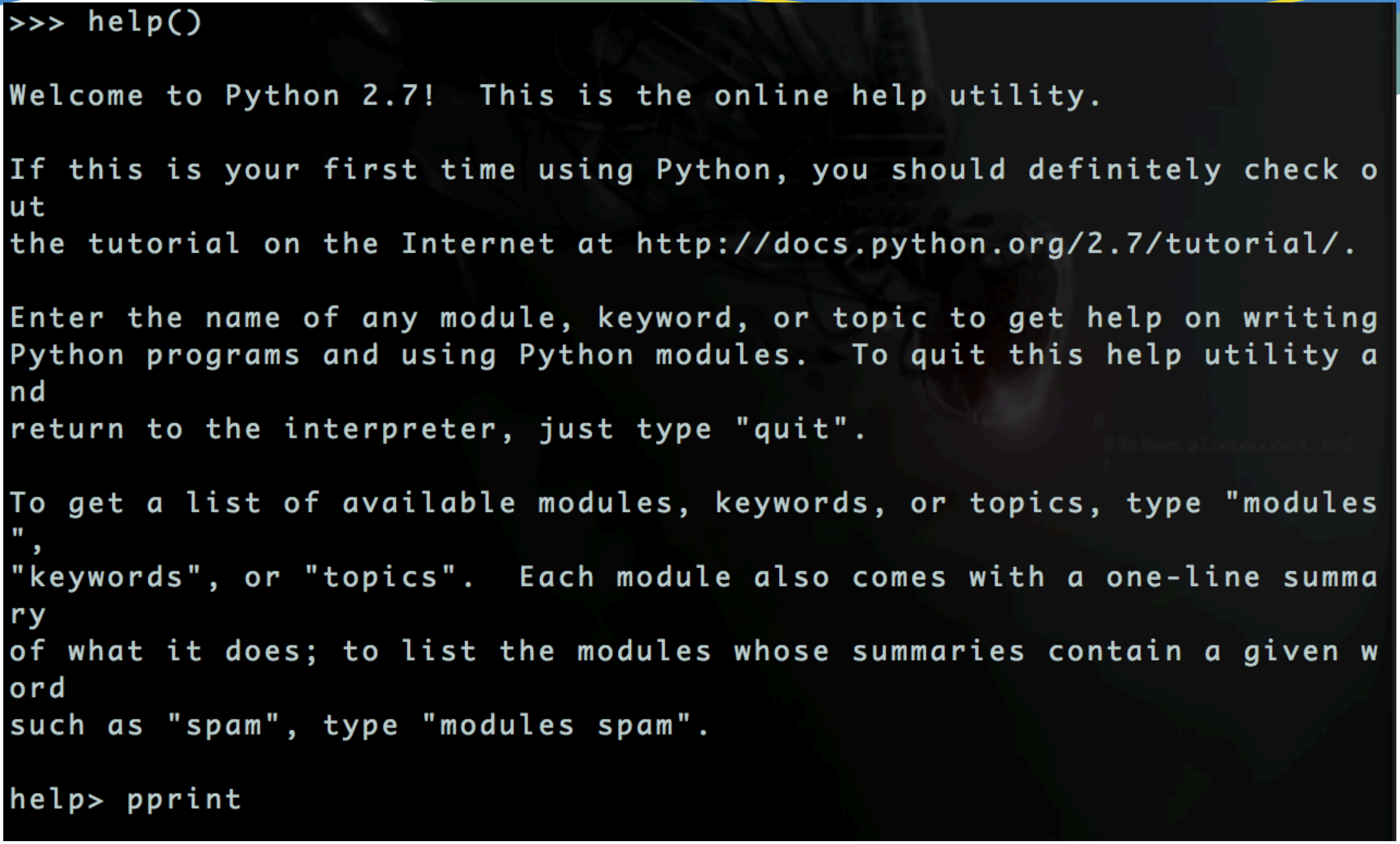
Call the built in function *print*, which displays whatever comes after the command. Put any message in quotes after the print command.



```
Hello World
>>> 
```

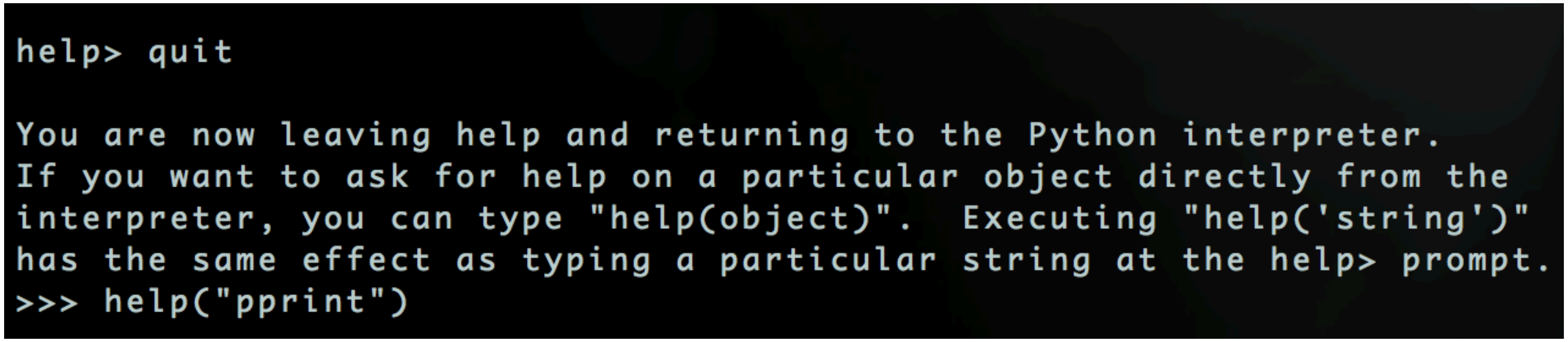
The command has finished and python is ready for the next command.
>>> means tell me what to do now!

Getting help - interactive



```
>>> help()

Welcome to Python 2.7!  This is the online help utility.

If this is your first time using Python, you should definitely check o
ut
the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility a
nd
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules
",
"keywords", or "topics".  Each module also comes with a one-line summa
ry
of what it does; to list the modules whose summaries contain a given w
ord
such as "spam", type "modules spam".

help> pprint
```

Getting help – single command



```
help> quit

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>> help("pprint")
```

But usually just Google!
If you got stuck on something, someone else probably has.

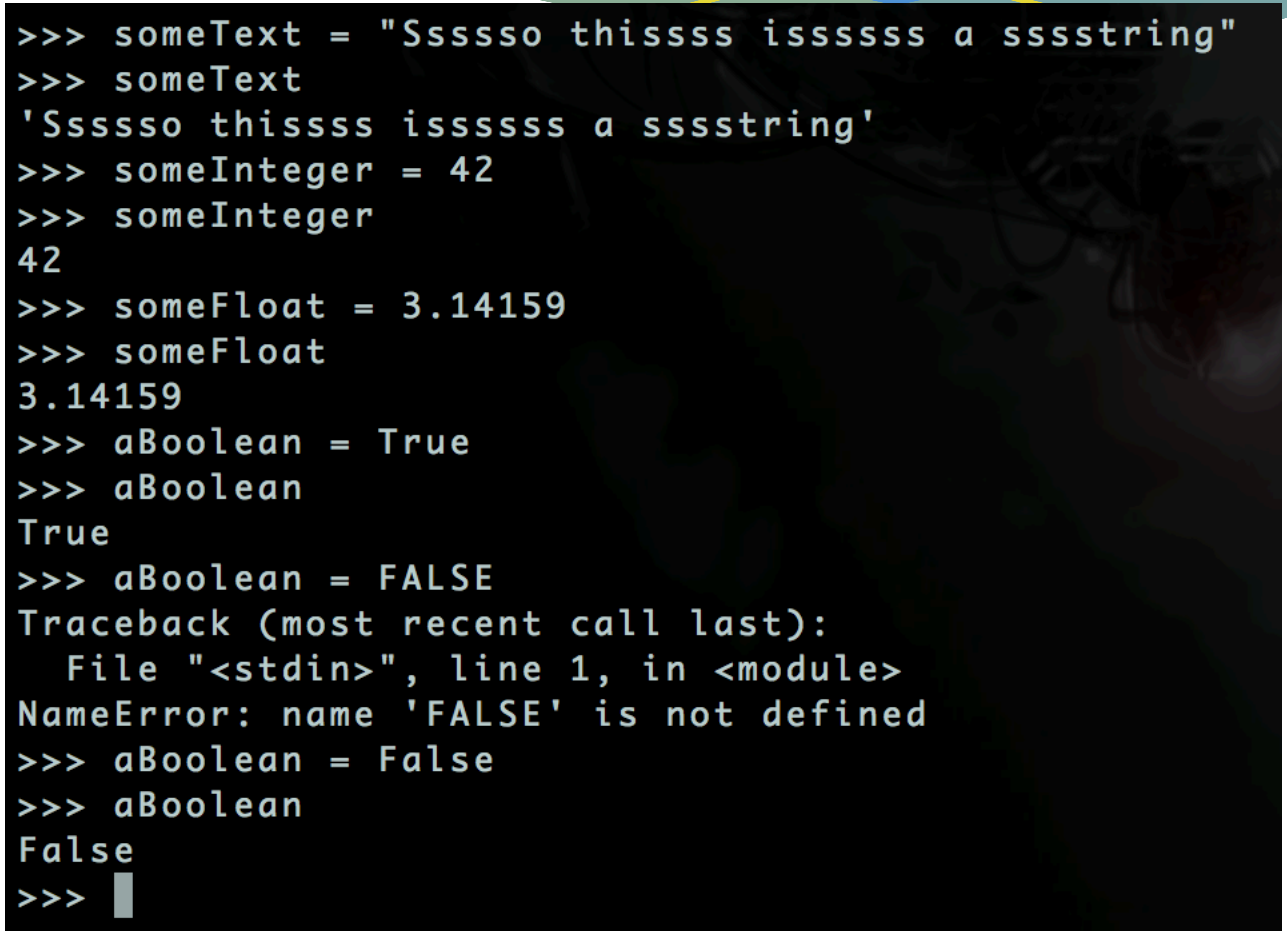
Let's get programming - Variables

Set a variable with equals

Display a variable by typing its name

Variables can be text, numbers, boolean (True/False) and many more things.

Capitalization is important for True/False



```
>>> someText = "Ssssso thissss issssss a sssstring"
>>> someText
'Ssssso thissss issssss a sssstring'
>>> someInteger = 42
>>> someInteger
42
>>> someFloat = 3.14159
>>> someFloat
3.14159
>>> aBoolean = True
>>> aBoolean
True
>>> aBoolean = FALSE
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'FALSE' is not defined
>>> aBoolean = False
>>> aBoolean
False
>>> 
```

Numeric Operators

Add +

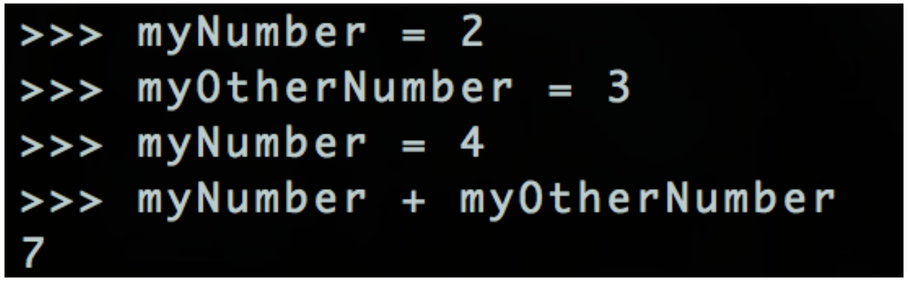
Subtract –

Multiply *

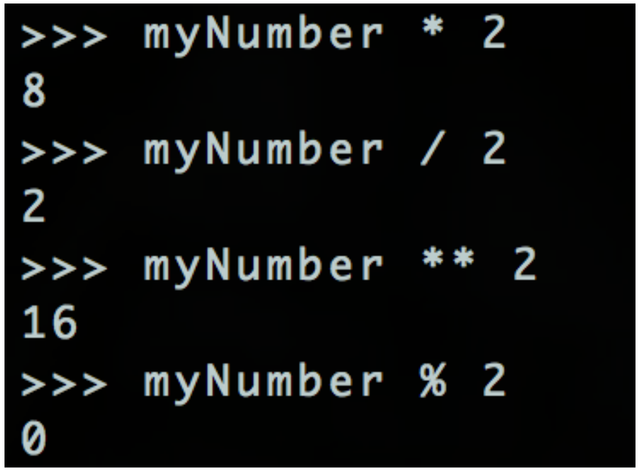
Divide /

Power **

Modulo (remainder) %



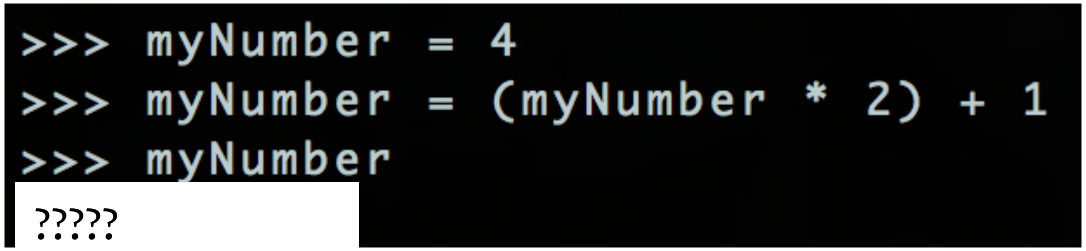
```
>>> myNumber = 2
>>> myOtherNumber = 3
>>> myNumber = 4
>>> myNumber + myOtherNumber
7
```



```
>>> myNumber * 2
8
>>> myNumber / 2
2
>>> myNumber ** 2
16
>>> myNumber % 2
0
```

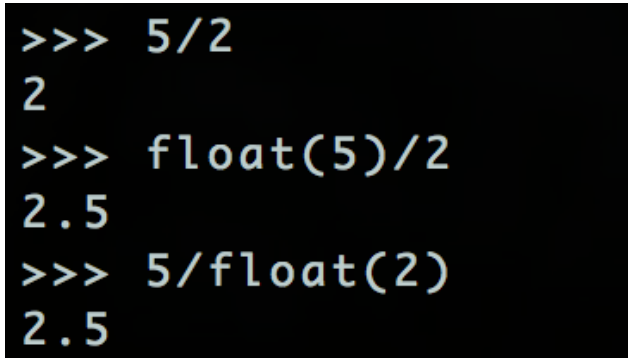
Reassigning Variables

Reassign with equals.
(Same as assigning)



```
>>> myNumber = 4
>>> myNumber = (myNumber * 2) + 1
>>> myNumber
?????
```

Warning!
In some version of python division might not do what you expect.
Integer division gives an integer result.

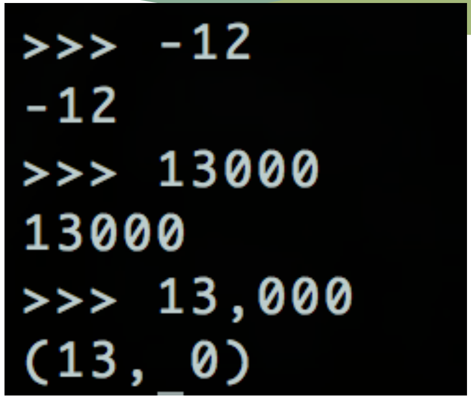


```
>>> 5/2
2
>>> float(5)/2
2.5
>>> 5/float(2)
2.5
```

Types of number

Integer:

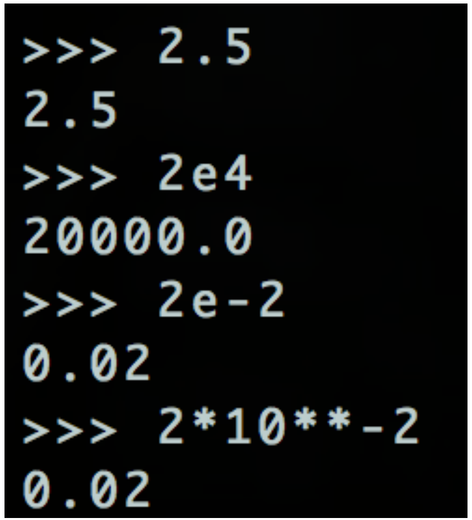
Plus and minus.
No decimal points or commas



```
>>> -12
-12
>>> 13000
13000
>>> 13,000
(13, 0)
```

Float:

Decimal points or scientific notation okay.
2e-2 = 2 x 10^{-2}



```
>>> 2.5
2.5
>>> 2e4
20000.0
>>> 2e-2
0.02
>>> 2*10**-2
0.02
```

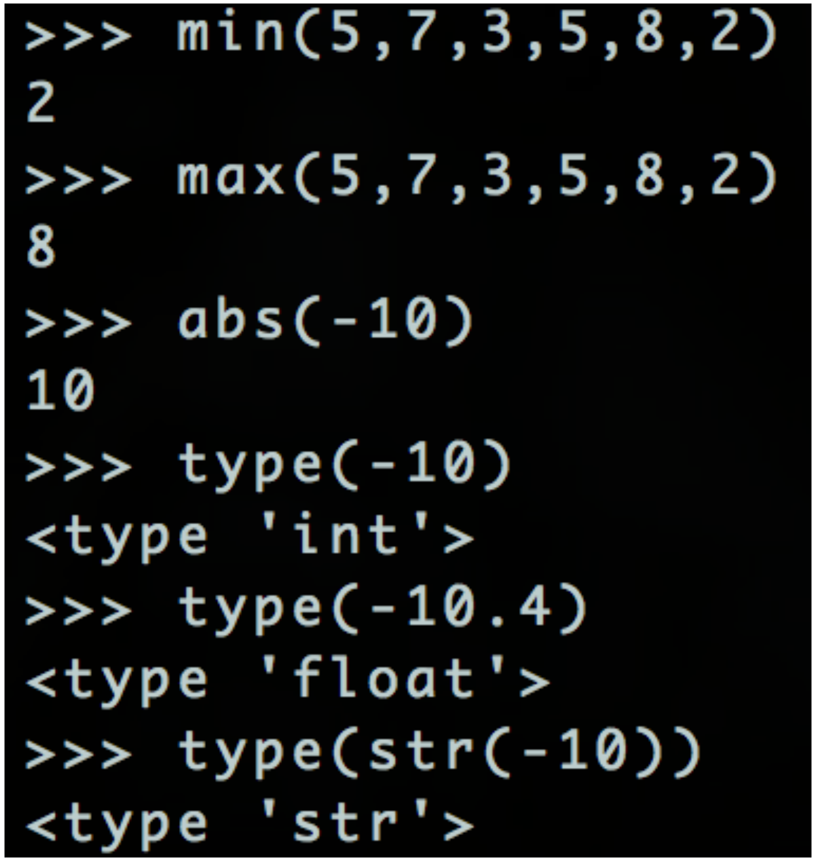
Working With Numbers

What is the **minimum** of these numbers:

What is the **maximum** of these numbers:

What **type** of variable is this?

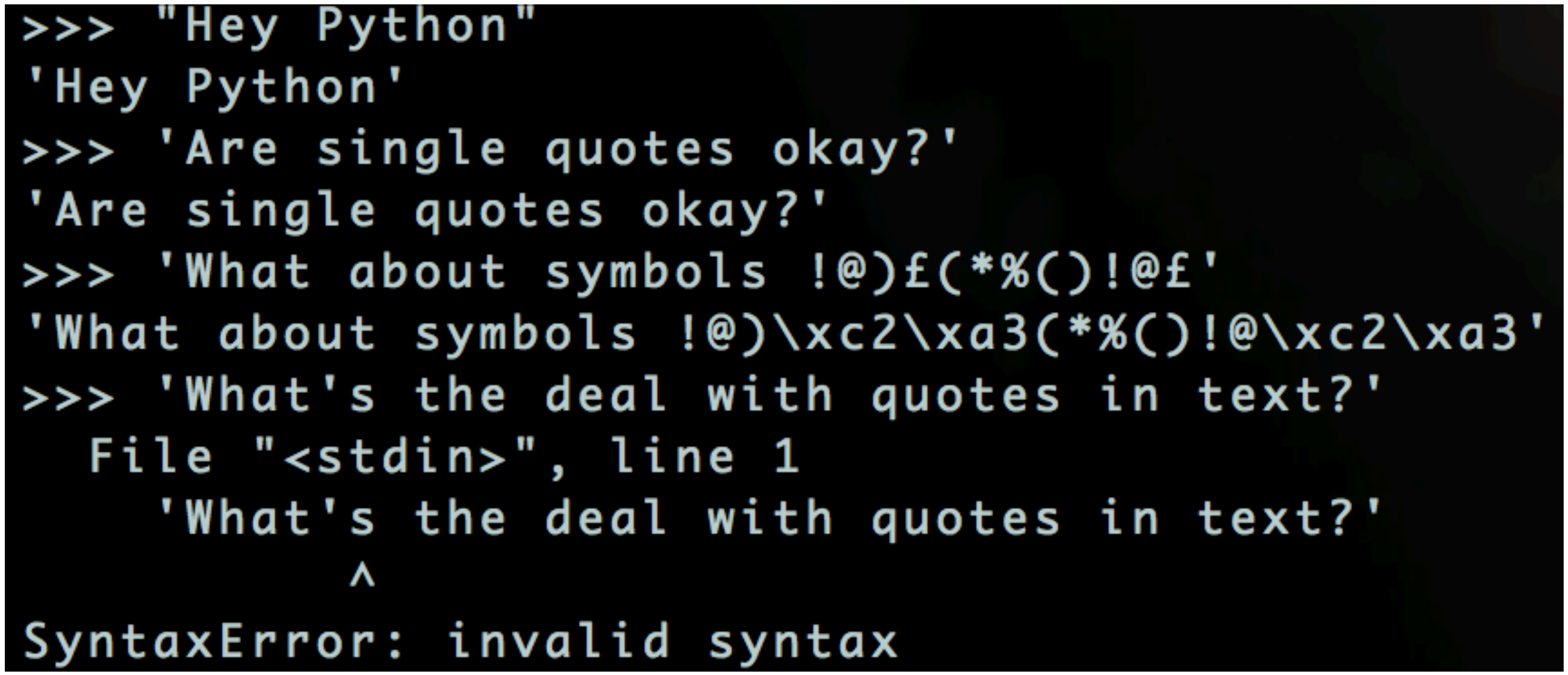
Remember that str(anything) makes that variable into a string:



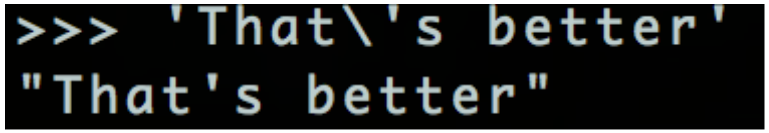
```
>>> min(5,7,3,5,8,2)
2
>>> max(5,7,3,5,8,2)
8
>>> abs(-10)
10
>>> type(-10)
<type 'int'>
>>> type(-10.4)
<type 'float'>
>>> type(str(-10))
<type 'str'>
```

Working With Text

Single or double quotes.
No *char* type. Just a single letter string.



```
>>> "Hey Python"
'Hey Python'
>>> 'Are single quotes okay?'
'Are single quotes okay?'
>>> 'What about symbols !@)£(*%()!@£'
'What about symbols !@)\xc2\xa3(*%()!@\xc2\xa3'
>>> 'What's the deal with quotes in text?'
  File "<stdin>", line 1
    'What's the deal with quotes in text?'
          ^
SyntaxError: invalid syntax
```



```
>>> 'That\'s better'
"That's better"
```

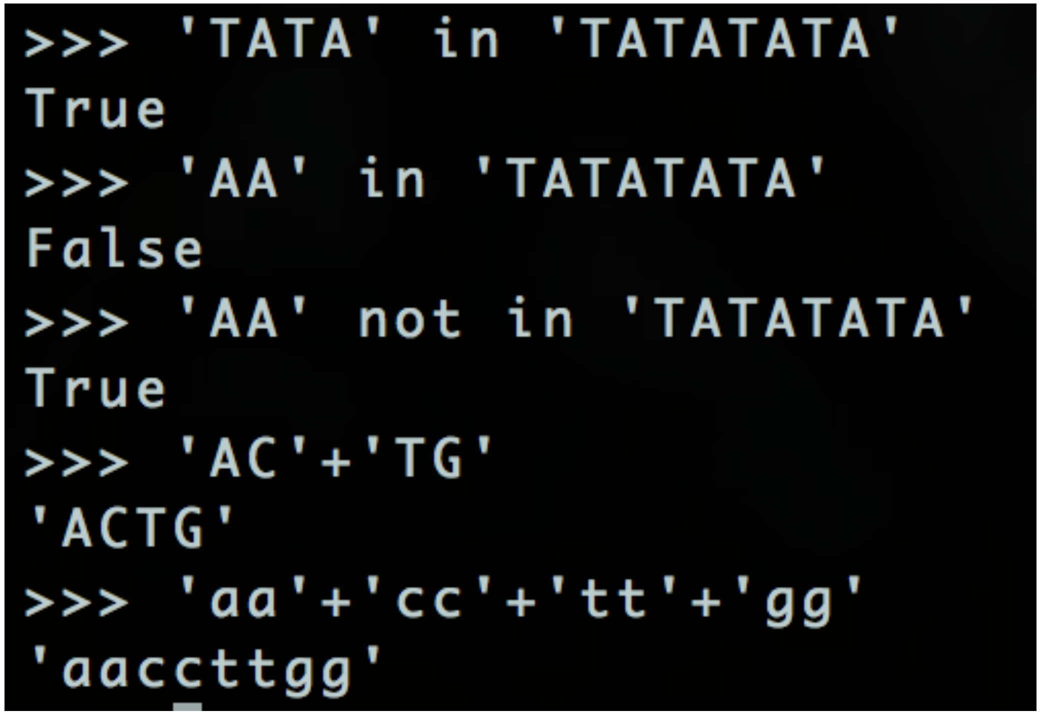
Escape character is \
\' types a quote.

Working With Text 2

Is a substring in a string?

Is a substring NOT in a string?

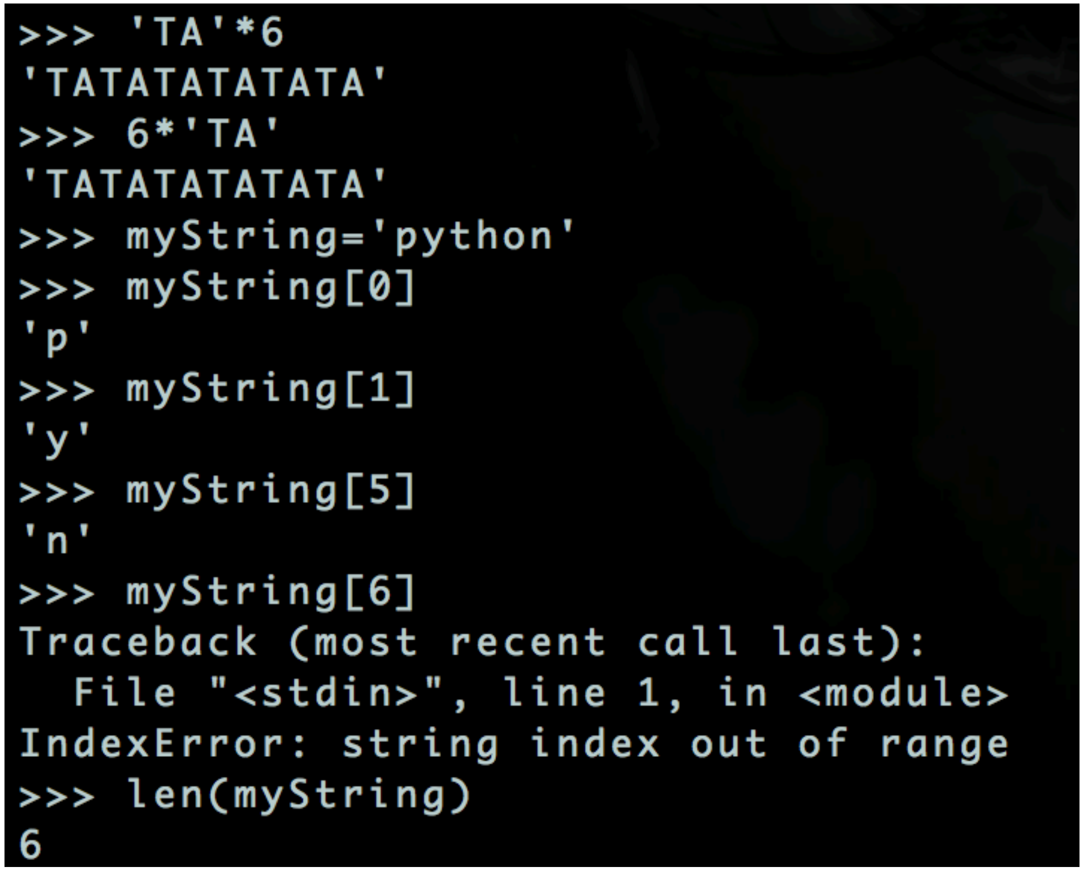
String concatenation:



```
>>> 'TATA' in 'TATATATA'
True
>>> 'AA' in 'TATATATA'
False
>>> 'AA' not in 'TATATATA'
True
>>> 'AC'+'TG'
'ACTG'
>>> 'aa'+'cc'+'tt'+'gg'
'aaccttgg'
```

Working With Text 3

- Multiply a string repeats it:
- Set variable *myString* to be 'python'
Each character in a string is a number
  - We start counting from **zero**!
- "String index out of range" error as we tried to reference a character beyond the end of the string.
- len(myString) gets the number of characters.

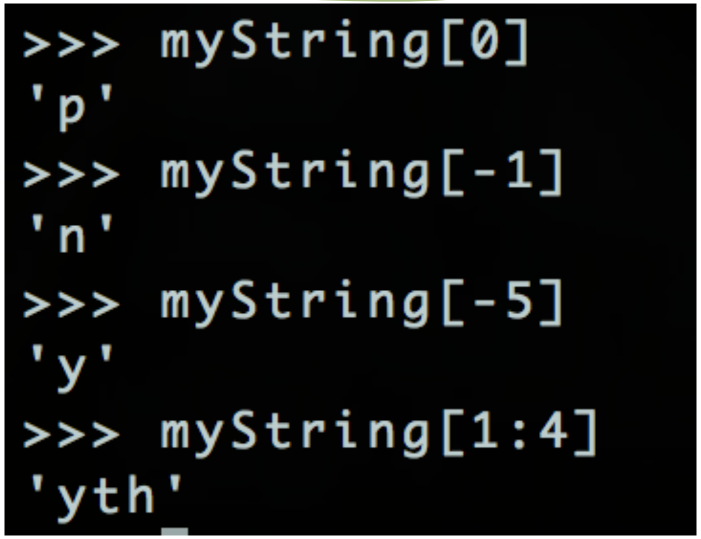


```
>>> 'TA'*6
'TATATATATATA'
>>> 6*'TA'
'TATATATATATA'
>>> myString='python'
>>> myString[0]
'p'
>>> myString[1]
'y'
>>> myString[5]
'n'
>>> myString[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> len(myString)
6
```

Working With Text 4

Negative index counts backwards from the last element.

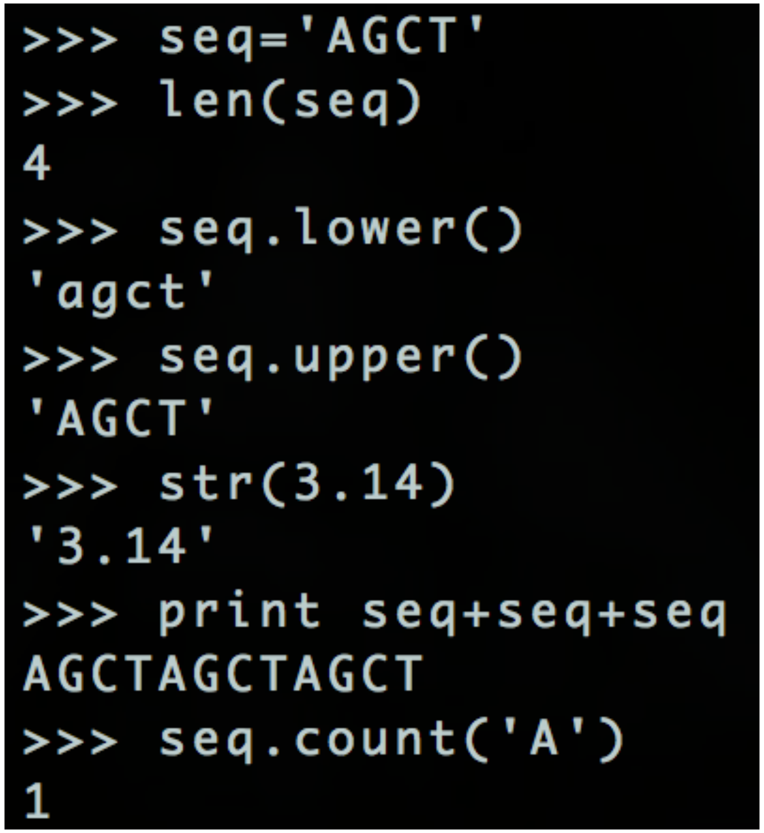
You can get a range of characters from a string.



```
>>> myString[0]
'p'
>>> myString[-1]
'n'
>>> myString[-5]
'y'
>>> myString[1:4]
'yth'
```

Working With Text 4

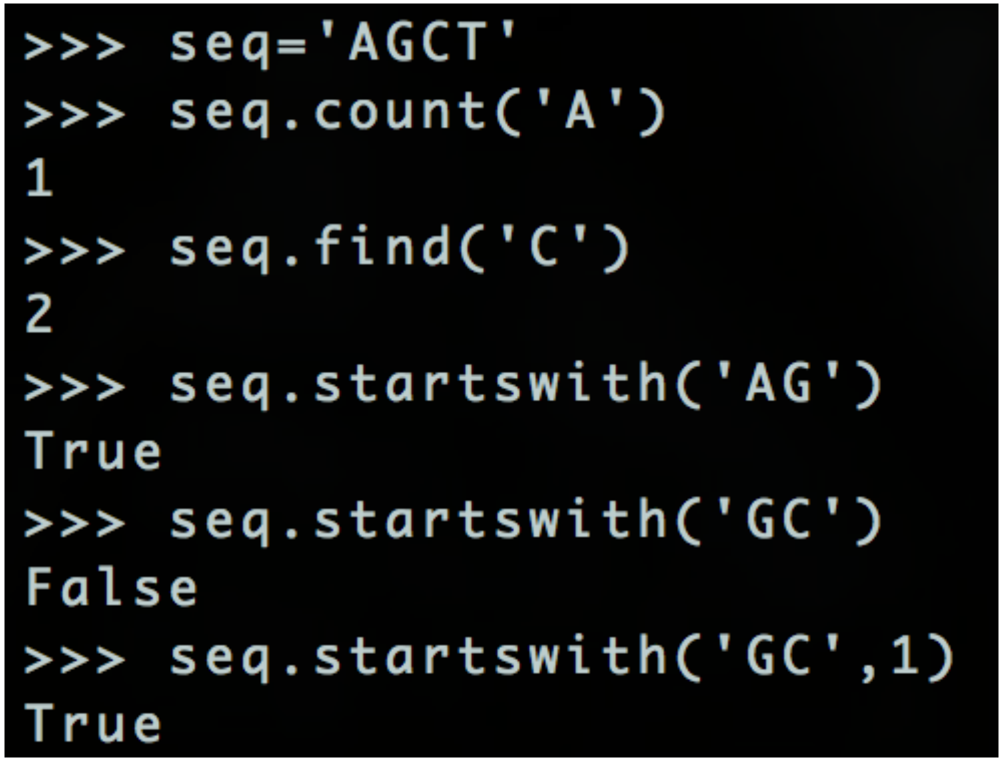
- Set the variable *seq* to be 'AGCT':
  - Get the number of characters in *seq:*
- Return the variable *seq* in all lower case characters:
- Return the variable *seq* in all upper case characters:
- Return the number 3.14 as a string:
- Display the variable *seq* repeated 3 times:
- Count the occurrences of 'A' in *seq:*



```
>>> seq='AGCT'
>>> len(seq)
4
>>> seq.lower()
'agct'
>>> seq.upper()
'AGCT'
>>> str(3.14)
'3.14'
>>> print seq+seq+seq
AGCTAGCTAGCT
>>> seq.count('A')
1
```

Working With Text 5

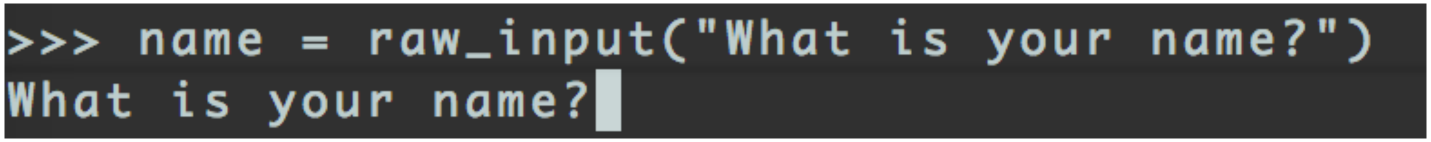
- Set the variable *seq* to be 'AGCT':
- Count the occurrences of 'A' in *seq:*
- Find which index in *seq* contains 'C'
- Does *seq* start with 'AG'
- Does *seq* start with 'GC'
- Does *seq* start with 'GC' if you start at the second letter.



```
>>> seq='AGCT'
>>> seq.count('A')
1
>>> seq.find('C')
2
>>> seq.startswith('AG')
True
>>> seq.startswith('GC')
False
>>> seq.startswith('GC',1)
True
```

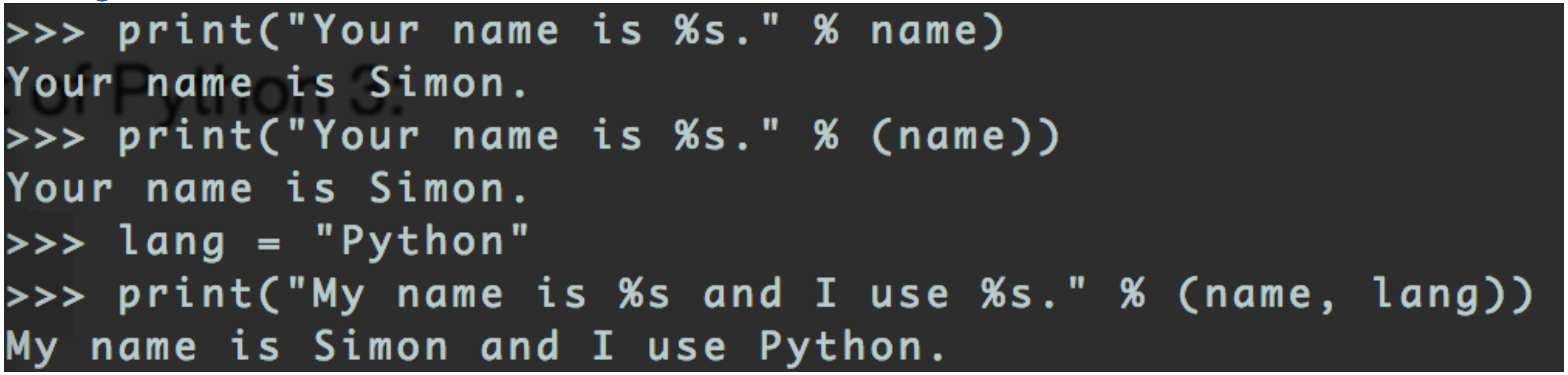
Working With Text 6

variable = raw_input(“text here”)
Prints the text in quotes and waits for user input.
Sets the variable on the left of = to whatever the user types.



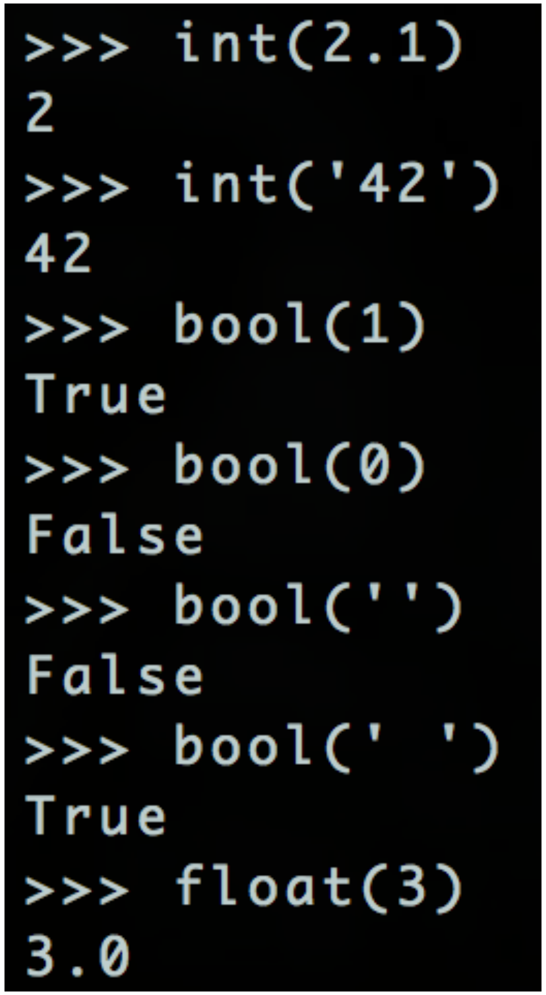
```
>>> name = raw_input("What is your name?")
What is your name?
```

print(“%s” % text-here)
Place a %s in a string to place a variable at that point in the string. The variables are given in order after a %.



```
>>> print("Your name is %s." % name)
Your name is Simon.
>>> print("Your name is %s." % (name))
Your name is Simon.
>>> lang = "Python"
>>> print("My name is %s and I use %s." % (name, lang))
My name is Simon and I use Python.
```

Changing a Variables Type



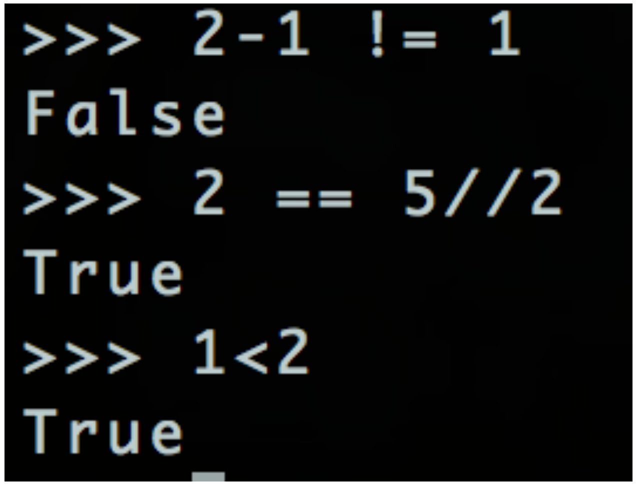
```
>>> int(2.1)
2
>>> int('42')
42
>>> bool(1)
True
>>> bool(0)
False
>>> bool('')
False
>>> bool(' ')
True
>>> float(3)
3.0
```

Cast a variable to another type.

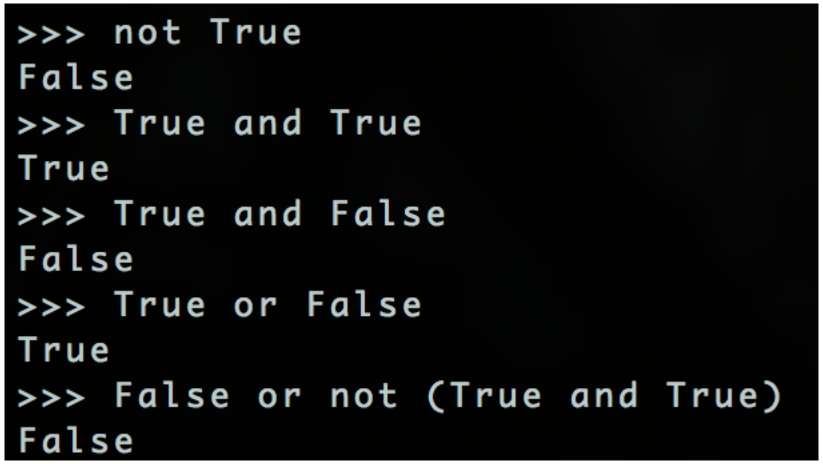
Note:
1 = True
0 = False

Empty strings = False
Any other string = True

True/False – conditional expressions



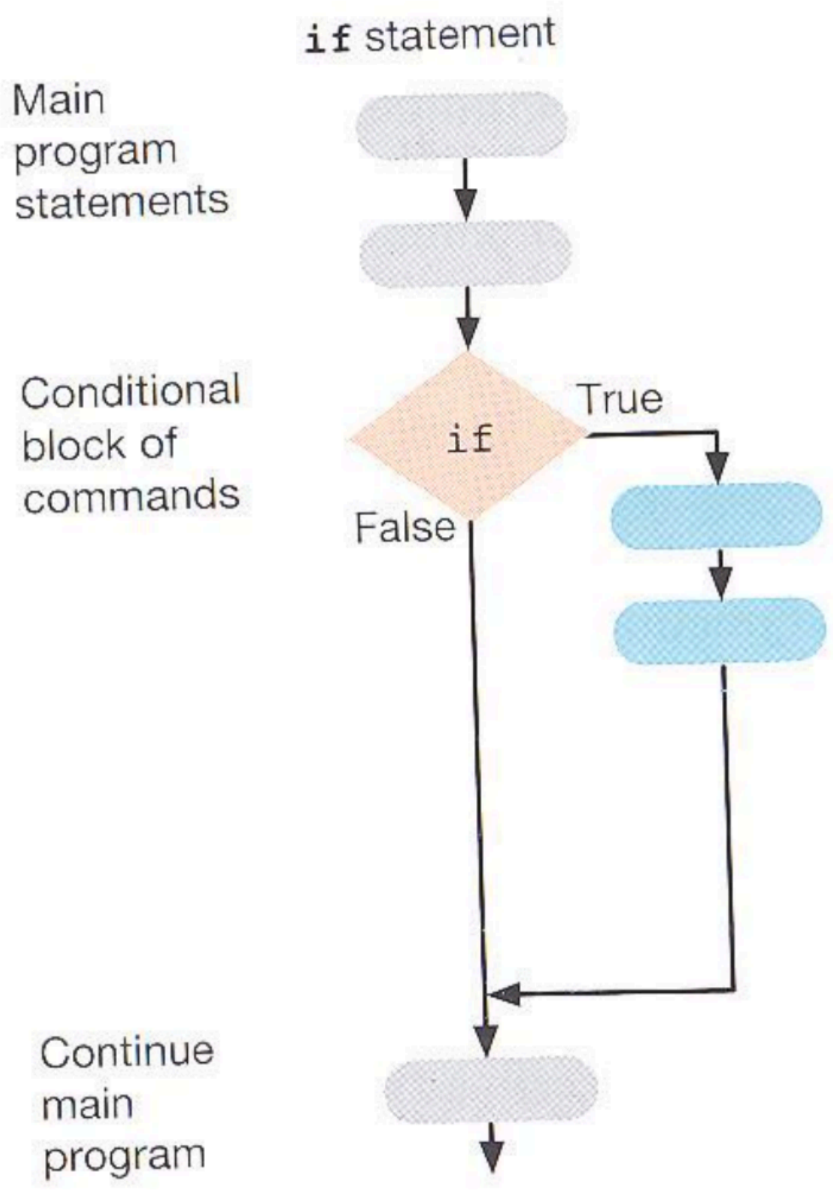
```
>>> 2-1 != 1
False
>>> 2 == 5//2
True
>>> 1<2
True
```

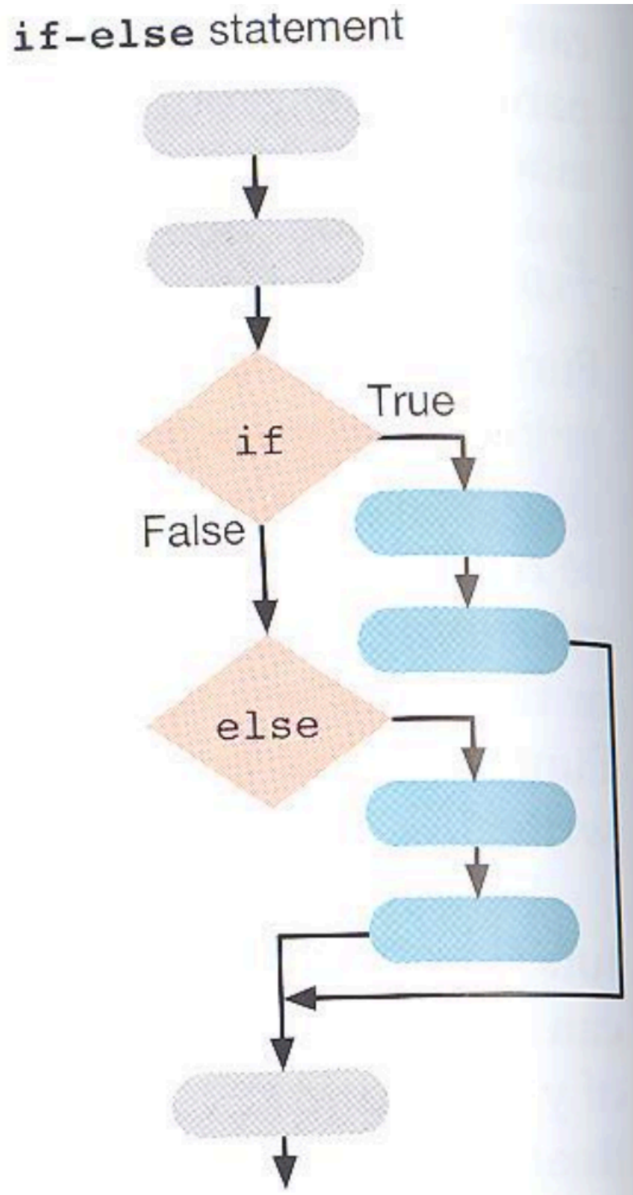


```
>>> not True
False
>>> True and True
True
>>> True and False
False
>>> True or False
True
>>> False or not (True and True)
False
```

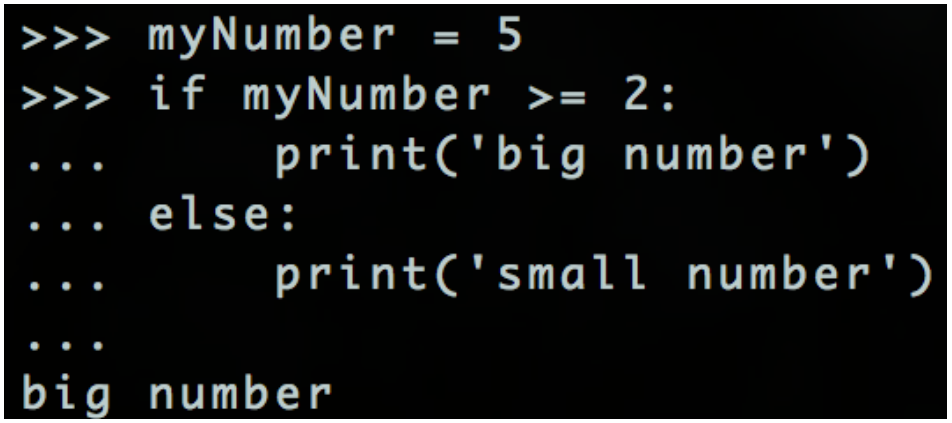
Equal to (==)
Not equal to (!=)
Less than (<)
Less than or equal to <=
Greater than (>)
Greater than or equal to (>=)

not
and
or


if statement
Main program statements
Conditional block of commands
Continue main program
if
True
False

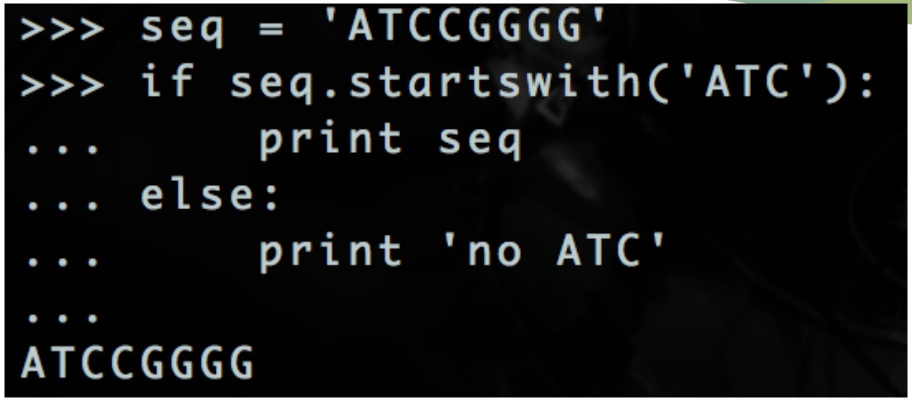

if-else statement
if
True
False
else

If Else Statements.

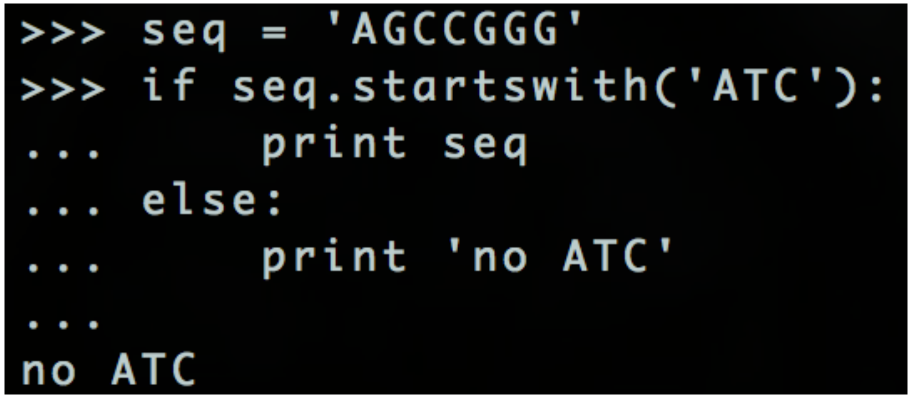


```
>>> myNumber = 5
>>> if myNumber >= 2:
...     print('big number')
... else:
...     print('small number')
...
big number
```

If Else Statements.



```
>>> seq = 'ATCCGGGG'
>>> if seq.startswith('ATC'):
...     print seq
... else:
...     print 'no ATC'
...
ATCCGGGG
```



```
>>> seq = 'AGCCGGG'
>>> if seq.startswith('ATC'):
...     print seq
... else:
...     print 'no ATC'
...
no ATC
```

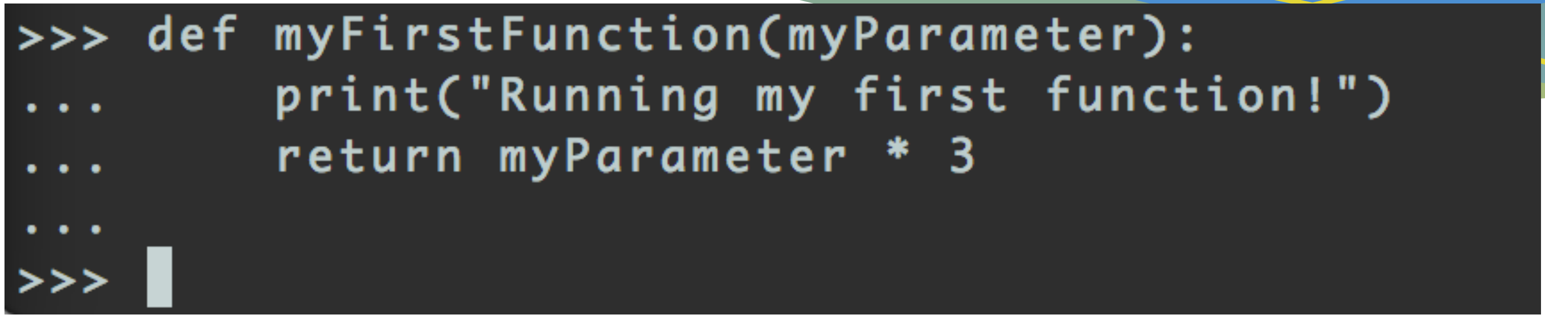
Write Code Once and Reuse

FUNCTIONS

- Might want to run the same code on million of sequences.
- Write a function once and use it whenever you have to do that task.

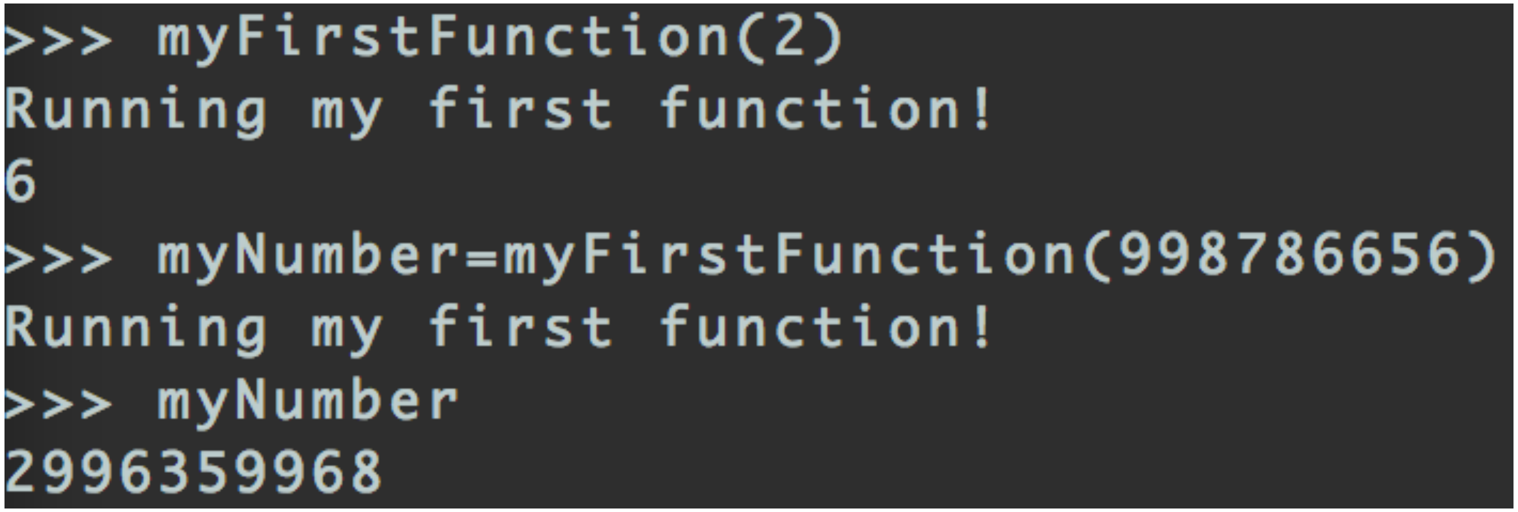
```
def function_name(parameter1,parameter2):
        any
        code
        here
        return result_of_function
```

Write Your First Function



```
>>> def myFirstFunction(myParameter):
...     print("Running my first function!")
...     return myParameter * 3
...
>>> 
```

Returned values can be assigned to variables outside functions.



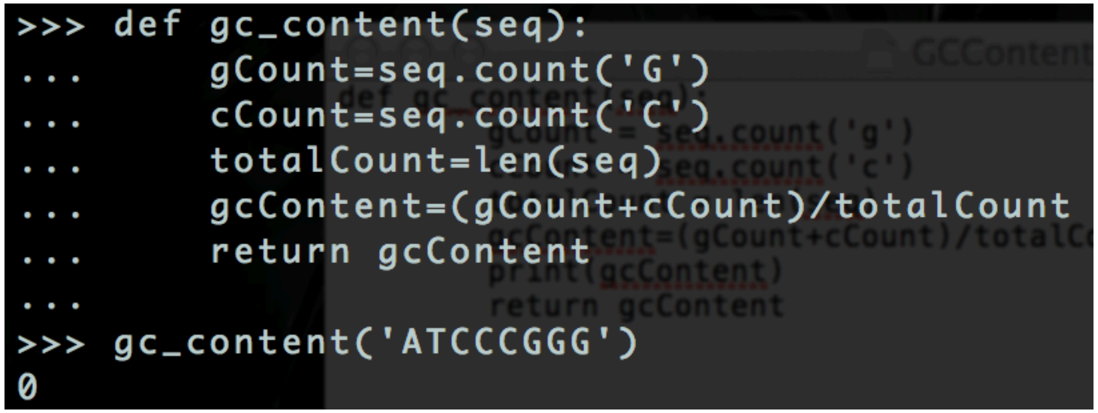
```
>>> myFirstFunction(2)
Running my first function!
6
>>> myNumber=myFirstFunction(998786656)
Running my first function!
>>> myNumber
2996359968
```

Your First USEFUL Function

Calculating GC Content:

- Let’s write pseudocode

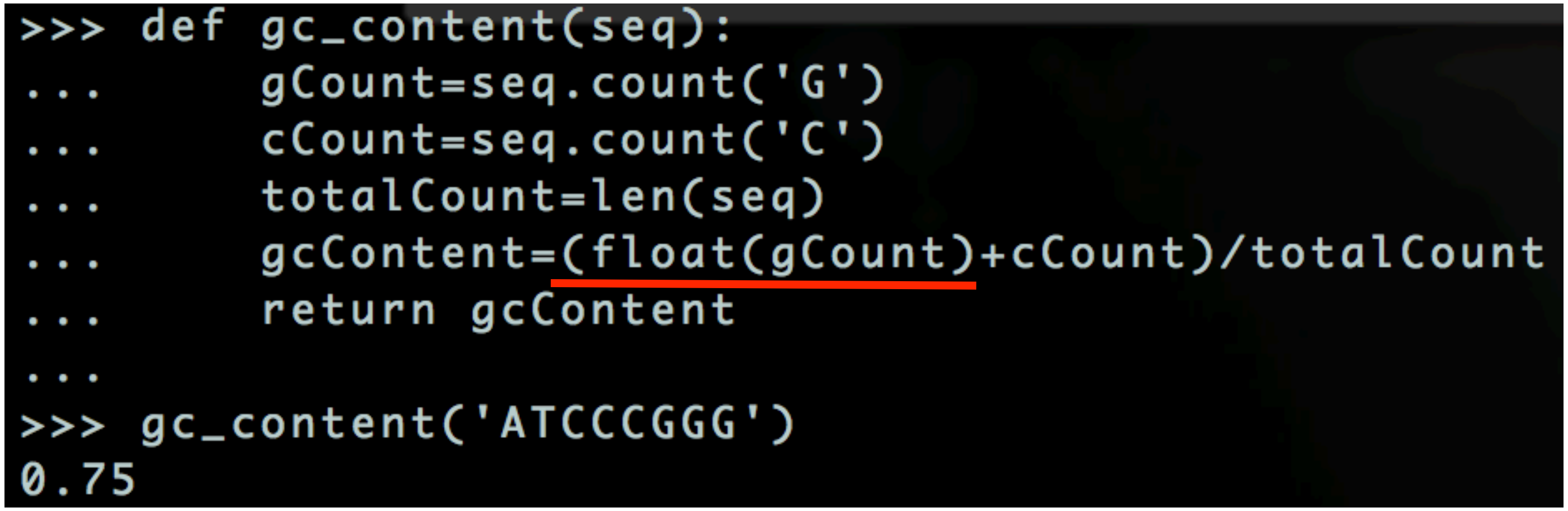
```
Input is a sequence
        count G occurrences
        count C occurrences
        sum G and C occurrences
        divide the sum by the total sequence length
        return the result
```



```
>>> def gc_content(seq):
...     gCount=seq.count('G')
...     cCount=seq.count('C')
...     totalCount=len(seq)
...     gcContent=(gCount+cCount)/totalCount
...     return gcContent
...
>>> gc_content('ATCCCGGG')
0
```

Who gets the right result?

Remember the integer division problem?

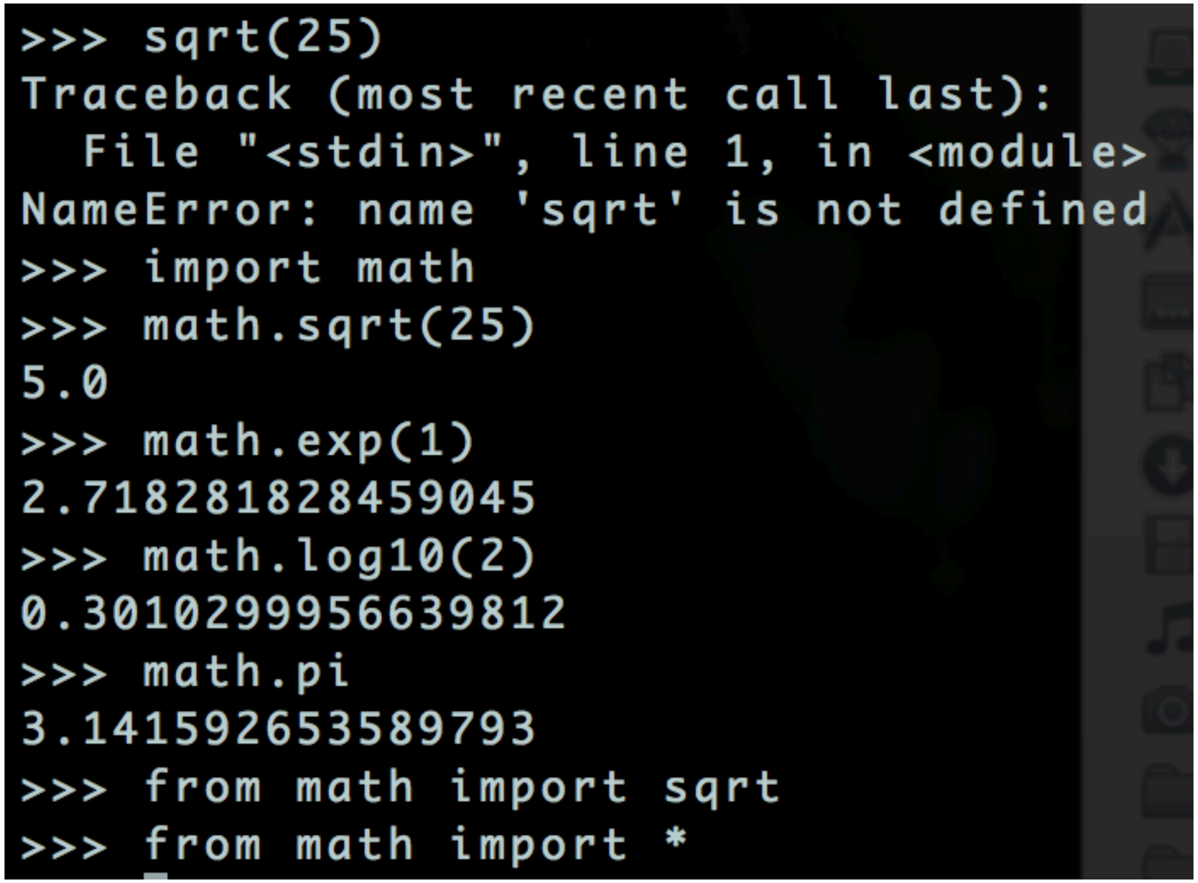


```
>>> def gc_content(seq):
...     gCount=seq.count('G')
...     cCount=seq.count('C')
...     totalCount=len(seq)
...     gcContent=(float(gCount)+cCount)/totalCount
...     return gcContent
...
>>> gc_content('ATCCCGGG')
0.75
```

3 Ways to Run Python Code

* Interactive environment
  * What we've been doing
* **Modules**
  * Groups of functions loaded into the interactive python session.
* **Scripts**
  * Run python code from outside the interactive python session. Typed into the Windows/OS X/Unix command line.

Importing Generic Modules



```
>>> sqrt(25)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> import math
>>> math.sqrt(25)
5.0
>>> math.exp(1)
2.718281828459045
>>> math.log10(2)
0.3010299956639812
>>> math.pi
3.141592653589793
>>> from math import sqrt
>>> from math import *
```

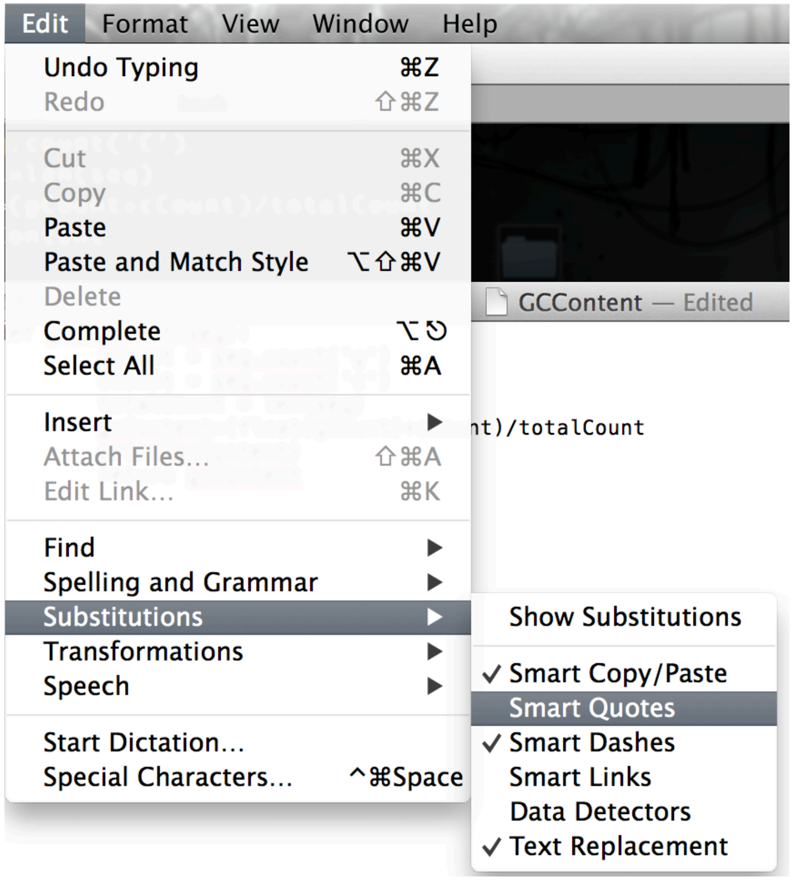
import MODULENAME

from MODULENAME import FUNCTION

from MODULENAME import *
(everything - caution)

Working in a Text Editor

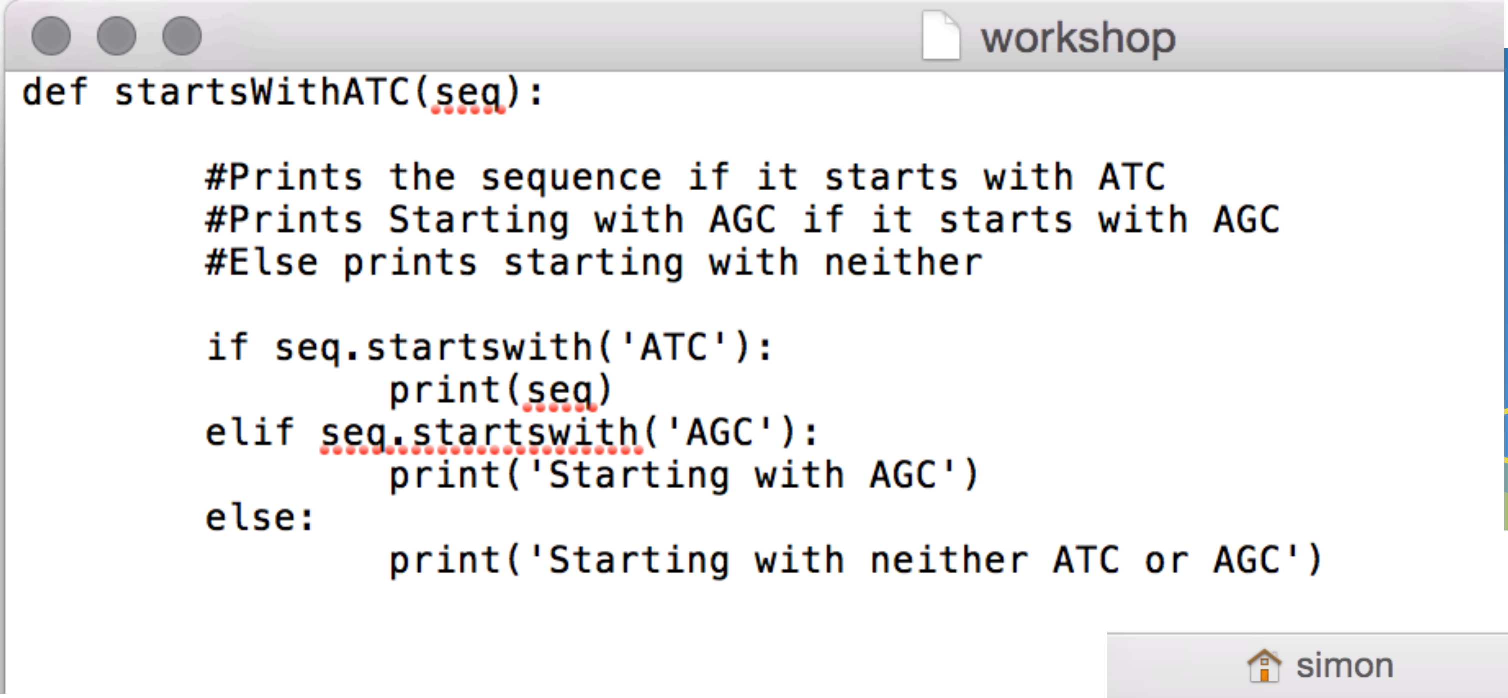
- Typing everything into the python environment can be inconvenient.
- Write your code into a text document
- Use a basic text editor
  - Notepad (windows)
  - TextEdit (Mac)
  - emacs/Vim!
- Save with a .py extension.
- Careful with TextEdit on Mac!



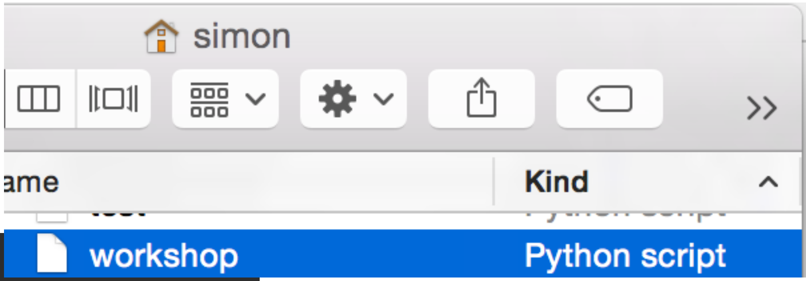
Combining Everything We've Learnt

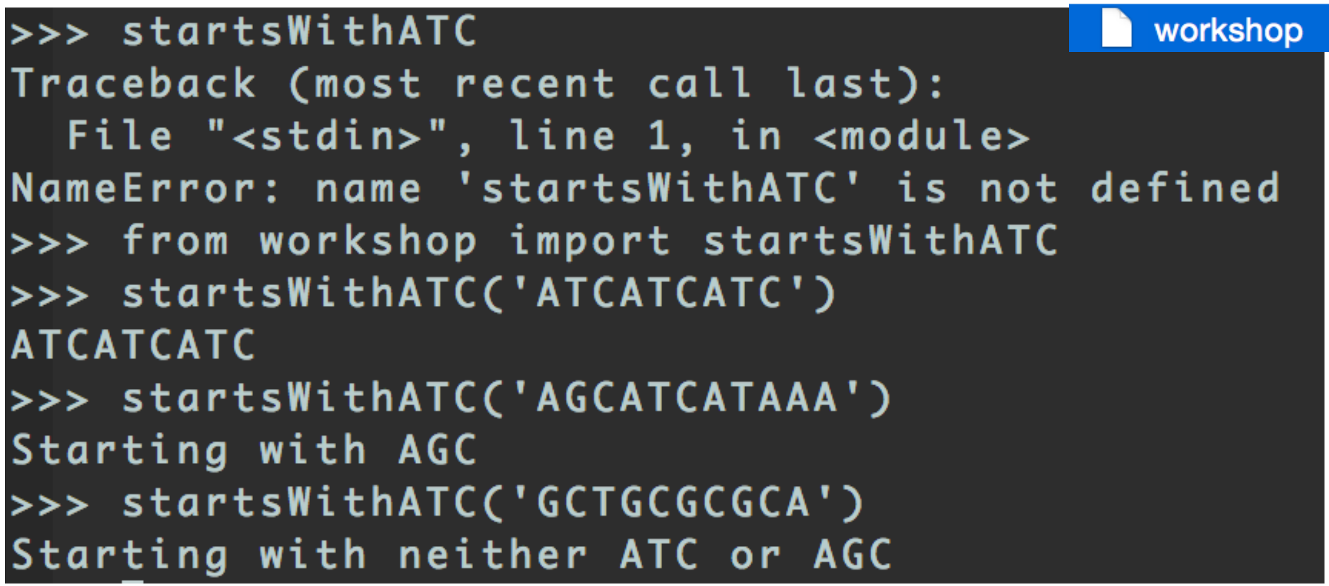
Let's write a function that:

* Takes a sequence as a parameter
* Prints the sequence if it starts with ATC
* If the sequence starts with AGC prints 'Starting with AGC'.
* If the sequence starts with neither print 'Starting with neither ATC or AGC'.



```
def startsWithATC(seq):

        #Prints the sequence if it starts with ATC
        #Prints Starting with AGC if it starts with AGC
        #Else prints starting with neither

        if seq.startswith('ATC'):
                print(seq)
        elif seq.startswith('AGC'):
                print('Starting with AGC')
        else:
                print('Starting with neither ATC or AGC')
```





```
>>> startsWithATC
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'startsWithATC' is not defined
>>> from workshop import startsWithATC
>>> startsWithATC('ATCATCATC')
ATCATCATC
>>> startsWithATC('AGCATCATAAA')
Starting with AGC
>>> startsWithATC('GCTGCGCGCA')
Starting with neither ATC or AGC
```