

Introduction to programming for Geoscientists

Revision Lecture 1

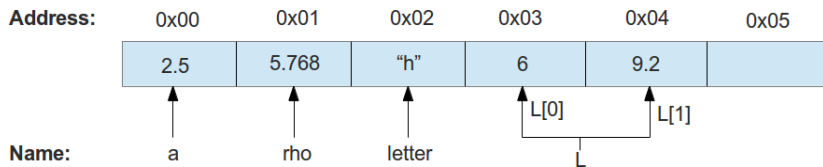
Christian Jacobs

c.jacobs10@imperial.ac.uk

Variables

Definition

- **Variable**: a place in the computer's memory which holds a value.
 - Memory **address** + symbolic **name**
 - You define the symbolic name in your Python program.
 - e.g. If a variable called `a` does not already exist, the statement `a = 5` stores the value 5 in an un-used block of memory.
 - The value can then be **referenced** (i.e. accessed) using the symbolic name, e.g. `print a`.



Variables

Key points

- Always make sure variables are defined **before trying to use them!**

The following block of code will not work:

```
b = 5
```

```
c = a*b
```

```
a = 10
```

- Variable names:
 - are **case sensitive**.
 - cannot start with a digit.
 - cannot be a Python keyword: and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, with, while, yield.

Printing

- Data held in variables can be printed to the screen using

```
b = 5.67560
print b
```
- Or, to present data in a nicer way, use **printf style formatting**:

```
print "The data held in variable b is: %.2f" % (b)
```
- The **format specifier** `%.2f` acts like a **placeholder**. When printing to the screen, Python substitutes this for the data in `b` and formats it accordingly:
 - `%.2f` prints out the data in `b` to 2 decimal places (i.e. `5.67`).
 - `%d` prints out the data in `b` as an integer (i.e. `5`).
 - `%g` prints out the data in `b` to the minimum number of significant figures (i.e. `5.6756`).
- If you see numbers like `5e-2`, this is just Python's way of writing 5×10^{-2} . It has nothing to do with the mathematical constant $e \approx 2.71828$.

Integer division

- In Python, dividing an integer by another integer will result in **another integer**.
- Python computes the result, and **drops the decimal point and everything after it**. e.g. $9/5 = 1.8$ will evaluate to 1
- This is a common error made in Python programs, so watch out for it.
- If in doubt, just make the numerator or denominator (or both) floating-point numbers. e.g. $3 \rightarrow 3.0$

Variable type conversion

- Converting a variable's data from one type to another.
 - `int(5.0) → 5`
 - `float(7) → 7.0`
 - `str(8.15) → "8.15"`
 - `int("5") → 5`
- Also known as **type casting**.
- You will most likely use casting:
 - to avoid integer division problems, e.g. `float(3)/5`
 - when you want to use numerical data read in from the keyboard using `raw_input`, e.g.

```
a = 5  
b = raw_input("Please enter a number.")  
c = a*float(b)
```

Operator precedence

- Expressions like $2 + 3/5.0$ are evaluated in a particular order, determined by **operator precedence**.
- Division has a **higher precedence** than addition, so $3/5.0$ is evaluated **first**, and 2 is then added on afterwards.
- If we wanted $2 + 3$ to be evaluated first, then we need to use **parentheses**: $(2 + 3)/5.0$.
- **BODMAS**: **B**rackets, **O**rder, **D**ivision, **M**ultiplication, **A**ddition, **S**ubtraction.

Importing modules

- Python **modules** are useful when you want to split your code up to make it more manageable, or to make a piece of code available for use in other programs.
- Mathematical functions like $\sin(x)$, $\cos(x)$, $\log(x)$ are available in the `math` module.
- There are two ways of importing functions from modules:
 - `import math`: Python will import all the functions in the `math` module, but will keep the functions in their own separate **namespace**. That is, you must use **prepend `math.` to the function's name** to use it, e.g. `x = 0.5; y = math.sin(x)`
 - `from math import *`: Python will import all the functions in the `math` module into the current namespace. That is, you can simply do `x = 0.5; y = sin(x)`. But: be careful that you do not have another function named `sin` in your program!

If statement

Definition

- The **if statement** is a programming construct that executes different blocks of code depending on whether a boolean **condition** evaluates to **True** or **False**.

```
if(boolean condition):  
    print "The condition is True"  
else:  
    print "The condition is False"
```

- Condition: it is raining (True or False)
- Possible actions: take an umbrella, don't take an umbrella.

```
if(it is raining):  
    Take an umbrella.  
else:  
    Don't take an umbrella.
```

If statements

Examples of conditions

- `b = 40`
- **Equality** condition: `b == 40` \rightarrow True
- **Negation** (also known as the logical complement):
 - `b != 40` \rightarrow False
 - `not(b == 40)` \rightarrow False
- **Or** condition: `b >= 40` \rightarrow True
 - `b == 40 or b > 40` \rightarrow True
- **And** condition: `b > 30 and b < 70` \rightarrow True
 - `b > 30 and b < 35` \rightarrow False

Logicals

And

		A	
		True	False
B	True	True	False
	False	False	False

Logicals

Or

		A	
		True	False
B	True	True	True
	False	True	False

Lists

Definition

- **List**: a Python data structure that can hold a **sequence** of **elements**/items/values. Elements can be added to, or removed from, a list.
- A list can be defined by enclosing the elements (separated by **commas**) in **square brackets**, e.g. `L = [4, 6, 2, -1]`
- **Append** an element to the end of a list by using the general form: `list_name.append(value)`
- Get the length of a list using the **len function**: `len(L)` returns a value of 4.

Lists

Referencing elements

- Each element of the list is assigned an **index**, with the first element's index being zero.
- To reference/access an element of the list, follow the general form:
list_name[element_index]
 - $L[0] \rightarrow 4$
 - $L[-1] \rightarrow -1$
 - $L[\text{len}(L)-1] \rightarrow -1$

Index:

0 1 2 3

Alternative index:

-4 -3 -2 -1

4	6	2	-1
---	---	---	----

Lists

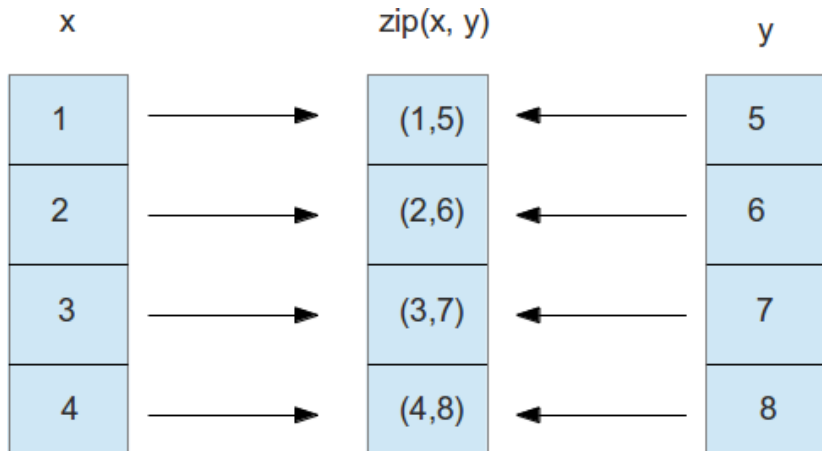
Slicing

- Sub-lists can also be extracted from a list. This is known as **slicing**.
- General usage: `list_name[start_index:end_index]`. By default, `start_index` is implicitly set to 0 if not provided by the user. Similarly, `end_index` is implicitly set to `len(list_name)` if not provided.
- $L[0:\text{len}(L)] \equiv L[:] \equiv L$
- Example: `L = [2, 5, 8, 0, 5, 1]`.
`A = L[:4] → A = [2, 5, 8, 0]`.

Lists

Zip two lists

- Elements from two lists can be combined using the **zip** function to form a new list: a **list of tuples**.



Loops

Definition

- **Loop**: a programming construct that allows a block of code to be executed multiple times.
- Two types of loop: `while` and `for`.

Loops

While loop

- Iterate indefinitely while some boolean condition is True. This condition is called the **loop invariant**.
- The invariant is evaluated before the start of each iteration. If it evaluates to True, Python executes all the statements in the indented code block.
- General form:

```
while(boolean condition is True):  
    Statements to be executed  
    within a single loop iteration.
```
- Remember to update any variables that the boolean condition depends on within the loop, e.g. if the condition is $i < 100$, you might do $i = i + 1$. Otherwise, i will never increase, the boolean condition will always be True, and the loop will never end.

Loops

For loop

- For loops must have **something to iterate over**. This is usually a list or an array.
- General form of a for loop:

```
for iterator in iterable_object:
```

Do some cool stuff, possibly involving the iterator.
- Example:
 - Iterator: `i`
 - Iterable object: `range(0, 3) → [0, 1, 2]`

```
for i in range(0, 3):  
    print i*2  
print "Out of the loop!"
```
 - Iteration 1: `i = 0`, Python prints out 0 to the screen.↷
 - Iteration 2: `i = 1`, Python prints out 2 to the screen.↷
 - Iteration 3: `i = 2`, Python prints out 4 to the screen.↷
 - No more elements to iterate over, so the loop ends.
 - Python prints "Out of the loop!"

Functions

Definition

- **Function**: a programming construct that expects zero or more **inputs**, and returns zero or more **outputs**.

- General form:

```
def function_name(input1, input2):
```

```
    The function's body. Compute any output values here.
```

```
    return output1, output2, output3
```

- The inputs are known as **arguments**.
- Example: the function `len` takes in 1 argument (a list/tuple/string/...) and returns 1 value (the length of that list/tuple/string).

User input

- User input can be read from the keyboard using the `raw_input` function. This takes 1 argument (a message that you want to show to the user, e.g. “Enter a number between 1 and 10”), and gives 1 output (the user’s input).
- This return/output value of the `raw_input` function is always a `string`.
- Remember: numerical data in string form needs to be `converted`, or `casted`, to a float or integer.

Exception handling

Definition

- **Exceptions:** errors that occur when Python cannot properly execute a line of code at run-time.
- Common examples include:
 - Trying to reference an element in a list that doesn't exist. e.g. `L = [1, 2]; print L[2]`
 - Trying to divide by zero.
- It is important that we handle these errors **gracefully**, because:
 - The standard exception error message will probably confuse an average user.
 - The program might not be able to continue executing properly.

Exception handling

try-except blocks

- **try-except blocks** are used to handle exceptions.
- Identify lines of your code where an exception may occur, and wrap them in a try block.
- In the except block, we decide how to handle the error. e.g.
try:

```
    number = float(raw_input("Enter a number:"))  
except ValueError:  
    print "Error:  you didn't enter a number."
```