>>> # An Interactive Session in the Python Shell.
>>> # When you type a statement in the Python Shell,
>>> # the statement is executed immediately. If the
>>> # the statement is an expression, its value is
>>> # displayed.
>>> # Lines that begin with the symbol #, like this one,
>>> # are comments--these are ignored by the interpreter.
>>> # Expressions.
>>> # We can use Python as a calculator. The expression below
>>> # calculates the wind-chill temperature (apparent temperature)
>>> # on a cold day, when the temperature is 10 degrees F and
>>> # the wind is blowing at 25 miles per hour. (Formula from
>>> # Wikipedia):

>>> 35.74 + 0.6215 * 10 - 35.75 * (25 ** 0.16) + 0.4275 * 10 * (25 ** 0.16)
-10.723832362544272

>>> # Note the symbols for addition, subtraction, multiplication
>>> # and exponentiation. Make sure you understand the precedence
>>> # rules for these operations, which govern when you need to
>>> # place parentheses. In fact, the parentheses in the above
>>> # expression are unnecessary! But I put them there to make the
>>> # formula more readable. If you are in doubt, you can always
>>> # insert parentheses.
>>> # We can also do division of course:

>>> 3 / 4
0.75

>>> # VARIABLES AND ASSIGNMENT

>>> # If you have a complicated computation to perform, particularly
>>> # one in which you are likely to re-use values or formulas, it is
>>> # more useful to break the computation into smaller pieces, saving
>>> # intermediate results by assigning them to variables. We will
>>> # discuss the rules for variable names in class. Here is the
>>> # same calculation carried out in this way.

>>> c1 = 35.74
>>> c2 = 0.6215
>>> c3 = 35.75
>>> c4 = 0.4275
>>> wind_speed = 25
>>> temp = 10
>>> wind_factor = wind_speed ** 0.16
>>> windchill_temp = c1 + c2 * temp - c3 * wind_factor + c4 * temp * wind_factor
>>> windchill_temp
-10.723832362544272
>>> # I had to type a lot more, but I can easily re-use the formula
>>> # with different values for temperature and speed if I have to perform
>>> # more calculations, and it is easier to read.

>>> WINDCHILL_TEMP
Traceback (most recent call last):
  File "<pyshell#54>" , line 1, in <module>
    WINDCHILL_TEMP
NameError: name 'WINDCHILL_TEMP' is not defined

>>> # You will see a lot of error messages. Here I asked for the value
>>> # of a variable WINDCHILL_TEMP, but was told that this variable is not
>>> # defined. Case matters!

>>> # In the next two errors, I break the rules for forming variable names,
>>> # resulting in new error messages:

>>> 2c=3
SyntaxError: invalid syntax
>>> if=3
SyntaxError: invalid syntax
>>> # But this is ok (or, at any rate, it's legal.)
>>> If=3
>>> If
3

>>> # INTEGER DIVISION

>>> # There is a second kind of division, denoted with a double slash:

>>> 4//3
1

>>> # This just throws away the fractional part of the quotient and
>>> # gives the integer part as the value. Be careful when you apply
>>> # this to negative arguments---the answer may not be what you expected:

>>> -4//3
-2

>>> # There is a complementary operation.

>>> # When you divide two integers this way, there are two results---the
>>> # quotient, as we calculated above, and the remainder:

>>> 4%3
1
>>> -4%3
2
>>> # We usually pronounce this operation 'mod': 'four mod three.'
# TYPES

Expressions in Python have 'types', and we have just seen two different types of values: float and int. The calculations below should give you a sense of this.

```python
>>> type(4)
<class 'int'>
>>> type(4.2)
<class 'float'>
>>> type(4.0)
<class 'float'>
>>> type(4/3)
<class 'float'>

>>> type(4//3)
<class 'int'>

>>> z=4*5
>>> type(z)
<class 'int'>

>>> type(4/2)
<class 'float'>

>>> type(4**2)
<class 'int'>

>>> type(4**(-2))
<class 'float'>
```

#Observe that 4 and 4.0 have different types. These numbers are represented differently in the computer’s memory.

# Generally speaking, if an operation on two integers always yields an integer, and you apply this operation to two values of type int, the result has type int. But if the operation can produce a non-integer (as in division), then the type of the result is float, even if, mathematically, the result is an integer. Exponentiation is a somewhat peculiar example: If the base has type int and the exponent has type int and a non-negative value, then the result has type int.

# A few more examples:

```python
>>> y=4
>>> type(y)
<class 'int'>
>>> y=y+2.1
>>> y
6.1
>>> type(y)
<class 'float'>
```

#The assignment statement y=y+2.1 looks odd if you insist on reading
>>> #the symbol '=' as 'equals', because this 'equation' makes no sense.
>>> #But what the expression means is, take the value of y, add 2.1 to it,
>>> #and assign the result to the variable y. Before the assignment
>>> #statement is executed, y has type int and 2.1 has type float. In order
>>> #to add these, Python has to convert the value of y to a float. The
>>> #result is a float, so when this result is assigned to y, the type of y
>>> #changes.
>>> 
>>> #Python has a number of built-in mathematical functions (trig functions,
>>> #logarithms, etc.) including a square root function denoted sqrt. But
>>> #you have to be careful about how you invoke it:

>>> sqrt(3)
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    sqrt(3)
NameError: name 'sqrt' is not defined

>>> #sqrt lives in the math library, so you need to call it as math.sqrt:

>>> math.sqrt(3)
1.7320508075688772
>>> math.sqrt(4)
2.0

>>> #That's 2.0, not 2: the result is a float.

>>> #STRINGS

>>> x='hello'
>>> x
'hello'
>>> x=
'hello'
>>> x=
"hello"
>>> x
'hello'

>>> #You can use either single or double quotes, or even
>>> #triple quotes

>>> x="""hello"
>>> x
'hello'
>>> type(x)
<class 'str'>
>>> # Slices
>>> x[0]
'h'
>>> x[4]
'o'
>>> x[5]
Traceback (most recent call last):
  File "<pyshell#16>" , line 1 , in <module>  
x[5]
IndexError: string index out of range

>>> # Whoops!
>>> x[1:3]
'el'
>>> # Observe that this gives you the substring between indices 1 and 2
>>> # inclusive (not between 1 and 3)
>>> # Some shortcuts:
>>> x[:3]
'hel'
>>> x[2:]
'loo'
>>> x[:=-1]
'hell'

>>> # 'Arithmetic with strings'
>>> 'hello'+'','+','goodbye'
'hello,goodbye'
>>> "2"+"2"
'22'
>>> '22' * 5
'2222222222'

>>> # EXPLICIT TYPE CONVERSION
>>> # Sometimes you will be given numbers represented as strings,
>>> # but you want to do normal numerical arithmetic with them.

>>> x='32.5'
>>> y='102'
>>> x+y
'32.5102'

>>> # That's not what we want!
>>> float(x)
32.5

>>> # The function float takes a string that can be interpreted as
>>> # a float, and returns that number is a value.

>>> # So you can convert first, then add:

>>> z=float(x)+float(y)
>>> z
134.5
>>> # If you need to, you can convert this back to a string:

>>> str(z)
'134.5'

>>> #print function
>>> # You can give this a string or a number as an argument, even
>>> # a sequence of arguments separated by commas.

>>> print("hello")
hello

>>> Observe that we don’t get the quotation marks when we print like this.

>>> print('134.5',27,"'bye, now")
134.5 27 'bye, now