Python lab 1: Functions

Dr Ben Dudson

Department of Physics, University of York

28th January 2011

http://www-users.york.ac.uk/∼bd512/teaching.shtml

From last time...

Last lecture covered the basics of programming in Python

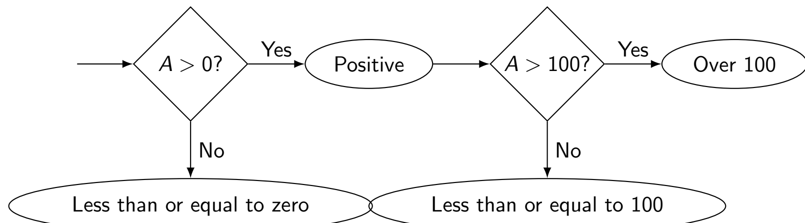
- Variables, calculations and how computers handle numbers
- **if** statements and making decisions (conditionals)
- **while** statements for repeating commands (loops)
- The main things to watch out for in Python: **case sensitive** (capitals matter), and **indentation** so the space in front of commands matters.

Today: Quick lecture, then Python lab problems

Basic python revisited

Last time we saw that in Python we can use **if** statements (conditionals) to make decisions, for example:

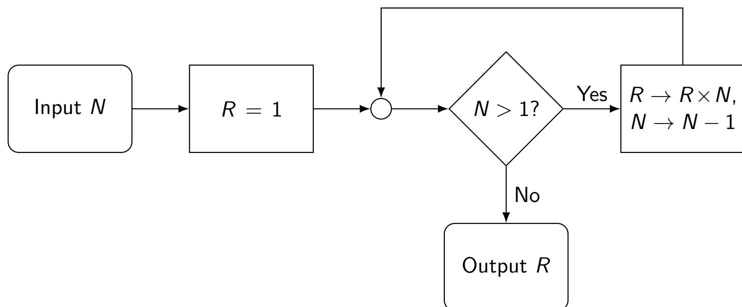
```
if A > 0:
    print "positive"
    if A > 100:
        print "Over 100"
    else:
        print "Less than or equal to 100"
else:
    print "Less than or equal to zero"
```



Basic python revisited

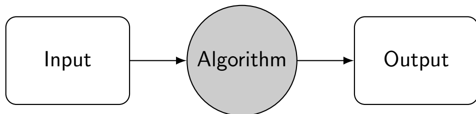
We can also use **while** to repeat commands, for example calculating the factorial $N!$ of a number N:

```
N = 10
R = 1
while N > 1:
    R = R * N
    N = N - 1
print "Result is ", R
```



Functions

In computing, we often want to take some input values, perform some set of commands on it, and produce some result.



In mathematics, this is called a **function**

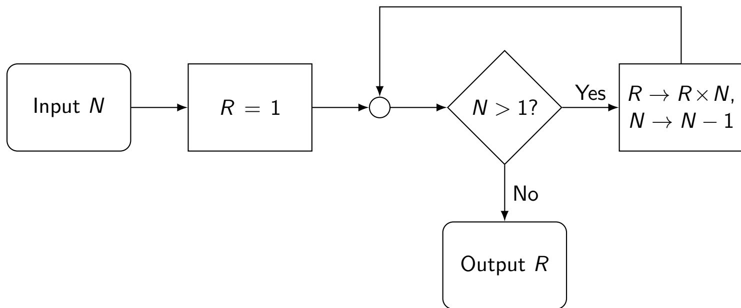
$$y = f(x) \qquad x \rightarrow_f y$$

Our entire program is one big function, but we can break big programs into steps, each of which might be another function.

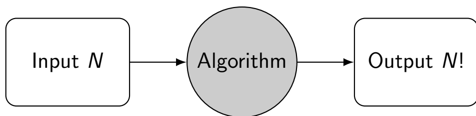
Combining functions together to make more complicated functions is one of the most powerful ideas in computing

Functions

Example: Our factorial calculation



can be represented as a function



Functions in Python

Creating functions in Python is done using **def**

```
def factorial(N):
    R = 1
    while N > 1:
        R = R * N
        N = N - 1
    return R
```

Functions in Python

Creating functions in Python is done using **def**

```
def factorial(N):
    R = 1
    while N > 1:
        R = R * N
        N = N - 1
    return R
```

We can now use our function as part of a program:

```
x = 10
result = factorial(x)
print "Result is ", result
```

Functions in Python

Creating functions in Python is done using **def**

```
def factorial(N):
    R = 1
    while N > 1:
        R = R * N
        N = N - 1
    return R
```

We can now use our function as part of a program:

```
x = 10
result = factorial(x)
print "Result is ", result
```

or

```
print factorial(10)
```

Function arguments

When creating a function, the format is

```
def function_name(variable1, variable2, ...):
    commands
```

Function arguments

When creating a function, the format is

```
def function_name(variable1, variable2, ...):
    commands
```

In the case of the factorial function, we only had one variable

```
def factorial(N):
    commands
```

Function arguments

When creating a function, the format is

```
def function_name(variable1, variable2, ...):
    commands
```

In the case of the factorial function, we only had one variable

```
def factorial(N):
    commands
```

but we can also create functions with two or more variables, separated by commas.

```
def myFunction(x, y):
    commands
```

These input variables are called **arguments**. When we use a function, we say we're **calling** a function, **passing** the arguments.

Function arguments

- An important point is that when we use a function we don't need to know what the arguments are called inside the function
- It should be possible to use a function without knowing anything about how it works inside
- When you call a function, it takes the values you give it and gives them names based on their position (first argument, second argument etc.)

When calling a function, only the position of the argument matters, not its name

Returning results

- When a function has calculated the result, it needs to **return** it so the result can be used. In our factorial example, this could only happen in one place:

```
def factorial(N):
    R = 1
    while N > 1:
        R = R * N
        N = N - 1
    return R
```

- When the function reaches a **return** statement, it takes the value(s) which follow and sends them back to the caller.

Returning results

- When a function has calculated the result, it needs to **return** it so the result can be used. In our factorial example, this could only happen in one place:
- When the function reaches a **return** statement, it takes the value(s) which follow and sends them back to the caller.
- In our example converting marks to letters, we could write a function

```
def markToLetter(mark):
    if mark > 70:
        return "A"
    elif mark > 40:
        return "B"
    else:
        return "C"
```

Recursive functions

If we can call functions, why shouldn't functions call themselves?
How does this version of factorial work?

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

Recursive functions

If we can call functions, why shouldn't functions call themselves?
How does this version of factorial work?

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

This just says that the factorial of anything less than 2 is 1, and the factorial of anything else is N times the factorial of $N-1$. For example:

```
result = factorial(5)
result = 5 * factorial(4)
```

If we can call functions, why shouldn't functions call themselves? How does this version of factorial work?

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

This just says that the factorial of anything less than 2 is 1, and the factorial of anything else is N times the factorial of $N - 1$. For example:

```
result = factorial(5)
result = 5 * factorial(4)
result = 5 * 4 * factorial(3)
result = 5 * 4 * 3 * factorial(2)
result = 5 * 4 * 3 * 2 * factorial(1)
result = 5 * 4 * 3 * 2 * 1
```

Functions summary

- Functions are a way to simplify programs by hiding the details of calculations
- When we create a function (using **def**), we specify how many inputs there are, and give them names which are used **inside** the function
- When we use (call) a function, we just need to know how many inputs there should be, and what order to give them in
- This allows us to write a function once, but use it many times
- Functions can even call themselves, a technique which can be used to replace loops

Functions example 1

Exercise: What does the following program print out?

```
x = 3
def double(x):
    x = x * 2
    return x
print x
```

Functions example 1

Exercise: What does the following program print out?

```
x = 3
def double(x):
    x = x * 2
    return x
print x
```

Answer: 3

The function double is defined (created), but is never run so doesn't do anything

def defines a function, but doesn't do any calculations.
Calculations are only performed when a function is called (used)

Functions example 2

Exercise: What does the following program print out?

```
x = 3
def double(y):
    y = y * 2
    return y

y = double(x)
x = double(y)
print x, y
```

Functions example 2

Exercise: What does the following program print out?

```
x = 3
def double(y):
    y = y * 2
    return y

y = double(x)
x = double(y)
print x, y
```

Answer: 12 6

Inside the function double, y is the name for the input value. Outside the function, we create a different variable called y which is set to double(3)

> When you use a variable name inside a function, Python first checks the list of inputs, then looks outside the function

Key points and Glossary

- Python is case sensitive, so watch out for capital letters
- Indentation matters, so always use spaces **or** tabs, not a mix
- Put a colon at the end of **if**, **elif**, **else**, **while**, and **def** statements, and increase indentation
- Functions are created using **def**. They can have one or more parameters (arguments) separated by commas, and **return** the result when they're finished