

INFO-664-01

Programming For Cultural Heritage

Intro to Python:
Python Basics

Agenda

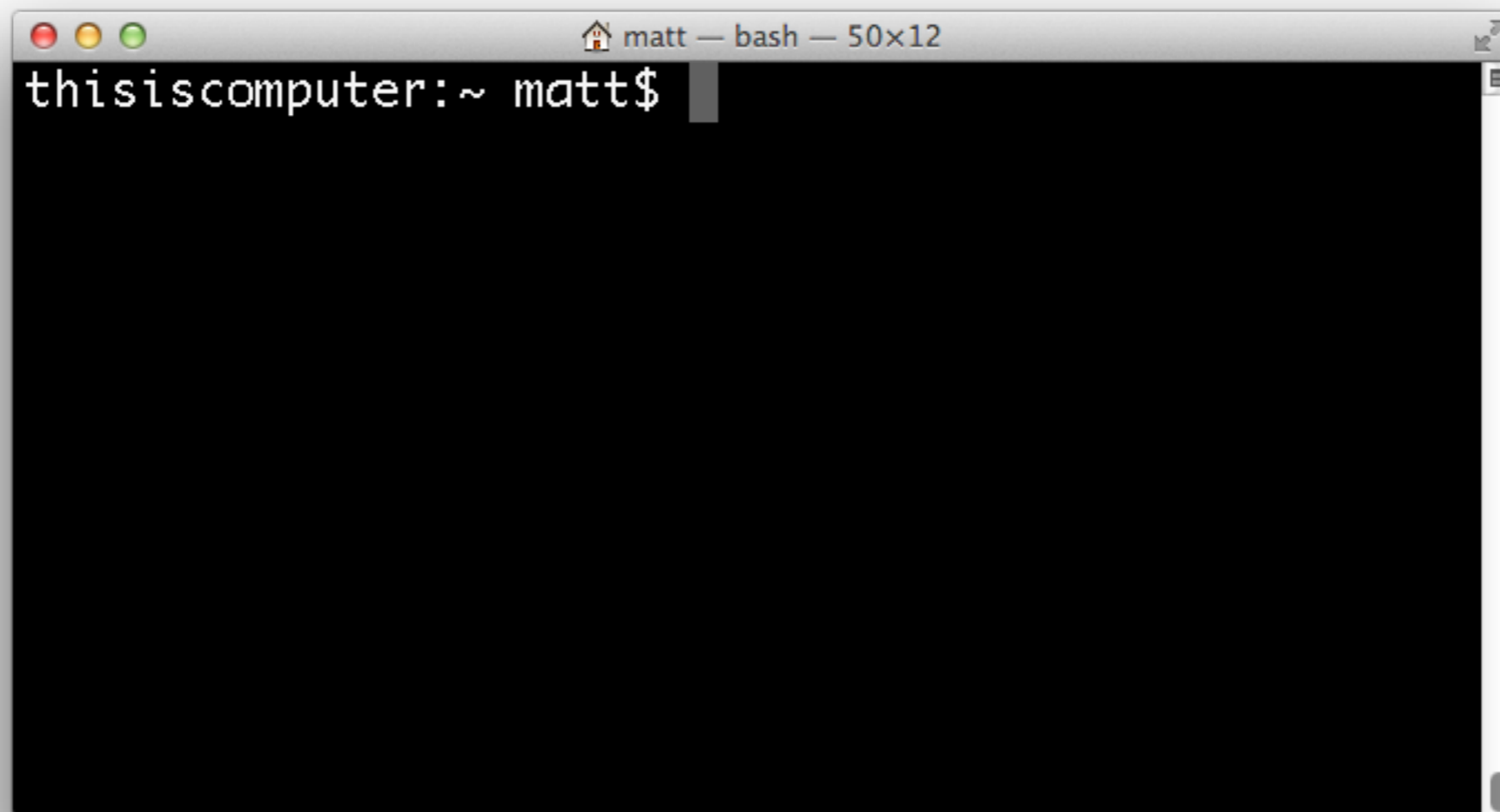
- About Python
- How do I Python
- Grammer
- Hello World
- Variables
- Control Structures
- Functions and Methods
- Modules
- Challenge

Python

- High level scripting language.
- Feed the interpreter a script and it executes synchronously.
- Is object oriented, but there are multiple ways to do the same thing.
- The script files normally have the file extension *.py*

Python Interpreter

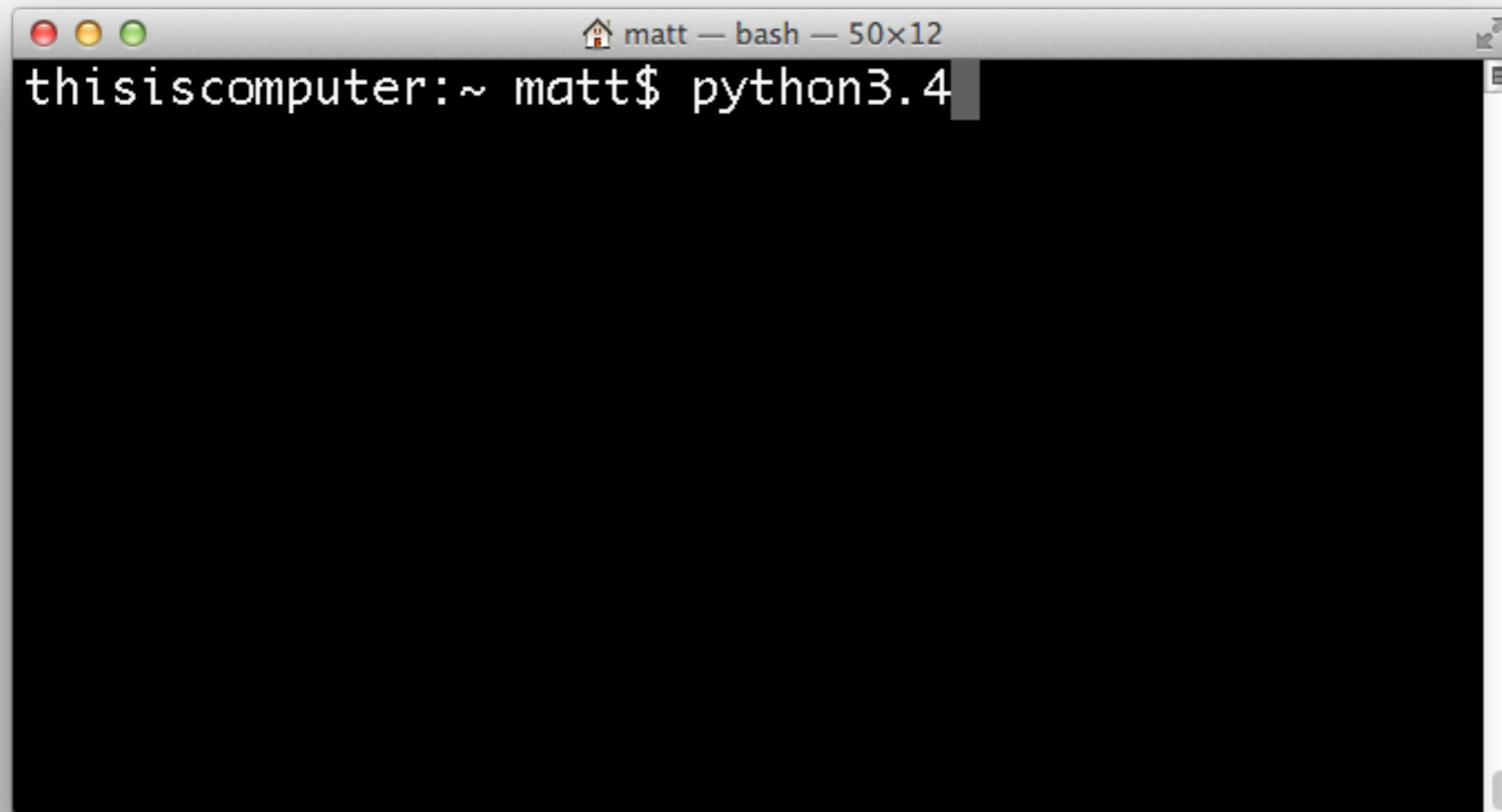
- Interactive Mode

A screenshot of a terminal window. The window title bar shows a home icon, the text "matt — bash — 50x12", and window control buttons. The terminal content shows the prompt "thisiscomputer:~ matt\$" with a cursor. The terminal background is black and the text is white.

```
thisiscomputer:~ matt$
```

Python Interpreter

- Interactive Mode

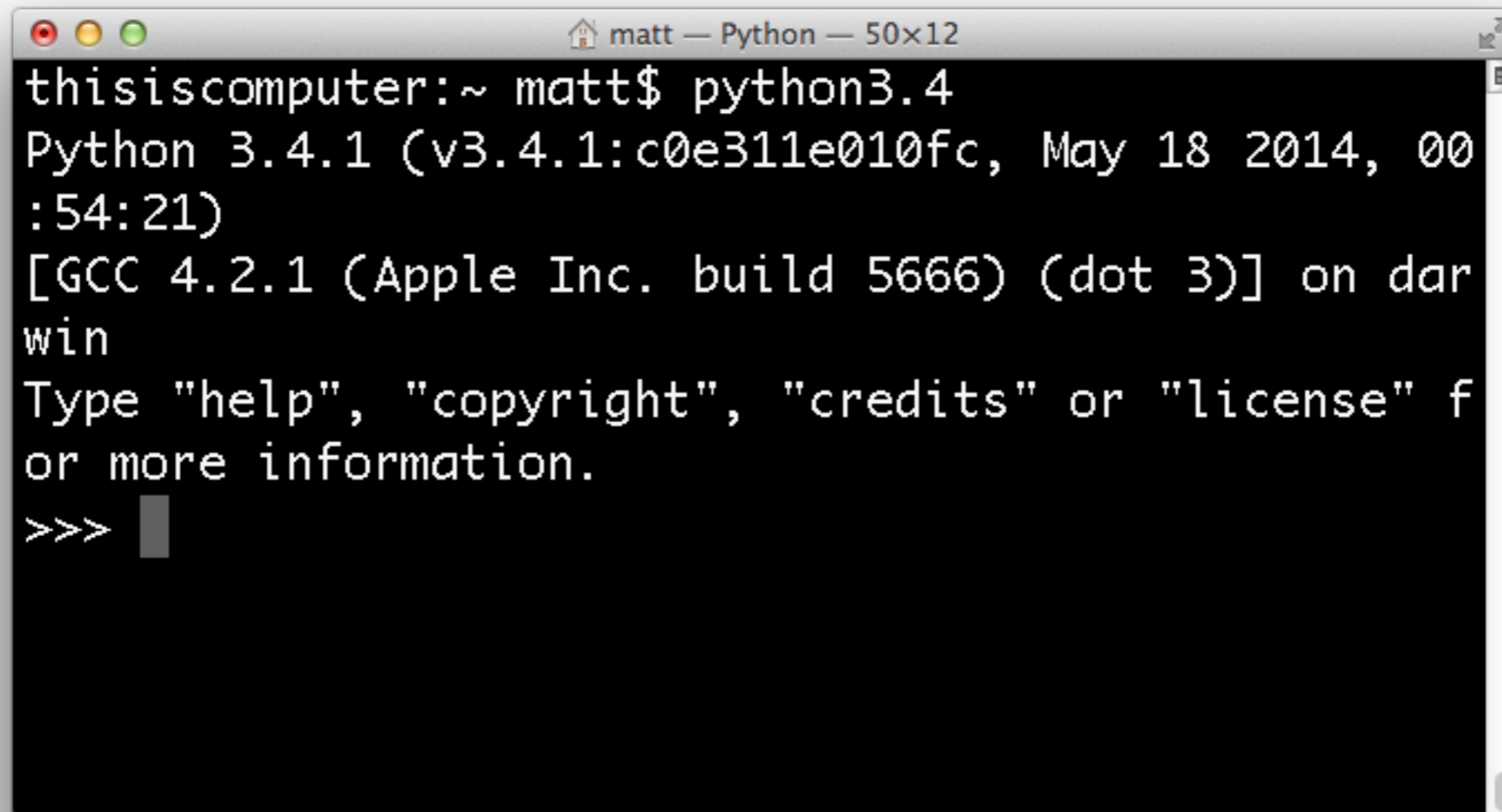


```
⌂ matt — bash — 50x12  
thisiscomputer:~ matt$ python3.4
```

A terminal window with a dark background and white text. The window title bar shows a home icon, the name 'matt', the shell 'bash', and the dimensions '50x12'. The terminal content shows the prompt 'thisiscomputer:~ matt\$' followed by the command 'python3.4' which is currently being typed, with a cursor at the end of the line.

Python Interpreter

- Interactive Mode



```
matt — Python — 50x12
thisiscomputer:~ matt$ python3.4
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python Interpreter

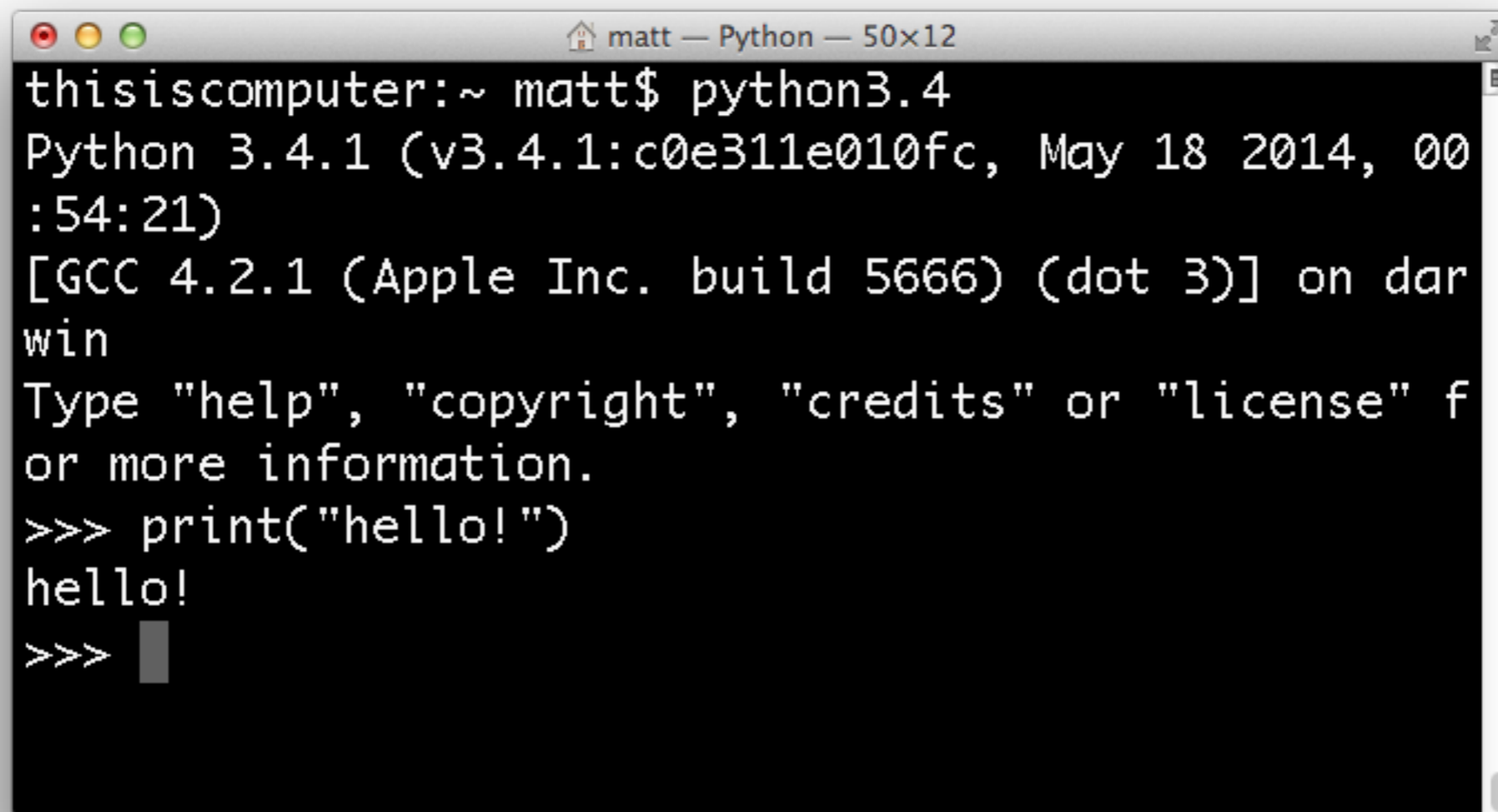
- Interactive Mode



```
matt — Python — 50x12
thisiscomputer:~ matt$ python3.4
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello!")
```

Python Interpreter

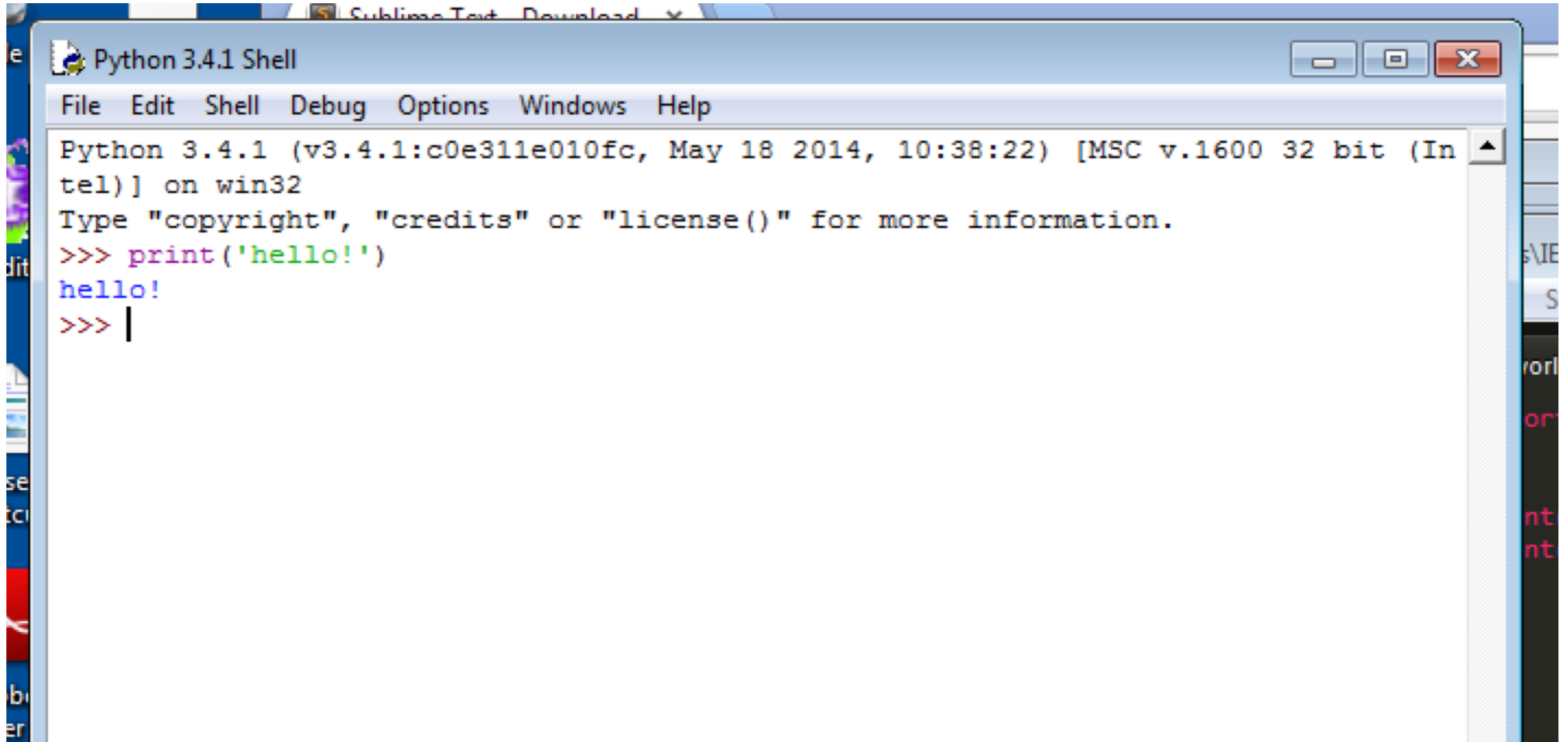
- Interactive Mode (control+D to exit)

A screenshot of a terminal window titled "matt — Python — 50x12". The terminal shows the command "python3.4" being executed, which starts the Python 3.4.1 interpreter. The output includes version information and a prompt. The user enters the command "print('hello!')", and the interpreter outputs "hello!". The prompt ">>>" is visible at the end of the line.

```
matt — Python — 50x12
thisiscomputer:~ matt$ python3.4
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello!")
hello!
>>>
```


Python Interpreter

- Interactive Mode (Windows)

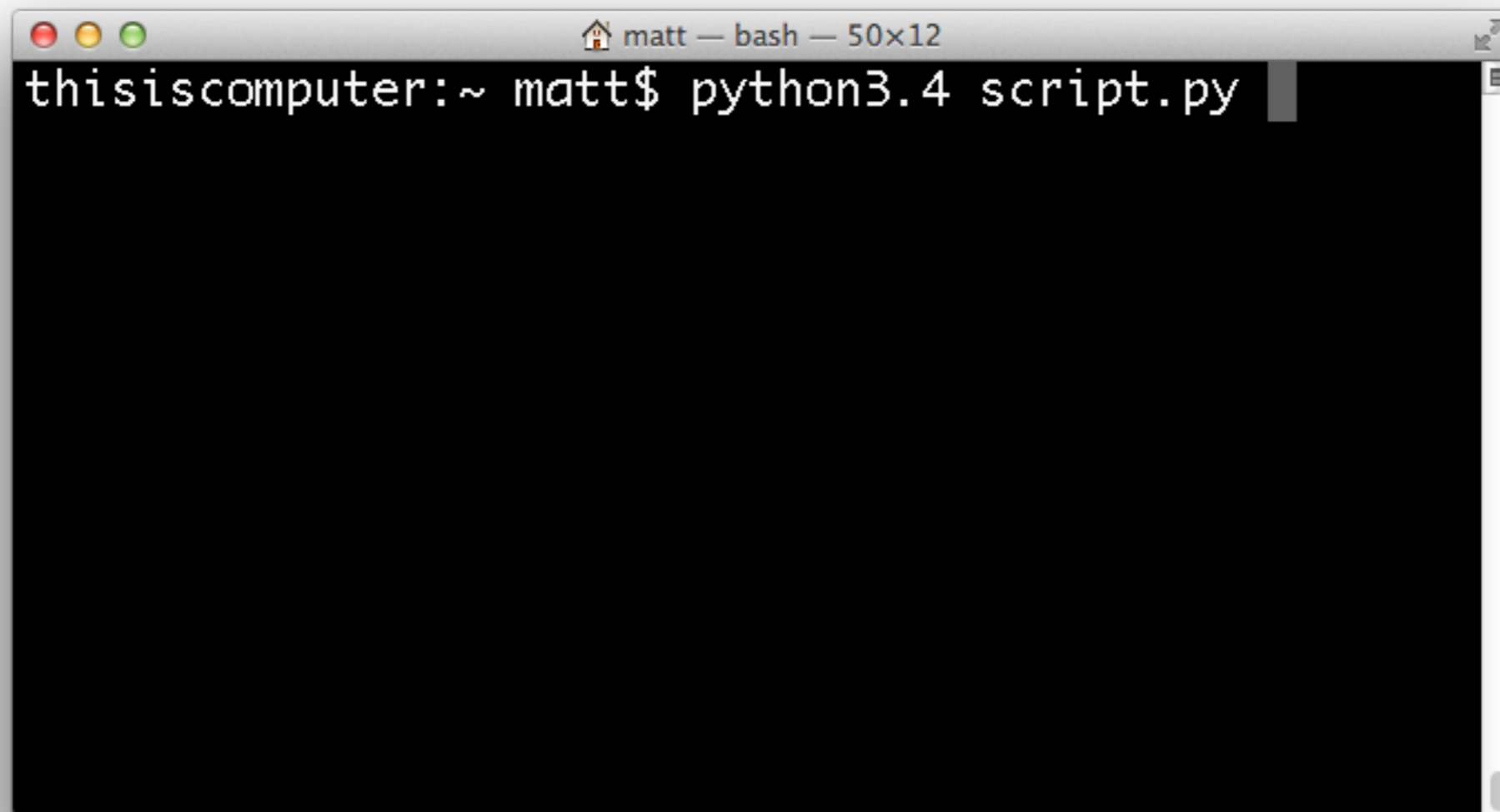
A screenshot of a Windows desktop environment showing a 'Python 3.4.1 Shell' window. The window has a standard Windows title bar with minimize, maximize, and close buttons. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Windows', and 'Help'. The main content area of the window displays the following text:

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print('hello!')
hello!
>>> |
```

The text is color-coded: 'print' is purple, 'hello!' is green, and 'hello!' is blue. A vertical scrollbar is visible on the right side of the window.

Python Interpreter

- Script Mode: specify the script to run.

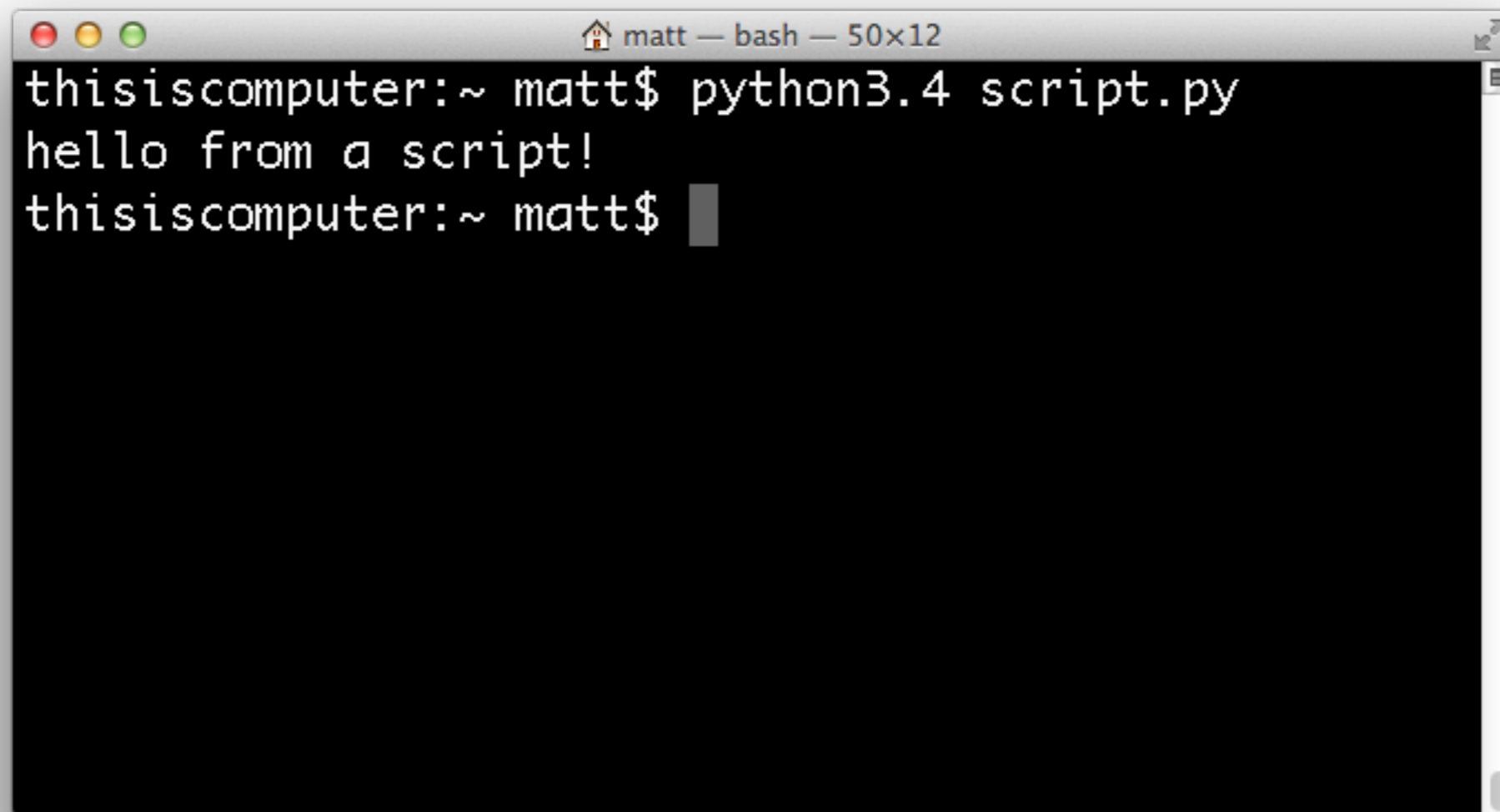


```
matt — bash — 50x12  
thisiscomputer:~ matt$ python3.4 script.py
```

A terminal window with a dark background and light text. The window title bar shows 'matt — bash — 50x12'. The terminal content shows the prompt 'thisiscomputer:~ matt\$' followed by the command 'python3.4 script.py' and a cursor. The window has standard macOS window controls (red, yellow, green buttons) and a scroll bar on the right.

Python Interpreter

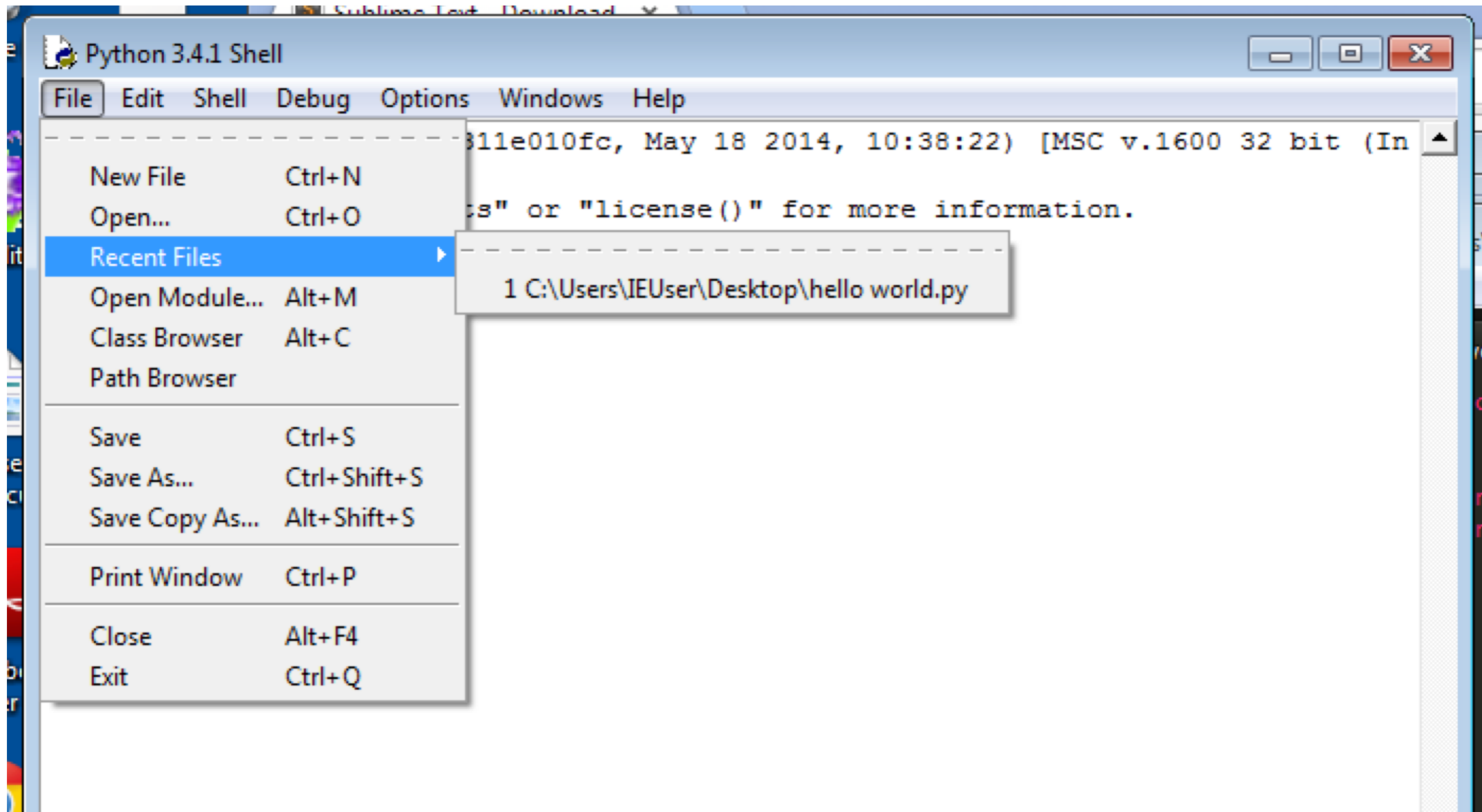
- Script Mode

A terminal window with a dark background and light text. The window title bar shows a home icon, the text 'matt — bash — 50x12', and window control buttons. The terminal content shows a command being executed and its output.

```
thisiscomputer:~ matt$ python3.4 script.py  
hello from a script!  
thisiscomputer:~ matt$ █
```

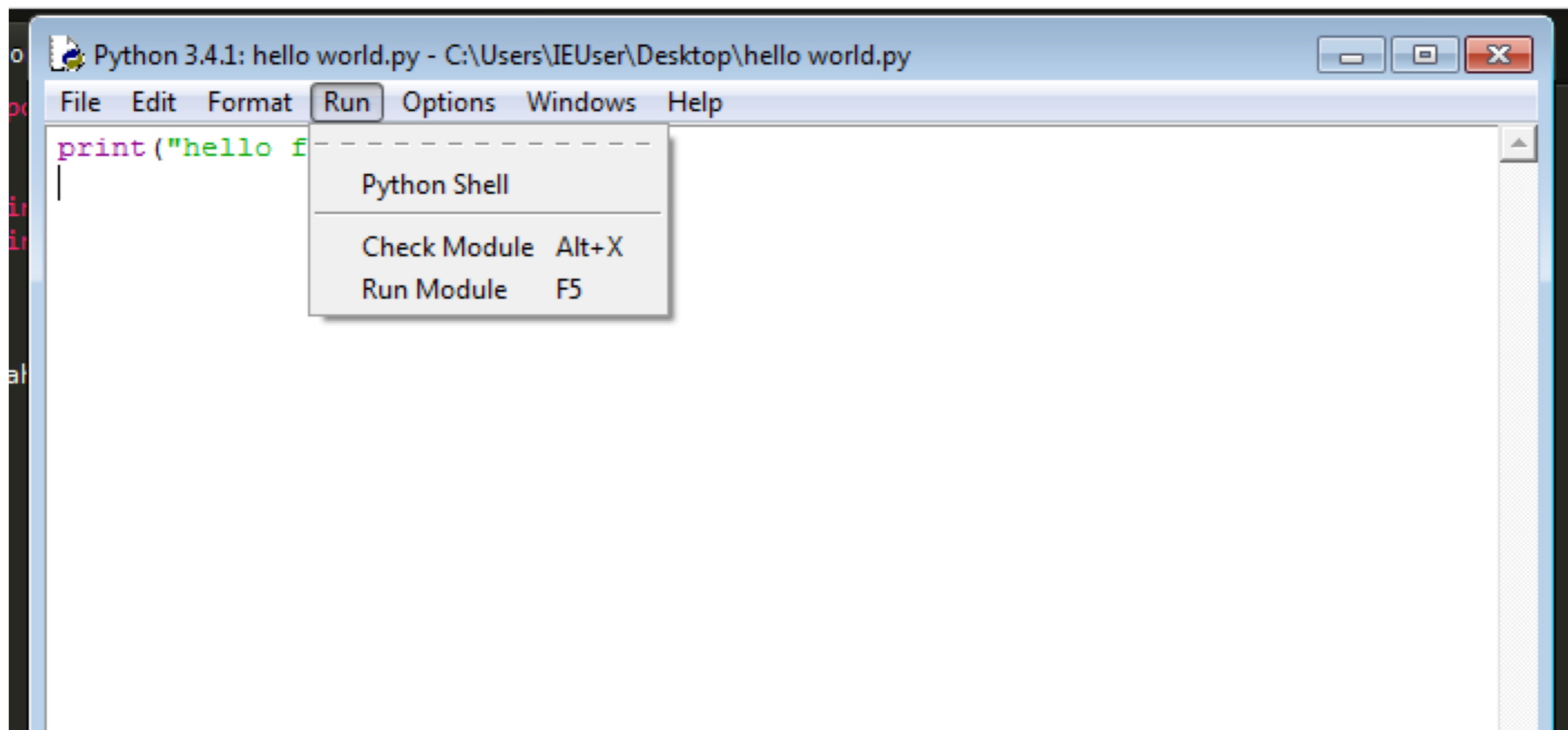
Python Interpreter

- Script Mode (Windows)



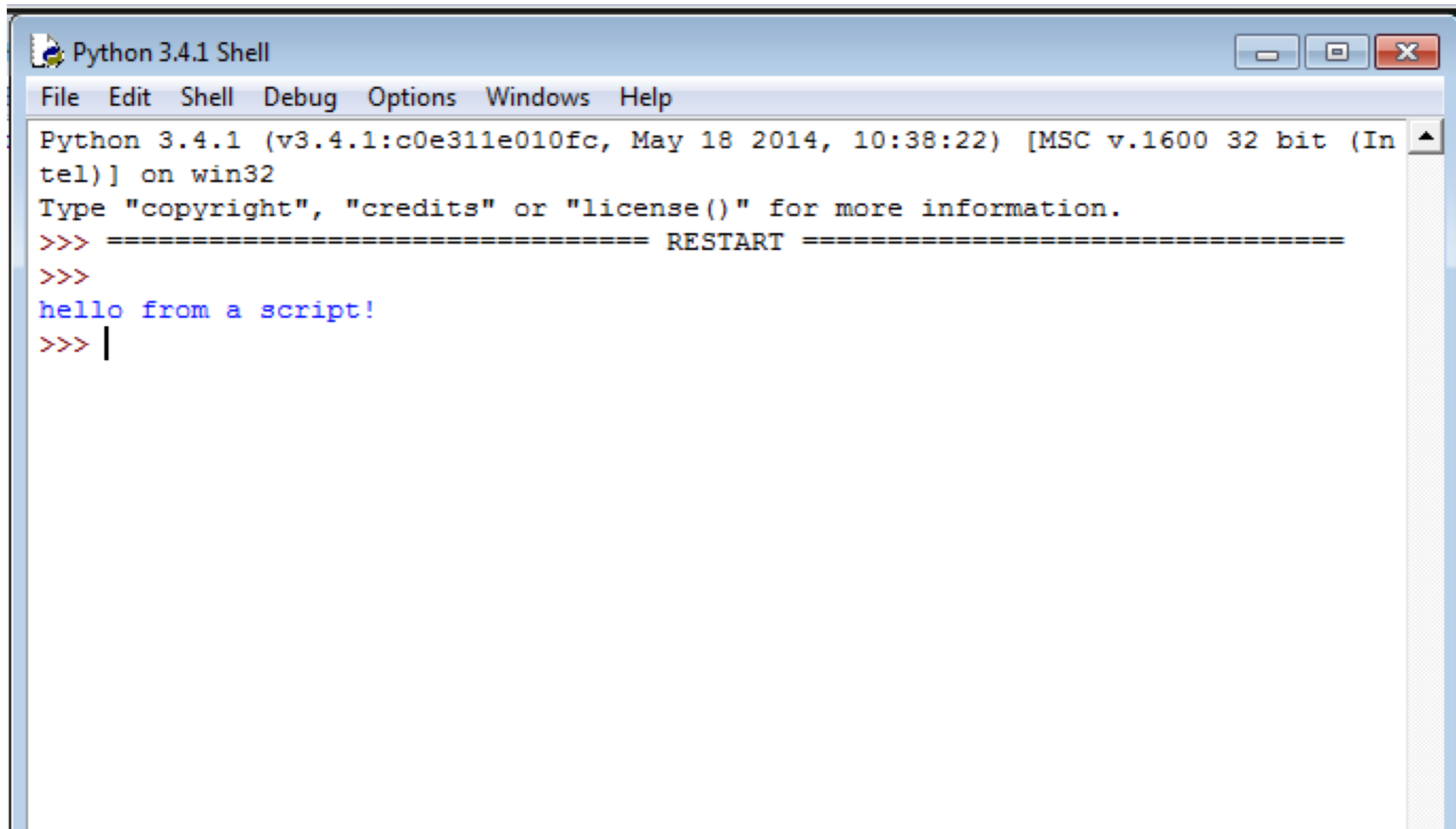
Python Interpreter

- Script Mode (Windows)



Python Interpreter

- Script Mode (Windows)



The screenshot shows a Windows-style window titled "Python 3.4.1 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main content area displays the following text:

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
hello from a script!
>>> |
```

Grammar

- Python relies on indentation and colons to control the flow of the script. Whitespace is important.

[Some Code...]

[Control Statement]:

Indent [A Line of Code]

Indent [A Line of Code]

Indent [Another Control Statement]:

Indent Indent [A Line of Code]

[Some More Code...]

Grammer

- Add comments using #

```
1  
2  
3 #this is a comment!  
4 print('hello!')  
5
```


Grammer

- Conventions
 - Sticking with a specific style. For example using underscores in your variable names, etc.

Python Basics

- Variables
- Control Structures
- Methods & Functions

Variables

```
1  
2 x = something  
3
```

Your Label

The Value

Variables - Integers

```
1  
2 x = 1  
3 hundred = 100  
4 something = 1000000  
5
```

Whole Numbers

Variables - Floats

```
1  
2 y = 0.5  
3 my_height = 1.81  
4
```

Decimal Numbers

Variables - Strings

```
1  
2 x = "A String!"  
3 x2 = 'A Sting with single quotes'  
4
```

Variables - Boolean

```
1  
2 x = True  
3 y = False  
4
```

Only two possible values

Variables - None

```
1  
2 x = None  
3  
4
```

Also known as: Null

Complex Variables - List

```
1
2 list_of_numbers = [1,2,3,5,7,9,10,10000]
3 list_of_strings = ["hello","goodbye","maybe"]
4 anything = [5,"hello",False,None,5.7]
5
```

Also know as: Array

Look for the brackets: []

Reference the things in the list using it's index (x[0],x[5],etc)

Complex Variables - Dictionary

```
1
2 x = {
3     'NY' : "New York",
4     'NJ' : "New Jersey",
5     "CT" : "Connecticut"
6 }
7
```

Also know as: Object

A Key/Value Store

Look for the curly brackets: {}

Complex Variables - Dictionary

```
1
2 x = {
3     'NY' : "New York",
4     'NJ' : "New Jersey",
5     "CT" : "Connecticut",
6     "codes" : ["NY", "NJ", "CT"]
7 }
8
```

Also know as: Object

A Key/Value Store

Look for the curly brackets: {}

Complex Variables - Dictionary

```
1
2 x = {
3     'NY' : "New York",
4     'NJ' : "New Jersey",
5     "CT" : "Connecticut",
6     "codes" : ["NY", "NJ", "CT"],
7     "pop" : {
8         "NY" : 8.3,
9         "NJ" : 8.8,
10        "CT" : 3.5
11    }
12 }
13
```

Exercise - List

- Write a List (in your editor or on paper) of all the names of cats or dogs (or whatever) you know.

Exercise - List

```
animals = ["Bert", "Miles", "Rulla"]
```

Exercise - Dictionary

- Write a Dictionary (in your editor or on paper) where the name is the key and the value is the type of animal it is

Exercise - Dictionary

```
{  
  "Bert" : "cat",  
  "Miles" : "dog",  
  "Rulla" : "cat"  
}
```


Exercise - Dictionary

- Write a Dictionary (in your editor or on paper) where the name is the key and the value another dictionary with the type of animal and their age

Exercise - Dictionary

```
{
  "Bert" : {
    "type": "cat",
    "age" : 10
  },
  "Miles" : {
    "type": "dog",
    "age" : 8
  },
  "Rulla" : {
    "type": "cat",
    "age" : 10
  }
}
```

Exercise - Both

- Make a List of dictionaries that describes the animals name, type and age

Exercise - Dictionary

```
[
  {
    "name": "Bert",
    "type": "cat",
    "age": 10
  },
  {
    "name": "Miles",
    "type": "dog",
    "age": 8
  },
  {
    "name": "Rulla",
    "type": "cat",
    "age": 10
  }
]
```

Control Structures - If Statement

```
1  
2 x = 5  
3  
4 if x == 5:  
5     print("x is 5!")  
6
```

Control Structures - If Else Statement

```
1
2 x = 55
3
4 if x == 5:
5     print("x is 5!")
6 else:
7     print("x is something else!")
8
```

Control Structures - If Else Elif Statement

```
1
2 x = 55
3
4 if x == 5:
5     print("x is 5!")
6 elif x == 55:
7     print("x is 55!")
8 else:
9     print("x is something else!")
10
```

Control Structures - If Statement

```
1  
2 x = 55  
3  
4 if x != 5:  
5     print("x is not 5!")  
6  
7
```


Control Structures - If Statement

```
1  
2 x = 5  
3  
4 if x is 5:  
5     print("x is 5!")  
6  
7
```

Control Structures - If Statement

```
1  
2 x = 55  
3  
4 if x is not 5:  
5     print("x is not 5!")  
6  
7
```

Control Structures - If Statement

```
1  
2 x = 55  
3  
4 if x > 0 and x < 100:  
5     print("x is something 0 to 100")  
6  
7
```

Control Structures - If Statement

```
1  
2 x = [1,2,3,5,6,7,8,9,10]  
3  
4 if 5 in x:  
5     print("5 is in x!")  
6
```

```
1
2 state_list = {
3     'NY' : "New York",
4     'NJ' : "New Jersey",
5     "CT" : "Connecticut",
6     "codes" : ["NY", "NJ", "CT"],
7     "pop" : {
8         "NY" : 8.3,
9         "NJ" : 8.8,
10        "CT" : 3.5
11    }
12 }
13
14 for x in state_list:
15     print(x)
16
```

Control Structures - For Statement

```
1  
2 a_string = "HeLLLLLloooooo"  
3  
4  
5 for x in a_string:  
6     print (x)  
7
```

Control Structures - While Loop

```
1  
2 x = 0  
3  
4 while x < 10:  
5     print(x)  
6     x = x + 1  
7
```

Control Structures - Try Statement

```
1  
2 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']  
3  
4 print(letters[100])  
5
```


Control Structures - Try Statement

```
Traceback (most recent call last):  
  File "try.py", line 4, in <module>  
    print(letters[100])  
IndexError: list index out of range
```

Control Structures - Try Statement

```
1
2 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
3
4 try:
5     print(letters[100])
6 except IndexError:
7     print("Uh-oh, that index does not exist")
8
```

Functions

- Defined code that accepts parameters and returns a values. They can be “built-in”, written by you or others.

Functions

- Defined code that accepts parameters and returns a values. They can be “built-in”, written by you or others.

Functions

```
1  
2 a_number = 5  
3  
4 a_string = "hello"  
5  
6 print(a_number + a_string)  
7
```

Functions

```
Traceback (most recent call last):  
  File "functions.py", line 6, in <module>  
    print(a_number + a_string)  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Functions

```
1  
2 a_number = 5  
3  
4 a_string = "hello"  
5  
6 a_number = str(a_number)  
7  
8 print(a_number + a_string)  
9
```

Functions

```
10  
11 a_numer_list = [1,2,3,4,5,6,7,8,9]  
12  
13 print( len(a_numer_list) )  
14
```


Functions

```
15  
16 a_numer_list = [5,7,2,4,1,0,5,2,9]  
17  
18 a_numer_list = sorted(a_numer_list)  
19  
20 print( a_numer_list )
```

Methods

- Methods are functions that exist as part of an object and uses that object's value as the parameter passed. Pretty much everything in python is an object and has methods

Methods

```
1  
2 a_string = "hi there"  
3  
4 print( a_string.upper() )  
5
```

S.count(substring[, start[, end]])	Count occurrences of <i>substring</i>
S.endswith(suffix[, start[, end]])	True if S ends with <i>suffix</i>
S.find(substring [,start [,end]])	Find first occurrence of <i>substring</i> and return its index number; if not found, return -1
S.index(substring [,start [,end]])	Find first occurrence of <i>substring</i> and return its index number; if not found, raise ValueError
S.isalnum()	True if S has only alphanumeric characters
S.isalpha()	True if S has only alphabetic characters
S.isdigit()	True if S has only digits
S.isspace()	True if S has only whitespace characters
S.join(iterable)	Using S as a separator, stick together the strings in <i>iterable</i>
S.lower()	Convert S to lowercase
S.lstrip([chars])	Remove whitespace (or <i>chars</i>) from front (left) of S
S.replace (old, new[, count])	Replace <i>old</i> (a substring) with <i>new</i>
S.rfind(substring [,start [,end]])	Find the last (rightmost) occurrence of <i>substring</i> and return its index number; if not found, return -1
S.rindex(substring [,start [,end]])	Find the last (rightmost) occurrence of <i>substring</i> and returns its index number; if not found, raise ValueError
S.rstrip([chars])	Remove whitespace (or <i>chars</i>) from end (right) of S
S.split([separator [,maxsplit]])	Split S using whitespace (or <i>separator</i>) and return a list of substrings
S.startswith(prefix[, start[, end]])	True if S starts with <i>prefix</i>
S.strip([chars])	Remove characters at beginning and end of S; default is whitespace characters
S.upper()	Convert S to uppercase

Modules

- Modules add new functions to python beyond the built-in ones that are available all the time. But they need to be declared at the start of the script so the processor knows it needs to include them.
- Some modules are shipped with python by default and some are written by users that you can download and include in your projects.

Modules - os

```
1  
2 import os  
3  
4 files = os.listdir()  
5  
6 print(files)  
7
```

Modules - os

```
1  
2 from os import listdir  
3  
4 files = listdir()  
5  
6 print(files)  
7
```

Challenge

- Let's loop through three structures of increasingly complex data:
 - `pratt_schedule1.py` - A list
 - `pratt_schedule2.py` - A dictionary of lists
 - `pratt_schedule3.py` - A dictionary of dictionaries which hold lists.
- Loop through them successfully and print out specific information.