

# FUNCTIONS

# Agenda

- What is Function and types of functions
- Call a function
- Significance of indentation
- How to function return a value
- Arguments in functions

# Functions

- In Python, function is a group of related statements that perform a specific task.
- Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable

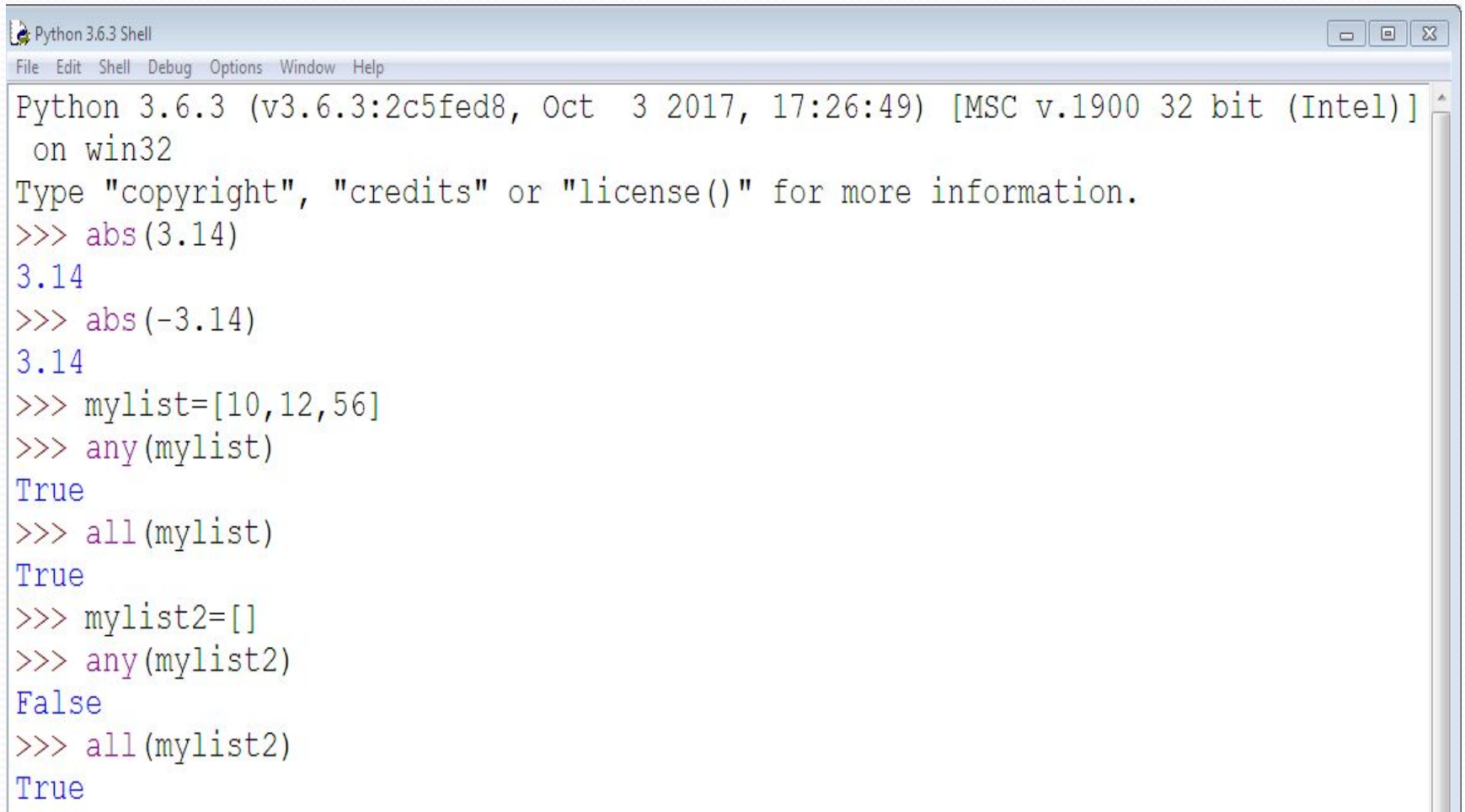
# Types of functions

- **Built-in Functions:** The Python interpreter has a number of functions that are always available for use. These functions are called built-in functions.
- example: `print()` function prints the given object to the standard output device (screen) or to the text stream file
- **User defined Functions:** Functions that we define ourselves to do certain specific task are referred as user-defined functions.

# Some Built in Functions

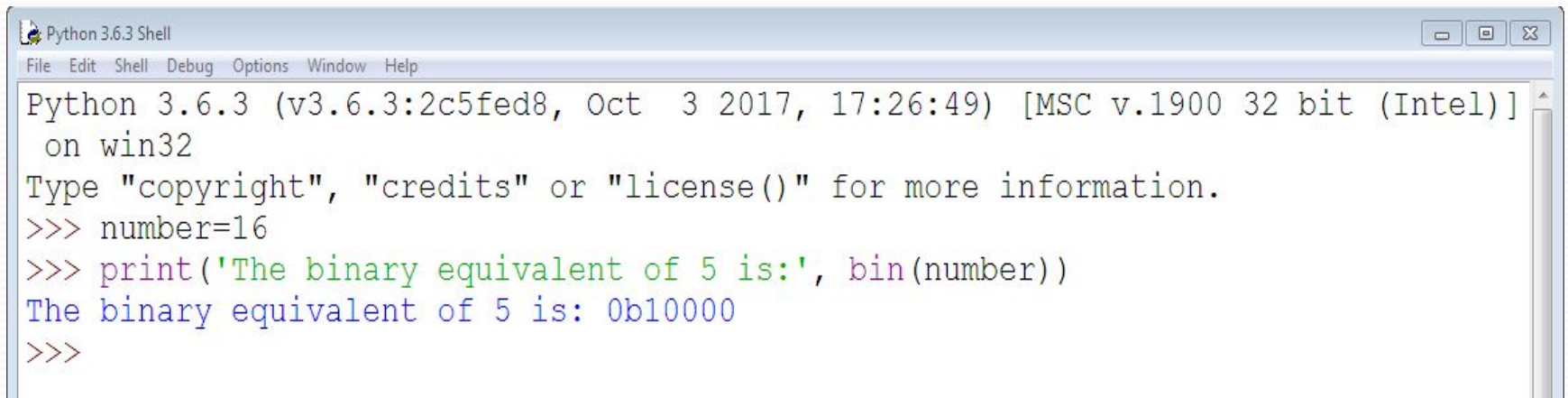
Method	Description
Python abs()	returns absolute value of a number
Python all()	returns true when all elements in iterable is true
Python any()	Checks if any Element of an Iterable is True
Python ascii()	Returns String Containing Printable Representation
Python bin()	converts integer to binary string
Python bool()	Coverts a Value to Boolean
Python bytearray()	returns array of given byte size
Python bytes()	returns immutable bytes object

# Continue....

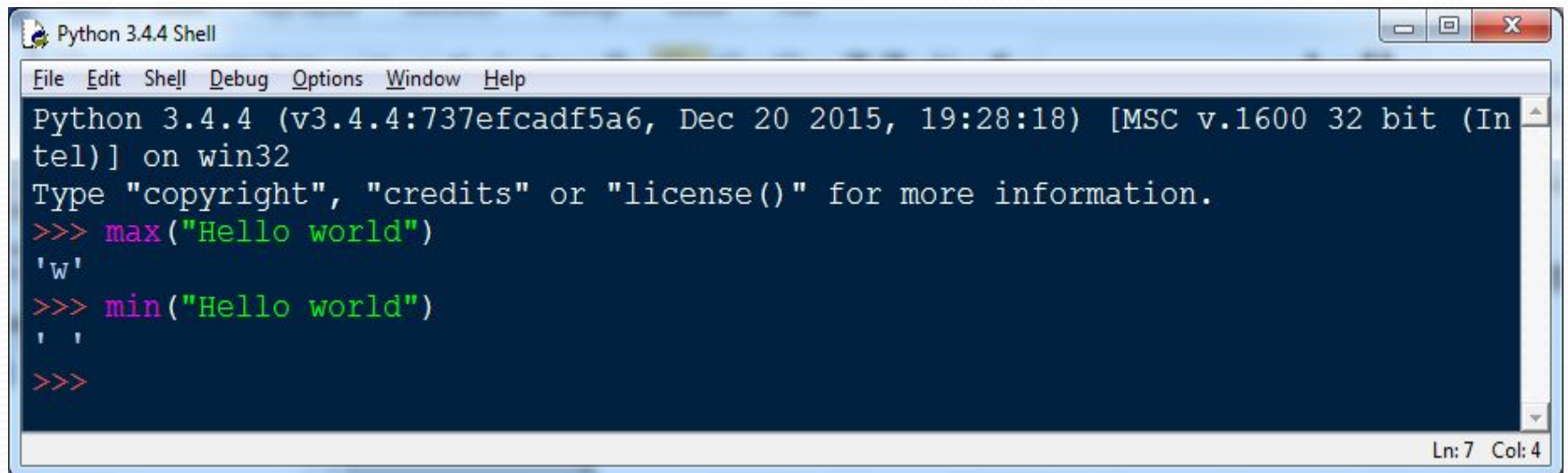
A screenshot of a Python 3.6.3 Shell window. The window has a title bar with the text 'Python 3.6.3 Shell' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area of the window displays the following text:

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> abs(3.14)
3.14
>>> abs(-3.14)
3.14
>>> mylist=[10,12,56]
>>> any(mylist)
True
>>> all(mylist)
True
>>> mylist2=[]
>>> any(mylist2)
False
>>> all(mylist2)
True
```

# Continue...

A screenshot of a Python 3.6.3 Shell window. The window has a title bar with the text 'Python 3.6.3 Shell' and standard Windows window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the Python 3.6.3 startup message, followed by user input and output. The code is color-coded: '>>>' is red, 'number=16' is blue, 'print' is green, and 'bin' is purple. The output 'The binary equivalent of 5 is: 0b10000' is in blue. The window has a vertical scrollbar on the right side.

```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> number=16
>>> print('The binary equivalent of 5 is:', bin(number))
The binary equivalent of 5 is: 0b10000
>>>
```

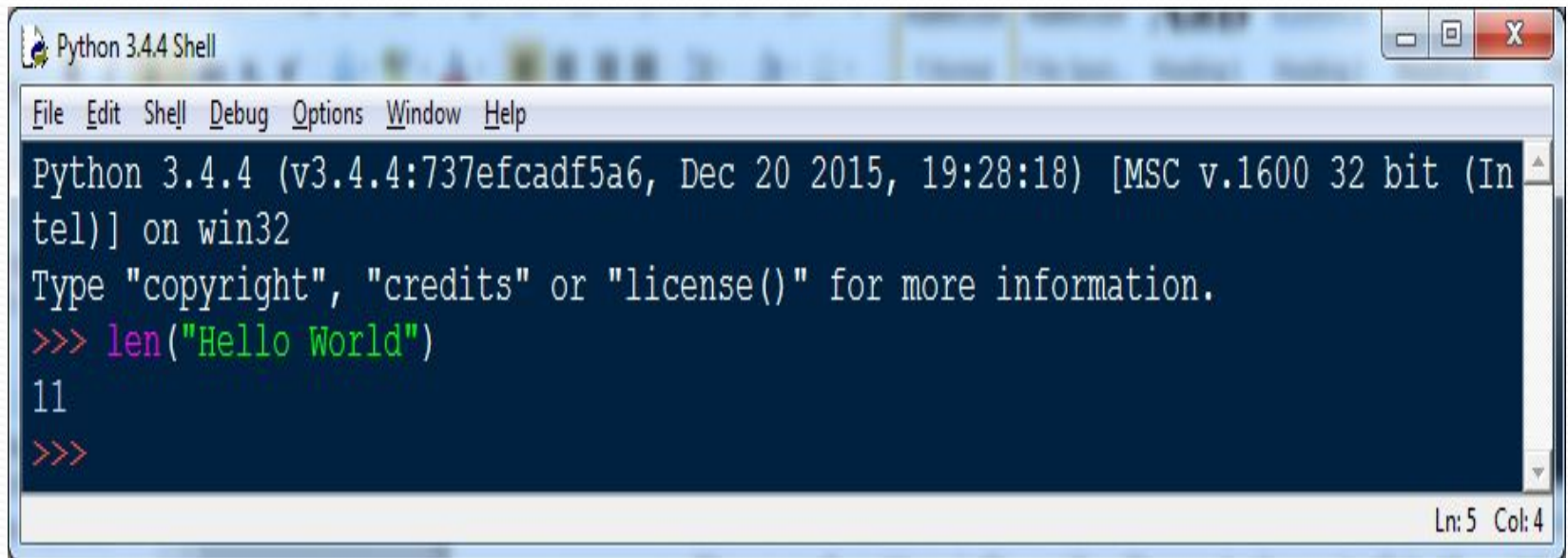
A screenshot of a Python 3.4.4 Shell window. The window has a title bar with the text 'Python 3.4.4 Shell' and standard Windows window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the Python 3.4.4 startup message, followed by user input and output. The code is color-coded: '>>>' is red, 'max' and 'min' are green, and the string 'Hello world' is blue. The output 'w' and 'l' are in blue. The window has a vertical scrollbar on the right side. At the bottom right, a status bar shows 'Ln: 7 Col: 4'.

```
Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> max("Hello world")
'w'
>>> min("Hello world")
'l'
>>>
Ln: 7 Col: 4
```



# Continue....

- **len**: Return the length (the number of items) of an object. The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).



```
Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len("Hello World")
11
>>>
```

Ln: 5 Col: 4



# Continue....

- **Round**: Return number rounded to *ndigits* precision after the decimal point. If *ndigits* is omitted or is None, it returns the nearest integer to its input.
- **Sorted**: Return a new sorted list from the items in iterable
- **Power**: The `pow()` method returns *x* to the power of *y*
- **Sum**: `sum(iterable)` sums the numeric values in an iterable such as a list, tuple, or set. `sum(iterable)` does not work with strings because you can't do math on strings (when you add two strings you are really using an operation called concatenation).

# Continue.....

- **Help**: The **help()** function is your new best friend. Invoke the built-in help system on any object and it will return usage information on the object.

```
>>> round(1.2)
1
>>> a=[5,2,1,5,7,27]
>>> sorted(a)
[1, 2, 5, 5, 7, 27]
>>> pow(2,4)
16
>>> pow(2,5)
32
>>> sum(a)
47
>>> help(len)
Help on built-in function len in module builtins:

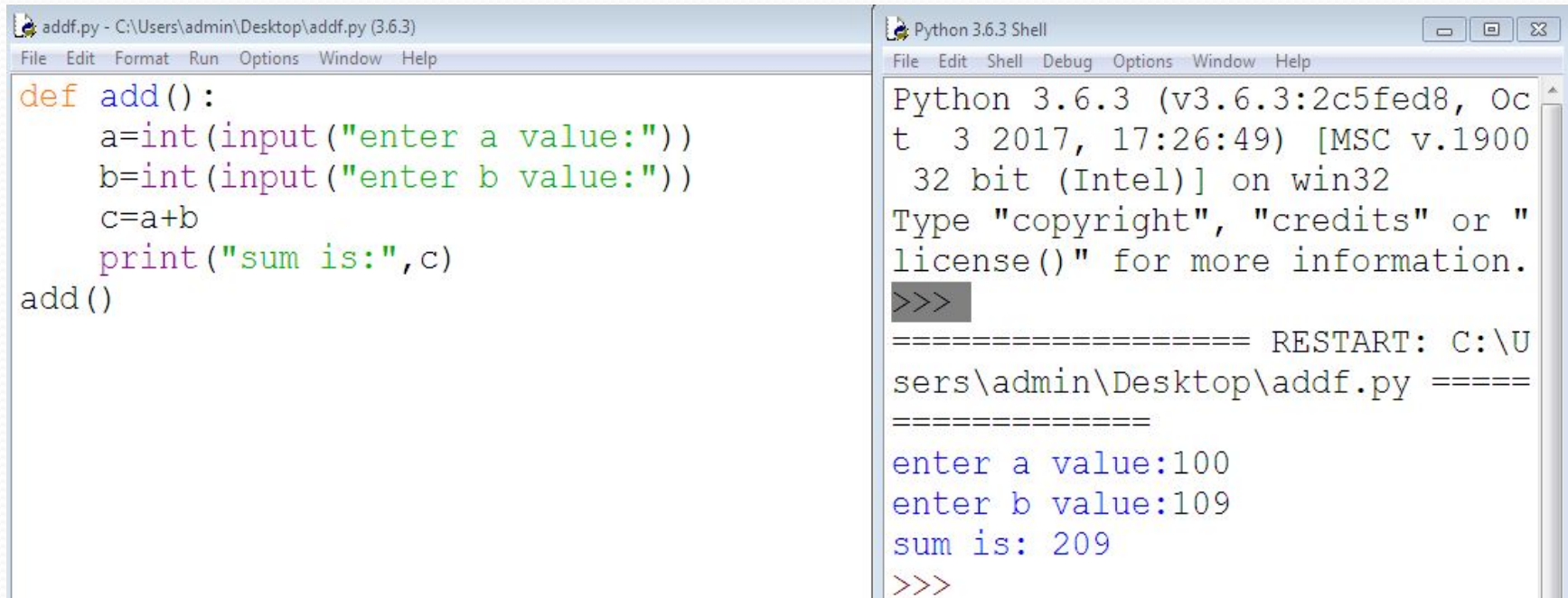
len(obj, /)
    Return the number of items in a container.

>>>
```

# User defined Functions

- **Syntax:**

def function\_name(parameters):  
 statement(s)



The image shows a screenshot of a Python IDE with two windows. The left window, titled 'addf.py - C:\Users\admin\Desktop\addf.py (3.6.3)', contains the following code:

```
def add():  
    a=int(input("enter a value:"))  
    b=int(input("enter b value:"))  
    c=a+b  
    print("sum is:",c)  
add()
```

The right window, titled 'Python 3.6.3 Shell', shows the output of running the script. It displays the Python version and system information, followed by a restart message and the execution of the 'add' function with inputs 100 and 109, resulting in a sum of 209.

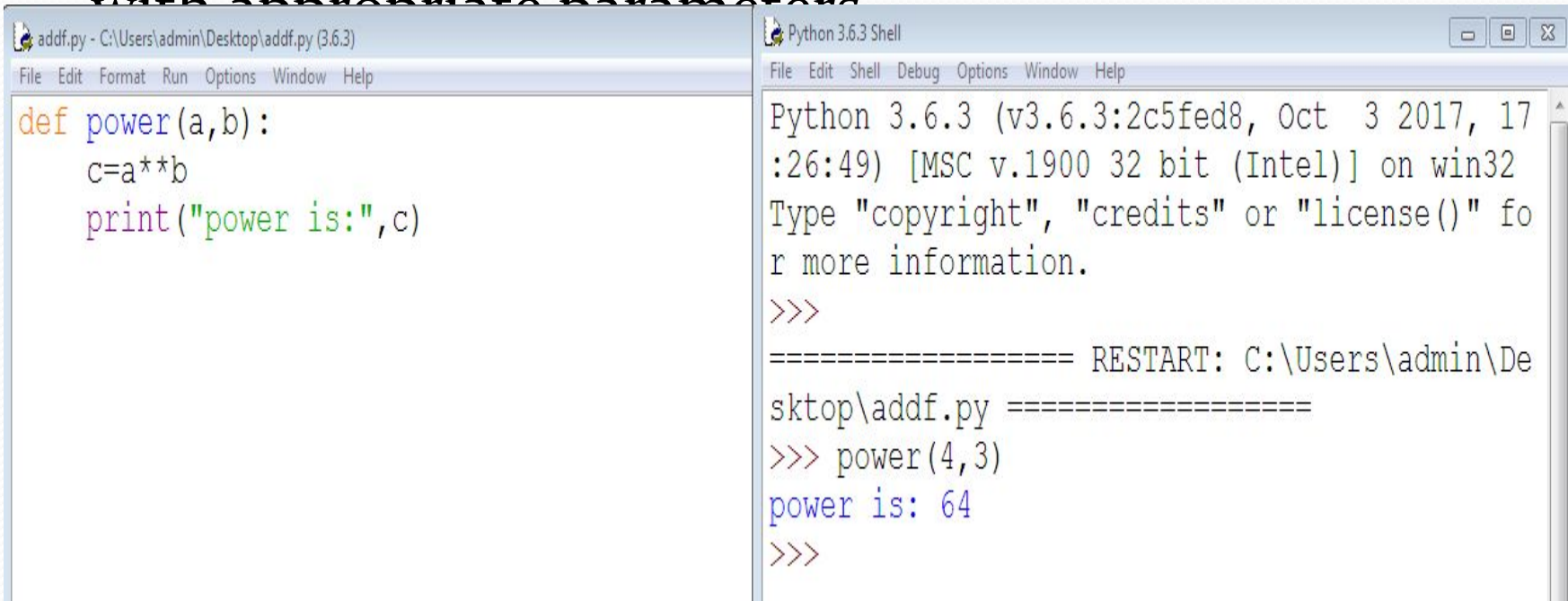
```
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900  
32 bit (Intel)] on win32  
Type "copyright", "credits" or "  
license()" for more information.  
>>>  
===== RESTART: C:\U  
sers\admin\Desktop\addf.py =====  
enter a value:100  
enter b value:109  
sum is: 209  
>>>
```

# Advantages of user defined functions

- User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
- If repeated code occurs in a program, function can be used to include those codes and execute when needed by calling that function.
- Programmers working on large project can divide the workload by making different functions.

# Call a function

- Once we have defined a function, we can call it from another function, program or even the Python prompt.
- To **call** a function we simply type the function name with appropriate parameters



The image shows two windows from a Python IDE. The left window, titled 'addf.py - C:\Users\admin\Desktop\addf.py (3.6.3)', contains the following code:

```
def power(a,b):  
    c=a**b  
    print("power is:",c)
```

The right window, titled 'Python 3.6.3 Shell', shows the execution of the script. It displays the Python version and system information, followed by a restart message and the output of the function call:

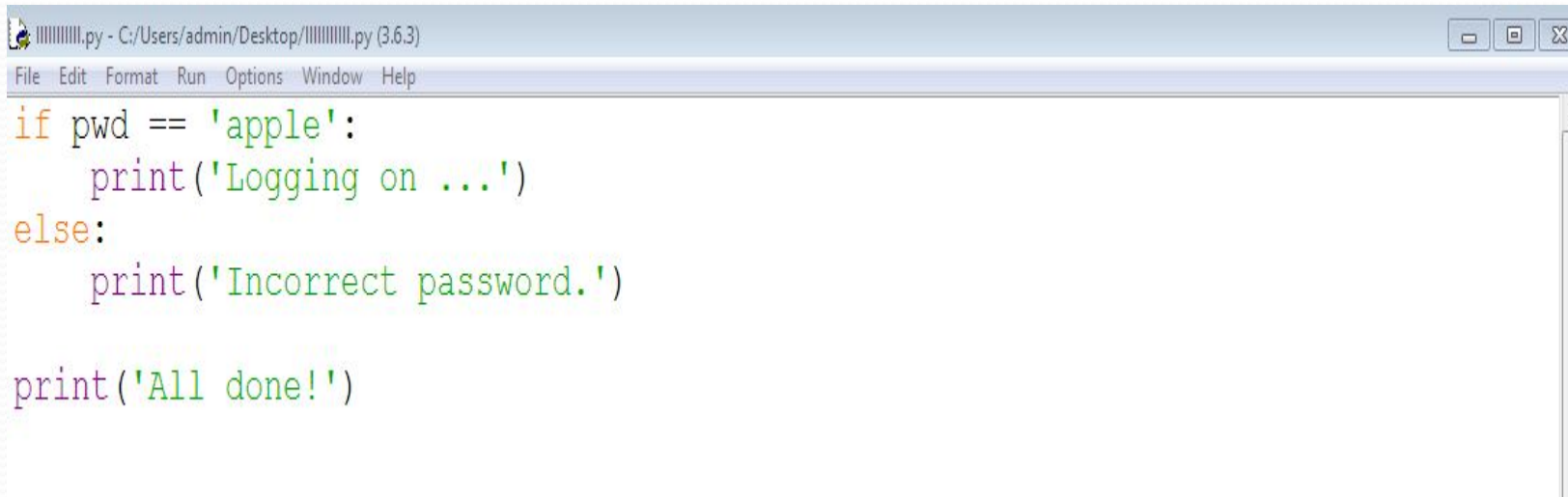
```
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:\Users\admin\Desktop\addf.py =====  
>>> power(4,3)  
power is: 64  
>>>
```

# Significance of Indentation

- Indentation in a writing context typically describes the space between the page border and the start of the text.
- In a programming context, it describes the space between the edge of the editor and the start of the code.
- Python is unusual among programming languages by having what is sometimes called "significant whitespace", Where the amount of space before the code starts affects the structure of the program.

# Code Blocks and Indentation

- Consider the if-statement from our simple password-checking program:

A screenshot of a Python IDE window titled 'IIIIIIII.py - C:/Users/admin/Desktop/IIIIIIII.py (3.6.3)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code editor displays the following Python code:

```
if pwd == 'apple':  
    print('Logging on ...')  
else:  
    print('Incorrect password.')  
  
print('All done!')
```

The code uses color coding: 'if' is orange, 'print' is purple, and strings are green. The indentation of the code block is clearly visible.

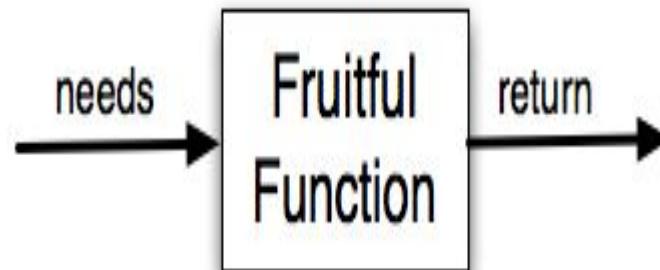


# Continue.....

- The lines `print('Logging on ...')` and `print('Incorrect password.')` are two separate code blocks.
- To indicate a block of code in Python, you must indent each line of the block by the same amount.
- The two blocks of code in our example if-statement are both indented four spaces, which is a typical amount of indentation for Python.
- In Python, it is required for indicating what block of code a statement belongs to. For instance, the final `print('All done!')` is not indented, and so is not part of the else-block.

# How to function return a value

- Functions that return values are sometimes called **fruitful functions**.
- In many other languages, a function that doesn't return a value is called a **procedure**.
- But we will stick here with the Python way of also calling it a function, or if we want to stress it, a non-fruitful function.



# Example

- The square function will take one number as a parameter and return the result of squaring that number



```
return.py - C:/Users/Admin/python programs/return.py (3.4.4)
File Edit Format Run Options Window Help
def square(x):
    y = x * x
    return y
tosquare = 10
result = square(tosquare)
print("The result of" + str(tosquare) + "squared is " + str(result))
Ln: 8 Col: 0
```

```
Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Admin/python programs/return.py =====
The result of10squared is 100
>>>
Ln: 6 Col: 4
```

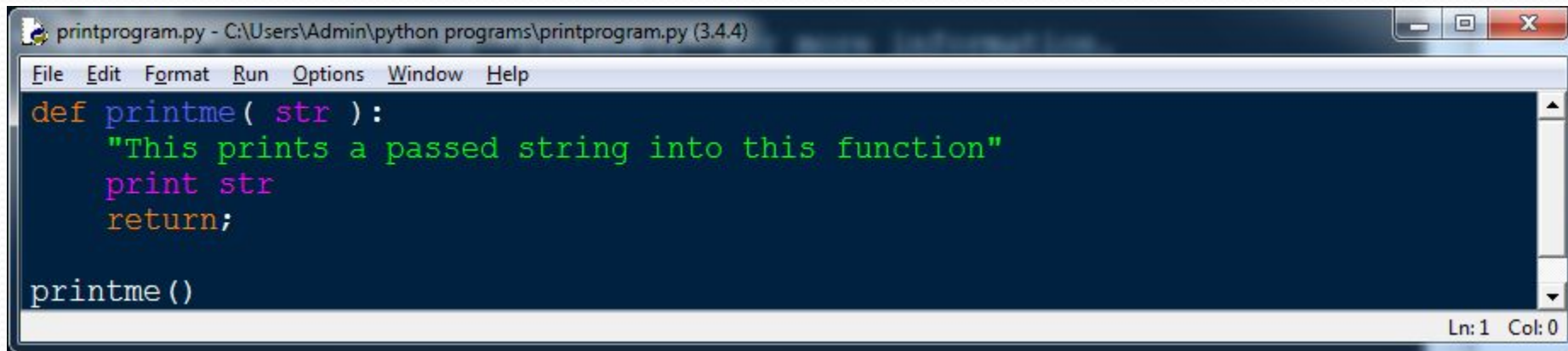
# Arguments in Functions

- You can call a function by using the following types of formal arguments:
- 1.Required arguments
- 2. Keyword arguments
- 3.Default arguments
- 4.Variable-length arguments

# Required arguments

- Required arguments are the arguments passed to a function in correct positional order.
- Here, the number of arguments in the function call should match exactly with the function definition.
- To call the function `printme()`, you definitely need to pass one argument, otherwise it gives a syntax error as follows

# Required arguments

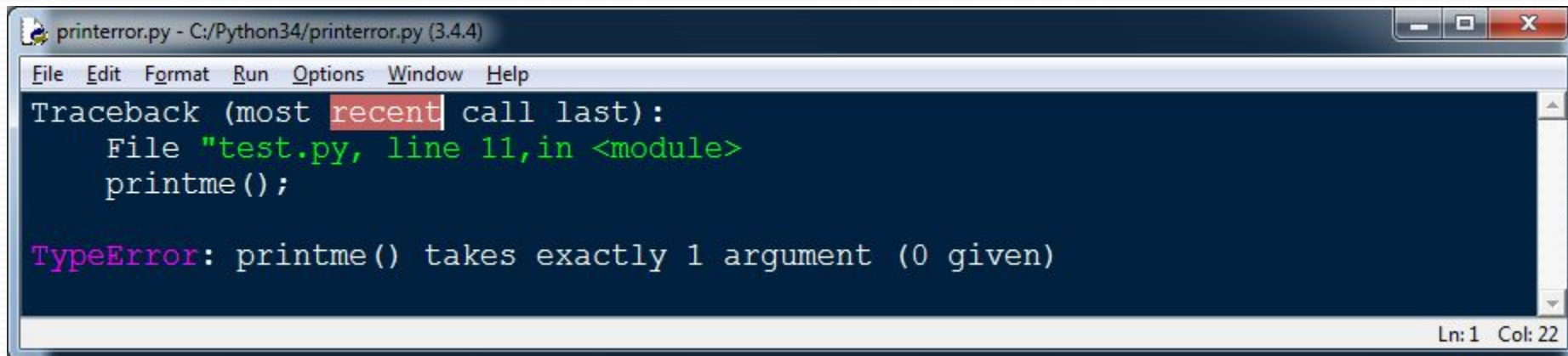


A screenshot of a Python IDE window titled "printprogram.py - C:\Users\Admin\python programs\printprogram.py (3.4.4)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code editor shows a function definition: 

```
def printme( str ) :  
    "This prints a passed string into this function"  
    print str  
    return;  
  
printme()
```

 The status bar at the bottom right indicates "Ln: 1 Col: 0".

```
def printme( str ) :  
    "This prints a passed string into this function"  
    print str  
    return;  
  
printme()
```



A screenshot of a Python IDE window titled "printerror.py - C:/Python34/printerror.py (3.4.4)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code editor shows a traceback error: 

```
Traceback (most recent call last):  
  File "test.py, line 11, in <module>  
    printme();  
  
TypeError: printme() takes exactly 1 argument (0 given)
```

 The status bar at the bottom right indicates "Ln: 1 Col: 22".

```
Traceback (most recent call last):  
  File "test.py, line 11, in <module>  
    printme();  
  
TypeError: printme() takes exactly 1 argument (0 given)
```

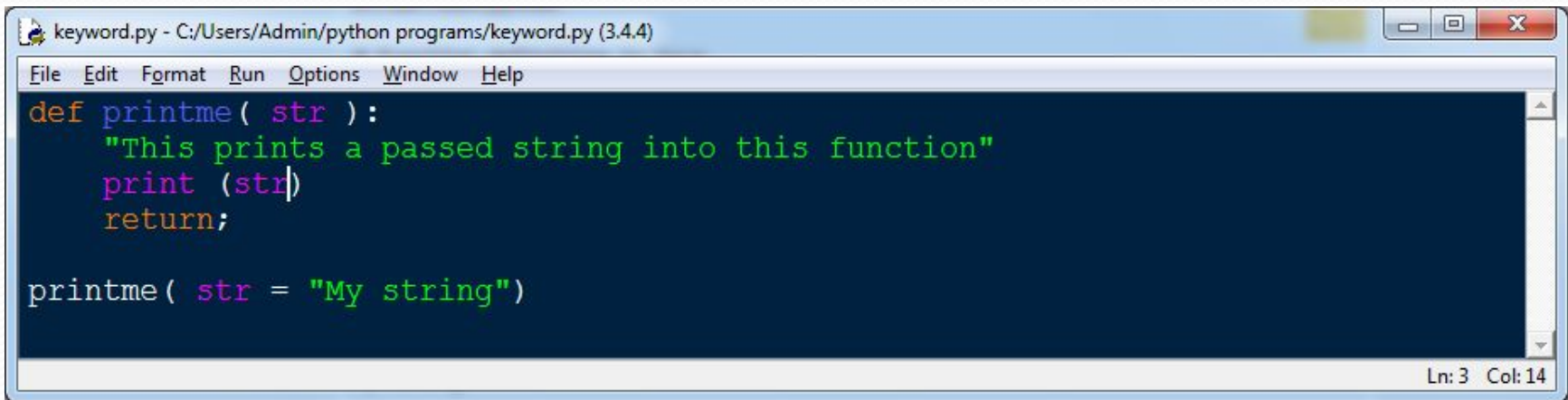
# Keyword arguments

- Keyword arguments are related to the function calls.  
When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.



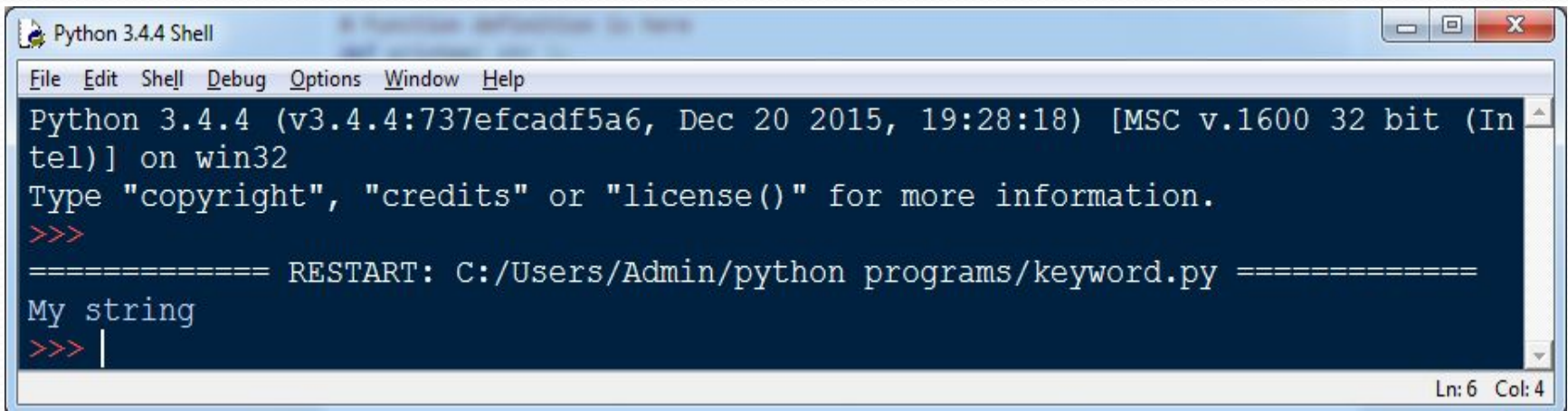
# Continue....

- You can also make keyword calls to the printme() function in the following ways –



```
keyword.py - C:/Users/Admin/python programs/keyword.py (3.4.4)
File Edit Format Run Options Window Help
def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return;

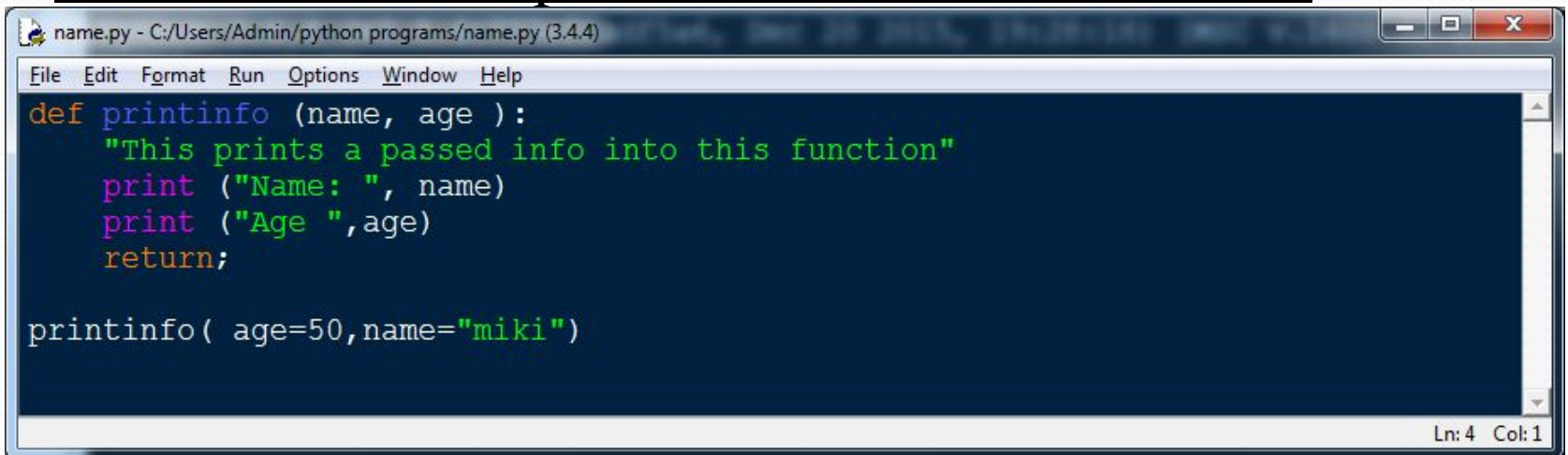
printme( str = "My string")
Ln: 3 Col: 14
```



```
Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/Admin/python programs/keyword.py =====
My string
>>> |
Ln: 6 Col: 4
```

# Continue...

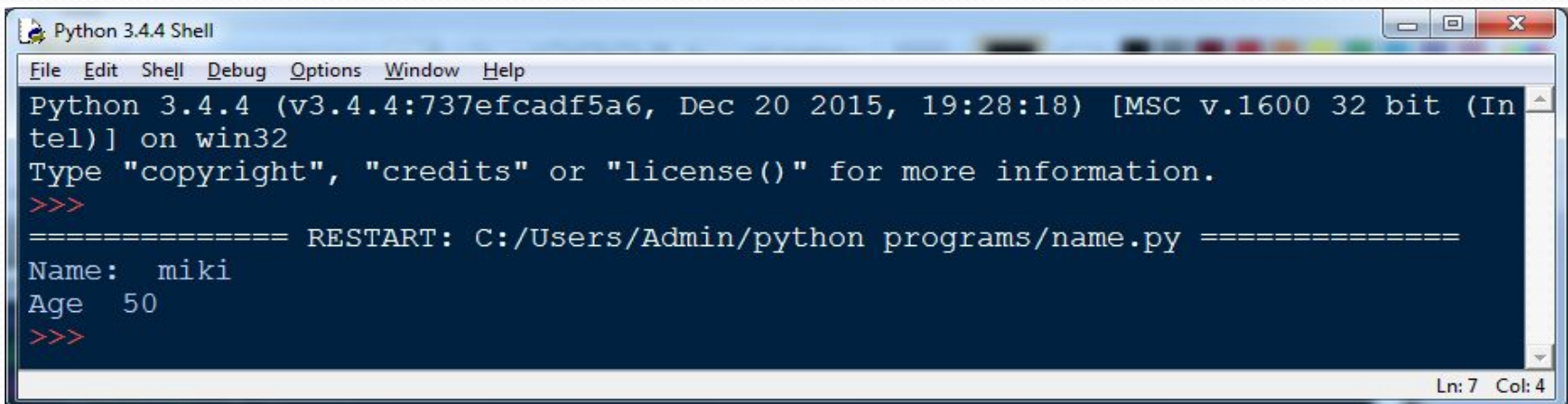
- The following example gives more clear picture. Note that the order of parameters does not matter



A screenshot of a Python IDE window titled "name.py - C:/Users/Admin/python programs/name.py (3.4.4)". The window contains the following Python code:

```
def printinfo (name, age ):  
    "This prints a passed info into this function"  
    print ("Name: ", name)  
    print ("Age ",age)  
    return;  
  
printinfo( age=50,name="miki")
```

The status bar at the bottom right indicates "Ln: 4 Col: 1".



A screenshot of a Python 3.4.4 Shell window titled "Python 3.4.4 Shell". The window shows the output of running the script:

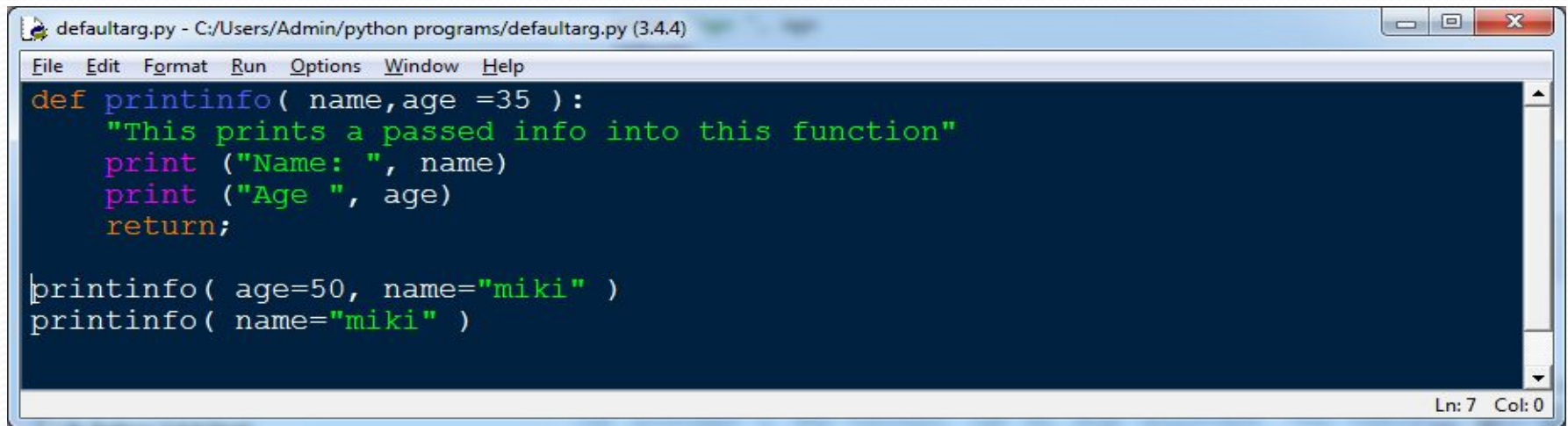
```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/Admin/python programs/name.py =====  
Name: miki  
Age 50  
>>>
```

The status bar at the bottom right indicates "Ln: 7 Col: 4".

# Default arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.
- The following example gives an idea on default arguments, it prints default age if it is not passed

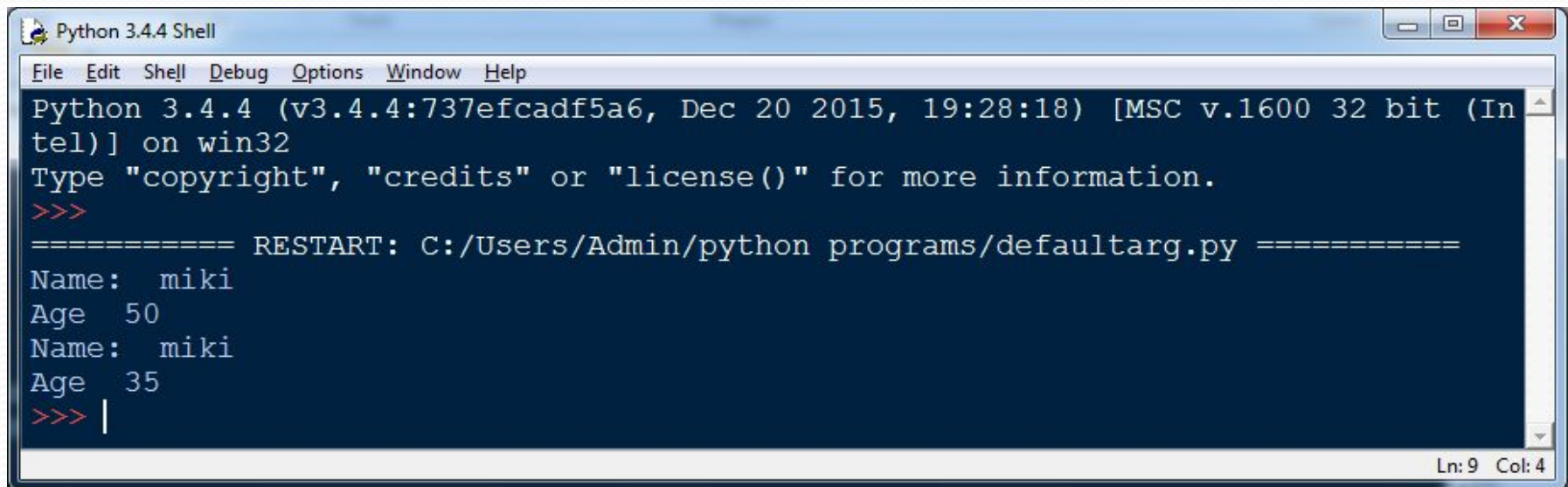
# Continue...



A screenshot of a Python IDE window titled "defaultarg.py - C:/Users/Admin/python programs/defaultarg.py (3.4.4)". The window has a menu bar with File, Edit, Format, Run, Options, Window, and Help. The code editor shows a function definition and two function calls. The status bar at the bottom right indicates "Ln: 7 Col: 0".

```
def printinfo( name,age =35 ):
    "This prints a passed info into this function"
    print ("Name: ", name)
    print ("Age ", age)
    return;

printinfo( age=50, name="miki" )
printinfo( name="miki" )
```



A screenshot of a Python 3.4.4 Shell window titled "Python 3.4.4 Shell". The window has a menu bar with File, Edit, Shell, Debug, Options, Window, and Help. The shell displays the Python version and build information, followed by a prompt to type "copyright", "credits", or "license()". It then shows the execution of the script "C:/Users/Admin/python programs/defaultarg.py", which prints the output of the function calls. The status bar at the bottom right indicates "Ln: 9 Col: 4".

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Admin/python programs/defaultarg.py =====
Name: miki
Age 50
Name: miki
Age 35
>>> |
```

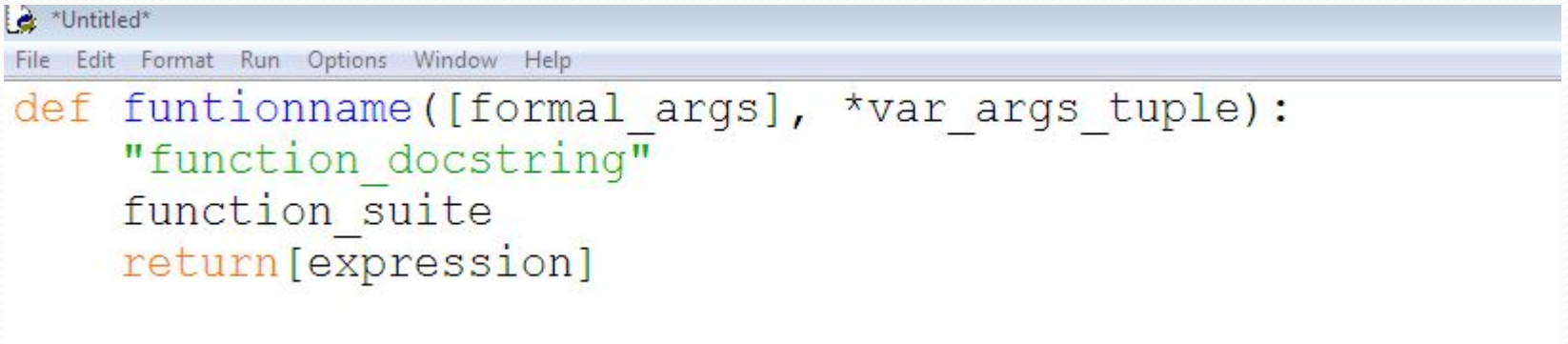
# Variable length arguments

- You may need to process a function for more arguments than you specified while defining the function
- These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.



# Continue....

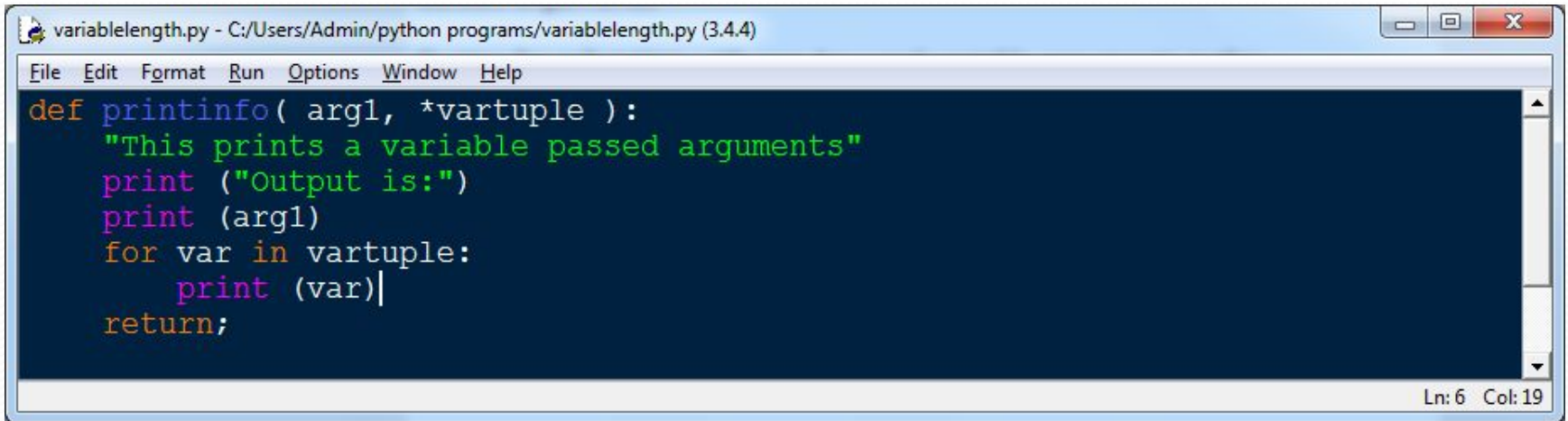
- Syntax for a function with non-keyword variable arguments



```
def funtionname([formal_args], *var_args_tuple):  
    "function_docstring"  
    function_suite  
    return [expression]
```

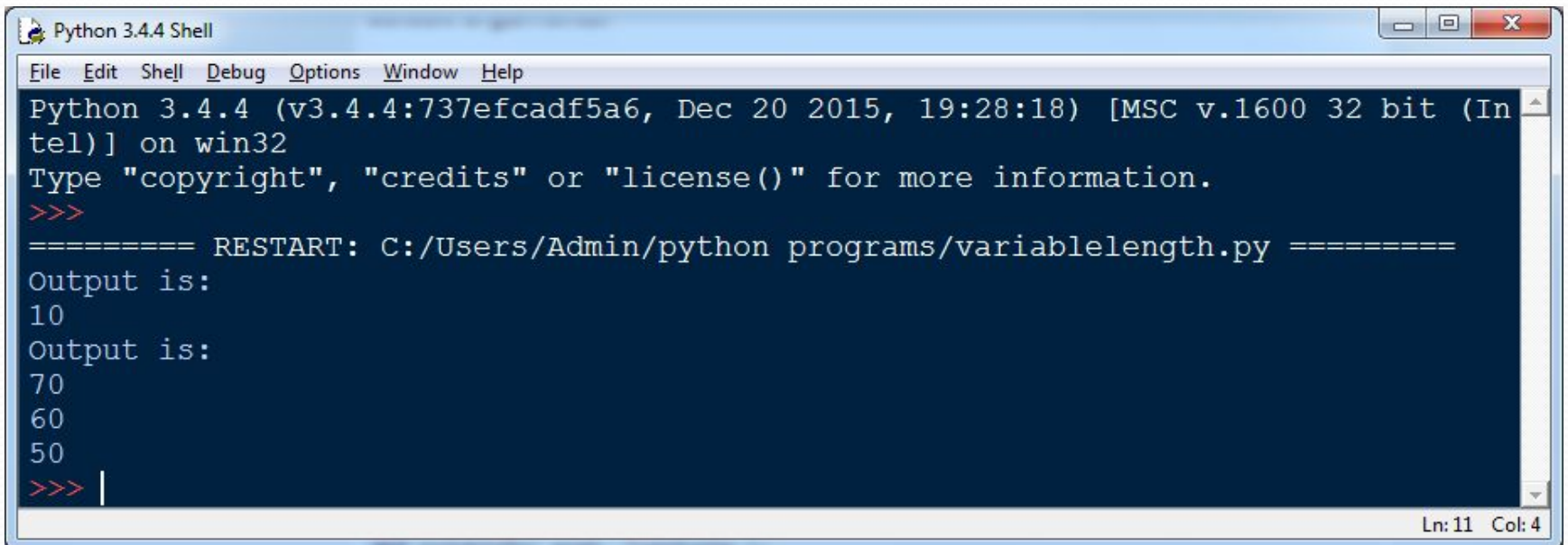
An asterisk (\*) is placed before the variable name that holds the values of all no keyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call

# Continue.....



The screenshot shows a Python IDE window with a menu bar (File, Edit, Format, Run, Options, Window, Help) and a dark blue code editor. The code defines a function `printinfo` that takes `arg1` and a variable-length tuple `*vartuple`. It prints a message, `arg1`, and each element of `vartuple`. The status bar at the bottom right indicates 'Ln: 6 Col: 19'.

```
def printinfo( arg1, *vartuple ):  
    "This prints a variable passed arguments"  
    print ("Output is:")  
    print (arg1)  
    for var in vartuple:  
        print (var)|  
    return;
```



The screenshot shows a Python 3.4.4 Shell window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). It displays the output of running the script, including the Python version, architecture, and the output of the `printinfo` function. The status bar at the bottom right indicates 'Ln: 11 Col: 4'.

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/Admin/python programs/variablelength.py =====  
Output is:  
10  
Output is:  
70  
60  
50  
>>> |
```