# Python lab session 1

Dr Ben Dudson, Department of Physics, University of York

28th January 2011

## Python labs

Before we can start using Python, first make sure:

- You can log into a computer using your username and password
- You have a CD with portable Python on it. This should work on all Windows computers, so you should be able to use this on your laptops or at home as well

If you have any trouble with this, ask one of the demonstrators.

---

As you go through this lab script, you'll see tasks to complete in these boxes. Each week, create a new Word document to write your answers in. Put your answer to each task in a different section. At the end of each lab session, email this document to `phys-python@york.ac.uk`

---

If you have problems at any point, or don't understand anything in this lab script then please ask one of the demonstrators. Each section is independent, so if you get stuck with one problem you can try another one and come back to it later.

## Getting started

Start by creating a new folder called `python` in your `My Documents` directory, where you can put all your Python programs. From now on I'll refer to this as your `python` directory.

Put the Python CD in the drive, and double-click on SPE-portable. SPE is a so-called "Integrated Development Environment", which allows you to write and run Python programs interactively, and gives you help on functions and finding mistakes. It is possible to use something like Notepad to write Python programs, but using programs like SPE make programming easier and quicker.

As is traditional, we'll start with a "Hello, World!" program. In the editing window on the right, enter the following:

```
print "Hello, World!"
```

Save this file as "hello.py" in your `python` directory (using the usual File → Save As menu). **Note**: All python files should end in ".py" to avoid any problems. This just makes it clear to the computer that it's a Python program and not just an ordinary text document.

To run this program, press `Ctrl-Shift-R`. At the bottom of the window, you should see the program run and print out "Hello, World!". Congratulations! You've written your first Python program. As you go through this lab script, you should create a new python file for each task.

# Calculations

Python includes the following basic operations

| | | |
|---|---|---|
| Power | A ** B | $A^B$ |
| Multiply | A * B | $A \times B$ |
| Divide | A / B | $A/B$ |
| Add | A + B | $A+B$ |
| Subtract | A − B | $A-B$ |

**Task 1** Calculations (15%)

Write programs to read in a value of $x$, then calculate the following expressions:

`(a)` $\dfrac{x-2}{x^2+1}$ `(b)` $2^{x+1}$ `(c)` $x^{-2x}$

For each program, you can start with the following code

```
x = 1.638453
print #####
```

where you need to replace ##### with your code. Copy and paste your 3 programs into your lab document, and give the result for each one.

**Task 2** Create a function (15%)

Create a function ctof which converts temperature from Celsius to Fahrenheit using

$$F = \frac{9}{5}C + 32$$

Check that `ctof(21.)` produces the result `69.8`. Start with the following incomplete code:

```
def ctof(C):
  return # Missing code

print ctof(21.)
```

Copy and paste your program into your lab document.

One of the advantages of using functions is that once we've got it working once, we can reuse it many times.

**Task 3** Looping (20%)

Write a program which uses your ctof function to print out a table of Fahrenheit and Celsius. Starting at $0^o$C, go up in steps of $10^o$C to $100^o$C. Print out column headers so your output looks something like

```
Celsius     Fahrenheit
  0.0          32.0
 10.0          50.0
```

(don't worry about getting the columns to line up) Copy and paste your program and output table into your lab document.

# Recursive functions

We can call functions to get results, but functions can also call themselves. These are then called **recursive** functions, and are often a concise way to define functions.

If a function is calling itself, then it needs some way to stop: otherwise the function will just keep calling itself forever. We therefore need to define some cases which can be calculated without calling itself. For example, the function for calculating the factorial was:

```
def factorial(N):
  if N < 2:
    return 1
  else
    return N * factorial(N-1)
```

Here we first say that the factorial of any number less than 2 is 1. We can then define all the other results in terms of these cases. This is very similar to proof by induction in mathematics.

Another example of a problem which can be solved elegantly using recursive functions involves the Fibonacci sequence of numbers. This is

$$\mathrm{F} = 1,1,2,3,5,8,13,21,34,55,...$$

The first two are defined as 1, but then all others are the sum of the two before it:

$$f(N) = \begin{cases} 1 & \text{if } N < 3, \\ f(N-1) + f(N-2) & \text{otherwise} \end{cases}$$

---

**Task 4** Fibonacci sequence (15%)

---

a) Write a function called fibonacci to calculate the $N^{th}$ Fibonacci number. Use it to calculate the 10th, 20th and 30th Fibonacci number and put those (along with your program) into your lab document.

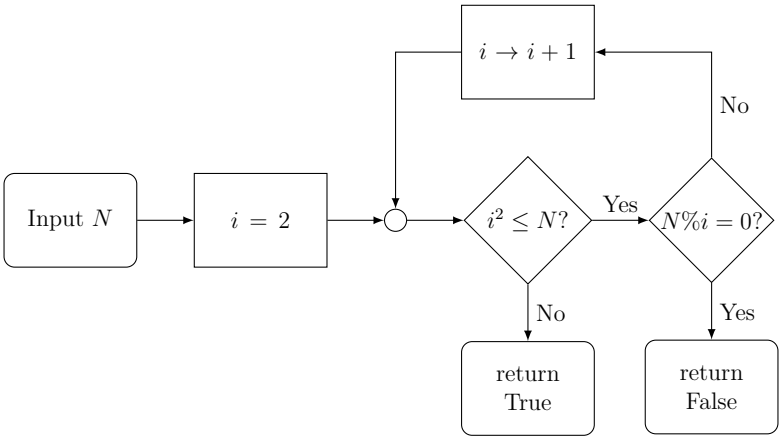
b) Why is the recursive method not very efficient for this problem? Write your answer in your lab document

---

# Prime numbers

In addition to the operations `**`, `*`, `/`, `+` and `-`, Python has another operation on numbers: the remainder or **modulus**. When you divide one integer by another, you get an integer result, so 7/3 gives the result 2, with remainder 1 (since $7 = 2 \times 3 + 1$). To get this remainder in Python, you can use the % operation, so 7 % 3 gives the remainder 1. One use for this is to test if x is a multiple of y, because this only happens when the remainder is zero.

A prime number is one which can only be divided by 1 and itself. The following flow chart gives an algorithm for checking if a number is prime or not.




In Python, we can use True and False as values. When we have an **if** statement like

```
if A > 4:
    print "A is greater than 4"
```

what happens is that Python compares A against 4 to calculate (A>4). This produces a result True or False. The **if** statement runs the following commands only if it is given True. Therefore, we could write

```
if True:
    print "This code will always run"
```

The **if** statement just needs a True or False value, but this could also come from a function. Therefore we could also write

```
def test(number):
    if number > 4:
        return True
    else
        return False

if test(A):
    print "A passes the test"
```

Since the result of number $> 4$ is True or False, we could even write this as

```
def test(number):
    return number > 4

if test(A):
    print "A passes the test"
```

---

**Task 5** Prime number function (15%)

Write a function to implement the algorithm above: given a number $N$, return either True or False. You can start with the following

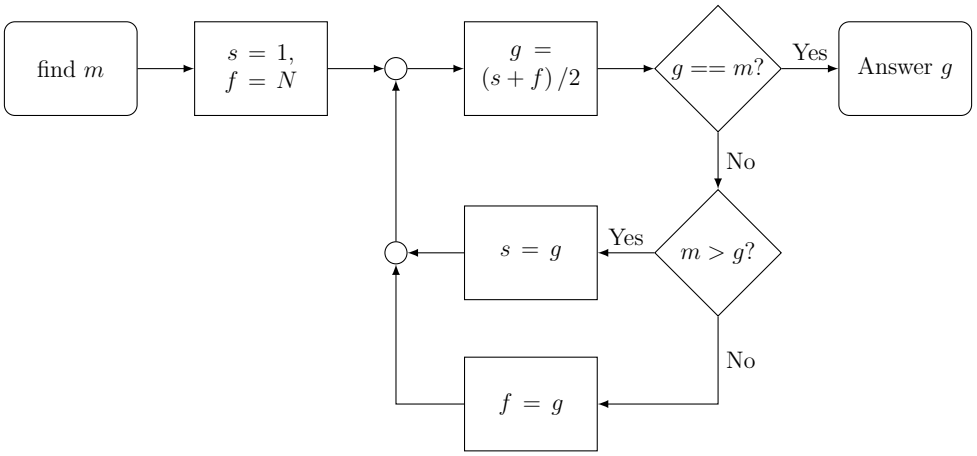
```
def isPrime(N):
    #####

i = 2011
if isPrime(i):
    print "Prime number"
else:
    print "Not a prime number"
```

Is 2011 a prime number or not? Answer in lab document along with your program

# Divide and conquer

In the problem sheet, the following method was used to guess a number $m$ between 1 and $N$.




This works by searching for an answer between $s$ and $f$. Each time around the loop, it looks at the half-way point $g = (s + f)/2$. If this number is too high, then the result must be in the lower half of the range, and if it's too low then the answer must be in the upper half of the range. The range $s \to f$ is adjusted and the loop begins again.

**Task 6** Square root function (10%)

Adapt the above search algorithm to find the square root $\sqrt{x}$ of a number $x$ i.e the number $g$ which satisfies $g^2 = x$. The size of the range $(f - s)$ is the error in the result; keep searching until this error is less than $10^{-5}$. Write this as a function sqrt which takes a number as input and returns the result. You should start with the following code:

```
def sqrt(x):
    #####

x = 42.4242
print "square root of x is ", sqrt(x)
```

Copy and paste the program and the answer into your lab document

The function you have just written can be generalised to find the $y^{th}$ root of a number i.e. the number $r$ such that $r^y = x$. This needs to be a function of two parameters ($x$ and $y$).

**Task 7** General root function (10%)

Adapt your square root function to calculate the $y^{th}$ root of x. You should start with the following code:

```
def root(x,y):
    #####

x = 53.936
y = 3
r = root(x,y)
print "the %fth root of %f is %f" % (y, x, r)
```

Note that we've used a slightly different way to print out our results. We could have written it by separating the numbers and text with commas:

```
print "the ",y,"th root of ",x, "is", r
```

This is good for small number of outputs, but can become difficult to read. The alternative format uses a %f to mean a floating point number, and takes them in order from the brackets at the end of the line. Hence the first %f is replaced with y, the second with x, and the third with r.

Both formats for printing output are valid, it's just a matter of personal preference which you use. In general the % method allows more control over what the output looks like. For example you can use %.2f to print only 2 decimal places, and %E to print in scientific notation (e.g. 1.32e-2).