before we start, everybody make sure python is installed
`which python`

start the python command prompt
`python`

ctrl+d to exit the command prompt

who has got any kind of programming experience?

- used a for() loop
- used C
- used C++
- used Java
- used matlab
- used python

python is a non-compiled or *interpreted* language

- doesn't produce machine-code (directly)
- sends instructions to an *interpreter*
- think of it as a virtual machine or an emulator
- black box that does all the heavy lifting

'heavy lifting':

- declaring a variable x, let it hold the value '0.5'
- how do we do this in C?
- what *type* is 0.5? (float)
- `float x = 0.5`
- let's see how to do it in python

```
x = 0.5
```

how do we know whether this worked? (print)

```
print x
```

how about printing "hello world"?
will this work?:
`print hello world`
no, because we want to literally print the words "hello[space]world" and not the value of a variable *hello* and/or *world*
`print "hello world"`

let's define "hello world" as a variable that we can call later

again, how do we do this in compiled languages?

```
string s = “hello world”;    // java
```

or

```
char[12] s = “hello world”;  // C
```

let’s do this in python:

```
s = “hello world”
print s
```

python is a *weakly typed* or *dynamically typed* language
you don’t need to tell python what type of data a variable holds, it figures it out for you.
in fact, we can even re-declare x to hold value “hello world”

```
x = “hello world”
```

let’s start doing some simple maths:

```
a = 1
b = 2
print a+b
print a*b
```

declare two strings

```
s1 = “cheese”
s2 = “cake”
dinner = s1+s2
print dinner
```

operator *overloading* (what is an operator? what is overloading?)
(operator does different things depending on the input types)

```
number = 10
```

what do you expect this to produce (aside on what a comment is):

```
print dinner*number    # result: ???
```

how about:

```
print dinner+number    # result: ???
```

syntax error

not showing this to confuse you. to emphasise that python does *most* of the work for you, but this can result in some ambiguities, or surprising results. just because we don’t see or define the types, doesn’t mean the values held by the variables don’t *have* types.

what if we want to produce the string “cheesecake10”?

we *convert* the type (or *cast* the variable):

```
print dinner + str(number)
```

What is ‘int’?
Integer -- i.e. no decimal point (what do we call number with decimal point? float)

```
print number / 20 #expect: ???
```

Computer is wrong? no - this is correct answer for *integer division*. but most of us were probably expecting (what?) 0.5

```
print number / 20.0
```
(or `print number / float(20)`)

Let’s go through some more operators:

Remainder `print 10 % 3`
Exponent `print 2 ** 3`
Exponent `print 16 ** 0.5`
Remainder `print 10 % 3`

Complex numbers:
```
result = print (1+4j) + (2+5j)
```
What did I do wrong?
```
result = (1+4j) + (2+5j)
print result.real
print result.imag
```

In-place operator
```
count = 0
count = count + 1
print count
count += 1
```

Works with other operators
```
count /= 2
print count
number = 10
number %= 3
```

Conditionals
```
9<10
```

```
9<9
9<=9
9==9
9!=9
```

When do we use these? Conditional statements, if clauses, for loops (will get to these today or next week)

Exercise:

1) declare these variables:

```
s1 = ”dog”
s2 = ”cat”
n1 = 1
n2 = 4
n3 = “0”
```

2) use these variables ONLY (i.e. no literal numbers or new strings) to produce the following output:
   a. ‘catdog’
   b. ‘cat01’
   c. ‘cat256’
   d. ‘dogdogdogdogcat’
   e. ‘dog255catcatcatcatcat’

Solutions:

```
s2+s1
s2+n3+str(n1)
s2+str(n2**n2)
s1*n2+s2
s1+str(n2**n2-n1)+cat*n2+cat
```

Lists

```
breakfast = [‘eggs’,’bacon’,’pancakes’]
print len(breakfast)
print len([])
```
(what is [] ?)

how do we get ‘eggs’ from the list?
```
print breakfast[0]
```
indexing from 0

(beware, matlab counts from 1, but most sensible languages count from 0)

```
print breakfast[len(breakfast)]    # (beware of brackets]
print breakfast[len(breakfast)-1]
print breakfast[-1]
print breakfast[-4]
```

you can change items in a list (i.e. lists are *mutable*)

```
breakfast[1] = ‘cheesecake’
print breakfast
```

you can add different types of values to the list
```
mixed = [‘stuff’, 1, 2.5, True]

del mixed[3]
```

del is an in-built python *function*

```
mixed.remove(‘stuff’)
```

remove() is a list *method*

```
breakfast[4] = ‘syrup’
breakfast.append(‘syrup’)
breakfast.append(‘syrup’)
breakfast.append(‘syrup’)
breakfast.count(‘syrup’)
breakfast.index(‘eggs’)
breakfast.sort()  # why empty parantheses? (sort is a method)
breakfast.reverse()
```

```
‘cheesecake’ in breakfast
‘bacon’ in breakfast
```

creating lists of numbers:

```
print range(10)
print range(5,10)
print range(1,10,3)
```

let’s do something more fancy: iteration

```
for food in breakfast:
      print food
```

Did it work for everyone? (probably not)

Where other languages use braces, python uses *indentation*

Try again:

```
for food in breakfast: [enter]
[tab]print food [enter]
[enter]
```

The command-prompt is a great place for quick experiments using python.
In most cases we want to write *scripts*, which will be our next topic.

Scripts

```
string = ‘hello,1,2,3’
print string.split(‘,’)
s1,s2,s3,s4 = string.split(‘,’)
```

CTRL-D to exit python

nano exercise1.python

```
total = 0
for i in range(1,100):
        total += i
print total
```

make a file data.txt with following content

```
Date,Species,Count
2012.04.28,marlin,2
2012.04.28,turtle,1
2012.04.28,shark,3
2012.04.27,marlin,4
```

```
source = open('data.txt','r')
for line in source:
      print line
source.close()
```

Exercise: count all marlins