Introduction to Python

Sophia Bethany Coban

Problem Solving By Computer

March 26, 2014

Introduction to Python

- Python is a general-purpose, high-level programming language.
- It offers readable codes, and its syntax allows us to write programs in fewer lines, compared to C/C++.
- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch
- and these versions are incompatible!
- However, the latest major version of 2.x (2.7) is close to 3.x.
- Python 2.6 or 2.7 is a good place to start but Python 3.x is more important! So make sure you learn the differences between them.

Introduction to Python

- Python is a general-purpose, high-level programming language.
- It offers readable codes, and its syntax allows us to write programs in fewer lines, compared to C/C++.
- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch
- and these versions are incompatible!
- However, the latest major version of 2.x (2.7) is close to 3.x.
- Python 2.6 or 2.7 is a good place to start but Python 3.x is more important! So make sure you learn the differences between them.

Introduction to Python

- Python is a general-purpose, high-level programming language.
- It offers readable codes, and its syntax allows us to write programs in fewer lines, compared to C/C++.
- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch
- and these versions are incompatible!
- However, the latest major version of 2.x (2.7) is close to 3.x.
- Python 2.6 or 2.7 is a good place to start but Python 3.x is more important! So make sure you learn the differences between them.

Introduction to Python

- Python is a general-purpose, high-level programming language.
- It offers readable codes, and its syntax allows us to write programs in fewer lines, compared to C/C++.
- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch
- and these versions are incompatible!
- However, the latest major version of 2.x (2.7) is close to 3.x.
- Python 2.6 or 2.7 is a good place to start but Python 3.x is more important! So make sure you learn the differences between them.

Introduction to Python

- Python is a general-purpose, high-level programming language.
- It offers readable codes, and its syntax allows us to write programs in fewer lines, compared to C/C++.
- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch
- and these versions are incompatible!
- However, the latest major version of 2.x (2.7) is close to 3.x.
- Python 2.6 or 2.7 is a good place to start but Python 3.x is more important! So make sure you learn the differences between them.

Introduction to Python

- Python is a general-purpose, high-level programming language.
- It offers readable codes, and its syntax allows us to write programs in fewer lines, compared to C/C++.
- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch
- and these versions are incompatible!
- However, the latest major version of 2.x (2.7) is close to 3.x.
- Python 2.6 or 2.7 is a good place to start but Python 3.x is more important! So make sure you learn the differences between them.

About IPython

- Python lacks a shiny GUI (compared to MATLAB).
- BUT there exists a solution called IPython!
- IPython *notebook* is an interface that looks a lot like a Mathematica notebook. It has quickly became the standard as an interactive, numerical computing tool for Python.
- We will not go into how IPython works but you are welcome to experiment with it as you like.
- The Windows computers in ATB do not have IPython (Anaconda recommended) but it can be downloaded via http://ipython.org/install.html (the download and the set up take about 3 minutes).

About IPython

- Python lacks a shiny GUI (compared to MATLAB).
- BUT there exists a solution called IPython!
- IPython *notebook* is an interface that looks a lot like a Mathematica notebook. It has quickly became the standard as an interactive, numerical computing tool for Python.
- We will not go into how IPython works but you are welcome to experiment with it as you like.
- The Windows computers in ATB do not have IPython (Anaconda recommended) but it can be downloaded via http://ipython.org/install.html (the download and the set up take about 3 minutes).

About IPython

- Python lacks a shiny GUI (compared to MATLAB).
- BUT there exists a solution called IPython!
- IPython *notebook* is an interface that looks a lot like a Mathematica notebook. It has quickly became the standard as an interactive, numerical computing tool for Python.
- We will not go into how IPython works but you are welcome to experiment with it as you like.
- The Windows computers in ATB do not have IPython (Anaconda recommended) but it can be downloaded via http://ipython.org/install.html (the download and the set up take about 3 minutes).

About IPython

- Python lacks a shiny GUI (compared to MATLAB).
- BUT there exists a solution called IPython!
- IPython *notebook* is an interface that looks a lot like a Mathematica notebook. It has quickly became the standard as an interactive, numerical computing tool for Python.
- We will not go into how IPython works but you are welcome to experiment with it as you like.
- The Windows computers in ATB do not have IPython (Anaconda recommended) but it can be downloaded via http://ipython.org/install.html (the download and the set up take about 3 minutes).

About IPython

- Python lacks a shiny GUI (compared to MATLAB).
- BUT there exists a solution called IPython!
- IPython *notebook* is an interface that looks a lot like a Mathematica notebook. It has quickly became the standard as an interactive, numerical computing tool for Python.
- We will not go into how IPython works but you are welcome to experiment with it as you like.
- The Windows computers in ATB do not have IPython (Anaconda recommended) but it can be downloaded via http://ipython.org/install.html (the download and the set up take about 3 minutes).

Why Python?

- Python is easy to read!
- A good transition between a language like MATLAB to more C/C++ and scripting languages.
- It is free and open source, which is why most companies prefer Python over MATLAB.
- Something else to add on your CV!!

Why Python?

- Python is easy to read!
- A good transition between a language like MATLAB to more C/C++ and scripting languages.
- It is free and open source, which is why most companies prefer Python over MATLAB.
- Something else to add on your CV!!

Why Python?

- Python is easy to read!
- A good transition between a language like MATLAB to more C/C++ and scripting languages.
- It is free and open source, which is why most companies prefer Python over MATLAB.
- Something else to add on your CV!!

Why Python?

- Python is easy to read!
- A good transition between a language like MATLAB to more C/C++ and scripting languages.
- It is free and open source, which is why most companies prefer Python over MATLAB.
- Something else to add on your CV!!

Learning Objectives

In these slides, we will go through

- Basic maths and string operations,
- Variables, their types and lists (arrays),
- If statements, for/while loops, functions and
- Modules, packages, and how to use them.

But most of all, the one thing we need to learn today is how important TABBING is in Python.
We will give examples both in MATLAB and Python, and compare the codes.

Learning Objectives

In these slides, we will go through

- Basic maths and string operations,
- Variables, their types and lists (arrays),
- If statements, for/while loops, functions and
- Modules, packages, and how to use them.

But most of all, the one thing we need to learn today is how important TABBING is in Python.
We will give examples both in MATLAB and Python, and compare the codes.

Learning Objectives

In these slides, we will go through

- Basic maths and string operations,
- Variables, their types and lists (arrays),
- If statements, for/while loops, functions and
- Modules, packages, and how to use them.

But most of all, the one thing we need to learn today is how important TABBING is in Python.
We will give examples both in MATLAB and Python, and compare the codes.

Learning Objectives

In these slides, we will go through

- Basic maths and string operations,
- Variables, their types and lists (arrays),
- If statements, for/while loops, functions and
- Modules, packages, and how to use them.

But most of all, the one thing we need to learn today is how important TABBING is in Python.
We will give examples both in MATLAB and Python, and compare the codes.

Learning Objectives

In these slides, we will go through

- Basic maths and string operations,
- Variables, their types and lists (arrays),
- If statements, for/while loops, functions and
- Modules, packages, and how to use them.

But most of all, the one thing we need to learn today is how important TABBING is in Python.
We will give examples both in MATLAB and Python, and compare the codes.

Learning Objectives

In these slides, we will go through

- Basic maths and string operations,
- Variables, their types and lists (arrays),
- If statements, for/while loops, functions and
- Modules, packages, and how to use them.

But most of all, the one thing we need to learn today is how important TABBING is in Python.

We will give examples both in MATLAB and Python, and compare the codes.

Learning Objectives

In these slides, we will go through

- Basic maths and string operations,
- Variables, their types and lists (arrays),
- If statements, for/while loops, functions and
- Modules, packages, and how to use them.

But most of all, the one thing we need to learn today is how important TABBING is in Python.
We will give examples both in MATLAB and Python, and compare the codes.

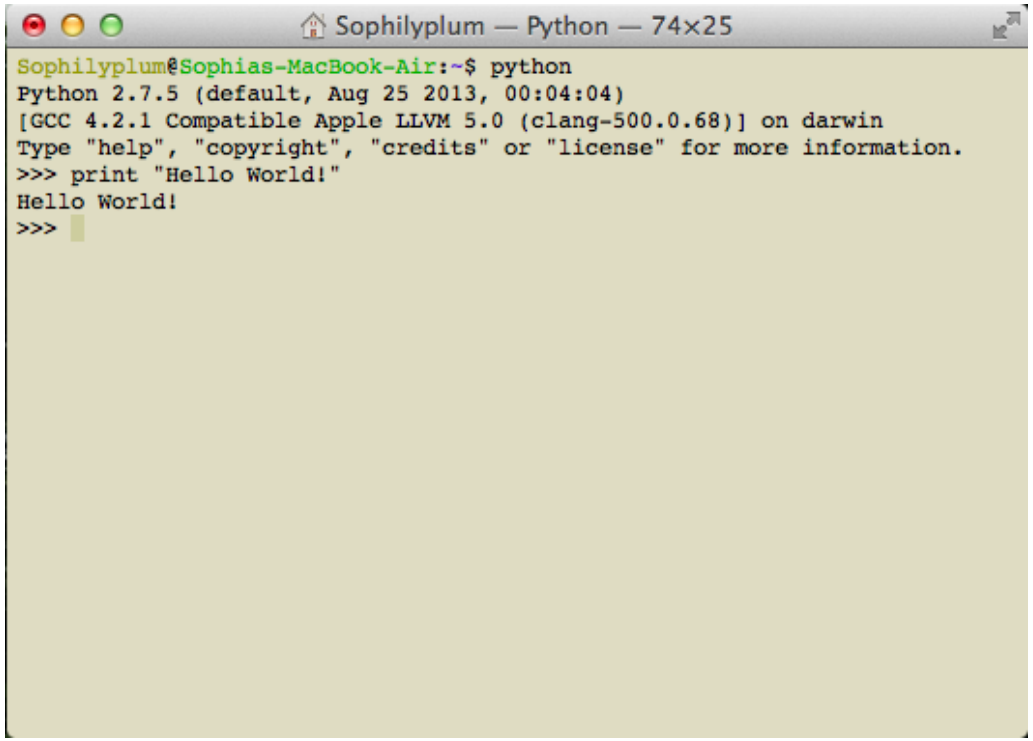
Running Python - Mac OSX

In these slides, we either run Python in a Terminal (similar to using MATLAB's command window), or use a text editor for the code and run the script in the Terminal for an output:

Running Python - Mac OSX

In these slides, we either run Python in a Terminal (similar to using MATLAB's command window), or use a text editor for the code and run the script in the Terminal for an output:

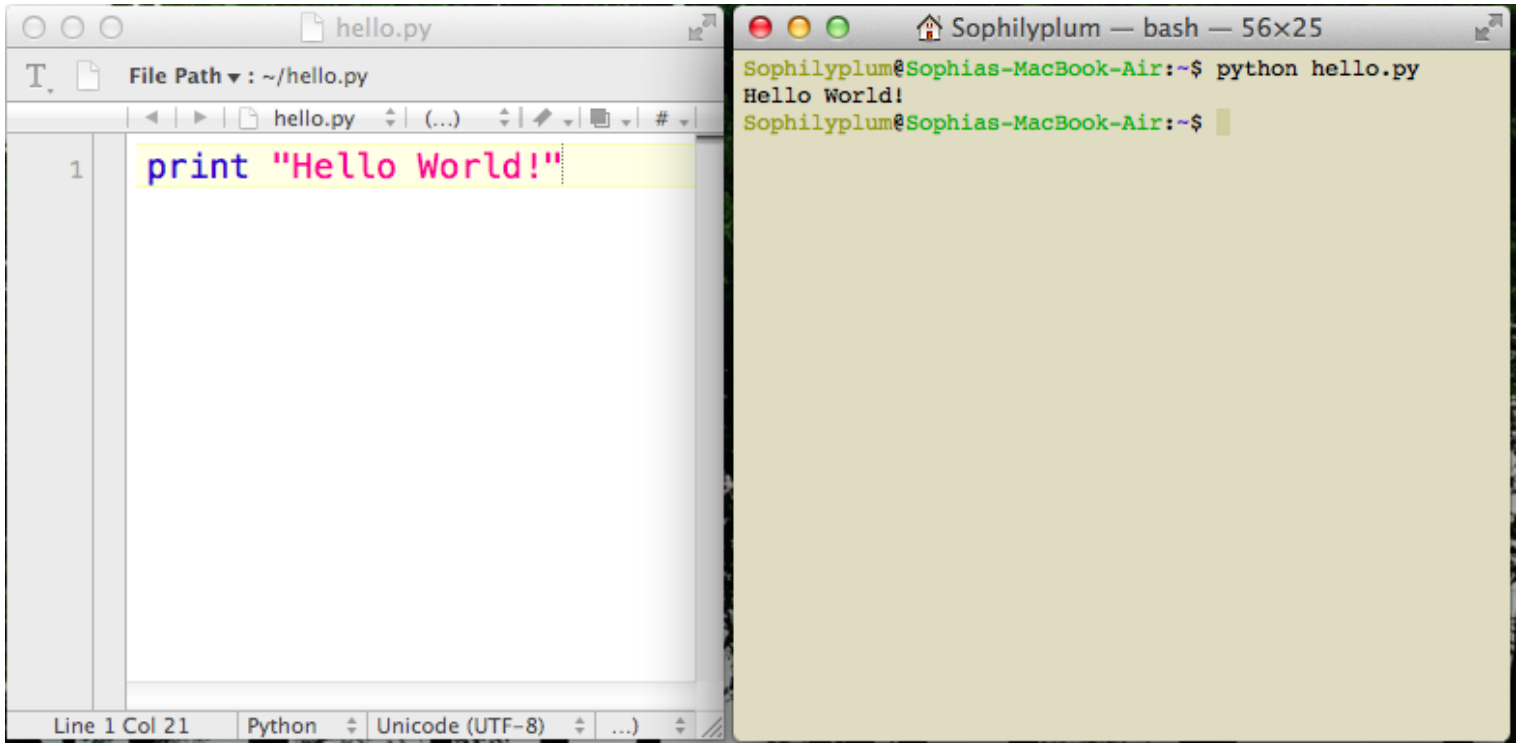
Python in Terminal



Running Python - Mac OSX

In these slides, we either run Python in a Terminal (similar to using MATLAB's command window), or use a text editor for the code and run the script in the Terminal for an output:

Python with a text editor (TextWrangler)



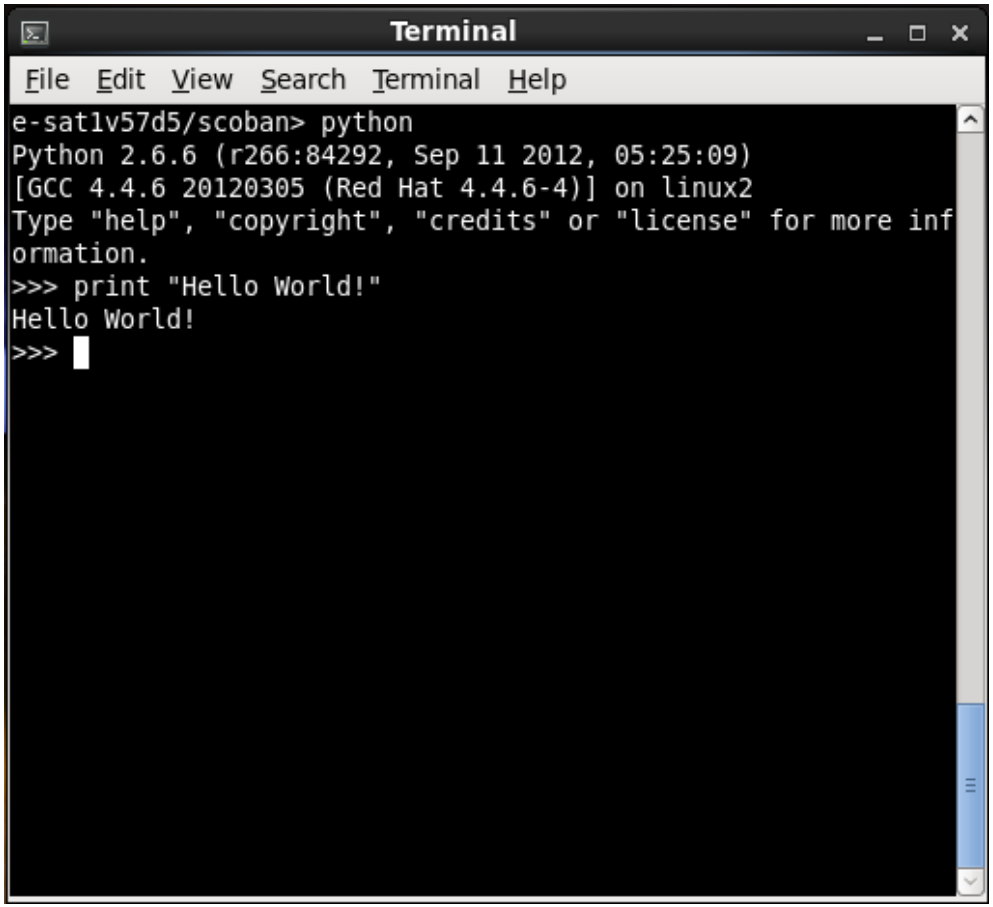
Running Python - Linux

Same steps can be taken to run Python in Linux as well (not surprising):

Running Python - Linux

Same steps can be taken to run Python in Linux as well (not surprising):

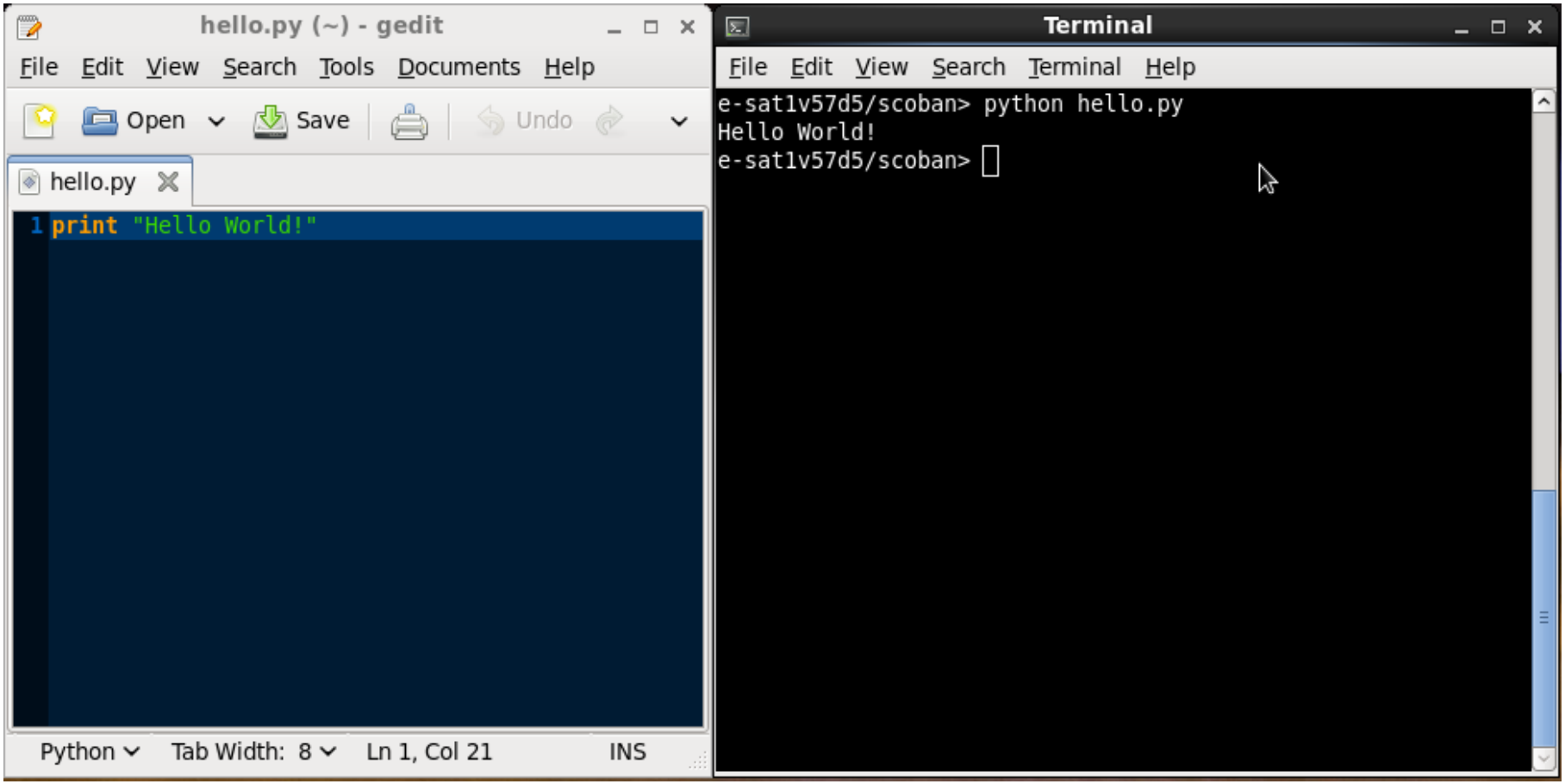
Python in Terminal



Running Python - Linux

Same steps can be taken to run Python in Linux as well (not surprising):

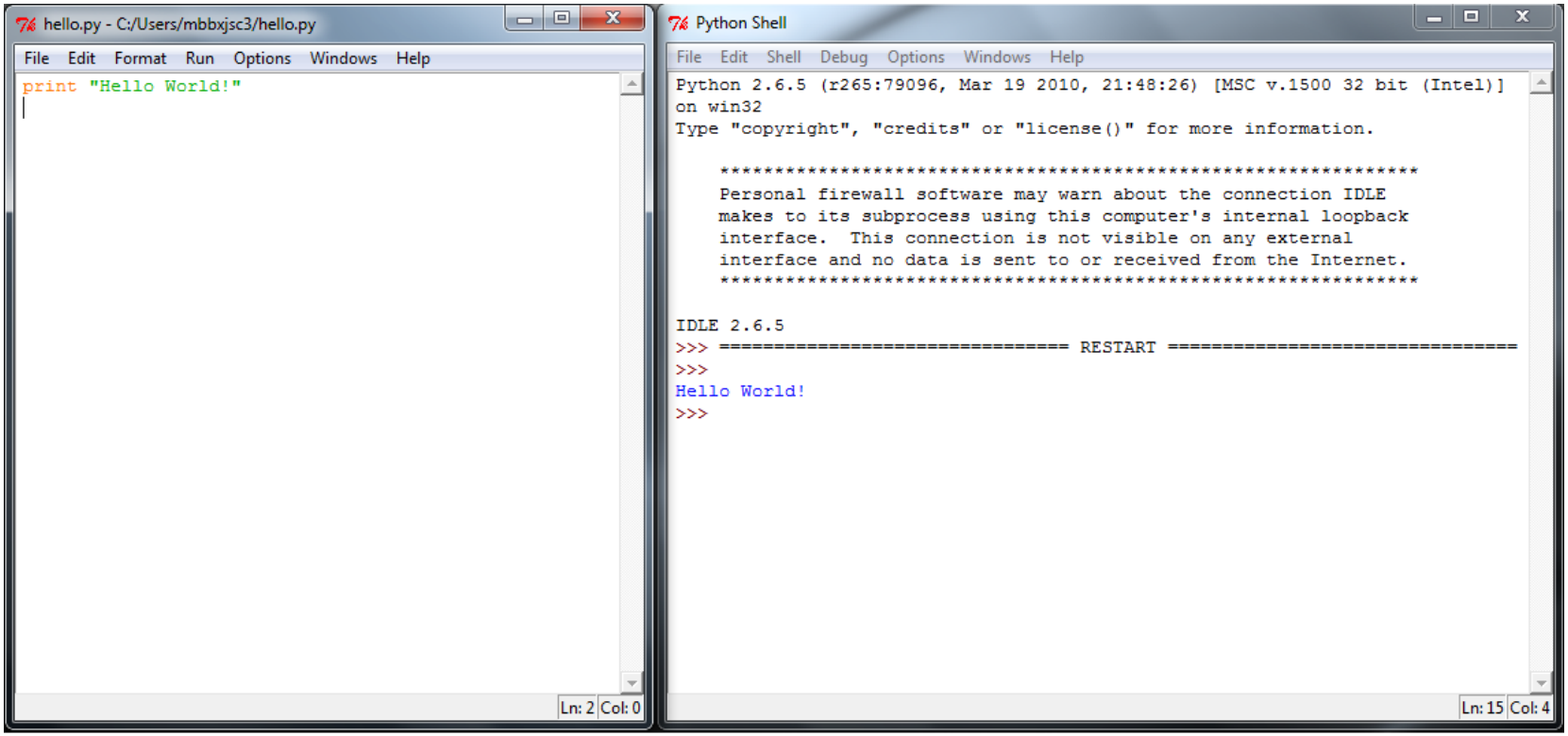
Python with a text editor (Gedit)



Running Python - Windows

In the ATB clusters, run Python via Start>Python IDLE. This program is similar to MATLAB in a sense that it has its own Editor and debug mode.

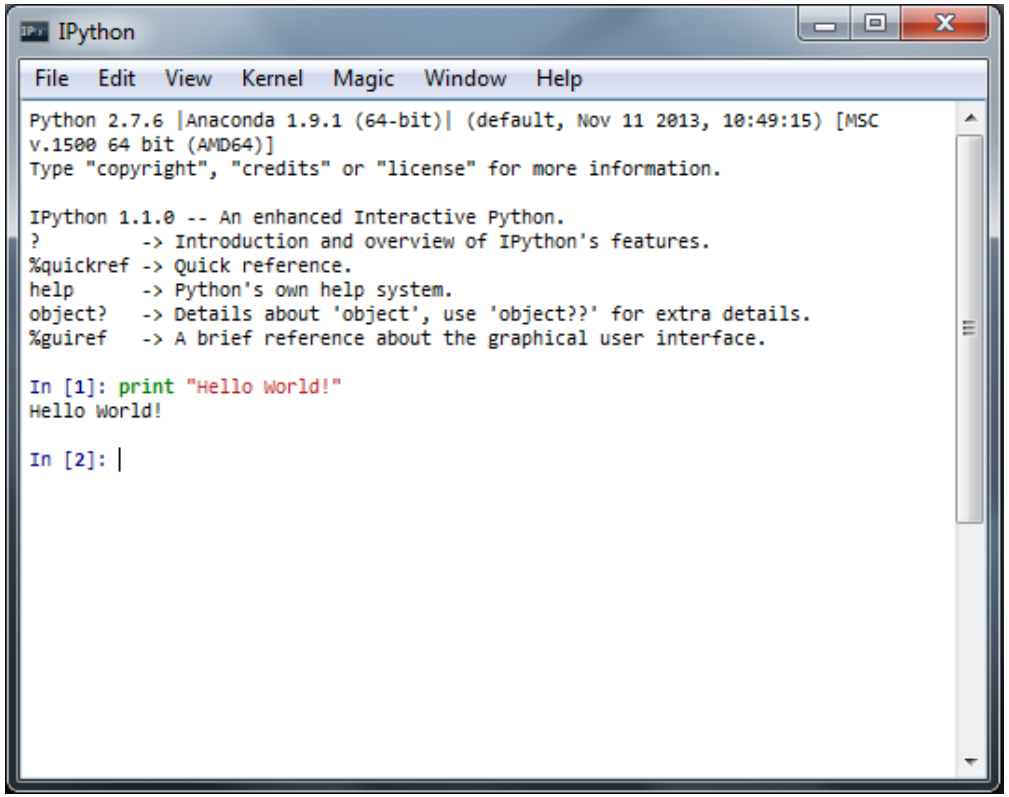
Python Shell



Running Python - Windows

In the ATB clusters, run Python via Start>Python IDLE. This program is similar to MATLAB in a sense that it has its own Editor and debug mode.

IPython QT Console



Basic Operations

- Python (as any other language) has addition, subtraction, multiplication and division implemented.

Basic Operations

- Python (as any other language) has addition, subtraction, multiplication and division implemented.
- If we were to calculate

$$c = 2 + 5 - 3 \times 6/9,$$

would Python follow the correct order?

Basic Operations

- Python (as any other language) has addition, subtraction, multiplication and division implemented.
- If we were to calculate

$$c = 2 + 5 - 3 \times 6/9,$$

would Python follow the correct order?

MATLAB

```
>> c = 2 + 5 - 3 * 6 / 9

c =

     5
```

Python

```
>>> c = 2 + 5 - 3 * 6 / 9
>>> print c
5
>>>
```

Variables and Types

Every variable in Python is an object and you do not need to declare their type before using them (something we do in C/C++).

- Below would be declared as an integer in Python:

  $>>> n\_int = 5$

- This would be a floating point number:

  $>>> n\_float = 5.0$

- We also have strings declared in single or double quotation marks:

  $>>> a\_string =$ "Hello World!"

- Finding type of a variable:

```
>>> print isinstance(c,int)
True
>>>
```

String Operations

You can do a lot with strings in Python.

```
>>> name, age = "John", 21
>>> print "It's %s's birthday today! %s is %d years old" % (name, name, age)
It's John's birthday today! John is 21 years old
>>>
```

String Operations

You can do a lot with strings in Python.

```
>>> name, age = "John", 21
>>> print "It's %s's birthday today! %s is %d years old" % (name, name, age)
It's John's birthday today! John is 21 years old
>>>
```

Print can also work as `fprintf`:

String Operations

You can do a lot with strings in Python.

```
>>> name, age = "John", 21
>>> print "It's %s's birthday today! %s is %d years old" % (name, name, age)
It's John's birthday today! John is 21 years old
>>>
```

Print can also work as `fprintf`:

MATLAB

```
>> my_float = c + 0.567;
>> if isa(my_float,'float')==true
fprintf('%1.4f is a float!\n',my_float)
end
5.5670 is a float!
>> |
```

Python

```
>>> my_float = c + 0.567
>>> if isinstance(my_float,float)==True:
...     print "%1.4f is a float!" % my_float
...
5.5670 is a float!
>>>
```

String Operations

- You can add two strings together:

MATLAB

```
>> string1 = 'Hello';
>> string2 = 'to you!';
>> string1 + ' ' + string2
Error using  +
Matrix dimensions must agree.
```

Python

```
>>> string_one = "Hello"
>>> string_two = "to you!"
>>> print string_one + " " + string_two
Hello to you!
>>>
```

- or multiply a string with a number:

MATLAB

```
>> hello1 = 'Hello ';
>> hello1 * 10

ans =

  Columns 1 through 3

        720        1010        1080

  Columns 4 through 6

       1080        1110         320
```

Python

```
>>> hello1 = "Hello "
>>> print hello1*5
Hello Hello Hello Hello Hello
>>>
```

- There is **a lot** you can do with strings in Python!

String Operations

- You can add two strings together:

MATLAB

```
>> string1 = 'Hello';
>> string2 = 'to you!';
>> string1 + ' ' + string2
Error using  +
Matrix dimensions must agree.
```

Python

```
>>> string_one = "Hello"
>>> string_two = "to you!"
>>> print string_one + " " + string_two
Hello to you!
>>>
```

- or multiply a string with a number:

MATLAB

```
>> hello1 = 'Hello ';
>> hello1 * 10

ans =

  Columns 1 through 3

        720        1010        1080

  Columns 4 through 6

       1080        1110         320
```

Python

```
>>> hello1 = "Hello "
>>> print hello1*5
Hello Hello Hello Hello Hello
>>>
```

- There is **a lot** you can do with strings in Python!

Lists

Lists are similar to arrays or vectors but they can contain variables of any type.

- We can declare a list in the following way:

$$>>> \ mylist = [1, 2, 3]$$

- or initialise as empty, and append elements individually:

$$>>> \ mylist = [\,]$$
$$>>> \ mylist.\text{append}(\text{“Hello!”})$$

- Let us try appending a string to a list of integers:

```
>>> mylist = [1,2,3]
>>> mylist.append("Hello!")
>>> for x in mylist:
...     print x
...
1
2
3
Hello!
>>>
```

Lists

- To access a specific element in a list, we use

$$>>> \ print = mylist[< index >]$$

- **BUT** indexing starts from 0 in Python (like in C/C++ but not in MATLAB)!

```
>>> for x in mylist:
...     print x
...
1
2
3
Hello!
>>> print mylist[0]
1
>>> print mylist[2]
3
>>> print mylist[3]
Hello!
>>>
```

Conditions and If Statements

- We have the boolean values True and False to evaluate conditions in Python (same as in MATLAB):

```
>>> print c
5
>>> c == 2
False
>>> c > 2
True
```

```
>> c == 2

ans =

     0
```

False

```
>> c > 2

ans =

     1
```

True

Conditions and If Statements

- We have the boolean values True and False to evaluate conditions in Python (same as in MATLAB):

```
>>> print c
5
>>> c == 2
False
>>> c > 2
True
```

```
>> c == 2

ans =

     0
```

False

```
>> c > 2

ans =

     1
```

True

- The syntax for `if statemets` is a little different:

MATLAB

```
>> if c == 5
disp('c == 5 is True.')
end
c == 5 is True.
>> |
```

Python

```
>>> if c == 5:
...     print "c == 5 is True"
...
c == 5 is True
>>>
```

Conditions and If Statements

- We have the boolean values True and False to evaluate conditions in Python (same as in MATLAB):

```
>>> print c
5
>>> c == 2
False
>>> c > 2
True
```

```
>> c == 2

ans =

     0
```

False

```
>> c > 2

ans =

     1
```

True

- The syntax for `if statemets` is a little different:

MATLAB

```
>> if c == 5
disp('c == 5 is True.')
end
c == 5 is True.
>> |
```

Python

```
>>> if c == 5:
...    print "c == 5 is True"
...
c == 5 is True
>>>
```

TAB

press enter for output

Conditions and If Statements

- We have the boolean values True and False to evaluate conditions in Python (same as in MATLAB):

```
>>> print c
5
>>> c == 2
False
>>> c > 2
True
```

```
>> c == 2

ans =

     0
```

False

```
>> c > 2

ans =

     1
```

True

- The syntax for `if statemets` is a little different:

MATLAB

```
>> if c == 5
disp('c == 5 is True.')
end
c == 5 is True.
>> |
```

Python

```
>>> if c == 5:
...    print "c == 5 is True"
...
c == 5 is True
>>>
```

TAB

press enter for output

- Of course in an editor, you would not have to press enter for output **BUT** you would have to put the **tab**!

Conditions and If Statements

- For when we have multiple conditions, we use AND and OR (much like && and || in MATLAB).
- We also have NOT for inverting the boolean value – similar to "∼=" (not equal) in MATLAB.
- When we have more than one statement to check? We use if/elseif in MATLAB but in Python, this is **if/elif**.

Conditions and If Statements

- For when we have multiple conditions, we use AND and OR (much like && and || in MATLAB).
- We also have NOT for inverting the boolean value – similar to "~=" (not equal) in MATLAB.
- When we have more than one statement to check? We use if/elseif in MATLAB but in Python, this is **if/elif**.

CODE:

```
>> if isa(c,'integer') && c>2
disp('c is an integer and greater than 2')
elseif isa(c,'float') && c>2
disp('c is a float and greater than 2')
else
disp('I dont know!')
end
```

```
>>> if isinstance(c,int) and c>2:
...     print "c is an integer and greater than 2"
... elif isinstance(c,float) and c>2:
...     print "c is a float and greater than 2"
... else:
...     print "I don't know!"
...
```

Output??

Conditions and If Statements

- For when we have multiple conditions, we use AND and OR (much like && and || in MATLAB).
- We also have NOT for inverting the boolean value – similar to "~=" (not equal) in MATLAB.
- When we have more than one statement to check? We use if/elseif in MATLAB but in Python, this is **if/elif**.

CODE:

```
>> if isa(c,'integer') && c>2
disp('c is an integer and greater than 2')
elseif isa(c,'float') && c>2
disp('c is a float and greater than 2')
else
disp('I dont know!')
end
```

```
>>> if isinstance(c,int) and c>2:
...     print "c is an integer and greater than 2"
... elif isinstance(c,float) and c>2:
...     print "c is a float and greater than 2"
... else:
...     print "I don't know!"
...
```

Output??

```
c is a float and greater than 2
```

```
c is an integer and greater than 2
```

Loops

- Just as in MATLAB, we have two different types of looping in Python: For and While.

- The syntax is very similar to the if statements:
  - Put a colon after the **for/while** <statement> line, and
  - Use **tabs** to continue with the loop or go back a tab to end the loop.

Loops

- Just as in MATLAB, we have two different types of looping in Python: For and While.

- The syntax is very similar to the if statements:
  - Put a colon after the **for/while** <statement> line, and
  - Use **tabs** to continue with the loop or go back a tab to end the loop.

MATLAB

```
>> mylist = [1,2,3];
>> for i=1:length(mylist)
disp(i)
end
     1

     2

     3
```

Python

```
>>> mylist=[1,2,3]
>>> for x in mylist:
...     print x
...
1
2
3
>>>
```

Loops

- Just as in MATLAB, we have two different types of looping in Python: For and While.

- The syntax is very similar to the if statements:
  - Put a colon after the **for/while** <statement> line, and
  - Use **tabs** to continue with the loop or go back a tab to end the loop.

MATLAB

```
>> mylist = [1,2,3];
>> for i=1:length(mylist)
disp(i)
end
     1

     2

     3
```

Python

```
>>> mylist=[1,2,3]
>>> for x in mylist:
...     print x
...
1
2
3
>>>
```

TAB

press enter for output

Loops

Below is an example of a while loop (and more tabbing!) in Python.

Loops

Below is an example of a while loop (and more tabbing!) in Python.

MATLAB

```
>> t = 0;
>> while t <= 5
disp(t)
if t==5
disp('End of the while loop!')
end
t = t + 1;
end
     0

     1

     2

     3

     4

     5

End of the while loop!
>>
```

Python

```
>>> t = 0
>>> while t <= 5:
...     print t
...     if t == 5:
...             print "End of the while loop!"
...     t += 1
...
0
1
2
3
4
5
End of the while loop!
>>>
```

Loops

Below is an example of a while loop (and more tabbing!) in Python.

MATLAB

```
>> t = 0;
>> while t <= 5
disp(t)
if t==5
disp('End of the while loop!')
end
t = t + 1;
end
     0

     1

     2

     3

     4

     5

End of the while loop!
>>
```

Python

```
>>> t = 0
>>> while t <= 5:
...    print t
...    if t == 5:
...        print "End of the while loop!"
...    t += 1
...
0
1
2
3
4
5
End of the while loop!
>>>
```

TAB

TAB

TAB

TAB

press enter for output

Inside IF statement

Outside IF statement

Loops

We also have Break and Continue in Python, which work exactly the same way as in MATLAB (and in C/C++):

Code (note they are now in .m and .py scripts)

Loops

We also have Break and Continue in Python, which work exactly the same way as in MATLAB (and in C/C++):

Code (note they are now in .m and .py scripts)

MATLAB

teaching.m

```
for t = 1:5;
    if t == 3
        disp('Skip t=3')
        continue
    end
    disp(t)
end
```

Python

teaching.py (no symbol selected)

```
for t in range(1,6):
    if t==3:
        print('Skip t=3')
        continue
    print t
```

Loops

We also have Break and Continue in Python, which work exactly the same way as in MATLAB (and in C/C++):

Code (note they are now in .m and .py scripts)

MATLAB

teaching.m

```
for t = 1:5;
    if t == 3
        disp('Skip t=3')
        continue
    end
    disp(t)
end
```

Python

teaching.py (no symbol selected)

```
for t in range(1,6):
    if t==3:
        print('Skip t=3')
        continue
    print t
```

No tab means end of loop!

Output??

Loops

We also have Break and Continue in Python, which work exactly the same way as in MATLAB (and in C/C++):

Code (note they are now in .m and .py scripts)

MATLAB

teaching.m

```
for t = 1:5;
    if t == 3
        disp('Skip t=3')
        continue
    end
    disp(t)
end
```

Python

teaching.py (no symbol selected)

```
for t in range(1,6):
    if t==3:
        print('Skip t=3')
        continue
    print t
```

No tab means end of loop!

Output??

MATLAB

```
     1

     2

Skip t=3
     4

     5
```

Python

```
1
2
Skip t=3
4
5
```

Loops

We also have Break and Continue in Python, which work exactly the same way as in MATLAB (and in C/C++):

Code (note they are now in .m and .py scripts)

MATLAB

teaching.m

```
for t = 1:5;
    if t == 3
        disp('Skip t=3')
        continue
    end
    disp(t)
end
```

Python

teaching.py (no symbol selected)

```
for t in range(1,6):
    if t==3:
        print('Skip t=3')
        continue
    print t
```

No tab means end of loop!

Notice we used `range(1,6)` for t = 1:5 – This is the range from 1 to 6, (with 6 not included).
We could have also used `xrange`.

(HW) Learn the difference between `range` and `xrange`!

Functions

We create and call functions in the following ways.

Functions

We create and call functions in the following ways.

- A function without any input arguments:

MATLAB

print_hello.m

```
function print_hello
disp('Hello!')
end
```

Output:

```
>> print_hello
Hello!
>>
```

Python

teaching.py (no symbol selected)

```
# This is a function with no input arguments
def print_hello():
    print "Hello!\n"

# Calling the function:
print_hello()
```

Output:

```
Sophilyplum $ python teaching.py
Hello!
```

Functions

We create and call functions in the following ways.

- A function with one input and one output argument:

MATLAB

print_hello.m

```
function greet = print_hello(name)
greet = ['Hello ' name '!'];
end
```

Output:

```
>> ch = print_hello('John')

ch =

Hello, John!
```

Python

teaching.py (no symbol selected)

```
# This is a function with one input
# and one output argument
def print_hello(name):
    greet = "Hello %s!\n" % name
    return greet

# Calling the function:
ch = print_hello('John')
print ch
```

Output:

```
Sophilyplum $ python teaching.py
Hello John!
```

Functions

We create and call functions in the following ways.

- A function with one input and one output argument:

MATLAB

print_hello.m

```
function greet = print_hello(name)
greet = ['Hello ' name '!'];
end
```

Output:

```
>> ch = print_hello('John')

ch =

Hello, John!
```

Python

teaching.py (no symbol selected)

```
# This is a function with one input
# and one output argument
def print_hello(name):
    greet = "Hello %s!\n" % name
    return greet

# Calling the function:
ch = print_hello('John')
print ch
```

Input

Output

Output:

```
Sophilyplum $ python teaching.py
Hello John!
```

Functions

We create and call functions in the following ways.

- A function with multiple input and output arguments:

MATLAB

add_things.m

```
function [n3,n4] = add_things(n1,n2,n3)
%This function compares adds n1 and
% n2 to obtain n4, and prints a
% message when the total is not n3.
n4 = n1 + n2;

if n4 ~= n3
    disp('n4 ~= n3')
    fprintf('n3 = %d\n',n3)
end
fprintf('total = %d\n',n4)
end
```

Python

teaching.py (no symbol selected)

```
# Function with multiple input/output args:
def my_function(n1,n2,n3): # multiple inputs
    n4 = n1 + n2
    if not n4==n3:
        print "n4 ~= n3\nn3 = %d" % n3
    print "total = %d\n" % n4
    return (n3,n4) # multiple outputs

# Calling the functions
n3,n4 = my_function(1,2,3)

n3,n4 = my_function(5,2,3)
```

Functions

We create and call functions in the following ways.

- A function with multiple input and output arguments:

MATLAB – Output

```
>> [n3,n4]=add_things(2,1,3);
total = 3
>>
>> [n3,n4]=add_things(5,2,3);
n4 ~= n3
n3 = 3
total = 7
>> |
```

Python – Output

```
Sophilyplum $ python teaching.py
total = 3

n4 ~= n3
n3 = 3
total = 7
```

Modules and Packages

- How to call a function in a different script? Use IMPORT!

Modules and Packages

- How to call a function in a different script? Use IMPORT!
- IMPORT is used to call modules.

Modules and Packages

- How to call a function in a different script? Use IMPORT!
- IMPORT is used to call modules.
- A *module* is a .py script with a number of functions.

Modules and Packages

- How to call a function in a different script? Use IMPORT!
- IMPORT is used to call modules.
- A *module* is a .py script with a number of functions.
- Let us call the previous function we used to add and compare numbers:

Modules and Packages

- How to call a function in a different script? Use IMPORT!

Calling functions in another .py script:

teaching.py (no symbol selected)

```python
# Module for adding + comparing things...

# This function compares two numbers.
def compare_things(num1,num2):
    if not num1==num2:
        print "n4 ~= n3\nn3 = %d" % num1

# This function adds two numbers, and
# compares the total with the third number.
def add_things(n1,n2,n3):
    n4 = n1 + n2
    compare_things(n3,n4)
    print "total = %d\n" % n4
    return (n3,n4)
```

run_functions.py (no symbol selected)

```python
import teaching

# Calling the functions
n3,n4 = teaching.add_things(5,2,3)

n3,n4 = teaching.add_things(2,1,3)
```

Output:

```
Sophilyplum $ python run_functions.py
n4 ~= n3
n3 = 3
total = 7

total = 3
```

Modules and Packages

- How to call a function in a different script? Use IMPORT!

Calling functions in another .py script:

teaching.py (no symbol selected)

```python
# Module for adding + comparing things...

# This function compares two numbers.
def compare_things(num1,num2):
    if not num1==num2:
        print "n4 ~= n3\nn3 = %d" % num1

# This function adds two numbers, and
# compares the total with the third number.
def add_things(n1,n2,n3):
    n4 = n1 + n2
    compare_things(n3,n4)
    print "total = %d\n" % n4
    return (n3,n4)
```

run_functions.py (no symbol selected)

```python
import teaching

# Calling the functions
n3,n4 = teaching.add_things(5,2,3)

n3,n4 = teaching.add_things(2,1,3)
```

Output:

```
Sophilyplum $ python run_functions.py
n4 ~= n3
n3 = 3
total = 7

total = 3
```

Modules and Packages

- How to call a function in a different script? Use IMPORT!
- IMPORT is used to call modules.
- A *module* is a .py script with a number of functions.
- We can use DIR to print the functions in the imported module:

Modules and Packages

- How to call a function in a different script? Use IMPORT!
- IMPORT is used to call modules.
- A *module* is a .py script with a number of functions.
- We can use DIR to print the functions in the imported module:

  Printing functions in a module:

run_functions.py

```
1  import teaching
2
3  print dir(teaching)
4
```

```
Sophilyplum $ python run_functions.py
['__builtins__', '__doc__', '__file__', '__name__',
'__package__', 'add_things', 'compare_things']
```

Modules and Packages

- How to call a function in a different script? Use IMPORT!
- IMPORT is used to call modules.
- A *module* is a .py script with a number of functions.
- We can use DIR to print the functions in the imported module.
- We can also use HELP to learn about the functions in the imported module:

Modules and Packages

- How to call a function in a different script? Use IMPORT!
- IMPORT is used to call modules.
- We can use DIR to print the functions in the imported module.
- We can also use HELP to learn about the functions in the imported module:

  Using HELP on functions in a module:

run_functions.py (no symbol selected)

```
1  import teaching
2
3  print help(teaching.add_things)
4
```

```
Help on function add_things in module teaching:

add_things(n1, n2, n3)
    # This function adds two numbers, and
    # compares the total with the third number.
(END)
```

Modules and Packages

- How to call a function in a different script? Use IMPORT!
- IMPORT is used to call modules.
- A *module* is a .py script with a number of functions.
- We can use DIR to print the functions in the imported module.
- We can also use HELP to learn about the functions in the imported module.
- Packages are a collection of modules put together (can be useful for different projects).
  (HW) Find out how to create packages!

Closing Notes

- Learning a new coding language often means a different way of thinking.
- Python is available in all platforms and running scripts is very common in/for all the devices, so it is very well integrated.
- We would be very impressed with you for being able to code in Python!
- You must remember that you would be more independent in finding solutions to your problems if you choose to code in Python.
- Don't be put off by the lack of a fancy gui and remember, practice makes perfect.

Closing Notes

- Learning a new coding language often means a different way of thinking.
- Python is available in all platforms and running scripts is very common in/for all the devices, so it is very well integrated.
- We would be very impressed with you for being able to code in Python!
- You must remember that you would be more independent in finding solutions to your problems if you choose to code in Python.
- Don't be put off by the lack of a fancy gui and remember, practice makes perfect.

Closing Notes

- Learning a new coding language often means a different way of thinking.
- Python is available in all platforms and running scripts is very common in/for all the devices, so it is very well integrated.
- We would be very impressed with you for being able to code in Python!
- You must remember that you would be more independent in finding solutions to your problems if you choose to code in Python.
- Don't be put off by the lack of a fancy gui and remember, practice makes perfect.

Closing Notes

- Learning a new coding language often means a different way of thinking.
- Python is available in all platforms and running scripts is very common in/for all the devices, so it is very well integrated.
- We would be very impressed with you for being able to code in Python!
- You must remember that you would be more independent in finding solutions to your problems if you choose to code in Python.
- Don't be put off by the lack of a fancy gui and remember, practice makes perfect.

Closing Notes

- Learning a new coding language often means a different way of thinking.
- Python is available in all platforms and running scripts is very common in/for all the devices, so it is very well integrated.
- We would be very impressed with you for being able to code in Python!
- You must remember that you would be more independent in finding solutions to your problems if you choose to code in Python.
- Don't be put off by the lack of a fancy gui and remember, practice makes perfect.

Useful Links

These slides can be found on
www.maths.manchester.ac.uk/∼scoban/python_lecture_psbc.pdf

Also check out:

- Teach Yourself Python:
  http://www.codecademy.com/tracks/python
- Python Standard Library (choose your version on top):
  http://docs.python.org/2/library/
- Differences between the versions:
  https://wiki.python.org/moin/Python2orPython3
- A huge collection of Python videos: http://pyvideo.org/
- Educational/interesting videos about IPython:
  http://ipython.org/videos.html